ALAGAPPA UNIVERSITY

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

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

KARAIKUDI – 630 003

Directorate of Distance Education

BCA

V - Semester

101 53

OPERATING SYSTEMS

Authors:

VIKAS® PUBLISHING HOUSE PVT. LTD.
E-28, Sector-8, Noida - 201301 (UP)
Phone: 0120-4078900 • Fax: 0120-4078999
Regd. Office: A-27, 2nd Floor, Mohan Co-operative Industrial Estate, New Delhi 1100 44
• Website: www.vikaspublishing.com • Email: helpline@vikaspublishing.com

Work Order No. AU/DDE/DE-12-02/Preparation and Printing of Course Materials/2020 Dated 30.01.2020 Copies - 1000

SYLLABI-BOOK MAPPING TABLE

Operating Systems

Syllabi	Mapping in Book
BLOCK - I: INTRODUCTION 1. Introduction - Definition of Operating Systems - Computer System Organization. 2. Computer System Architecture - Operating System Structure - Operating System Operations. 3. System Structures: Operating System Services - System Calls - System Programs - Operating System Design and Implementation.	**Unit 1:** Introduction to Operating System **(Pages 1-18);** **Unit 2:** Computer System Architecture **(Pages 19-42);** **Unit 3:** System Structure **(Pages 43-68)**
BLOCK - II: PROCESS CONCEPT 4. Process Concept: Process Scheduling - Operations on Processes - Inter Process Communication. 5. Process Scheduling: Scheduling Concepts - Scheduling Criteria - Scheduling Algorithms - Multiple Processor Scheduling.	**Unit 4:** Processes in Operating System **(Pages 69-92);** **Unit 5:** Process Scheduling **(Pages 93-120)**
BLOCK - III: SYNCHRONIZATION 6. Synchronization: The Critical Section Problem - Synchronization Hardware - Semaphores - Classic Problems of Synchronization - Monitors. 7. Deadlocks: Deadlocks Characterization - Methods for Handling Deadlocks. 8. Deadlock Prevention - Avoidance - Detection -Recovery from Deadlock.	**Unit 6:** Process Synchronization **(Pages 121-146);** **Unit 7:** Deadlocks **(Pages 147-154);** **Unit 8:** Deadlock Prevention **(Pages 155-172)**
BLOCK - IV: MEMORY MANAGEMENT 9. Memory Management Strategies: Swapping - Contiguous Memory Allocation - Paging - Segmentation.	**Unit 9:** Memory Management Strategies **(Pages 173-210)**

BLOCK - V: FILE SYSTEM

10. File Concept - Access Methods - Directory.
11. Structure - File System Mounting - File Sharing - Protection.
12. Implementing File Systems: File System Structure - File System Implementation.
13. Directory Implementation - Allocation Methods - Free Space Management.
14. Secondary Storage Structure: Overview of Mass Storage Structure - Disk Structure - Disk Attachment - Disk Scheduling - Disk Management.

Unit 10: File System, Access Methods and Directory **(Pages 211-226);**
Unit 11: File Structure **(Pages 227-246);**
Unit 12: Implementing File Systems **(Pages 247-254);**
Unit 13: Directory Implementation **(Pages 255-270);**
Unit 14: Secondary Storage Structure **(Pages 271-304)**

CONTENTS

INTRODUCTION

NOTES

An Operating System (OS) is system software that manages computer hardware, software resources, and provides common services for computer programs. Characteristically, an Operating System or OS is referred as an interface between a computer user and computer hardware. An operating system is a software which typically performs all the basic tasks, such as file management, memory management, process management, handling input and output, and controlling peripheral devices, for example the disk drives and the printers. Some popular Operating Systems include Linux Operating System, Windows Operating System, VMS, OS/400, AIX, z/OS, etc.

As per the definition, "An Operating System or OS is a program that acts as an interface between the user and the computer hardware and controls the execution of all kinds of programs".

There are various types of operating systems which are precisely scheduled to perform different specified tasks. Time-sharing operating systems, for example have scheduled tasks for efficient use of the system and may also include accounting software for cost allocation of processor time, mass storage, printing, and other resources. For hardware functions, such as input and output and memory allocation, the operating system acts as an intermediary between programs and the computer hardware, although the application code is usually executed directly by the hardware and frequently makes system calls to an operating system function or is interrupted by it. Operating systems are found on many devices that contain a computer – from cellular phones and video game consoles to web servers and supercomputers. Consequently, an operating system is the most essential and vital part of any computer system.

Process scheduling is an operating system task that schedules processes of different states like ready, waiting, and running. Process scheduling allows operating system to allocate a time interval of CPU execution for each process. Another important reason for using a process scheduling system is that it keeps the CPU (Central Processing Unit) busy all the time.

In concurrent computing, a deadlock is a state in which each member of a group waits for another member, including itself, to take action, such as sending a message or more commonly releasing a lock. Deadlock is a common problem in multiprocessing systems, parallel computing, and distributed systems, where software and hardware locks are used to arbitrate shared resources and implement process synchronization. In an operating system, a deadlock occurs when a process or thread enters a waiting state because a requested system resource is held by another waiting process, which in turn is waiting for another resource held by another waiting process. If a process is unable to change its state indefinitely because the resources requested by it are being used by another waiting process, then the system is said to be in a deadlock. Deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.

This book, *Operating Systems*, is divided into five blocks, which are further subdivided into fourteen units. The topics discussed include operating systems, computer system organization, computer system architecture, operating system structure and operations, operating system services, operating system design and implementation, process scheduling, inter process communication, scheduling criteria, scheduling algorithms, synchronization, semaphores, deadlocks, deadlocks characterization, deadlock prevention, memory management strategies, swapping, contiguous memory allocation, paging, segmentation, file concept and access methods, directory, file system mounting, file sharing and protection, file system structure, file system implementation, directory implementation, secondary storage structure, overview of mass storage structure and disk management.

The book follows the Self-Instructional Mode (SIM) wherein each unit begins with an 'Introduction' to the topic. The 'Objectives' are then outlined before going on to the presentation of the detailed content in a simple and structured format. 'Check Your Progress' questions are provided at regular intervals to test the student's understanding of the subject. 'Answers to Check Your Progress Questions', a 'Summary', a list of 'Key Words', and a set of 'Self-Assessment Questions and Exercises' are provided at the end of each unit for effective recapitulation. This book provides a good learning platform to the people who need to be skilled in the area of operating system functions. Logically arranged topics, relevant examples and illustrations have been included for better understanding of the topics.

BLOCK - I
INTRODUCTION

NOTES

UNIT 1 INTRODUCTION TO OPERATING SYSTEM

Structure

1.0 INTRODUCTION

An Operating System (OS) is an interface between a computer user and computer hardware. An operating system is a software which performs all the basic tasks like file management, memory management, process management, handling input and output, and controlling peripheral devices, such as disk drives and printers. Some popular Operating Systems include Linux Operating System, Windows Operating System, VMS, OS/400, AIX, z/OS, etc. Operating systems have evolved from slow and expensive systems to present-day technology where computing power has reached exponential speeds and relatively inexpensive costs.

In the beginning, computers were manually loaded with program code to control computer functions and process code related to business logic. The salient points about the Computer System Organisation are 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 and the device controllers use an interrupt to inform the CPU that I/O operation is completed.

In this unit, you will study the basic definition and evolution of operating system, computer system organization.

1.1 OBJECTIVES

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

- Understand the basic concept of operating system
- Analyse the concept of user and system view of an operating system
- Discuss the evolution of operating system
- Compute the significance of computer system organization
- Explain the I/O structure of an operating system

1.2 DEFINITION OF OPERATING SYSTEM

1.2.1 Definition

In simple terms, an operating system is defined as the most essential and indispensable program that is running at all times on the computer (usually called the **kernel**). It is a program that acts as an interface between the computer users and the computer hardware (Refer Figure 1.1). It manages the computer hardware and controls, and coordinates the use of hardware among various application programs. An operating system also provides a platform on which the various computer resources, such as hardware, software and the data can be acceptably and efficiently run in order to perform basic tasks.

Functions of Operating System

Essentially, there are two basic objectives of an operating system: To provide convenience and efficiency to the user who is interacting with the hardware. An operating system is designed in such a way that it makes the computer system more convenient to use and allows the system to use its resources to achieve maximum efficiency thereby, aiding in the successful execution of desired function. Some operating systems are designed for convenience (for example, PC or Personal Computer operating systems), some for efficiency (for example, mainframe operating systems), and some for the combination of both.

NOTES

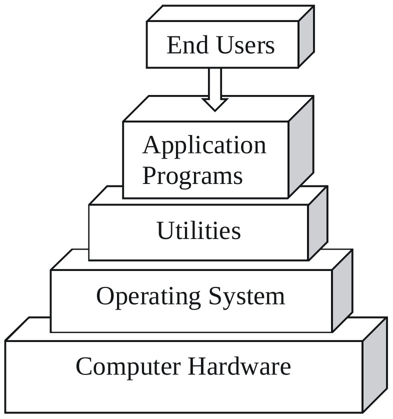


Fig. 1.1 *Components of a Computer System*

An operating system resembles the working pattern of a government. Just as a government does not perform any useful function by itself, but it provides an environment for the other programs so that they can do useful work with optimal performance. There are two viewpoints from which we can vividly understand the role of an operating system: the user point of view and the system point of view.

User View

In case of a stand-alone environment, where a single user sits in front of a personal computer, an operating system is designed basically for the **ease of use** and some attention is also paid to the **system performance**. However, since these systems are fabricated with the intention to facilitate the single user to monopolize the resources, there is no sharing of hardware and software among multiple users. Therefore, no attention is paid to resource utilization.

In case of a networked environment, where multiple users share resources and may exchange information, an operating system is designed to make the optimal use of **resources.** In this case, an operating system ensures the efficient management of the available processor, time, memory and I/O devices, and no individual user tries to monopolize the system resources. In case, the various users are connected to a **mainframe** or a **minicomputer** via their terminals, no attention is paid to usability of individual systems. However, in case the users are connected to the **servers** via their **workstations**, a compromise between individual usability and resource utilization is made while designing an operating system.

In case of handheld systems, the operating system is basically designed for individual usability and manoeuvrability as these systems are mostly stand-alone units for individual users. Finally, the computers which have little or no user view, such as embedded systems, the operating system for such systems is basically designed to ensure that these system will run without user intervention.

System View

As discussed earlier, the computer system consists of many resources, such as CPU time, memory and I/O devices, which are required to solve a computing problem. It is the responsibility of operating system to provide a conducive environment for effective management of these resources and allocate them to various programs and users in a way such that the computer system can execute a progam in an efficient, and fair manner. Thus, from the system's point of view, an operating system primarily acts as a **resource allocator**.

An operating system also acts as a **control program** or **an interface** that manages the execution of user programs to avoid possible errors and improper use of computer system. It also monitors the performance of the I/O devices and their operations.

Check Your Progress

1. Define operating system.
2. What are the two viewpoints from which we can understand the role of an operating system?
3. How an operating system acts as a control program or an interface?

1.3 EVOLUTION OF OPERATING SYSTEM

The operating system may process its work serially or concurrently. That is, it can dedicate all the computer resources to a single program until the program finishes or can dynamically assign the resources to various currently active programs. The execution of multiple programs in an interleaved manner is known as *multiprogramming*. In this section, we will discuss how operating systems have evolved from serial processing to multiprogramming systems.

Serial Processing

During the late 1940s to the mid 1950s, there were no operating systems. Programmers used to interact directly with the computer hardware by writing programs in machine language. The programs and the data were entered into the computer with the help of some input device such as a card reader. In addition to input devices, the machines also consisted of display lights that were used to indicate an error condition in case an error occurred during any operation. If the execution of programs were completed successfully, the output appeared (after minutes, hours, or days) in printed form using the line printers attached to the machine.

Whenever a programmer needed to operate the computer system, he had to first reserve the machine time by signing up for a block of time on a hardcopy sign-up sheet. After signing up, the programmer used to enter the machine room and spend the desired block of time working on the system. Once the desired block of time of the programmer was over, the next programmer in the queue was supposed to perform the same procedure. Thus, all the users were allowed to access the computer sequentially (one after the other), hence this was known as **serial processing**.

The main problem associated with serial processing was that in some cases, a programmer might sign up for two hours but finished his work in one and half hour. In such situations, the computer processing time would be wasted, since no other programmer was allowed to enter the machine room during that time.

With advancement in technologies, various system software tools were developed that made serial processing more efficient. These tools include language translators, loaders, linkers, debuggers, libraries of common routines, and I/O routines. Programmers could now code their programs in a programming language, which could then be translated into executable code with the help of *language translator* such as a compiler or an interpreter. The *loader* automated the process of loading executable programs into memory (it automatically transfers the program and the data from the input device to the memory). *Debuggers* assisted the programmers in detecting and examining the errors that occur during program execution (run-time errors). *Linkers* were used to link the precompiled routines with the object code of the program so that they could be executed along with the object code to produce the desired output.

Though the use of system software tools made the serial processing a bit more efficient, serial processing still resulted in low utilization of resources. Not only was user productivity low, but users had to wait for their turns.

Batch Processing

In the mid 1950s, transistors were introduced, changing the entire scenario. Computers now became more reliable and were bought by customers with the expectation that they would continue to work reliably for a long time, thus giving consumers the confidence that they could take on work on a long-term basis. These machines were named **mainframes**, and were generally operated by professional operators. However, they were so expensive that only major government agencies and big corporations could afford to buy them.

A clear distinction was made between computer operators and programmers. Programmers used to prepare a job that consisted of instructions, data and some control information about the nature of the job, and submit it to the computer operator. The jobs were generally in the form of punched cards. When the currently running job was finished, the operator would take off the output using a printer (which could be kept in another room), and the programmer could collect the output later at any time. The operator performed the same process for all the card decks submitted by the programmers. Much computer time was wasted while the operator was moving from one room to another.

To reduce this wasted time and speed up the processing, the operator used to batch together the jobs with similar requirements, and run these batches one by one. This system was known as **batch processing system**. For example, the jobs that need FORTRAN compiler can be batched together so that the FORTRAN compiler can be loaded only once to process all these jobs. Note that the jobs in a batch are independent of each other and belong to different users.

To improve resource utilization and user productivity, the first operating system was developed by General Motors for IBM 701 in the mid 1950s. This operating system was termed as **batch operating system**. Its major task was to transfer control automatically from one job to next job in the batch without the operator's intervention. This was achieved by automating the transition from execution of one job to that of the next in the batch.

Batch processing is implemented by the kernel (also known as **batch monitor**), which is the memory-resident portion of the batch operating system (Refer Figure 1.2). The rest of the memory is allocated to the user jobs one at a time.

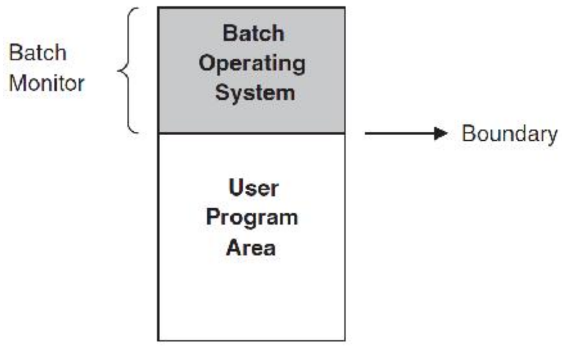


Fig. 1.2 *Memory Layout for a Batch System*

When a batch of similar jobs is submitted for processing, the batch monitor reads the card reader and loads the first job in the memory for processing. The beginning and end of each job in the batch is identified by the `JOB_START` and `JOB_END` command respectively. When the batch monitor encounters the `JOB_START` command, it starts the execution of the job, and when it encounters the `JOB_END` command, it searches for another job in the batch. Finally, when all the jobs in the batch are finished, the batch monitor waits for the next batch to be submitted by the operator. Hence, the operator intervention is required only at the time of start and end of a batch.

The main disadvantage of batch processing is that during execution, the CPU is often idle, because of the speed difference between the CPU and I/O devices. To overcome the problem of speed-mismatch, the concept of **SPOOLing** (Simultaneous Peripheral Operation On-line) came into existence. Instead of inputting the jobs from the card readers, the jobs were first copied from the punched cards to the magnetic tape. The magnetic tape was then mounted on a tape drive, and the operating system read the jobs from the input tape. Similarly, instead of getting output directly on the printer, it was sent to the magnetic tape. Once the jobs were finished, the output tape was removed and connected to the printer (which was not connected to the main computer) for printing the output. Since magnetic tapes proved to be much faster than card readers and printers, they reduced the CPU idle time by solving the problem of speed-mismatch.

With the introduction of disk technology, the batch operating system started keeping all the jobs on the disk rather that on the tapes. Disks are much faster than magnetic tapes and allow direct access, hence the problem of speed-mismatch was further reduced.

Multiprogramming

Though the batch processing system attempted to utilize the computer resources like CPU and I/O devices efficiently, it still dedicates all resources to a single job at a time. The execution of a single job cannot keep the CPU and I/O devices busy at all times because during execution, the jobs sometimes require CPU and sometimes require I/O devices but not both at one point of time. Hence, when the job is busy with CPU, the I/O devices have to wait, and when the job is busy with I/O devices, the CPU remains idle.

For example, consider two jobs P_1 and P_2, both of which require CPU time and I/O time alternatively. The serial execution of P_1 and P_2 is shown in Figure 1.4(a). The shaded boxes show the CPU activity of the jobs, and the white boxes show their I/O activity. It is clear from the figure that when P_1 is busy in its I/O activity, the CPU is idle even if P_2 is ready for execution.

The idle time of CPU and I/O devices can be reduced by employing multiprogramming which allows multiple jobs to reside in the main memory at the same time. If one job is busy with I/O devices, the CPU can pick another job and start executing it. To implement multiprogramming, the memory is partitioned into several partitions, where each partition can hold only one job. The jobs are organized in such a way that the CPU always has one job to execute. This increases the amount of CPU utilization by minimizing the CPU idle time.

The basic idea behind multiprogramming is that the operating system loads multiple jobs into the memory from the job pool on the disk. It then picks up one job from the pool and starts executing it. When this job needs to perform I/O activity, the operating system simply picks up another job and starts executing it. Again, when this job requires I/O activity, the operating system switches to third job, and so on. When the I/O activity of the job gets finished, it gets the CPU back. Therefore, as long as there is at least one job to execute, the CPU will never remain idle. The memory layout for a multiprogramming batched system is shown in Figure 1.3.

NOTES

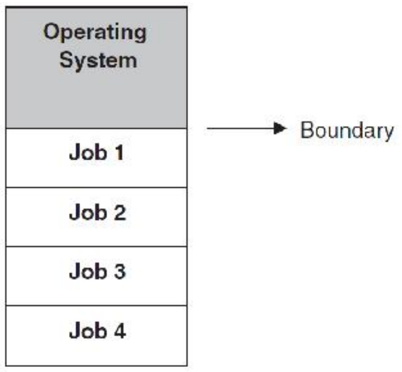
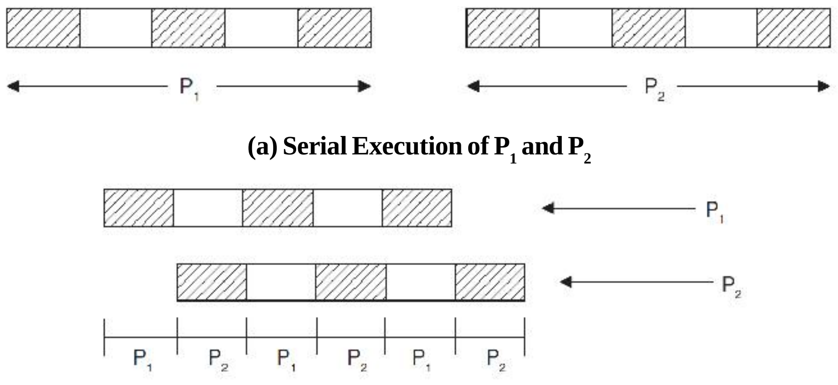


Fig. 1.3 *Memory Layout for a Multiprogramming System*

Figure 1.4(b) shows the multiprogrammed execution of jobs P_1 and P_2; both are assumed to be in memory and waiting to get CPU time. Further assume that job P_1 gets the CPU time first. When P_1 needs to perform its I/O activity, the CPU starts executing P_2. When P_2 needs to perform I/O activity, the CPU again switches to P_1, and so on. This type of execution of multiple processes is known as **concurrent execution**.



(a) Serial Execution of P_1 and P_2

(b) Multiprogrammed Execution of P_1 and P_2

Fig. 1.4 *Serial and Multiprogrammed Execution*

Note that, for sake of simplicity, we have considered the concurrent execution of only two programs P_1 and P_2. However, in real life, there are generally more than two programs that compete for system resources at any point of time. The number of jobs competing to get the system resources in multiprogramming environment is known as **degree of multiprogramming**. In general, the higher the degree of multiprogramming, more will be the resource utilization.

In multiprogrammed systems, the operating system is responsible for making decisions for the users. When a job enters the system, it is kept in the job pool on the disk which contains all those jobs that are waiting for the allocation of main memory. If there is not enough memory to accommodate all these jobs, the operating system must select which ones among them are to be loaded into the main memory. Making this decision is known as **job scheduling**. To keep multiple jobs in the main memory at the same time, some kind of memory management is required. Moreover, if multiple jobs in the main memory are ready for execution at the same time, the operating system must choose one of them. Making this decision is known as **CPU scheduling**.

The main drawback of multiprogramming systems is that the programmers have to wait for several hours to get their output. Moreover, these systems do not allow the programmers to interact with the system. To overcome these problems, an extension of multiprogramming systems, called time-sharing systems is used. In **time-sharing** (or **multitasking**) **systems**, multiple users are allowed to interact with the system through their terminals. Each user is assigned a fixed time-slot in which he or she can interact with the system. The user interacts with the system by giving instructions to the operating system or to a program using an input device such as keyboard or a mouse, and then waits for the response.

The **response time** should be short—generally within one second. The CPU in time-sharing system switches so rapidly from one user to another that each individual user gets the impression that only he or she is working on the system, even though the system is being shared by multiple users simultaneously. A typical time-sharing system is shown in Figure 1.5.

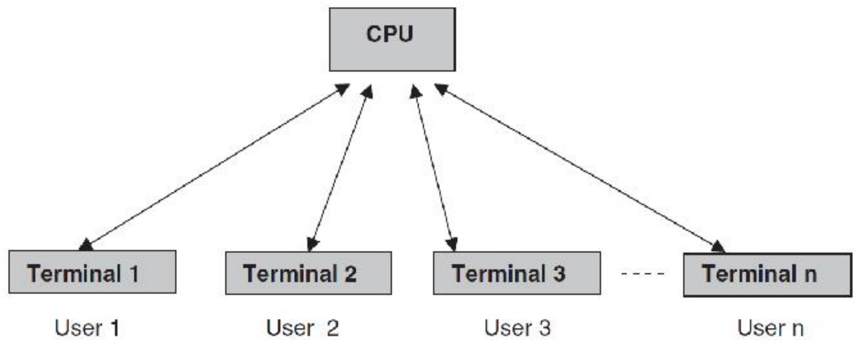


Fig. 1.5 *Time-Sharing System*

The main advantage of time-sharing systems is that they provide a convenient environment in which the users can develop and execute their programs. Unlike batch processing systems, they provide quicker response time, and allow users to debug their program interactively under the control of a debugging program. Moreover, the users are allowed to share the system resources in such a way that each user gets an impression that he or she has all the resources to himself or herself.

Though the concept of time-sharing was demonstrated in early 1960s, but since it was expensive and difficult to implement at that time, they were not in use until the early 1970s. However, these days, most of the systems are time sharing.

NOTES

Check Your Progress

4. What is the basic idea behind multiprogramming?
5. Define the purpose of linker, loader and a translator.
6. What is meant by batch processing system and batch operating system?
7. State the main disadvantage of batch processing.
8. Define degree of multiprogramming.

1.4 COMPUTER SYSTEM ORGANIZATION

One of the important job of computers is storing and managing data in a retrievable form as and when required. These days the basic constitutents that make up a computer system are: One or more processors (CPUs), several device controllers and the memory. All these components are connected through a common bus that provides access to shared memory. Each device controller acts as an interface between a particular I/O device and an operating system. Thus, a device controller plays an important role in operating a particular device. For example, the disk controller helps in operating disks, USB controller in operating mouse, keyboard and printer, graphics adapter in operating monitor, sound card in operating audio devices, and so on. In order to access the shared memory, the memory controller is also provided that synchronizes the access to the memory. The interconnection of various components via a common bus is shown in Figure 1.6.

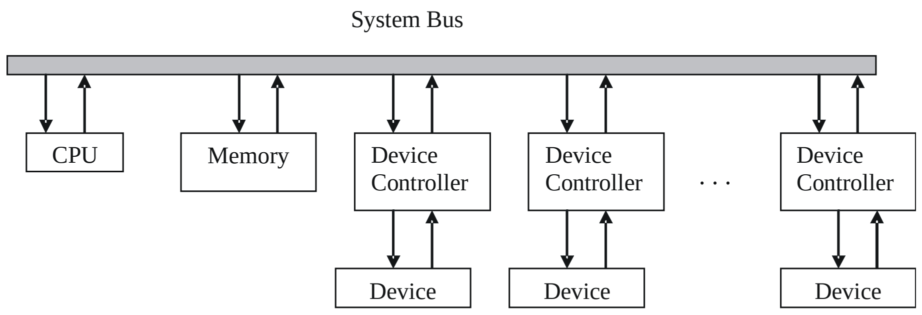


Fig. 1.6 *Bus Interconnection*

Computer System Operation

When the system originally is turned on, it runs a well-defined set of initial programs known as **bootstrap program**. The bootstrap program is typically stored in Read-Only Memory (ROM) or Electrically Erasable Programmable ROM (EEPROM). During the booting process, all the aspects of the system checking like CPU registers, device controllers and memory contents are initialized, and then an operating system is loaded into the memory. Once an operating system is loaded, the first process, such as “init” is executed and operating system then waits for some special sequence of events to occur.

The event notification is done with the help of an **interrupt** that is stimulated either by the hardware or the software. When the hardware needs to trigger an interrupt, it can do so by sending a signal to the CPU via the system bus. When the software needs to trigger an interrupt, it can do so with the help of a **system call** (or **monitor call**).

Whenever, an interrupt is fired, the CPU suspends the current task for the time being and jumps to a predefined location in the kernel's address space, which contains the starting address of the service routine for the interrupt (known as **interrupt handler**). It then executes the interrupt handler and once the execution is completed, the CPU resumes the task that it was previously doing.

To quickly handle the interrupts, a table of pointers to interrupt routines is used. The table contains the addresses of the interrupt handlers for the various devices and is generally stored in the low memory (say first 100 locations or so). The interrupt routine can be called indirectly with the help of this table. This array of addresses is known as **interrupt vector**. The interrupt vector is further indexed by a unique device number, given with the interrupt request, to provide the address of the interrupt handler for the interrupting device.

Storage Structure

Whenever, a program needs to be executed, it must be first loaded into the main memory (called **Random-Access Memory** or commonly known by the acronym **RAM**) where it is stored. RAM is the only storage area that can be directly accessed by the CPU. RAM consists of an array of memory words, where each word has its unique address. The two instructions, namely, load and store are used to interact with the RAM memory.

- The load instruction is used to move a word from the main memory to the CPU register.
- The store instruction is used to move the content of the CPU register to the main memory.

We know that a program is basically a set of instructions that a computer can read to direct an intended task. The execution of the program instructions takes place in the CPU registers, which are primarily used as temporary storage areas and have restricted storage margin. Usually, an instruction–execution cycle consists of the following steps.

(i) Whenever, the CPU needs to execute an instruction, it first fetches it from the main memory and stores it in **Instruction Register** (**IR**).

(ii) Once the instruction has been loaded into the IR, the control unit examines and decodes the fetched instruction.

(iii) After decoding the instruction, the operands (if required) are fetched from the main memory and stored in one of the internal registers.

(iv) The instruction is executed on the operands and the result is stored back to the main memory.

Ideally all the programs and data should be stored in the main memory permanently for fast execution and better system performance because RAM is the only storage area that allows direct accessibility of the stored data by the CPU. But, practically it is not possible because RAM is exorbitantly proceed and offers limited storage capacity. Secondly, it is volatile in nature, that is, the information is vanished the moment the power is switched off.

Therefore, to meet the requirements of sufficient data storage, we need some storage area that can hold large amount of data permanently. Such a type of storage is called **secondary storage**. Secondary storage is non-volatile in nature, that is, the data are retained even when the power is switched off or if the system crashes. However, data on the secondary storage devices are not directly accessed by the CPU as they are used to store the data that are not being concurrently processed. Therefore, it needs to be transferred to the main memory so that the CPU can access it. **Magnetic disk** (generally called disk) is the most widespread form of secondary storage means for computer. It offers intense storage capacity for enormous amount of data and easy accessibility. It is used to hold on-line data for a long term.

In addition to RAM and magnetic disk, some other form of storage devices also exist, which include cache memory, flash memory, optical discs and magnetic tapes. The basic function of all the storage devices is to store the data in an easy and retrievable form. However, they differ in terms of their speed, cost, storage capacity and volatility. On the basis of their characteristics, such as cost per unit of data and speed with which data can be accessed, they can be arranged in a hierarchical manner as shown in Figure 1.7.

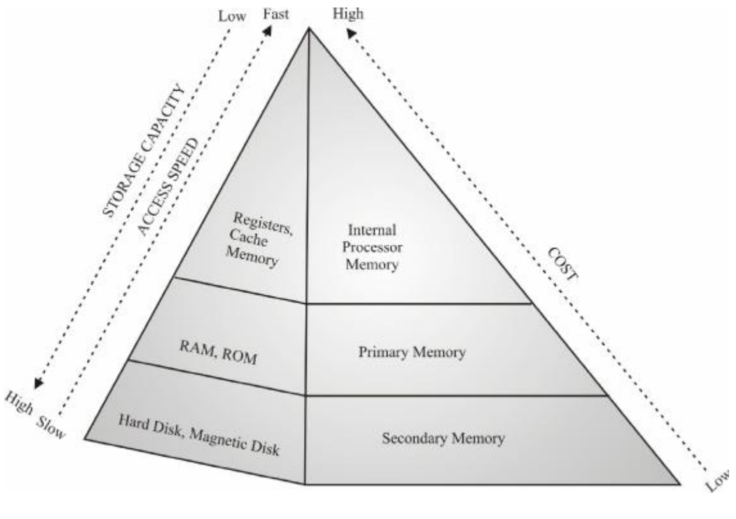


Fig. 1.7 *Memory Hierarchy*

I/O Structure

Handling I/O devices and getting all these parts work together is a major problem and that is what an operating system does. An operating system, therefore, provides an effective link between the components and also clearly outlines how each component should function with the help of codes. One reason for this is the varying nature of I/O devices. An operating system must issue commands to the devices, catch interrupts, handle errors and provide an interface between the devices and the rest of the system.

NOTES

As already mentioned, a computer system consists of one or more processors and multiple device controllers that are connected through a common bus. Each device controller controls a specific type of device and depending on the device controller one or more devices may be attached to it. For example, a **Small Computer-System Interface** (**SCSI**) controller may have seven or more devices attached to it. To perform its job, device controller maintains some local buffer storage and a set of special-purpose registers. An operating systems usually have a **device driver** for each device controller. The device driver acts as an interface to the device to the rest of the system. This interface should be uniform, that is, it should be same for all the devices to the extent possible.

To start an I/O operation, the device driver loads the appropriate registers within the device controller, which in turn examines the contents of registers to determine the action to be taken. Suppose, the action is to read the data from the keyboard, the controller starts transferring data from the device to its local buffer. Upon completion of data transfer, the controller informs the device driver (by generating an interrupt) that the transfer has been completed. The device driver then returns the control along with the data or pointer to the data to an operating system. This form of I/O is interrupt-driven I/O, and this scheme amounts to colossal wastage of CPU's time because CPU requests data from the device controller one byte at a time. This is one of the major drawbacks of this scheme as it is not feasible to transfer a large amount of data with this scheme.

To solve this problem, another scheme, that is, **Direct Memory Access** (**DMA**) is commonly used. This scheme, after setting up the registers to inform the controller to know what to transfer and where, reduces the overhead burden of CPU and relieves the CPU to perform other tasks. The device controller can now complete its job, that is, transfer a complete block of data between its local buffer and memory without CPU intervention. Once the block of data has been transferred, an interrupt is generated to inform the device driver that the operation has been successfully executed.

Check Your Progress

9. How do you define the bootstrap program?
10. Write the steps involved in an instruction- execution cycle.
11. Define the usage of direct memory access scheme.

1.5 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

NOTES

1. An operating system is defined as the most essential and indispensable program that is running at all times on the computer (usually called the kernel). It is a program that acts as an interface between the computer users and the computer hardware. It manages the computer hardware and controls, and coordinates the use of hardware among various application programs.
2. The two viewpoints from which we can understand the role of an operating system are user point of view and the system point of view.
3. An operating system also acts as a control program or an interface that manages the execution of user programs to avoid possible errors and improper use of computer system. It also monitors the performance of the I/O devices and their operations.
4. The basic idea behind multiprogramming is that the operating system loads multiple jobs into the memory from the job pool (the jobs kept on the disk). The jobs are organized in such a way that the CPU always has one job to execute. This increases the CPU utilization by minimizing the CPU idle time.
5. With advancement in technologies, various system software tools were developed that made serial processing more efficient. These tools include language translators, loaders, linkers, debuggers, libraries of common routines, and I/O routines. Programmers could now code their programs in a programming language, which could then be translated into executable code with the help of *language translator* such as a compiler or an interpreter. The *loader* automated the process of loading executable programs into memory (it automatically transfers the program and the data from the input device to the memory). *Debuggers* assisted the programmers in detecting and examining the errors that occur during program execution (run-time errors). *Linkers* were used to link the precompiled routines with the object code of the program so that they could be executed along with the object code to produce the desired output.
6. To reduce this wasted time and speed up the processing, the operator used to batch together the jobs with similar requirements, and run these batches one by one. This system was known as batch processing system. For example, the jobs that need FORTRAN compiler can be batched together so that the FORTRAN compiler can be loaded only once to process all these jobs. Note that the jobs in a batch are independent of each other and belong to different users.

To improve resource utilization and user productivity, the first operating system was developed by General Motors for IBM 701 in the mid 1950s. This operating system was termed as batch operating system. Its major task was to transfer control automatically from one job to next job in the batch without the operator's intervention. This was achieved by automating the transition from execution of one job to that of the next in the batch.

7. The main disadvantage of batch processing is that during execution, the CPU is often idle, because of the speed difference between the CPU and I/O devices. To overcome the problem of speed-mismatch, the concept of SPOOLing (Simultaneous Peripheral Operation On-line) came into existence. Instead of inputting the jobs from the card readers, the jobs were first copied from the punched cards to the magnetic tape.
8. However, in real life, there are generally more than two programs that compete for system resources at any point of time. The number of jobs competing to get the system resources in multiprogramming environment is known as degree of multiprogramming.
9. When the system originally is turned on, it runs a well-defined set of initial programs known as bootstrap program. The bootstrap program is typically stored in Read-Only Memory (ROM) or Electrically Erasable Programmable ROM (EEPROM). During the booting process, all the aspects of the system checking like CPU registers, device controllers and memory contents are initialized, and then an operating system is loaded into the memory. Once an operating system is loaded, the first process, such as "init" is executed and operating system then waits for some special sequence of events to occur.
10. Usually, an instruction–execution cycle consists of the following steps.
    (i) Whenever, the CPU needs to execute an instruction, it first fetches it from the main memory and stores it in Instruction Register (IR).
    (ii) Once the instruction has been loaded into the IR, the control unit examines and decodes the fetched instruction.
    (iii) After decoding the instruction, the operands (if required) are fetched from the main memory and stored in one of the internal registers.
    (iv) The instruction is executed on the operands and the result is stored back to the main memory.
11. Direct Memory Access (DMA) is commonly used. This scheme, after setting up the registers to inform the controller to know what to transfer and where, reduces the overhead burden of CPU and relieves the CPU to perform other tasks. The device controller can now complete its job, that is, transfer a complete block of data between its local buffer and memory without CPU intervention. Once the block of data has been transferred, an interrupt is generated to inform the device driver that the operation has been successfully executed.

1.6 SUMMARY

NOTES

- An operating system is defined as a program that is running at all times on the computer (usually called the kernel). It is a program that acts as an interface between the computer users and the computer hardware.
- An operating system is designed in such a way that it makes the computer system more convenient to use and allows the system to use its resources in an efficient manner.
- There are two viewpoints from which we can understand the role of an operating system: the user point of view and the system point of view.
- In case of a stand-alone environment, where a single user sits in front of a personal computer, an operating system is designed basically for the ease of use and some attention is also paid to system performance.
- In case of a networked environment, where multiple users share resources and may exchange information, an operating system is designed for resource utilization.
- In case of handheld systems, an operating system is basically designed for individual usability as these systems are mostly stand-alone units for individual users.
- From the system's point of view, the operating system primarily acts as a resource allocator.
- An operating system also acts as a control program that manages the execution of user programs to avoid errors and improper use of computer system.
- These days a computer system basically consists of one or more processors (CPUs), several device controllers, and the memory. All these components are connected through a common bus that provides access to shared memory
- When the system boots up, the initial program that runs on the system is known as bootstrap program.
- The event notification is done with the help of an interrupt that is fired either by the hardware or the software.
- Whenever, a program needs to be executed, it must be first loaded into the main memory (called Random-Access Memory or RAM).
- The two instructions, namely, load and store are used to interact with the memory.
- The execution of the program instructions takes place in the CPU registers, which are used as temporary storage areas, and have limited storage space.
- RAM is expensive, offers limited storage capacity, and is volatile in nature, that is, it loses its contents when power supply is switched off.

- Secondary storage is non-volatile in nature, that is, the data is permanently stored and survives power failure and system crashes.
- Magnetic disk (generally called disk) is the primary form of secondary storage that enables storage of enormous amount of data.
- A significant portion of code of an operating system is dedicated to manage I/O. One reason for this is the varying nature of I/O devices.
- Each device controller controls a specific type of device and depending on the type of device controller one or more devices may be attached to it.
- Interrupt-driven I/O wastes CPU's time because CPU requests data from the device controller one byte at a time. To solve this problem, another scheme, that is, Direct Memory Access (DMA) is commonly used.

1.7 KEY WORDS

- **Operating system**: The most essential and indispensable program that is running at all times on the computer (usually called the kernel). It is a program that acts as an interface between the computer users and the computer hardware.
- **SPOOLing:** SPOOLing stands for Simultaneous Peripheral Operation On-line.
- **Degree of multiprogramming:** The number of jobs competing to get the system resources in multiprogramming environment is known as degree of multiprogramming.
- **Bootstrap program:** A well-defined set of initial programs.

1.8 SELF-ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. What are the two basic objectives of an operating system?
2. Differentiate between the concept of user view and system view.
3. State the major drawback of multiprogramming systems.
4. Define serial processing.
5. What are advantages of having mainframe machines?
6. What is the meaning of concurrent execution of processes?
7. What are the advantages of having a time-sharing system?
8. Define the system call or monitor call and state the significance of interrupt handler.

9. What is the importance of having a Small Computer-System Interface (SCSI) controller?
10. Define primary and secondary memory or storage.

Long-Answer Questions

1. Describe the two viewpoints from which we can understand the role of an operating system.
2. Write short notes on each of the following:
   (a) Direct memory access
   (b) Multiprogramming
   (c) Time-sharing systems
3. How does a computer system handle interrupts? Also, discuss how interrupts can be handled quickly?
4. Discuss the storage structure of a computer system with the help of a diagram.
5. Describe how an I/O operation is handled by the system with the help of a diagram.

1.9 FURTHER READINGS

Silberschatz, Abraham, Peter B. Galvin and Greg Gagne. 2008. *Operating System Concepts*, 8th Edition. New Jersey: John Wiley & Sons.

Tanenbaum, Andrew S. 2006. *Operating Systems Design and Implementation*, 3rd Edition. New Jersey: Prentice Hall.

Tanenbaum, Andrew S. 2001. *Modern Operating Systems*. New Jersey: Prentice Hall.

Deitel, Harvey M. 1984. *An Introduction to Operating Systems*. Boston (US): Addison-Wesley.

Stallings, William. 1995. *Operating Systems*, 2nd Edition. New Jersey: Prentice Hall.

Milenkovic, Milan. 1992. *Operating Systems: Concepts and Design*. New York: McGraw Hill Higher Education.

Mano, M. Morris. 1993. *Computer System Architecture*. New Jersey: Prentice Hall Inc.

UNIT 2 COMPUTER SYSTEM ARCHITECTURE

NOTES

Structure

2.0 INTRODUCTION

Depending on the number of processors used in a system, a computer system can be broadly categorized mainly into one of the two types: single-processor system or multiprocessor system. Every operating system has its own internal structure in terms of file arrangement, memory management, storage management, etc. The performance of the entire system depends on its structure. An operating system is composed of a kernel and user level libraries. But, knowing the structure of an operating system refers to know the basic concept of virtual machine, kernel, and CPU and I/O structure. For instance, programmer should not be bothered about how memory is allocated to their programs, where their programs are loaded in memory during execution, how multiple programs are managed and executed, how their programs are organized in files to reside on disk, how I/O devices are supervised, etc. Providing this environment in which programs can be successfully executed and the set of services to user programs are the responsibilities of an operating system.

Power on self-test is a diagnostic tool that provides the end user information about the computer. The motherboard basic input output system contains POST function. Booting up is a bootstrapping process that starts the operating system when the user switches on a computer. Kernel is the fundamental part of an operating system. It is a piece of software used for providing secure access to the machine's hardware to various computer programs.

In this unit, you will study the basics of computer system architecture, operating system structure, operating system operations, post and bootstrapping.

2.1 OBJECTIVES

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

- Describe the basics of computer system architecture
- Explain the definition of operating system structure
- Understand the concept and functions of virtual machine concept, modules
- Discuss about the study of kernel and layered approach
- Analyse the advantages and disadvantages of CPU and I/O structure of operating system
- Discuss different operating system operations, such as dual-mode and timer
- Understand the role and function of post and bootstrapping
- Define the importance of kernel of an operating system

2.2 BASICS OF COMPUTER SYSTEM ARCHITECTURE

Depending on the number of processors used in a system, a computer system can be broadly categorized mainly into one of the two types: single-processor system or multiprocessor system.

Single-Processor Systems

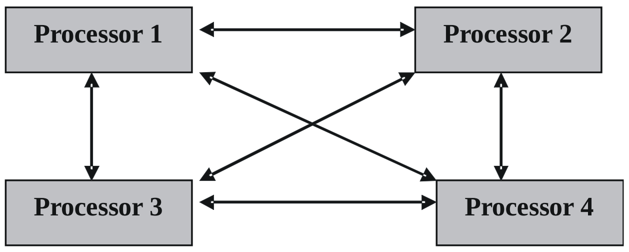
Single-processor systems consist of one main CPU that can execute a general-purpose instruction set, which includes instructions from user processes. Other than the one main CPU, most systems also have some special-purpose processors. These special-purpose processors may be in the form of device-specific processors, such as disk, keyboard, etc., or in mainframes, they may be I/O processors that move data among the system components and allow communication. Note that the special-purpose processors execute a limited instruction set and do not execute instructions from the user processes. Further, the use of special-purpose processors does not turn a single-processor system into a multiprocessor system.

In some systems, the special-purpose processors are managed by the operating systems and in others, they are low-level components built into the hardware. In the former case, an operating system monitors their status and sends them information for their next task. For example, the main CPU sends requests to access the disk to a disk controller microprocessor, which implements its own disk queue and disk scheduling algorithm. Doing this, the main CPU is relieved from the disk scheduling overhead. In the latter case, these special-purpose processors do their tasks autonomously and an operating system cannot communicate with them.

Multiprocessor Systems

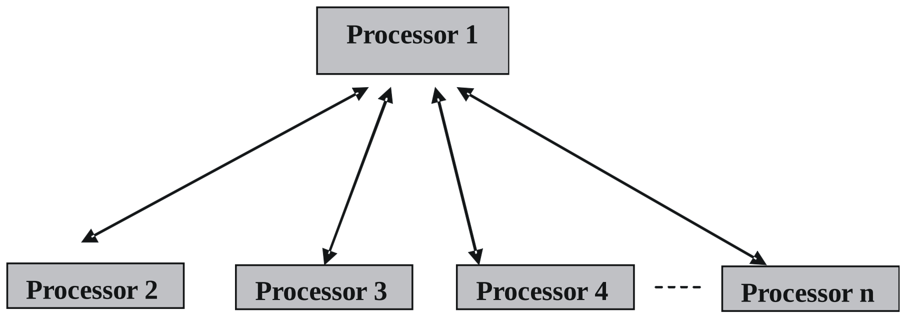
As the name suggests, the **multiprocessor systems** (also known as **parallel systems** or **tightly coupled systems**) consist of multiple processors in close communication in a sense that they share the computer bus and even the system clock, memory, and peripheral devices. The main advantage of multiprocessor systems is that they increase the system **throughput** by getting more work done in less time. Another benefit is that it is more economic to have a single multiprocessor system than to have multiple single-processor systems. In addition, the multiprocessor systems are more reliable. If one out of N processors fails, then the remaining N-1 processors share the work of the failed processor amongst them, and thereby preventing the failure of the entire system.

Multiprocessor systems are of two types, namely, *symmetric* and *asymmetric*. In **symmetric multiprocessing systems**, all the processors are identical and perform identical functions. Each processor runs an identical copy of an operating system and these copies interact with each other as and when required. All processors in symmetric multiprocessor system are peers—no master–slave relationship exists between them. On the other hand, in **asymmetric multiprocessing systems**, processors are different and each of them performs a specific task. One processor controls the entire system and hence, it is known as **master processor**. Other processors, known as **slave processors**, either wait for the master's instructions to perform any task or have predefined tasks. This scheme defines a master–slave relationship. The main disadvantage of asymmetric multiprocessing systems is that the failure of the master processor brings the entire system to a halt. Figure 2.1 shows symmetric and asymmetric multiprocessor system.



(a) Symmetric Multiprocessing System

NOTES



(b) Asymmetric Multiprocessing System

Fig 2.1 *Symmetric and Asymmetric Multiprocessing Systems*

2.3 OPERATING SYSTEM STRUCTURE

Every operating system has its own internal structure in terms of file arrangement, memory management, storage management, etc. The performance of the entire system depends on its structure. The internal structure of an operating system provides an idea of how the components of the operating system are interconnected and blended into the kernel.

2.3.1 Central Processing Unit and Input/Output Structure

The Arithmetic and Logical Unit (ALU) and the Control Unit (CU) of a computer system are jointly known as the central processing unit. You may call CPU as the brain of any computer system. It takes all major decisions, makes all sorts of calculations and directs different parts of the computer functions by activating and controlling the operations.

For a computer to start running, it needs to have an initial program to run. This initial program, also known as bootstrap program, tends to be simple. It is stored in CPU registers. The role of the initial program or the bootstrap program is to load the operating system for the execution of the system. The operating system starts executing the first process, such as 'init' and waits for some event to occur. Event is known to occur by an interrupt from either the hardware or the software. Hardware can interrupt through system bus whereas software through system call.

When a CPU is interrupted, it immediately stops whatever it is doing and returns to a fixed location. This fixed location usually contains the starting address where the service routine for the interrupt is located.

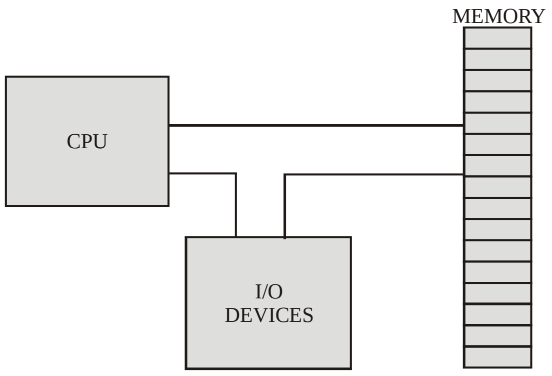


Fig. 2.2 *Structure of a Computer System*

Input/Output Structure

There are various types of I/O devices that are used for different types of applications. They are also known as peripheral devices because they surround the CPU and make a communication between computer and the outer world. Following types of devices are implemented with system unit:

Input Devices: Input devices are necessary to convert our information or data into a form, which can be understood by the computer. A good input device should provide timely, accurate and useful data to the main memory of the computer for processing. Keyboard, mouse and scanner are the most useful input devices.

Output Devices: Visual Display Unit (VDU), terminals and printers are the most commonly used output devices.

2.3.2 Simple Structure

The elementary approach to structure an operating system is known as a simple structure. In this approach, the structure of the operating system is not well-defined. MS-DOS is an operating system designed with this approach in view. It was initially designed to be simple and small in size having limited scope, but with time, it outgrew its scope. Designed with a view to providing more functionality within less space; it was not carefully divided into modules. Figure 2.3 shows the structure of MS-DOS system.

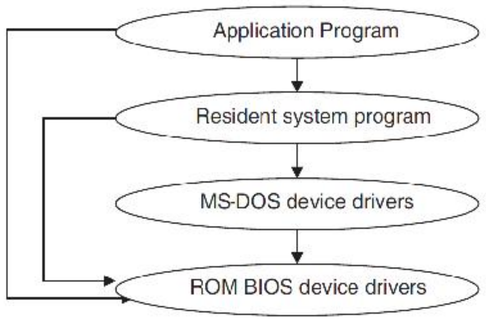


Fig. 2.3 *Structure of MS-DOS System*

NOTES

Though MS-DOS has a limited structuring, there is no clear separation between the different interfaces and level of functionality. For example, application programs can directly call the basic I/O routines to read/write data on disk instead of going through a series of interfaces. This exemption makes the MS-DOS system susceptible to malicious programs which may lead to system crash. Moreover, due to lack of hardware protection and dual-mode operation in Intel 8088 system (for which MS-DOS system was developed), the base hardware was directly accessible to the application programs.

2.3.3 Virtual Machine Concept

The operating system provides applications with a virtual machine. This type of situation is analogous to the communication line of a telephony company which enables separate and isolated conversations over the same wire(s). An important aspect of such a system is that the user can run an operating system as per choice.

The virtual machine concept can be well understood by understanding the difference between conventional multiprogramming and virtual machine multiprogramming. In conventional multiprogramming, processes are allocated a portion of the real machine resources, i.e., a resource from the same machine is distributed among several resources (Refer Figure 2.4).

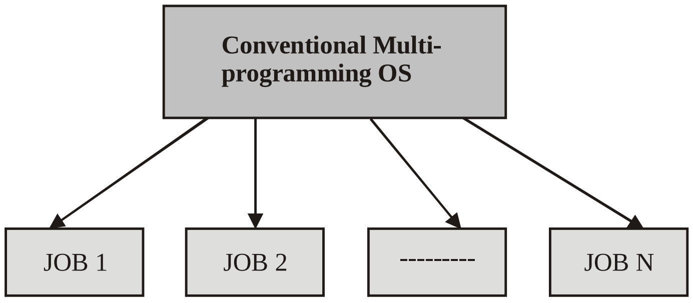


Fig. 2.4 *Conventional Multiprogramming*

In the virtual machine multiprogramming system, a single machine gives an illusion of many virtual machines each of them having its own virtual processor and storage space which can be handled through process scheduling (Refer Figure 2.5).

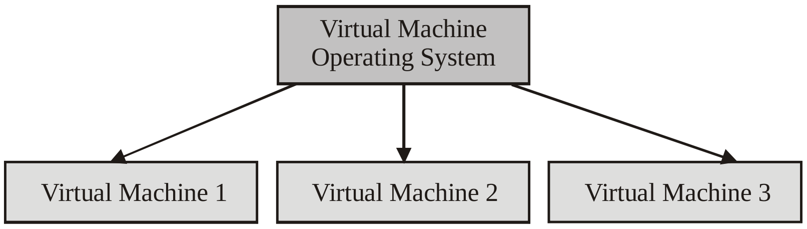


Fig. 2.5 *Virtual Machine Multiprogramming*

Advantages

Following are the advantages of virtual machine:

- Each user is allocated with a machine which eliminates mutual interference between users.
- A user can select an OS of the choice for executing his virtual machine. Hence, the user can simultaneously use different operating systems on the same computer system.

2.3.4 Layered Approach

In layered approach, the operating system is organized as a hierarchy of layers with each layer built on the top of the layer below it. The topmost layer is the user interface, while the bottommost layer is the hardware. Each layer has a well-defined function and comprises data structures and a set of routines. The layers are constructed in such a manner that a typical layer (say, layer n) is able to invoke operations on its lower layers, and the operations of layer n can be invoked by its higher layers.

'THE system' was the first layer-based operating system developed in 1968 by E.W. Dijkstra and his students. This operating system consisted of six layers (0-5) and each layer had a pre-defined function as shown in Figure 2.6.

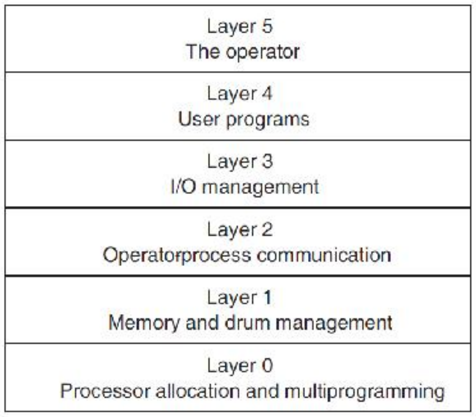

Layer 5
The operator
Layer 4
User programs
Layer 3
I/O management
Layer 2
Operatorprocess communication
Layer 1
Memory and drum management
Layer 0
Processor allocation and multiprogramming

***Fig. 2.6** Layers in THE System*

The layered design of operating system provides some benefits, which are as follows:

- It simplifies the debugging and verification of the system. As the lowest layer uses merely the base hardware, it can be debugged without referring to the rest of the system. Once it has been verified, its correct functioning can be assumed while the second layer is being verified. Similarly, each higher-level layer can be debugged independent of the lower layer. If, during verification, any bug is found, it will be on the layer being debugged as lower layers have already been verified.

- It supports information hiding. Each higher-level layer is required to know only what operations the lower layers provide and not how they are being implemented.

The layered approach has some limitations too, which are as follows:

- As each higher-level layer is allowed to use only its lower-level layers, the layers must be defined carefully. For example, the device driver of a physical disk must be defined at a layer below the one containing memory-management routines. This is because memory management needs to use the physical disk.
- The time taken in executing a system call is much longer as compared to that of non-layered systems. This is because any request by the user has to pass through a number of layers before the action could be taken. As a result, system overheads increase and efficiency deceases.

2.3.5 Kernel Approach

Kernel is the central part of an operating system which directly controls the computer hardware. Following are the advantages of kernel.

- Kernel lies below system call interface and above the physical hardware.
- It provides large number of functions, such as CPU scheduling, memory management, I/O management, synchronization of processes, inter process communication and other operating system functions.

Microkernels: Initially, the size of kernel was small; the era of large monolithic kernels began with Berkley UNIX (BSD). The **monolithic kernel** runs every basic system service like scheduling, interprocess communication, file management, process and memory management, device management, etc., in the kernel space itself. However, the inclusion of all basic services within the kernel space increases the size of the kernel. In addition, these kernels were difficult to extend and maintain. The addition of new features required recompilation of the whole kernel, which was time and resource consuming.

To overcome these problems, an approach called **microkernel** was developed that emphasized on modularizing the kernel. The idea was to remove the less-essential components from the kernel, keeping only a subset of mechanisms typically included in a kernel, thereby reducing its size as well as number of system calls. The components moved outside the kernel are implemented either as system or user-level programs. MACH system and OS X are the examples of operating systems designed using the microkernel approach.

The main advantage of a microkernel approach is that the operating system can be extended easily; the addition of new services in the user space does not cause any changes at the kernel level. In addition, microkernel offers high security and reliability as most services are running as user processes rather than kernel processes. Thus, if any of the running services fail, the rest of the system remains unaffected.

Note: *Though the microkernel approach reduced the size of the kernel, there is still an issue regarding which services are to be included in the kernel and which services to be implemented at the user level.*

2.3.6 Modules

The module-based approach employs object-oriented programming techniques to design a modular kernel. In this approach, the operating system is organized around a core kernel and other loadable modules that can be linked dynamically with the kernel either at boot time or at run time. The idea is to make the kernel provide only core services, while certain services can be added dynamically. An example of a module-based operating system is Solaris, which consists of a core kernel and seven loadable kernel modules: scheduling classes, file systems, loadable system calls, executable formats, streams modules, miscellaneous, and device and bus drivers.

The modular approach is similar to a layered approach in the sense that each kernel module has well-defined interfaces. However, it is more flexible than a layered approach as each module is free to call any other module.

Check Your Progress

1. Classify computer system based on the number of processors used in a system.
2. What are the two types of multiprocessor systems? Define any one of them.
3. What are special-purpose processors?
4. What problems may arise when resources are shared among several programs?
5. State the limitations of layered approach of the structure of an operating system.
6. Define the advantages of microkernel.

2.4 OPERATING SYSTEM OPERATIONS

As discussed earlier, modern operating systems are interrupt driven. When there is management and coordination of activities to be performed, that is, no processes for execution, no I/O activities, and no user to whom to respond, an operating system will go into the dormant mode and sit idle. Whenever an event occurs, it is signalled by triggering an interrupt or a trap. For each type of interrupt, there exists a code segment in the operating system that specifies the actions to be taken. The part of the operating system called **Interrupt Service Routine** (**ISR**) executes the appropriate code segment to deal with the issued interrupt.

In case of multiprogrammed environment, the computer resources are shared among several users simultaneously. Though the sharing of resources enhances the resource utilization, it also multiplies the complications. An error in one user program can adversely affect the execution of other programs. It may also happen that the erroneous program modifies another program, or data of another program, or the operating system itself. Without the protection against such type of errors, only one process must be allowed to execute at a given moment.

However, for effective and improvized resource utilization, it is imperative to allow resource sharing among multiple programs simultaneously. Therefore, to cope up with such an environment, an operating system should be fabricated keeping in mind that an incorrect program does not adversely impact the execution of other programs, or an operating system itself.

Dual-Mode Operation

In order to ensure the proper functioning of the computer system, dual-mode operator requires an operating system and all other programs and their data to ensure that an incorrect program does not hamper the execution and, therefore, must be protected against such programs. To achieve this protection, two modes of operations, namely, **user mode** and **monitor mode** (also known as **supervisor mode**, **system mode**, **kernel mode**, or **privileged mode**) are provided for supporting. A **mode bit** is associated with the computer hardware to indicate the current mode of operation. The value '1' indicates the user mode and '0' indicates the monitor mode. When the mode bit is 1, it implies that the execution is being done on behalf of the user, and when it is 0, it implies that the execution is being done on behalf of an operating system.

Originally, when the system begins to initialize (or booted), it is in monitor mode. Then, the operating system is loaded and the user processes are started in the user mode. When a trap or an interrupt occurs, the hardware switches from user mode to the monitor mode by changing the mode bit value to 0. Therefore, whenever an operating system has the control on the computer, it is in the monitor mode. Contrary to this, whenever the control needs to be passed to the user program, the hardware must switch the mode to the user mode before passing the control to the user program.

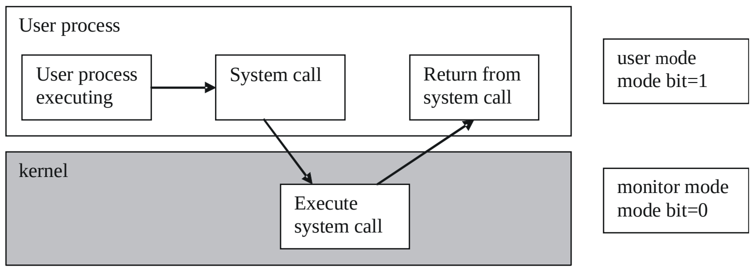


Fig. 2.7 *Dual-Mode Operation*

This dual mode of operation helps in protecting an operating system and the other programs, from malicious programs. To achieve this protection, some of the machine instructions that may cause harm are designated as **privileged**

instructions. These privileged instructions are allowed to be executed only in the monitor mode. If an attempt is made to execute a privileged instruction in user mode, the hardware treats it as an illegal instruction and traps it to the operating system without executing it. The instruction used to switch from kernel mode to user mode is an example of a privileged instruction.

***Note:** Recent operating systems, such as Windows 2000 and IBM OS/2 provide greater protection for an operating system by supporting privileged instructions.*

Timer

When a process starts executing, then it is quite possible that it gets stuck in an infinite loop and never returns the control to the operating system. Therefore, it is necessary to prevent a user program from gaining the control of the system for an infinite time. For this, a **timer** is maintained, which interrupts the system after a specified period and checks the sequence of events. This period can be fixed or variable. A **variable timer** is usually implemented by a fixed-rate clock and a counter.

It is the responsibility of an operating system to set the counter which is decremented with every clock tick. Whenever, the value of counter reaches 0, an interrupt occurs. In this way, the timer prevents a user program from running too long. Initially, when a program starts, a counter is initialized with the amount of time that a program is allowed to run. The value of counter is decremented by 1 with each clock tick and once it becomes negative, an operating system terminates the program for exceeding the assigned time limit. Note that the instructions that modify the operations of the timer are also designated as privileged instructions.

2.5 POST AND BOOTSTRAPPING

Power On Self Test (POST) is a diagnostic tool that provides the end user information about the computer. The motherboard Basic Input Output System (BIOS) contains POST function. It provides BIOS related diagnostic information in two forms and these two forms are known as POST codes and beep codes. When the Personal Computer (PC) is booted it first goes to the POST that builds diagnostic program. It verifies whether the hardware is functioning properly before the BIOS starts the process of actual booting. It then continues with other tests, such as the memory test. POST indicates wrong process in the machine. Beep patterns can be used for diagnosing many hardware problems with the PC. The actual patterns depend on the companies which manufactured the BIOS. The most common example is Award and American Megatrends Inc. (AMI) BIOS. Troubleshooting expert helps users to figure out the POST codes and solution for occurred problem. Sometimes, POST errors are considered as fatal error which halts the boot process immediately. POST, a hardware detecting program applicable for BIOS requires loading and detecting firmware, includes the following steps:

NOTES

- A POST program is run to test and detect a hardware program.
- When the hardware testing program sends back an error value it shows the output of error messages corresponding to the hardware testing program to a memory.
- It reboots the computer, loads next sequence of detecting firmware to the BIOS and runs the hardware testing program of detecting firmware.
- It continues to run the POST program when the hardware testing program sends back a correct value. If the hardware testing program sends back an error value then above process is repeated.

POST codes are different for all BIOS manufacturers. They provide visual characters readout to show what stage the POST is at. A POST card is required to see the coding that can be set in motherboard's POST codes. The recent motherboards are facilitated with built-in diagnostic capability tools. It is a 32-bit card that is allotted in Peripheral Component Interconnect (PCI) slot on any type of motherboard. The POST processes include verification test, register test, controller test, receive digital signal path test and basic front end test to perform its task efficiently and successfully. The POST is initiated, performed and completed within a short time, for example it can take three seconds when power is switched on. The tests are performed after the POST at the discretion of an operator. This process includes a transmit test, a transducer element test, a front end voltage test and a receive test. The various BIOS companies, such as Award, AMI and Phoenix provide own beep codes. Beep codes occur if motherboard has an inbuilt speaker. Most chassis have a speaker connector so if the motherboard does not come with a built-in speaker it usually comes with a 4-pin speaker header. These codes occur if the board either functions normally or get problem abruptly with system functioning. In most cases, one time beep is produced if the board POST is executed successfully. There are few beep codes coming in a chart format. It is easy and necessary to analyse the beep codes. Beeping sound makes the users aware if the computer does not boot successfully. Thus, beep sounds ensure that CPU works properly and POST performs its functions at this stage. Following are the various types of POST method:

Microsoft POST Operating System: This POST starts immediately after power is up. The Microsoft Windows XP operating system sends results to the front panel for POST that works as per instructions but the results are not saved in a file.

Application POST: The application POST executes when the real-time processor has booted up to check licensing requirements, missing Dynamic Link Library (DLL) socket communications, with the real-time processor and the data transfer processor.

Real-time Processor POST: The real-time processor POST is executed from its own boot ROM. Its processed results are written in the power up status file.

Data Transfer Processor POST: The data transfer processor POST is started by the real-time processor. Its processed results are written in the power up status file.

Hardware Control Processor POST: The real-time processor POST communicates with the hardware control processors. Real-time hardware includes down converter, motherboard and Remote File Input Output (RFIO).

NOTES

The POST was introduced with Reduced Instruction Set Computer Operating System (RISC OS) 3.0 and later versions of the OS. When computer is switched on the machine verifies the hardware for physical faults before using it. Then, it highlights major errors so that hardware damage would not be possible. ROM, RAM, VIDeo Controller (VIDC) and Input Output Controller (IOC) test are carried out in POST process in which the color of the screen becomes blue. A limited memory test is carried out simultaneously with a second test of the VIDC and IOC. When the screen again becomes purple the machine is tested for an ARM3 chip. At the end of the sequence the screen color is set to green for pass status or red for fail status. If POST is passed successfully through all processes the machine starts booting and the RISC OS 3.0 welcome screen is seen. If any test fails the screen remains red and the disk drive light blinks a fault code. A short flash is used for indicating a binary '0' and a long flash for binary '1'. The bits are grouped into eight nibbles, i.e., blocks of four bits with the most significant bit first. The block constituted by the lowest seven bits is known as status word. Each bit is coded in hexadecimal format as shown in Table 2.1.

Table 2.1 *Bit Values and Their Function*

Bit Value	**Function**
00000001	POST is processed when power is on.
00000002	POST is processed by interface hardware.
00000004	POST is processed by test link.
00000008	POST is processed by long memory test.
00000010	POST is processed if ARM ID detected.
00000020	It disables long memory test.
00000040	POST is processed to detect IO interfaces.
00000080	POST is processed to detect Virtual Random Access Memory (VRAM).

The fault code contains the bits value from bit-8 to bit-31. All bits are not used in fault code. If the code is marked as reserved RiscPC it means error number is currently unassigned. It states that older hardware is no longer sensible for the new machines. Table 2.2 shows the bit values and their functions.

NOTES

Table 2.2 *Bit Values and Their Functions*

Bit Value	Function
00000100	This bit value is used for CMOS RAM checksum error.
00000200	This bit value is used for ROM failed checksum test.
00000400	This bit value is used for failed MEMC CAM mapping.
00000800	This bit value is used for failed MEMC protection.
00001000	This bit value is used for reserved code on the RiscPC.
00002000	This bit value is used for reserved code on the RiscPC.
00004000	This bit value is used for Virq video interrupt.
00008000	This bit value is used for VIDC Sirq sound interrupt
00010000	CMOS unreadable.
00020000	RAM control line failure.
00040000	RAM test failure.
00080000	A reserved code on the RiscPC.

The RiscPC (codenamed Medusa) is Acorn Computers's next generation RISC OS/Acorn RISC Machine computer launched in the year of 1994 which superseded the Acorn Archimedes. Virq (video interrupt) and Sirq (sound interrupt) are assembled with microprocessor. If POST is failed with VIDC enhancer and machine is continued to work systematically then Beginner's All-purpose Symbolic Instruction Code (BASIC) program runs and saves COMS settings. The following code is written in BASIC language to save the CMOS settings:

```
REM saving POST CMOS settings
REM read byte
SYS "OS_Byte",161, &BC TO, byte%
REM EOR byte for mask bit 1
byte% =  byte% EOR %10000000
REM writing byte
SYS "OS_Byte",161, &BC TO, byte%
END
```

The result of above coding is to save 1-bit while preserving the other bit which is used to change the Complementary Metal Oxide Semiconductor (CMOS) setting.

Bootstrapping

In computing, booting (booting up) is a bootstrapping process that starts the operating system when the user switches on a computer. A boot sequence is the initial set of operations that the computer performs when switched on. The bootloader typically loads the main operating system for the computer.

One can boot an operating system in two conditions: (i) Where there is a single OS installed and (ii) Where there are multiple OSs installed on the computer.

Single OS Boot Process

Whenever a computer is turned on, BIOS takes control and performs many operations, such as checking hardware and ports and then loads the Master Boot Record (MBR) program into the memory (RAM).

In the next step, the MBR takes control of the booting process.

When only one OS is installed, the functions of MBR are as follows:

- The boot process starts by executing a code in the first sector of the disk.
- MBR looks over the partition table to find the 'Active Partition'.
- Control is passed to that Partition's Boot Record (PBR) to continue booting.
- The PBR locates the system specific boot files, such as Win98's io.sys or WinXP's ntoskrnl.
- Then these boot files continue the process of loading and initializing the rest of the OS.

Multiple OS Boot Process

When there are multiple OSs, be it multiple Windows or Windows with Linux, the booting process is slightly different. There can be two different types of booting processes in a multiple OS environment, the Microsoft way and non-Microsoft way or third party boot loader way.

2.6 KERNEL

Kernel is the fundamental part of an operating system. It is a piece of software used for providing secure access to the machine's hardware for various computer programs. Since, there are many programs which are used to access the hardware is limited to the kernel and also responsible for deciding when and how long a program should be able to make use of a piece of hardware in a technique. This process is known as multiplexing. Accessing the hardware directly is a complex task therefore the kernel is implemented for hardware abstractions. These abstractions are a way of hiding the complexity and providing a clean and uniform interface to the underlying hardware which makes it easier on application programmers. The kernel is the central part of an operating system that directly controls the computer hardware. Usually, the kernel is the first of the user installed software on a computer but booting directly after the BIOS. Operating system kernels are specific to the hardware on which they are running thus most operating systems are distributed with different kernel options that are configured when the system is installed. Changing major hardware components, such as the motherboard, processor or memory often requires a kernel update. The two major types of kernels available in the computer markets are the Windows kernel and the UNIX like kernels. The Windows kernel is available only with the series of Microsoft

NOTES

Windows operating systems. UNIX like kernel is a family of operating system kernels that are based upon the original Bell Labs UNIX operating system. Common examples of UNIX like kernels are the Linux kernel, Berkeley Software Distribution (BSD), Mac OS and Solaris. Following are the functions of kernel:

- The function of kernel in the operating system is to improve system security and performance.
- Kernel is used to increase the system stability and security.
- Microkernel, one of the prime types of kernel, is often used in embedded robotic or medical computers because most of the OS components reside in their own private or protected memory space.
- Exokernel which is also one of the prime types of kernel used for a very low level interface to the hardware lacking any of the higher level functionalities of other operating systems.
- The goal of kernel is to allow an application to request the specific piece of memory and specific disk block. The main function of kernel is to check whether the requested resource is free and the application is allowed to access it.

Figure 2.8 shows the two modes of operating system known as user mode and kernel mode. Both modes provide the system services for Hardware Abstraction Layer (HAL) as well as for microkernel too.

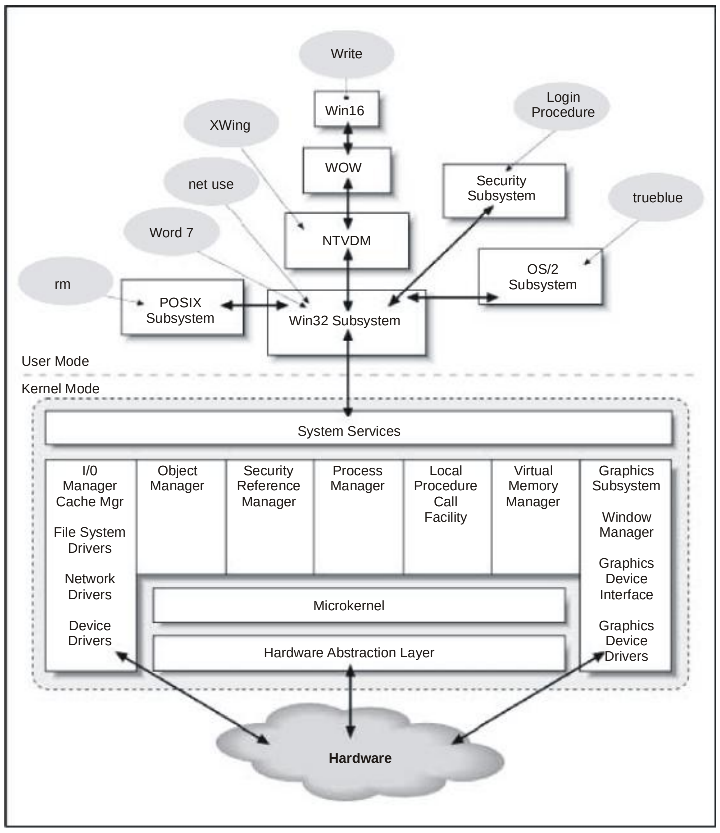


Fig. 2.8 *Kernel in an Operating System*

The structure of operating system is separated into two sections in which the upper section contains components that run in user mode and the lower section containing those that run in kernel mode. The heart of the various types of operating systems, such as Linux or Windows consists of the modules running in kernel mode. Most interactions with the computer hardware take place via the HAL although some device drivers also directly access the hardware. The core of the operating system refers to kernel which is the microkernel. It oversees the working of all of the other modules and handles communications between them and the HAL (refer Figure 2.8). In the other components of the kernel, each has a single specific area of responsibility. An I/O manager controls most input and output on the system. The role of object manager is to create, modify and delete the system objects. Data structures correspond to a specific instance of a resource, for example a file, a process or a port. The Security Reference Manager (SRM) is responsible for enforcing system security settings by granting or denying access to objects and system resources upon request from the object manager. This process relies on data structures known as Security Access Tokens (SAT). The Process Manager (PM) creates and manages system processes but process scheduling is handled by the microkernel. The Local Procedure Call (LPC) facility is responsible for communication between distinct processes known as interprocess communication. The Virtual Memory Manager (VMM) handles the allocation and use of the system's memory. The Graphics Subsystem (GS) provides services required for interfacing to graphical displays.

Check Your Progress

7. Name the two modes required for the protection in dual-mode operation of an operating system?
8. Define the timer and variable timer.
9. What is the use of the input devices?
10. What is meant by the POST?
11. What all POST process includes?
12. How do we define the term booting?
13. State the conditions required for booting the system.
14. Define kernel.

2.7 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. Depending on the number of processors used in a system, a computer system can be broadly categorized mainly into one of the two types; single processor system or multiprocessor system.

2. Multiprocessor systems are of two types, namely, symmetric and asymmetric. In symmetric multiprocessing systems, all the processors are identical and perform identical functions. Each processor runs an identical copy of the operating system and these copies interact with each other as and when required. All processors in symmetric multiprocessor system are peers–no master–slave relationship exists between them.
3. Special-purpose processors may be in the form of device-specific processors, such as disk, keyboard, etc., or in mainframes, they may be I/ O processors that move data among system components. Note that the special-purpose processors execute a limited instruction set and do not execute instructions from the user processes.
4. Though the sharing of resources has improved the resource utilization, it has also increased the problems. In single-user system, an error in a currently running program can cause problems for that program only as it is the only active program at that point of time. However, when the resources are shared among several programs, then an error in one program can adversely affect the execution of other programs. It may also happen that an erroneous program modifies another program, or data of another program, or the operating system itself.
5. The layered approach has some limitations too, which are as follows:
   - As each higher-level layer is allowed to use only its lower-level layers, the layers must be defined carefully. For example, the device driver of a physical disk must be defined at a layer below the one containing memory-management routines. This is because memory management needs to use the physical disk.
   - The time taken in executing a system call is much longer as compared to that of non-layered systems. This is because any request by the user has to pass through a number of layers before the action could be taken. As a result, system overheads increase and efficiency deceases.
6. The main advantage of a microkernel approach is that the operating system can be extended easily; the addition of new services in the user space does not cause any changes at the kernel level. In addition, microkernel offers high security and reliability as most services are running as user processes rather than kernel processes. Thus, if any of the running services fail, the rest of the system remains unaffected.
7. To achieve this protection, two modes of operations, namely, user mode and monitor mode (also known as supervisor mode, system mode, kernel mode, or privileged mode) are provided for supporting. A mode bit is

associated with the computer hardware to indicate the current mode of operation. The value '1' indicates the user mode and '0' indicates the monitor mode. When the mode bit is 1, it implies that the execution is being done on behalf of the user, and when it is 0, it implies that the execution is being done on behalf of an operating system.

8. A timer is maintained, which interrupts the system after a specified period and checks the sequence of events. This period can be fixed or variable. A variable timer is usually implemented by a fixed-rate clock and a counter.
9. Input devices are used to convert our information or data into a form, which can be understood by the computer.
10. Power On Self Test (POST) is a diagnostic tool that provides the end user information about the computer. The motherboard Basic Input Output System (BIOS) contains POST function. It provides BIOS related diagnostic information in two forms and these two forms are known as POST codes and beep codes.
11. The POST processes include verification test, register test, controller test, and receive digital signal path test and basic front end test to perform its task efficiently and successfully. The POST is initiated, performed and completed within a short time, for example it can take three seconds when power is switched on. The tests are performed after the POST at the discretion of an operator.
12. Booting (booting up) is a bootstrapping process that starts the operating system when the user switches on a computer. A boot sequence is the initial set of operations that the computer performs when switched on. The bootloader typically loads the main operating system for the computer.
13. One can boot an operating system in two conditions:
    (i) Where there is a single OS installed and
    (ii) Where there are multiple OSs installed on the computer.
14. Kernel is the fundamental part of an operating system. It is a piece of software used for providing secure access to the machine's hardware for various computer programs. The kernel is the central part of an operating system that directly controls the computer hardware.

2.8 SUMMARY

- Depending on the number of processors used in a system, a computer system can be broadly categorized mainly into one of the two types: single-processor system or multiprocessor system.

- Single-processor systems consist of one main CPU that can execute a general-purpose instruction set, which includes instructions from user processes. Other than the one main CPU, most systems also have some special-purpose processors.
- The multiprocessor systems (also known as parallel systems or tightly coupled systems) consist of multiple processors in close communication in a sense that they share the computer bus and even the system clock, memory, and peripheral devices.
- The main advantage of multiprocessor systems is that they increase the system throughput by getting more work done in less time.
- Multiprocessor systems are of two types, namely, *symmetric* and *asymmetric*. In symmetric multiprocessing systems, all the processors are identical and perform identical functions.
- Every operating system has its own internal structure in terms of file arrangement, memory management, storage management, etc. The performance of the entire system depends on its structure.
- The Arithmetic and Logical Unit (ALU) and the Control Unit (CU) of a computer system are jointly known as the central processing unit.

  There are various types of I/O devices that are used for different types of applications. They are also known as peripheral devices because they surround the CPU and make a communication between computer and the outer world.
- MS-DOS is an operating system designed with this approach in view. It was initially designed to be simple and small in size having limited scope, but with time, it outgrew its scope. Designed with a view to providing more functionality within less space; it was not carefully divided into modules.
- The operating system provides applications with a virtual machine. This type of situation is analogous to the communication line of a telephony company which enables separate and isolated conversations over the same wire(s). An important aspect of such a system is that the user can run an operating system as per choice.
- In layered approach, the operating system is organized as a hierarchy of layers with each layer built on the top of the layer below it. The topmost layer is the user interface, while the bottommost layer is the hardware. Each layer has a well-defined function and comprises data structures and a set of routines. The layers are constructed in such a manner that a typical layer (say, layer n) is able to invoke operations on its lower layers, and the operations of layer n can be invoked by its higher layers.
- 'THE system' was the first layer-based operating system developed in 1968 by E.W. Dijkstra and his students. This operating system consisted of six layers (0-5) and each layer had a pre-defined function.

NOTES

- Kernel is the central part of an operating system which directly controls the computer hardware.
- The monolithic kernel runs every basic system service like scheduling, interprocess communication, file management, process and memory management, device management, etc., in the kernel space itself. However, the inclusion of all basic services within the kernel space increases the size of the kernel.
- An example of a module-based operating system is Solaris, which consists of a core kernel and seven loadable kernel modules: scheduling classes, file systems, loadable system calls, executable formats, streams modules, miscellaneous, and device and bus drivers.
- The module-based approach employs object-oriented programming techniques to design a modular kernel.
- When there is management and coordination of activities to be performed, that is, no processes for execution, no I/O activities, and no user to whom to respond, an operating system will go into the dormant mode and sit idle. Whenever an event occurs, it is signalled by triggering an interrupt or a trap. For each type of interrupt, there exists a code segment in the operating system that specifies the actions to be taken. The part of the operating system called Interrupt Service Routine (ISR) executes the appropriate code segment to deal with the issued interrupt.
- To achieve this protection, two modes of operations, namely, user mode and monitor mode (also known as supervisor mode, system mode, kernel mode, or privileged mode) are provided for supporting. A mode bit is associated with the computer hardware to indicate the current mode of operation. The value '1' indicates the user mode and '0' indicates the monitor mode. When the mode bit is 1, it implies that the execution is being done on behalf of the user, and when it is 0, it implies that the execution is being done on behalf of an operating system.
- To achieve this protection, some of the machine instructions that may cause harm are designated as privileged instructions.
- A timer is maintained, which interrupts the system after a specified period and checks the sequence of events. This period can be fixed or variable. A variable timer is usually implemented by a fixed-rate clock and a counter.
- Power On Self Test (POST) is a diagnostic tool that provides the end user information about the computer. The motherboard Basic Input Output System (BIOS) contains POST function. It provides BIOS related diagnostic information in two forms and these two forms are known as POST codes and beep codes.
- In computing, booting (booting up) is a bootstrapping process that starts the operating system when the user switches on a computer. A boot sequence is the initial set of operations that the computer performs when switched on. The bootloader typically loads the main operating system for the computer.

- Whenever a computer is turned on, BIOS takes control and performs many operations, such as checking hardware and ports and then loads the Master Boot Record (MBR) program into the memory (RAM).
- Kernel is the fundamental part of an operating system. It is a piece of software used for providing secure access to the machine's hardware for various computer programs.
- The goal of kernel is to allow an application to request the specific piece of memory and specific disk block. The main function of kernel is to check whether the requested resource is free and the application is allowed to access it.
- The structure of operating system is separated into two sections in which the upper section contains components that run in user mode and the lower section containing those that run in kernel mode. The heart of the various types of operating systems, such as Linux or Windows consists of the modules running in kernel mode.

2.9 KEY WORDS

- **Single-processor system:** System one main CPU that can execute a general-purpose instruction set, which includes instructions from user processes.
- **Multiprocessor system:** A system with multiple processors in close communication in a sense that they share the computer bus and even the system clock, memory, and peripheral devices.
- **Master processor:** A processor that controls the entire system.
- **Timer:** Interrupts the system after a specified period and checks the sequence of events.
- **Kernel:** It is a piece of software used for providing secure access to the machine's hardware to various computer programs.
- **Power on self-test:** It is a diagnostic tool that provides the end user information about the computer.

2.10 SELF-ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. What is meant by single-processor systems?
2. Define significance of slave processors and master processor.

3. What can be included in the internal structure of an operating system?
4. State few advantages of the virtual machine and kernel.
5. Define monolithic kernel in an operating system.
6. What is the significance of interrupt service routine?
7. How do we define a privileged instructions?
8. Give few examples of input and output devices used for different types of applications.
9. Define POST.
10. How does a bootstrapping process works?
11. What is the goal of a kernel in an OS?

Long-Answer Questions

1. Describe single-processor and multiprocessor systems.
2. What is the difference between user interface system and protection system?
3. Explain briefly about the history and structure of an operating systems.
4. What is SPOOLing? Explain its functions and applications.
5. Compute the functions of time shared systems as well as single user systems?
6. Elaborate on the advantages and disadvantages of operating system operations.
7. Discuss briefly about the architecture of the virtual machine multiprogramming.
8. Differentiate between POST and BIOS with the help of examples.
9. Illustrate on the definitions and functions of kernel.

2.11 FURTHER READINGS

Silberschatz, Abraham, Peter B. Galvin and Greg Gagne. 2008. *Operating System Concepts,* 8th Edition. New Jersey: John Wiley & Sons.

Tanenbaum, Andrew S. 2006. *Operating Systems Design and Implementation*, 3rd Edition. New Jersey: Prentice Hall.

Tanenbaum, Andrew S. 2001. *Modern Operating Systems*. New Jersey: Prentice Hall.

Deitel, Harvey M. 1984. *An Introduction to Operating Systems*. Boston (US): Addison-Wesley.

NOTES

Stallings, William. 1995. *Operating Systems,* 2nd Edition. New Jersey: Prentice Hall.

Milenkovic, Milan. 1992. *Operating Systems: Concepts and Design*. New York: McGraw Hill Higher Education.

Mano, M. Morris. 1993. *Computer System Architecture*. New Jersey: Prentice Hall Inc.

UNIT 3 SYSTEM STRUCTURE

NOTES

Structure

3.0 INTRODUCTION

The components of modern operating systems are process management, main memory management, file management, I/O system management, secondary management, networking, protection system and command-interpreter system. A file manager or file browser is a computer program that provides a user interface to work with file systems. The device manager allows users to view and control the hardware attached to the computer. A process is a program in execution. A process needs resources including CPU time, memory, files and I/O devices to accomplish its task. These resources are either given to the process when it is created or when it is running. When the process completes, the OS reclaims all the resources. An operating system is a complex and normally huge software used to control and coordinate the hardware resources like CPU, memory and I/O devices to enable easy interaction of the computer with human and other applications. In a computer system, users have to run their programs, store programs or data permanently on secondary storage devices, and have to determine the malfunctioning programs and locate the information needed to identify the reasons for errors. Processes (running programs) may want to communicate with each other for sharing of data and for cooperated execution.

And processes may need to read data from input devices like keyboard, and write data to output devices like printer and monitor. A system call is the invocation of an operating system routine. System calls provide an interface between the processes of an operating system. These calls allow user-level processes to request some services from the operating system which process itself is not allowed to do.

In this unit, you will study the basics of operating system structures, operating system services, system calls, system programs, operating system design and implementations.

3.1 OBJECTIVES

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

- Describe the basics of operating system structures
- Understand the concept and functions of operating system services
- Discuss about the terms, such as system calls and system programs
- Analyse the advantages and disadvantages of operating system design and implementations.

3.2 OPERATING SYSTEMS STRUCTURES

Due to the complex nature of the modern operating systems, it is partitioned into smaller components. Each component performs the input and output functions. The components of modern operating systems are process management, main memory management, file management, I/O system management, secondary management, networking, protection system and command-interpreter system. Figure 3.1 illustrates the components used in operating systems structure.

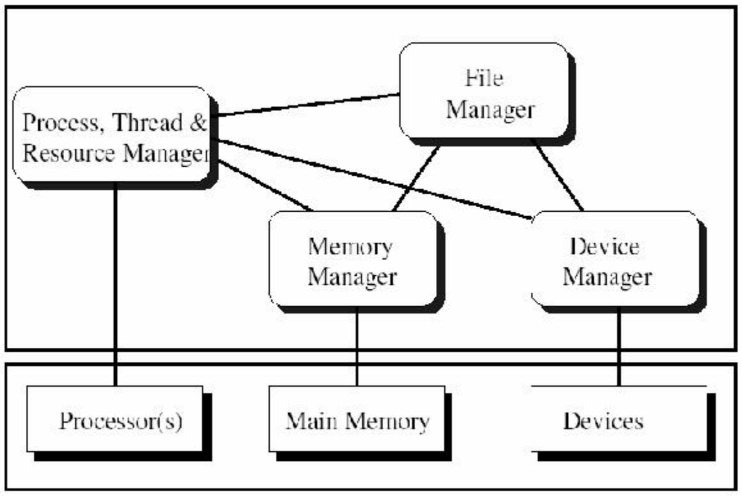


Fig. 3.1 *Components used in Operating Systems Structure*

A file manager or file browser is a computer program that provides a user interface to work with filesystems. The device manager allows users to view and control the hardware attached to the computer. A process is a program in execution. A process needs resources including CPU time, memory, files and I/O devices to accomplish its task. These resources are either given to the process when it is created or when it is running. When the process completes, the OS reclaims all the resources. The operating system is responsible for the following activities in connections with memory management:

NOTES

- It keeps track to check which parts of memory are currently being used and by whom.
- It decides which processes are used to load when memory space becomes available.
- It allocates and de-allocates memory space as needed.

Computers can store information on several different types of physical media, for example magnetic tape, magnetic disk, Compact Disk or CD, etc. For fast use of the computer system, the OS provides a uniform logical view of information storage. The kernel is the main component of most computer operating systems. It works as a bridge between applications and data processing done at the hardware level. The function of kernel includes managing the system's resources, and communicating between hardware and software components. Usually as a basic component of an operating system, a kernel can provide the lowest level abstraction layer for the resources especially processors and I/O devices that application software must control to perform its function. It typically makes these facilities available to application processes through interprocess communication mechanisms and system calls. Operating system structure is designed to decompose into smaller components with well-defined interfaces and dependences using layered approach, microkernels, modules and virtual machines. The features and functions of virtual machine are discussed in subsequent section.

Kernel Approach

- Kernel lies below system call interface and above the physical hardware.
- It provides large number of functions, such as CPU scheduling, memory management, I/O management, synchronization of processes, interprocess communication and other operating system functions.

CPU and I/O Structure

The Arithmetic and Logic Unit (ALU) and the Control Unit (CU) of a computer system are jointly known as the Central Processing Unit or CPU. You may call CPU as the brain of any computer system. It takes all major decisions, makes all sorts of calculations and directs different parts of the computer functions by activating and controlling the operations.

For a computer to start running, it needs to have an initial program to run. This initial program, also known as a bootstrap program, tends to be simple. It is stored in CPU registers. The role of the initial program or the bootstrap program is to load the operating system for the execution of the system. The operating system starts executing the first process, namely 'init', and waits for some event to occur. Event is known to occur by an interrupt from either the hardware or the software. Hardware can interrupt through system bus whereas software through system call.

When a CPU is interrupted, it immediately stops whatever it is doing and returns to a fixed location. This fixed location usually contains the starting address where the service routine for the interrupt is located.

I/O Structure

There are various types of I/O devices that are used for different types of applications. They are also known as peripheral devices because they surround the CPU and make a communication between computer and the outer world.

Input Devices: Input devices are necessary to convert our information or data into a form which can be understood by the computer. A good input device should provide timely, accurate and useful data to the main memory of the computer for processing. keyboard, mouse and scanner are the most useful input devices.

Output Devices: Visual Display Unit (VDU), terminals and printers are the most commonly used output devices.

Virtual Machines

The operating system provides applications with a virtual machine. This type of situation is analogous to the communication line of a telephony company, which enables separate and isolated conversations over the same wire(s). An important aspect of such a system is that the user can run an operating system of his/her choice.

The virtual machine concept can be well understood by understanding the difference between conventional multiprogramming and virtual machine multiprogramming. In conventional multiprogramming, processes are allocated a portion of the real machine resources, i.e., a resource from the same machine is distributed among several resources (Refer Figure 3.2(a)).

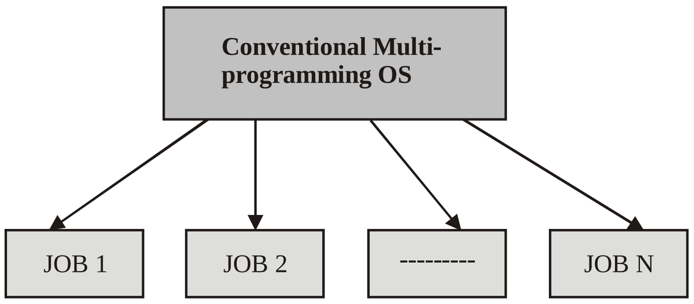


Fig. 3.2(a) *Conventional Multiprogramming*

In the virtual machine multiprogramming system, a single machine gives an illusion of many virtual machines, each of them having its own virtual processor and storage space which can be handled through process scheduling (Refer Figure 3.2(b)).

NOTES

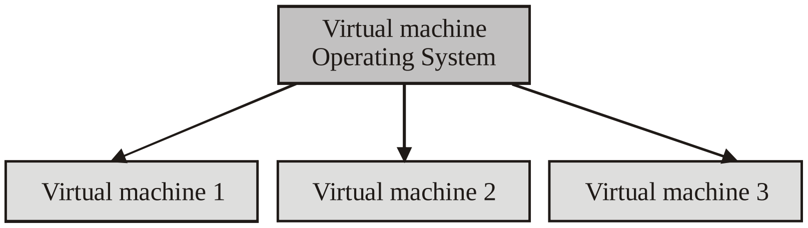


Fig. 3.2(b) *Virtual Machine Multiprogramming*

Advantages

Following are the advantages of virtual machine:

- Each user is allocated a machine which eliminates mutual interference between users.
- A user can select an OS of his/her choice for execution. Hence, the user can simultaneously use different operating systems on the same computer system.

3.3 SYSTEMS COMPONENTS

An operating system is a complex and normally huge software used to control and coordinate the hardware resources like CPU, memory and I/O devices to enable easy interaction of the computer with human and other applications. The objects or entities that an operating system manages or deals with include processes, memory space, files, I/O devices and networks. Let us first briefly describe each of these entities:

- **Process:** A process is simply the program in execution. For every program to execute, the operating system creates a process. A process needs resources like CPU, memory and I/O devices to execute a program. These resources are under the control of the operating system. In a computer, there will be many programs in the state of execution; hence a large number of processes demanding various resources are also needed to be maintained and managed in an operating system. When the execution is finished, the resources held by that process will be returned back to the operating system.
- **Memory Space:** As mentioned earlier, the execution of a program needs memory. The available memory is divided among various programs that are needed to execute simultaneously (concurrently) in a time multiplexed way. In normal type of memory, at a time only one memory location can be accessed. So, in a uniprocessor environment, because the secondary storage devices are much slower than main memory, the programs to be executed

NOTES

concurrently are loaded into memory and kept ready awaiting the CPU for fast overall execution speed. The memory is space multiplexed which is used to load and execute more than one program in a time interleaved way. Even if more than one CPU is available, at a time only one program memory area can be accessed. However, instruction that does not need main memory access (access can be from local memory or from CPU cache) can be executed simultaneously in a multiprocessor scenario.

- **Files:** Files are used to store sequence of bits, bytes, words or records. Files are an abstract concept used to represent storage of logically similar entities or objects. A file may be a data file or a program file. A file with no format for interpreting its content is called a plain unformatted file. Formatted file content can be interpreted and is more portable as one knows in what way the data inside is structured and what it represents. Example formats are Joint Photographic Experts Group or JPEG, Moving Picture Experts Group or MPEG, Graphics Interchange Format or GIF, Executable or EXE, MPEG Audio Layer III or MP3, etc.
- **I/O Devices:** The input/output devices of a computer include keyboard, monitor, mouse, pen, joystick, scanner, printer, modem, secondary storage devices like hard disk, floppy disk, CD (Compact Disk) ROMs (Read Only Memory), etc. Primary memory like RAM is volatile and the data or program stored there will be lost when we switch off the power to the computer system or when we shutdown it. Secondary storage devices are needed to permanently preserve program and data. Also, as the amount of primary storage that can be provided will be limited due to reasons of cost, a secondary storage is needed to store the huge amount data needed for various applications.
- **Network:** Network is the interconnection system between one computer and the other computers located in the same desk/room, same building, adjacent building or in any geographical location over the world. We can have wired or wireless network connections to other computers located anywhere in the world.

From the discussion above, it is clear that the operating system has various functions to carry out. So, the operating system consists of many subsystems to accomplish its various tasks. The major components or subsystems needed for an operating system are as follows:

- Process Management System
- Process Scheduling
- Memory Management System
- File and Secondary Storage Management System
- I/O System Management System
- Networking System

- Protection System
- User Interface System

The following subsections describe each of the above subsystems.

3.3.1 Process Management System

An operating system has to provide many services for the execution of user programs and system programs. Programs are executed as processes. A process is a program in execution. A program can be executed by many processes on behalf of different users. For example, a C++ compiler will be executed by many users as different processes. A running program has various data structures to represent its context. Code, data, Program Counter (PC), registers, open files and other resources in use altogether represent the context. A process encapsulates the whole context of a running program. Once the execution of a process is completed, it should be removed, or killed or deleted from the system. A process requesting for the resource that is not currently available must be made to wait or suspended and it must be resumed when the resource is available. More than one process sharing resources must be synchronized for the right and consistent use of such shared resources. A process should not be allowed to wait for a resource that is already held by another waiting process in a circular chain. In some cases, a job has to be executed as multiple cooperating processes. So, for the execution, control and deadlock-free cooperation of processes, the process management subsystem must provide services for the following:

- Creation and deletion of processes
- Suspension and resumption of processes
- Synchronizing execution of processes
- Communication between processes
- Preventing and handling deadlock situations

Modern operating systems also support the concept of threads which are lightweight processes. A process may have many threads to improve the response time of an application. In that case, services similar to that needed for the process management are required for thread management.

3.3.2 Process Scheduling System

In a computer system, there will be many processes in a ready state waiting for the CPU for execution. When and in what order to allocate the CPU for different processes, in the ready state, is called process scheduling or CPU scheduling. This scheduling is done based on various scheduling algorithms used for this purpose. In a multiuser computer system, the next process to be executed is selected based on the scheduling algorithm. A process will be allocated the CPU for a duration that is also dictated by the scheduling algorithms. After the expiry of permitted time, a hardware timer interrupts the current execution to enable the operating system to select the next process to execute based on the scheduling algorithm. The context of the outgoing process must be saved and the context of the new incoming process must be restored before handing over the CPU to the newly selected process. This is called a context switch. There are various scheduling algorithms that are in use for optimizing the performance of computer systems. Common among them are Round Robin, first-in-first-out, shortest time first, shortest remaining time next, etc. So, the process scheduling subsystem must provide the following functions for the execution of programs:

- Services for selecting next process to execute based on some scheduling algorithm
- Context switching
- Restarting the timer to interrupt at the end of the time duration allotted for the new process
- Transfer control to the new process

3.3.3 Memory Management System

The main memory is arranged as an array of bytes or words as dictated by the low-level architecture of a CPU. Every memory location has different addresses. These addresses can be from zero to the size of memory that can be accommodated in the CPU. This is called the address space of a CPU. These are the memory addresses that the CPU can access directly. As mentioned earlier, in a multiuser system, there will be many programs loaded and existing simultaneously in the main memory. The available main memory will be allocated to such programs/ processes. Processes may also request memory during the execution time. So, the operating system must allocate memory to satisfy the needs of different processes out of the free memory available at any time. When the processes complete their execution, the memory must be taken back to the free memory list. When sufficient main memory is not available, the memory occupied by some of the waiting processes or a part of the executing processes may be written to secondary storage, and the memory thus freed may be allocated to the demanding processes. So, the memory management subsystem has to provide the following facilities for the execution of programs:

- To keep track of the amount of memory spaces allocated to different processes and the addresses of such memory spaces.
- Allocate main memory when requested.
- De-allocate memory used by the process when it terminates and add it to the free space list for future allocation to other processes.

3.3.4 File Management System

File is an abstract concept for recoding the memory locations where the information is recorded on storage media. Files may be considered as information folders.

Information is recorded in secondary storage as sequences of bits, bytes, words or records. Files are stored on secondary storage media. Hard disk (magnetic disks), magnetic tapes and optical disks (CD-ROM, Digital Versatile Disk or DVD, etc.) are examples of secondary storage devices. Different storage devices are characterized by speed, capacity, data transfer rate and access methods. Files are organized into directories based on the information content or the purpose for which it is used, or based on the owner name. The various functions that a file management subsystem must provide are as follows:

NOTES

- Creation and deletion of files.
- Creation and deletion of directories.
- Opening, reading, writing and appending files and directories.
- Allocation of storage blocks to files and deallocation of storage blocks to free list.
- Keeping track of the usage of storage by files.
- Backing up files.

3.3.5 Input/Output System Management System

Users of a computer system interact mainly through I/O devices. Through the keyboard and mouse we can apply inputs and commands to the computer, and through the monitor we can get back the responses from the computer. I/O devices normally communicate with the computer through hardware-level interrupts. The operating system has to respond to such events by executing interrupt service routines stored as part of its code. The I/O subsystem provides utilities for all types of communication needed with the I/O devices. Moreover, every device type will have its own special program called **device driver** for interaction and communication with the operating system.

3.3.6 Networking System

Modern operating systems support execution of programs in a remote computer connected through communication network. Such a system is called network operating system. Network operating systems are loosely coupled software on loosely coupled hardware (independent computers connected to the network). Such a system must provide remote login facility for executing programs and remote copy command facility.

It is also possible to have a system with tightly coupled software on loosely coupled hardware. Such a system is called distributed operating system. A distributed operating system creates the impression of one large uniprocessor computer out of many networked computers. A single operating system runs in all the computers and they are tightly coupled through software. Files in such a system appear to have the same name (can have different access privileges) to different users working in any of the networked computers. A distributed system must provide a single global interprocess communication mechanism so that any process can communicate with any other process. Distributed operating system must also provide schemes for global protection of files.

3.3.7 Protection System

NOTES

In a multiuser system, there will be many users executing programs simultaneously/ concurrently. Also different users will have their own resources like programs and data files. Some users may permit their resources to be shared by other users in a controlled way. An operating system must provide facilities for protecting the shared use of resources like programs and files, and also provide facilities for controlled access rights to such resources by the users. The protection system must ensure that each resource of the system is accessed correctly and only by those processes that are allowed to do so.

3.3.8 User Interface System

Users interact with a computer using a command interpreter or shell program. Human users can also communicate through a human computer interface. They can enter commands through a terminal (keyboard/mouse and monitor) by typing on the keyboard or clicking on an icon or button on the desktop. The shell interprets commands and executes the operations by invoking operating system services. A user program can directly invoke the operating system services through software interrupts also known as system calls.

Check Your Progress

1. What are the functions of CPU?
2. Write the role of process scheduling in the virtual multiprogramming system.
3. Define the term network.
4. Name the resources present in the context of the running program.
5. What functions should be provided by the process scheduling sub system for the execution of programs?
6. What is the address space of the CPU?
7. What should the protection system ensure?

3.4 OPERATING SYSTEM SERVICES

In a computer system, users have to run their programs, store programs or data permanently on secondary storage devices, and have to determine the malfunctioning programs and locate the information needed to identify the reasons for errors. Processes (running programs) may want to communicate with each other for sharing of data and for cooperated execution. And processes may need to read data from input devices like keyboard, and write data to output devices like printer and monitor. It will not be practicable for each user program to build these facilities for accomplishing its task. Moreover, such facilities if built into each user program may lead to a system that is unsecured and undependable. So, the operating system must provide the services needed to accomplish all these tasks. The services provided by an operating system for these and other purposes may be grouped under the four headings given below:

NOTES

- Program Execution, Control and Communication.
- I/O Operations.
- File Manipulation.
- Error Detection.

Program Execution, Control and Communication Services

A computer system must provide facilities for execution of programs (processes) and for controlling these programs. A program needs memory for loading the program before the start of execution and other resources including the CPU for execution. In a system, there will be many users and each user will be executing many tasks all of which need memory, CPU and other resources. These resources will be normally limited and shared among processes in a space and/or time multiplexed fashion. In some situations, an application may be executed as multiple cooperating processes possibly on different computers connected through network. It may be necessary to exchange information between processes. Also, processes will be competing for using the available resources. So, these functions are to be done in a supervisory mode by the operating system. Thus, an OS must provide the following services:

- Creation and deletion of processes.
- Suspension and resumption of processes.
- Synchronizing execution of processes.
- Communication between processes on a single or different computers.
- Preventing and handling deadlock situations.
- Keeping track of the amount of memory spaces allocated to different processes and the addresses of such memory spaces.
- Allocating main memory when requested.
- De-allocating the main memory used by the process when it terminates and adding it to the free space list for future allocation to other processes.

Services for I/O Operations

User programs need to do I/O operations on different types of devices during their executions. Each device has its own controller and associated commands for I/O. It will be highly inefficient that a user learns all these low-level commands and incorporate these in the program. The operating system must hide from users the hardware details of the devices and provide high-level commands like open, read and write to handle I/O on all devices irrespective of their types. Also, from the protection of files and efficiency point of view, it is not advisable to do the I/O operations directly by a user program. Manufacturer of each device type will provide a program appropriate to the device called device driver. The device drivers can take high-level commands like open, read and write as inputs and translate them into low-level commands that the controller of the device can understand. So, an OS must provide the services like open, read and write to handle all devices in a uniform manner.

Services for File Manipulation

We need files for storing data and programs permanently. You may have heard of different file types like text, image, executable, music, video and program source files. A program takes input from some files and writes output to some other files. Generally speaking, we store organized information in files. Files are created on secondary storage devices. Filesystem is the part of the OS program that is used to create, manipulate, organize and maintain the files. Files are organized using directories in a filesystem. Users need not bother about how the file is stored and retrieved from secondary storage. It is the duty of operating system to hide the hardware specific details of secondary storage. The OS provides the following services to manipulate and maintain files in the system:

- Create files and directories
- Open files
- Close files
- Read files
- Write files
- Rename files
- Copy files
- Delete files and directories
- List files
- Search files and directories
- Allocate, de-allocate and manage secondary storage space

Services for Error Detection

A running program may have errors or bugs, which are to be detected and corrected. Errors may be due to many reasons including trying to execute illegal instruction or accessing of memory (intentionally or unknowingly) not allocated for a user or infinite loop in a process or trying to execute an illegal operation on I/ O devices. The operating system monitors the functioning of different programs to detect errors and to localize them. Users can confidently execute their programs without the worry of malfunctioning.

3.5 SYSTEM CALLS

NOTES

A system call is the invocation of an operating system routine. In order to understand how to use system calls, consider writing a simple program to read data from one file and to copy them to another file. There are two names of two different files: one is the input file and the other is the output file. One approach to understand system calls is to ask the user for the names of the two files in an interactive system. This approach will require the following sequence of system calls:

- Write a prompting message on the screen.
- Read from the keyboard the character that the two files have.

Once the two file names are obtained, the program must open the input file and create the output file. Each of these operations requires other system calls and may encounter errors. When the program tries to open the input file, it may find that no file of that name exists or that the file is protected against access. In these cases, the program prints a message on the screen and then terminates abnormally. This requires another system call. If the input file exists, then we must create the new output file. We may find an output file with the same name. This situation may cause the program to abort or we may delete the existing file and create a new one. After opening both the files, we may enter a loop that reads from input file and writes to an output file. Each read and write must return status information regarding various possible error conditions. Finally, after the entire file is copied, the program may close both files.

Types of System Calls

Some of the system calls are discussed as follows:

Create: In this system call, a new process is created with the specified attributes and identifiers. Some of the parameters which can be defined while a process is being created are as follows:

- Level of privilege, such as system or user
- Priority
- Size and memory requirements
- Maximum data area and/or stack size
- Maximum protection information and access rights
- Other system dependent data

Delete: The delete service is also called destroy, terminate or exit. Its execution causes the operating system to destroy the designated process and remove it from the system.

Abort: It is used to forcibly terminate the process. Although a process could conceivably abort itself, the most frequent use of this call is for involuntary terminations, such as removal of malfunctioning process from the system.

Fork/Join: Another method of process creation and termination is by means of fork/join pair, originally introduced as primitives for multiprocessor system. The fork operation is used for splitting a sequence of instructions into two concurrently executable sequences. Join operation is used to merge the two sequences of code divided by the fork; it is available to a parent process for synchronization with a child.

Suspend: The suspend system call is also called **block** in some systems. The designated process is suspended indefinitely and placed in the suspend state. A process may be suspended itself or by another process authorized to do so.

Resume: The resume system call is also called **wakeup** in some systems. This call resumes the target process which is presumably suspended. Obviously, a suspended process cannot resume itself because a process must be running to have its operating system call processed. So, a suspended process depends on a partner process to issue the resume.

Delay: The delay system call is also called **sleep**. The target process is suspended for the duration of the specified time period. The time may be expressed in terms of system clock ticks that are system-dependent and not portable or in terms of standard time units, such as seconds and minutes. A process may delay itself or, optionally, delay some other process.

Get Attributes: It is an enquiry to which the operating system responds by providing the current values of the process attributes, or their specified subset from the Process Control Block (PCB).

Change Priority: It is an instance of a more general set-process-attributes system call. Obviously, this call is not implemented in systems where process priority is static.

3.6 SYSTEM PROGRAMS

System calls provide an interface between the processes of an operating system. These calls allow user-level processes to request some services from the operating system which process itself is not allowed to do. In handling the trap, the operating system will enter in the kernel mode where it has access to privileged instructions and can perform the desired service on the behalf of user-level process. It is because of the critical nature of operations that the operating system itself does them every time they are needed, for example a process involves a system call telling the operating system to read or write particular area and this request is satisfied by the operating system for I/O. System programs provide basic functioning to users so that they do not need to write their own environment for program development (editors and compilers) and program execution (shells). In some cases, they are bundles of useful system calls. System programs provide a convenient environment for program development and execution with the help of following tasks:

- **File Manipulation:** This task is used to perform the process of creating, deleting, copying, renaming, printing and listing the files, etc.
- **Status Information:** This task is used to perform the process of displaying date, time, disk space, memory size, etc.
- **File Modification:** This task is used to perform the process of creating and modifying files using text editors.
- **Programming Language Support:** This task is performed by using compilers, assemblers, and interpreters to interpret the system programming.
- **Program Loading and Execution:** Loaders and linkers are used to perform this task.
- **Communications:** These tasks include the process of remote login, and sending and receiving messages.

Most users' view of the operation system is defined by system programs not the actual system calls.

In a programmer's view, processes achieve concurrent execution of programs. The main process for concurrent execution of a program is to create child processes. It should assign appropriate priorities to them to either achieve increased computation speed or execute some functions of high priority. The main process and the child process also have to interact to achieve their common goal. This interaction may involve the exchange of data or may require the two processes to coordinate their activities.

An operating system provides the following four types of operations to implement the programmer's view of processes:

- Creating child processes and assigning priorities to them
- Terminating the child processes
- Determining the status of child processes
- Sharing, communication and synchronization between processes

The following tasks are used for process management:

Create a Process: Creates a new process and assigns to it a priority and a unique identifier called its process id and returns the process id to the caller.

Status: Checks the status of a process and returns a code terminated or alive.

Terminate: Terminates the specified child process or terminates itself if no process is specified.

3.7 OPERATING SYSTEM DESIGN AND IMPLEMENTATION

NOTES

Modern operating systems are designed and developed carefully due to their size and complexity. A common approach is to divide the systems into small components. Following are the types of system structure according to the structure of the operating system:

Monolithic Architecture of Operating System

The components of monolithic operating system are organized in any design without any reservation. Similar to other operating systems, applications in monolithic operating system are separated from the operating system itself. Therefore, the operating system code runs in a privileged processor mode which is also known as kernel mode with access to system data and to the hardware. Various applications run in a non-privileged processor mode known as user mode with a limited set of interfaces available and with limited access to system data. The monolithic operating system structure with separate user and kernel processor mode is shown in Figure 3.3.

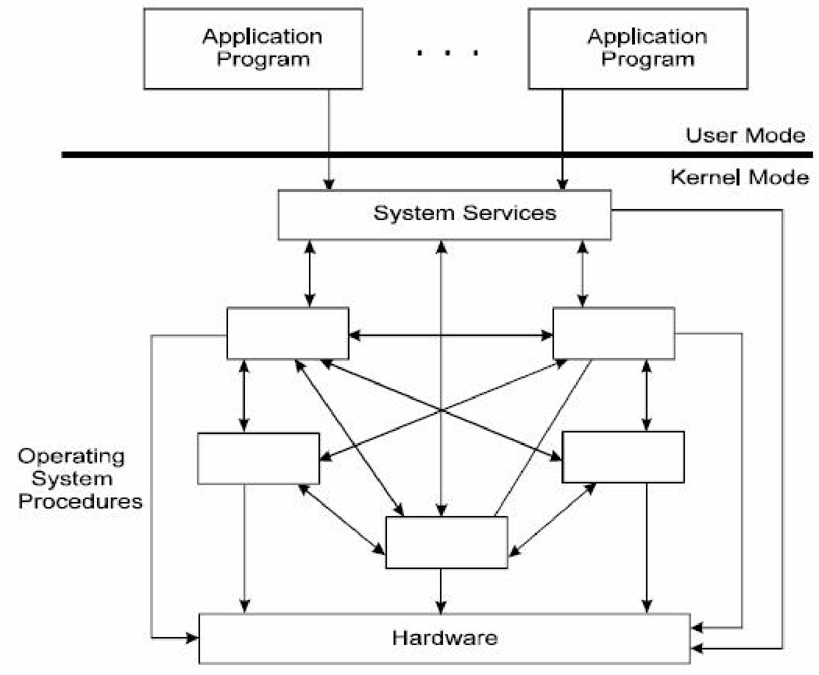


Fig. 3.3 *Monolithic Structure of Operating System*

When a user mode program calls a system service, the processor traps the call and then switches the calling thread to kernel mode. Completion of system service switches the thread back to the user mode by the operating system and allows the caller to continue. The monolithic structure does not enforce data hiding in the operating system.

Layered Operating System

The operating system is divided into a number of layers (levels), each built on top of lower layers. The bottom layer (layer 0) is the hardware whereas the highest (layer N) is the user interface. An operating system layer is an implementation of an abstract object that is the encapsulation of data and operations to manipulate those data. These operations (routines) can be invoked by higher level layers. The layer itself can invoke operations on lower level layers. Layered approach provides modularity. With modularity, layers are selected such that each layer uses functions (operations) and services of only lower level layers. Each layer is implemented by using only those operations that are provided by lower level layers. The major difficulty is appropriate definition of various layers. The components of layered operating system are organized into modules. Each module provides a set of functions that other module can call. Interface functions at any particular level can invoke services provided by lower layers but not the other way around. The layered operating system structure with hierarchical organization of modules is shown in Figure 3.4.

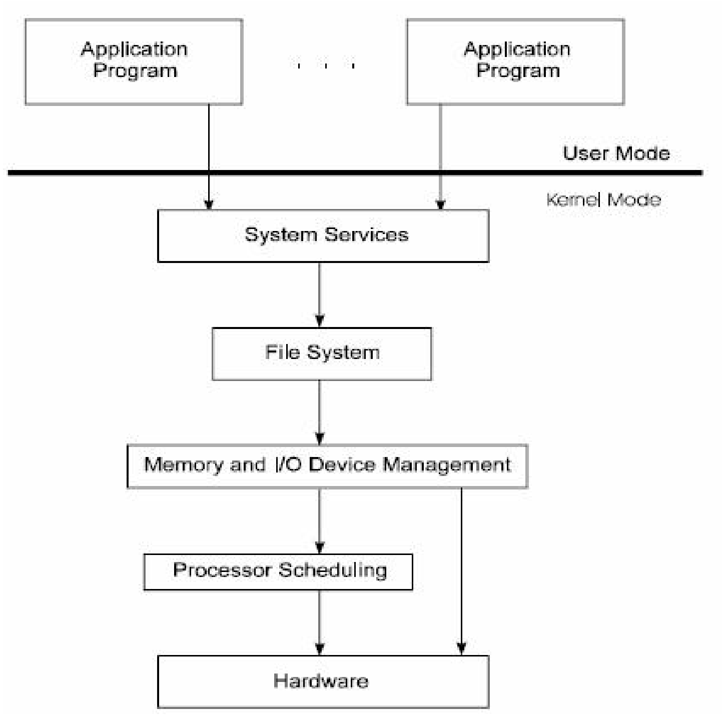


Fig. 3.4 *Layered Operating System*

One advantage of a layered operating system structure is that each layer of code is given access to only the lower level interfaces and data structures it requires, thus limiting the amount of code that wields unlimited power. In this approach, the *N*th layer can access services provided by the (*N*–1)th layer and provide services to the (*N*+1)th layer. This structure also allows the operating system to be debugged starting at the lowest layer, adding one layer at a time until the whole system works

NOTES

correctly. Layering also makes it easier to enhance the operating system; one entire layer can be replaced without affecting other parts of the system. Layered operating system delivers low application performance in comparison to monolithic operating system. Examples of layered operating systems are Virtual Address eXtension/Virtual Memory System or VAX/VMS, Multics, UNIX, etc.

Client-Server or Microkernel Operating System

In microkernel system structure, communication takes place between user modules using message passing. This structure moves from the kernel to the user space. This structure is easy to extend a microkernel and to port the operating system to new architectures. This structure is more reliable (less code is running in kernel mode) and also more secure. The advent of new concepts in operating system design, microkernel, is aimed at migrating traditional services of an operating system, out of monolithic kernel into the user level process. The idea is to divide the operating system into several processes and each implements a single set of services, for example I/O servers, memory servers, process servers, threads interface system, etc. Each server runs in user mode and provides services to the requested client. The client can be another operating system component or application program that requests a service by sending a message to the server. An operating system kernel or microkernel running in kernel mode delivers the message to the appropriate server and the server performs the operations and then microkernel delivers the results to the client in another message as illustrated in Figure 3.5.

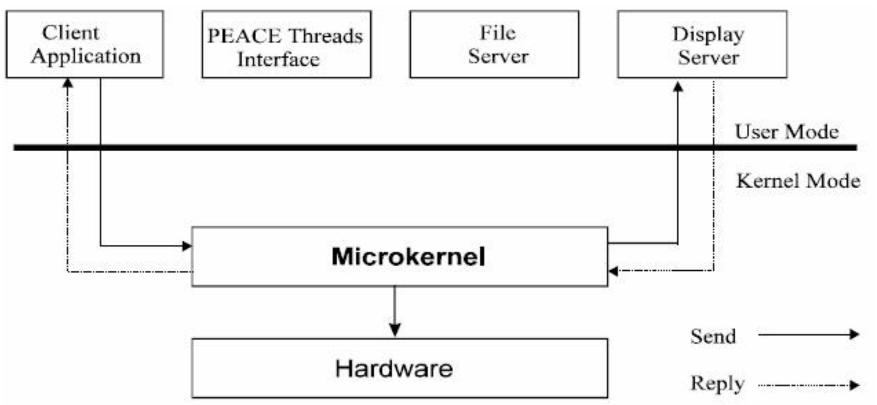


Fig. 3.5 *Microkernel Operating System*

Examples of microkernel operating systems are Centre or Develoment of advance Computing or C-DAC PARAS, Windows NT/95, Mach, QNX, Chorus, etc. The important characteristics of microkernel operating system are simplified base, traditional services of operating system have become peripheral, improved reliability, vertical style access instead of horizontal, message passing facilities leads to distributed computing model (transparent local or remote services), subsystem's (Portable Operating System Interface for UNIX (POSIX), database, file, network server, etc.), monolithic application performance competence, and foundations for modular and portable extensions.

Virtual Machine Architecture Operating System

The virtual machine concept provides complete protection of system resources since each virtual machine is isolated from all other virtual machines. This isolation, however, permits no direct sharing of resources. A virtual machine system is a perfect vehicle for operating system's research and development. System development is done on the virtual machine instead of on a physical machine and so does not disrupt normal system operation. The virtual machine concept is difficult to implement due to the effort required to provide an exact duplicate to the underlying machine.

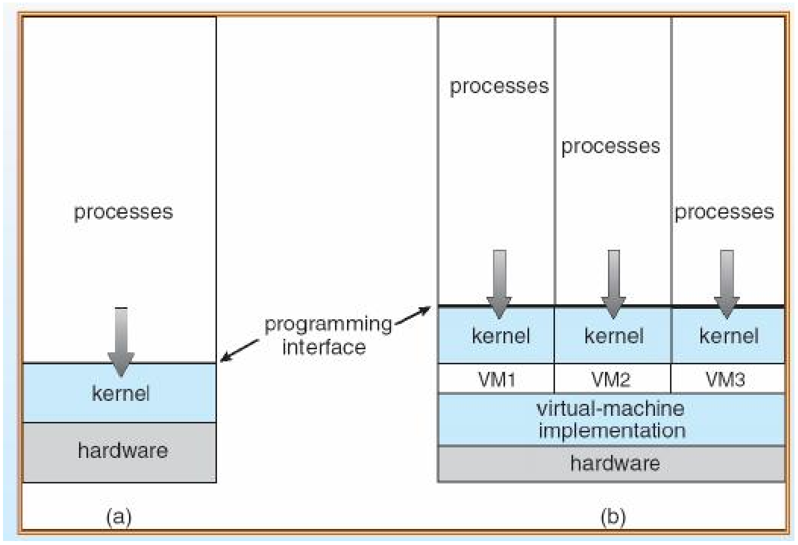


Fig. 3.6 *Virtual Memory Architecture of Operating System*

Figures 3.6 (a) and (b) display the structure of non-virtual and virtual machine, respectively. In the virtual machine, complete protection of system resources is provided since each virtual machine is isolated from all other virtual machines. This isolation permits no direct sharing of resources. A virtual machine system is a perfect vehicle for operating systems research and development. System development is done on the virtual machine instead of on a physical machine and so does not disrupt normal system operation. The resources of the physical computer are shared to create the virtual machines in which CPU scheduling can create the appearance that users have their own processor. Spooling and a filesystem can provide virtual card readers and virtual line printers whereas a normal user time sharing terminal serves as the virtual machine operator's console. Virtual machine is an illusion of a real machine. It is created by a real machine operating system, which makes a single real machine appear to be several real machines. The architecture of virtual machine is shown in Figure 3.5. The best example of virtual machine architecture is International Business Machines or IBM 370 computer. In this system each user can choose a different operating system. Actually, virtual machine can run several operating systems at once on its virtual machine. Its multiprogramming shares the resource of a single machine in different manner. Following factors are implemented to the virtual machine:

- **Control Program or CP:** Control program creates the environment in which virtual machine can execute. It gives to each user facilities of real machine, such as processor, storage I/0 devices, etc.
- **Conversation Monitor System or CMS:** Conversation monitor system is a system application having features of developing program. It contains editor, language translator and various application packages.
- **Remote Spooling Communication System or RSCS:** Remote spooling communication system provides virtual machine with the ability to transmit and receive file in distributed system.
- **Interactive Problem Control System or IPCS**: An interactive problem control system is used to fix the virtual machine software problems.

MS DOS System Structure

Microsoft Disk Operating System (MS DOS) is written to provide the functionality in the least space. To apply this concept, tasks are not divided into modules. Although MS DOS supports simple structure but its interfaces and levels of functionality are not well separated. Figure 3.7 displays the structure of MS DOS system in which the resident system program resides in application programs whereas MS DOS device drivers are connected to Read Only Memory Basic Input/Output System (ROM BIOS) device drivers which support a special kind of program. This program is required to enable the CPU to talk to other devices. A ROM chip stores these programs and these programs are collectively known as the BIOS. In computing, a device driver or software driver is a computer program allowing higher level computer programs to interact with a hardware device. A driver typically communicates with the device through the computer bus or communications subsystem to which the hardware connects. When a calling program invokes a routine in the driver, then the role of driver is to issue the commands to the device. Once the device sends data back to the driver, the driver may invoke routines in the original calling program.

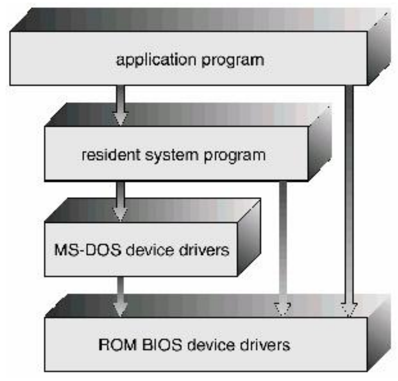


Fig. 3.7 *MS DOS System Structure*

UNIX System Structure

UNIX system structure is limited by hardware functionality. The original UNIX operating system had limited structuring. The UNIX operating system consists of two prime parts:

- Systems programs use kernel supported system calls to provide useful functions, such as compilation and file manipulation.
- The kernel consists of everything below the system call interface and above the physical hardware and provides the filesystem, CPU scheduling, memory management and other operating system functions.

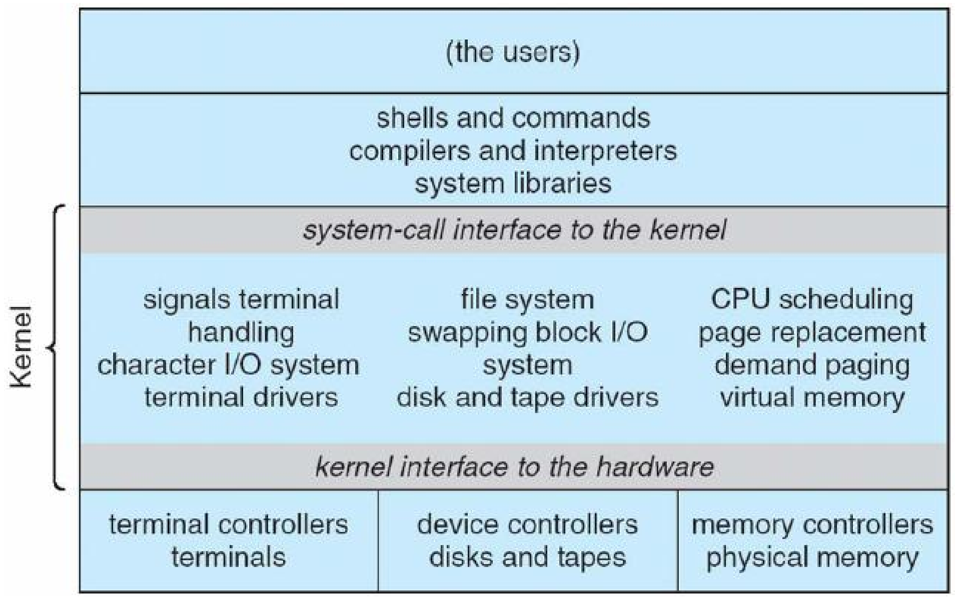


Fig. 3.8 *UNIX System Structure*

Figure 3.8 displays an arrangement of UNIX system structure where the users are in the first level whereas shells and commands, compilers and interpreters system libraries reside on the second level. System call interface to the kernel contains signals handling character I/O system terminal drivers, filesystem swapping block I/O system, disk and tape drivers, and also CPU scheduling, page replacement, demand paging, etc. The kernel interface to the hardware also resides on the same level. At last level, terminal controllers, device controllers, disks and tapes and memory controllers reside. The kernel consists of everything below the system call interface and above the physical hardware and provides the filesystem, CPU scheduling, memory management, and other operating system functions and also a large number of functions for one level.

NOTES

Check Your Progress

8. State the various services that should be provided by the OS.
9. What is system call?
10. What is the abort system call used for?
11. Which operations should be provided by the operating system to implement the programmer's view of processes?
12. What are the two prime parts of UNIX system structure?

3.8 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. You may call CPU as the brain of any computer system. It takes all major decisions, makes all sorts of calculations and directs different parts of the computer functions by activating and controlling the operations.
2. In the virtual machine multiprogramming system, a single machine gives an illusion of many virtual machines, each of them having its own virtual processor and storage space which can be handled through process scheduling.
3. Network is the interconnection system between one computer and the other computers located in the same desk/room, same building, adjacent building or in any geographical location over the world.
4. A running program has various data structures to represent its context. Code, data, Program Counter (PC), registers, open files and other resources in use altogether represent the context.
5. The process scheduling subsystem must provide the following functions for the execution of programs:
   - Services for selecting next process to execute based on some scheduling algorithm
   - Context switching
   - Restarting the timer to interrupt at the end of the time duration allotted for the new process
   - Transfer control to the new process
6. The main memory is arranged as an array of bytes or words as dictated by the low-level architecture of a CPU. Every memory location has different addresses. These addresses can be from zero to the size of memory that can be accommodated in the CPU. This is called the address space of a CPU.

7. The protection system must ensure that each resource of the system is accessed correctly and only by those processes that are allowed to do so.
8. An OS must provide the following services:
   - Creation and deletion of processes
   - Suspension and resumption of processes
   - Synchronizing execution of processes
   - Communication between processes on a single or different computers
   - Preventing and handling deadlock situations
   - Keeping track of the amount of memory spaces allocated to different processes and the addresses of such memory spaces
   - Allocating main memory when requested
   - De-allocating the main memory used by the process when it terminates and adding it to the free space list for future allocation to other processes.
9. A system call is the invocation of an operating system routine.
10. The abort system call is used to forcibly terminate the process.
11. An operating system provides the following four types of operations to implement the programmer's view of processes:
    - Creating child processes and assigning priorities to them
    - Terminating the child processes
    - Determining the status of child processes
    - Sharing, communication and synchronization between processes
12. The UNIX operating system consists of two prime parts:
    - Systems programs use kernel supported system calls to provide useful functions, such as compilation and file manipulation.
    - The kernel consists of everything below the system call interface and above the physical hardware and provides the filesystem, CPU scheduling, memory management and other operating system functions.

NOTES

3.9 SUMMARY

- Operating system is a software program that enables the computer hardware to communicate and operate with the computer software.
- Kernel lies below system call interface and above the physical hardware.
- The arithmetic and logic unit and the control unit of a computer system are jointly known as the central processing unit.

NOTES

- There are various types of I/O devices that are used for different types of applications.
- In the virtual machine multiprogramming system, a single machine gives an illusion of many virtual machines, each of them having its own virtual processor and storage space which can be handled through process scheduling.
- An operating system is a complex and normally huge software used to control and coordinate the hardware resources like a CPU, memory and I/O devices to enable easy interaction of the computer with human and other applications. The objects or entities that an operating system manages or deals with include processes, memory space, files, I/O devices and networks.
- An operating system has to provide many services for the execution of user programs and system programs.
- When and in what order to allocate the CPU for different processes, in the ready state, is called process scheduling or CPU scheduling. This scheduling is done based on various scheduling algorithms used for this purpose.
- The operating system must allocate memory to satisfy the needs of different processes out of the free memory available at any time.
- File is an abstract concept for recoding the memory locations where the information is recorded on storage media. The various functions of file management subsystem are, creation and deletion of files; creation and deletion of directories; opening, reading, writing and appending files and directories; allocation and deallocation of storage blocks, etc.
- The I/O subsystem provides utilities for all types of communication needed with the I/O devices.
- Networking systems must provide remote login facility for executing programs and remote copy command facility.
- The protection system must ensure that each resource of the system is accessed correctly and only by those processes that are allowed to do so.
- Users interact with a computer using a command interpreter or shell program.
- In a computer system, users have to run their programs, store programs or data permanently on secondary storage devices, and have to determine the malfunctioning programs and locate the information needed to identify the reasons for errors.
- A computer system must provide facilities for execution of programs (processes) and for controlling these programs.
- The operating system must hide from users the hardware details of the devices and provide high-level commands like open, read and write to handle I/O on all devices irrespective of their types.

- Operating system provides various services for file manipulation, like creating, opening, closing, reading writing, renaming, copying, deleting, listing and searching files.
- The operating system monitors the functioning of different programs to detect errors and to localize them.
- A system call is the invocation of an operating system routine.
- Some of the frequently used system calls are create, delete, abort, fork/join, suspend, resume, delay, get attributes, change priority, etc.
- In a programmer's view, processes achieve concurrent execution of programs. The main process for concurrent execution of a program is to create child processes.
- The various types of system structure according to the structure of the operating system are monolithic architecture, layered, client-server or microkernel, virtual machine architecture, MS DOS system and UNIX system structure.

3.10 KEY WORDS

- **OS:** A software program that enables the computer hardware to communicate and operate with the computer software
- **Process:** A program in execution
- **Network:** The interconnection system between one computer and the other computers located in any geographical location over the world
- **System call:** Invocation of an operating system routine

3.11 SELF-ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. Define the I/O structure.
2. What is the importance of virtual machine concept?
3. Name the entities which the operating system manages.
4. Give the major components needed for an operating system.
5. Define the term networking system.
6. What are the services provided by operating systems?
7. Name the parameters of create system call.

8. Which system calls are used to implement the programmer's view of processes?
9. What do you understand by microkernel operating system?

NOTES

Long-Answer Questions

1. Describe the various objectives of operating systems.
2. Write a note on operating system structures.
3. Explain the subsystems needed for an operating system.
4. Describe the various services provided by an operating system with the help of examples.
5. Discuss the various types of system calls.
6. Illustrate the programmer's view of processes.
7. Describe system structures with the help of illustration.

3.12 FURTHER READINGS

Silberschatz, Abraham, Peter B. Galvin and Greg Gagne. 2008. *Operating System Concepts*, 8th Edition. New Jersey: John Wiley & Sons.

Tanenbaum, Andrew S. 2006. *Operating Systems Design and Implementation*, 3rd Edition. New Jersey: Prentice Hall.

Tanenbaum, Andrew S. 2001. *Modern Operating Systems*. New Jersey: Prentice Hall.

Deitel, Harvey M. 1984. *An Introduction to Operating Systems*. Boston (US): Addison-Wesley.

Stallings, William. 1995. *Operating Systems*, 2nd Edition. New Jersey: Prentice Hall.

Milenkovic, Milan. 1992. *Operating Systems: Concepts and Design*. New York: McGraw Hill Higher Education.

Mano, M. Morris. 1993. *Computer System Architecture*. New Jersey: Prentice Hall Inc.

BLOCK - II
PROCESS CONCEPT

NOTES

UNIT 4 PROCESSES IN OPERATING SYSTEM

Structure

4.0 INTRODUCTION

Previously, there was a boundation of loading only one program into the main memory for execution at a time. This program was very multifaceted and resourceful as it had access to all the hardware resources, such as memory, CPU time, I/O devices, and so on. With time improvements were accepted as new systems incorporate a variety of new and powerful features that dramatically improved the efficiency and functionality of your overall system. Modern computer systems corroborate multiprogramming, which allows a number of programs to reside in the main memory at the same time. These programs have the potential to run a number of programs simultaneously thereby requiring the system resources to be shared among them. Multi-programmed systems need to distinguish among the multiple executing programs, which is accomplished with the concept of a process (also called task on some systems). A process is a program under execution or a set of executable machine instructions. A process can be either a system process executing the system's code or a user process executing the user's code.

When multiple processes run on a system concurrently and more than one process requires the CPU at the same time, then it becomes essential to select any one process to which the CPU can be allocated. To serve this purpose, logical scheduling is required. In addition, several processes being in operation on a system also need to intercommunicate in order to reciprocate some data or information. This kind of intercommunication between several processes is referred to as Inter-Process Communication (IPC).

In this unit, you will study the basic concepts of processes, definition and types of process scheduling, operations to be performed on processes, inter-process communication.

4.1 OBJECTIVES

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

- Introduce the basic concepts of processes
- Discuss about the various states of a process and the transition between these states
- Explain the various operations that can be performed on processes
- Understand the term process scheduling
- Provide an overview of inter-process communication

4.2 BASIC CONCEPTS OF THE PROCESS

In this, we will discuss some basic concepts of processes.

The Process

As stated earlier, a program undergoing an execution procedure is termed as a process. There is a hairline difference between the program and process in the sense that a program is a passive entity that does not initiate anything by itself whereas a process is an active entity that performs all the actions specified in a particular program. A process comprises of not only the program code (known as **text section**) but also a set of global variables (known as **data section**) and the Process Control Block (PCB). The process control block of a process contains some additional information about the process like a program counter to specify the current activity in progress, the contents of CPU's registers, a process stack to store temporary data like function parameters, local variables used in a function, return addresses, and so on.

Note*: The set of instructions, data, and stack together form the* ***address space*** *of a process.*

There can be either one-to-one or one-to-many relationship between programs and processes. A one-to-one relationship exists in case only a single instance of a program is running on the system. Regardless of whether the multiple instances of a single program are running simultaneously or a concurrent program (A program that requires some of its parts to be executed concurrently) is being run, there exists one-to-many relationship between programs and processes. In this case, the text section of the multiple instances will be same but the data section will be different.

One important thing to notice about the processes is that some processes involve higher computation than I/O operations, thereby, demanding more use of CPU than I/O devices during their life time. Such processes where the speed of computation is governed by CPU are called **CPU-bound** or **compute-bound**. Contrastive to this, are some processes where there are a lot of involvement I/O operations as compared to computation during their life time. Such processes are called **I/O-bound** as the speed of execution is governed by the input/output device not by the speed of the CPU**.**

Process States

Each process in the operating system tagged with a '**State**' variable—an integer value that helps the operating system to settle conclusively what should be done with the active process and also gives the evidence of the nature of the current activity in a process. A process may be in one of the following states depending on the current activity of the process.

- **New**: A process is said to be in a 'New' state if it is being created for the first time.
- **Ready**: A process is said to be in a 'Ready' state if it is ready for the execution and waiting for the CPU to be allocated to it.
- **Running**: A process is said to be in a 'Running' state if CPU has been allocated to it and it is being executed.
- **Waiting**: A process is said to be in a 'Waiting' state (also called '**blocked**' state) if it has been blocked by some event. Unless that event recurs, the process cannot resume the execution procedure. Examples of such blocking events are completion of some I/O operation, reception of a signal, and so on. Note that a process in waiting state is unable to run even if the CPU is available.
- **Terminated**: A process is said to be in a 'Terminated' state if it has completed its execution normally or it has been terminated abruptly by the operating system because of some error or killed by some other process.

***Note**: On a single processor system, only one process may be in running state at one time; however in a multiprocessor system with m CPUs, at most m processes may be in running state at one time.*

Each process undergoes changes in states during its life time. This change that occurs in the state of a process is known as **state transition** of a process. By and large, it is caused by the occurrence of some event in the system. There are many possible state transitions (Refer Figure 4.1) that may crop up along with their possible causes are as follows:

- **New $\rightarrow$ Ready**: This transition takes place if a new process has been loaded into the main memory and it is waiting for the CPU to be allocated to it.

- **Ready → Running**: This transition takes place if the CPU has been allocated to a ready process and it has started its execution.
- **Running → Ready**: This transition may occur if:
  (i) The time slice of the currently running process has expired.
  (ii) Some higher priority process gets ready for execution, and so on.

  In this case, the CPU is preempted from the currently executing process and allocated to some another ready process.
- **Running → Waiting**: This transition may take place if the currently running process.
  (i) Needs to perform some I/O operation.
  (ii) Has to wait for a message or some action from another process.
  (iii) Requests for some other resource.

  In this case, the CPU gets relieved by the process and can be allocated to some another ready process.
- **Running → Terminated**: This transition happens if the currently running process
  (i) Has successfully accomplished its task and requests to the operating system for its termination.
  (ii) Is terminated by its parent in case the function performed by it is no longer required.
  (iii) Is terminated by the kernel because it has exceeded its resource usage limit or involved in a deadlock.

  In this case, the CPU is preempted from the currently running process and allocated to some another ready process.
- **Waiting → Ready**: This transition takes place if an event (for example, I/O completion, signal reception, synchronization operation, etc.) for which the process was waiting, has occurred.

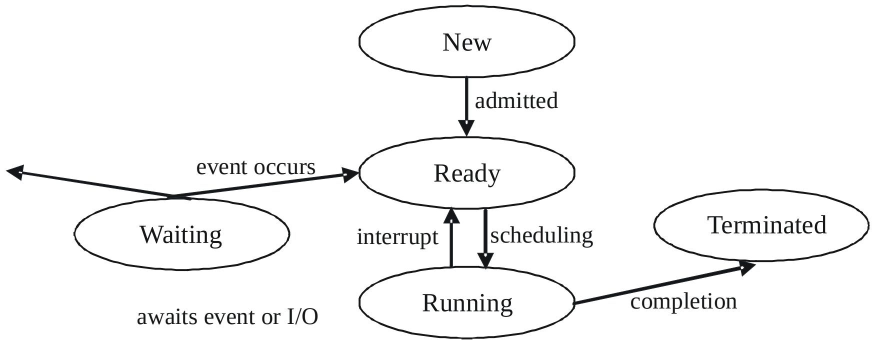


Fig. 4.1 *Process State Transition Diagram*

Process Control Block

The operating system maintains a structurally organized table called **process table** to keep track of and record all the processes in the system that includes an entry for each process. This entry is called **Process Control Block** (**PCB**)—a data structure created by the operating system for representing a process. A process control block stores descriptive information pertaining to a process such as its state, program counter, memory management information, information about its scheduling, allocated resources, accounting information, and so on, that is required to control and manage a particular process. The basic purpose is to apparently manifest the hitherto advancement of a process. Some of the important fields stored in a PCB are as follows:

- **Process ID**: Each process is assigned a unique identification number called **Process Identifier** (**PID**) by the operating system at the time of its creation. PID is used to refer the process in the operating system.
- **Process state**: It stores the current state of a process that can be new, ready, running, waiting, or terminated.
- **Parent Process ID**: It stores the PID of the parent, if the process has been created by some other process.
- **Child Process IDs**: It stores the PID s of all the child processes of a parent process.
- **Program Counter**: It contains the address of the instruction that is to be executed next in the process. Whenever CPU switches from one process to another, the program counter of the old process is saved so that the operating system could resume with the same instruction whenever the old process is restarted.
- **Event Information**: If the process is in waiting state, then this field contains the information about the event for which the process is waiting to happen. For example, if the process is waiting for an I/O device, then this field stores the ID of that device.
- **Memory Management Information**: It includes information related to the memory configuration for a process such as the value of base and limit registers, the page tables (If paging memory management technique has been used) or the segment tables (If segmentation memory management technique has been used).
- **CPU Registers**: It stores the contents of index registers, general purpose registers, condition code information, and so on, at the time when the CPU was last freed by the process or preempted from the process.
- **CPU Scheduling Information**: It includes information used by the scheduling algorithms such as the process priority number (In case the priority scheduling is to be used for the process), the pointers to appropriate scheduling queues

depending upon the current state of the process, the time when CPU was last allocated to the process, and so on.

- **I/O Status**: It includes information like I/O devices allocated to a process, pointers to the files opened by the process for I/O, the current position in the files, and so on.

4.3 PROCESS SCHEDULING

The main objective of multiprogramming is to keeps the jobs organized in such a manner that CPU has always one task to execute in the queue. This confirms that the CPU is utilized to the maximum level by reducing its idle time., This purpose can be very well achieved by keeping the CPU busy at all the times. However, when two or more processes compete for the CPU at the same time, then choice has to be made which process to allocate the CPU next. This procedure of determining the next process to be executed on the CPU is called **process scheduling** and the module of an operating system that makes this decision is called the **scheduler**.

Scheduling Queues

For scheduling purposes, there exist different queues in the system that are as follows:

- **Job Queue**: Each process enters the system for execution. As soon as they enter the execution phase, they get into a queue called job queue (Or **input queue**) on a mass storage device, such as hard disk.
- **Ready Queue**: From the job queue, the processes which are ready for the throughput are shifted into the main memory, where they are kept into a queue called ready queue. In other words, the ready queue contains all those processes that are waiting for the CPU.
- **Device Queue**: For each I/O device in the system, a separate queue called device queue is maintained. The process that needs to perform I/O during its execution is kept into the queue of that specific I/O device and waits there until it is served by the device.

  Generally, both the ready queue and device queue are maintained as linked lists that contain PCBs of the processes in the queue as their nodes. Each PCB includes a pointer to the PCB of the next process in the queue (Refer Figure 4.2). In addition, the header node of the queue contains pointers to the PCBs of the first and last process in the queue.

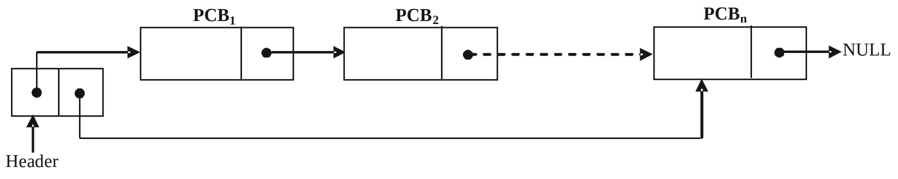


***Fig. 4.2** Ready Queue and Device Queue Maintained as Linked List*

Whenever, a process in the job queue is in the state of readiness and waiting to be executed, it is brought into the ready queue where it waits for the CPU allocation. Once CPU is allocated to it (that is, the process switches to the running state), the following transitions may happen.

(i) If the process needs to perform some I/O operation during its execution, it is removed from the ready queue and put into the appropriate device queue. After the process completes its I/O operation and is ready for the execution, it is switched from the device queue to ready queue.

(ii) If an interrupt occurs, the CPU can be forcibly taken away from the currently executing process and the process has to wait until the interrupt is handled. After that, the process is put back into the ready queue.

(iii) If the time slice (in case of time sharing systems) of the process has expired, the process is put back into the ready queue.

(iv) If the process creates a new process and has to wait until the child process terminates, the parent process is suspended. After the execution of child process, it is again put back into the ready queue.

(v) If the process has successfully completed its task, it is terminated. The PCB and all the resources allocated to the process are deallocated.

All these transitions can be represented with the help of a **queueing diagram** as shown in Figure 4.3.

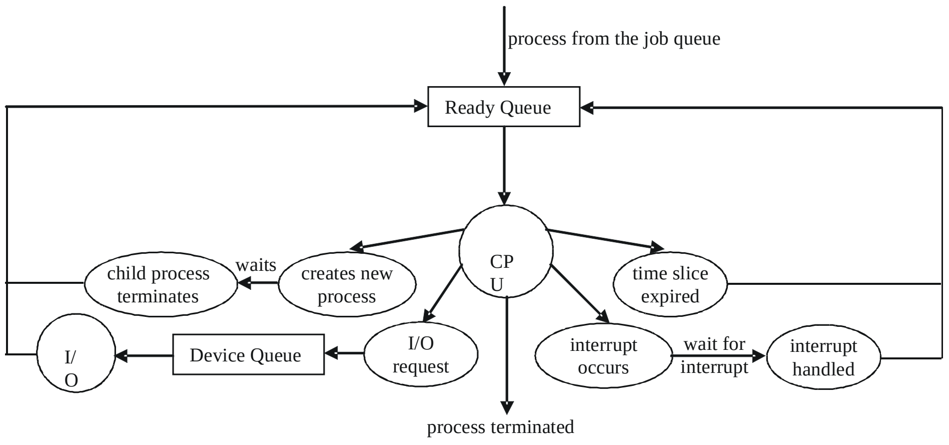


Fig. 4.3 *Queueing Diagram*

***Note**: In a single processor system, since there can be only one running process at a time, there is no need to maintain a queue for the running processes.*

Types of Schedulers

The following types of schedulers (Refer Figure 4.4) may coexist in a complex operating system.

- **Long-Term Scheduler**: It is also known as **job scheduler** or **admission scheduler** works with the job queue. It chooses the next process to be executed from the job queue and loads it into the main memory for execution. The long-term scheduler must select the processes in such a way that some of the processes are CPU-bound while others are I/O-bound. This is because if all the processes are CPU-bound, then for maximum number of times the other devices will remain idle and unused. On the contrary, if all the processes are I/O-bound, then CPU will remain idle most of the time. Thus, to make the best use of and optimize both, a balanced mix of CPU-bound and I/O-bound processes must be selected. The scheduler helps to control the degree of multiprogramming (that is, the number of processes in the ready queue) in order to achieve the maximum efficiency of the processor utilization at the desired level. For this, the long-term scheduler may admit new processes in the ready queue in case of poor processor utilization or may reduce the rate of admission of processes in the ready queue in case the processor utilization is high. By and large, the long-term scheduler is invoked only when a process exits from the system. Thus, the frequency of invocation of long-term scheduler entirely dependant on the system and workload and is much lower than other two types of schedulers.

- **Short-Term Scheduler**: It is also known as **CPU scheduler** or **process scheduler** selects a process from the memory from the ready queue geared up for the execution and allocates CPU to it. This scheduler needs repeated invocation to carry out the execution as compared to the long-term scheduler, the primary reason being a process executes for a short period under normal working conditions and then it may have to wait either for I/O or some other reason. At that time, CPU scheduler is required to select some other process and allocate CPU to it. Thus, the CPU scheduler must be fast in order to provide the least time gap between execution procedure.
- **Medium-Term Scheduler**: It is also known as **swapper** comes into play whenever a process is to be removed from the ready queue (or from the CPU in case it is being executed) thereby reducing the degree of multiprogramming. This process is stored at some other fragment on the hard disk and later brought into the memory to restart the execution from the point where it left off. This task of temporarily switching a process in and out of main memory is known as **swapping**. The medium-term scheduler selects a process among the partially executed or unexecuted swapped-out processes and swaps it in the main memory. The medium-term scheduler is usually invoked when there is some unoccupied space in the memory made by the termination of a process or the supply of ready process reduces below a specified limit.

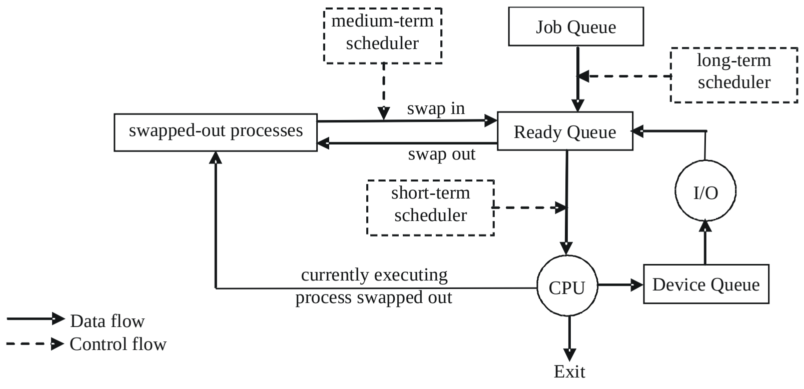


Fig. 4.4 *Types of Schedulers*

Context Switch

Transferring the control of CPU from one process to another demands for saving the context of the currently running process and loading the context of another ready process. This mechanism of saving and restoring the context of the ongoing process is known as **context switch**. The portion of the process control block including the process state, memory management information and CPU scheduling information together constitutes the **context** (also called **state information**) of a process. Context switch may occur due to varied reasons, some of which are as follows:

- The current process terminates and exits from the system.
- The time slice of the current process expires.
- The process has to wait for I/O or some other resource.
- Some higher priority process enters the system.
- The process relinquishes the CPU by invoking some system call.

Context switching involves a two step procedure, which are as follows:

(i) **Save Context**: In this step, the kernel saves the context of the ongoing executing process in its PCB of the process so that the proficient restoration of the context can easily take place at some point in the future. This is only achieved when the processing phase of the ongoing process is successfully over and now the execution of the suspended process can be recommenced.

(ii) **Restore Context**: In this step, the kernel loads the saved context of a different process that is to be executed next in the line. Note that if the process to be executed is newly created and CPU has not yet been allocated to it, there will be no saved context. In this case, the kernel loads the context of the new process. However, if the process to be executed was in waiting state due to I/O or some other reason, there will be saved context that can be restored in future.

One of the major detriments of using context switching is that it incurs a heavy cost to the system in terms of real time and CPU cycles. Owing to the fact that during context switching the system does not perform any productive work. Therefore, context switching should be generally refrained from as far as possible otherwise it would amount to reckless use of time. Figure 4.5 shows context switching between two processes P_1 and P_2.

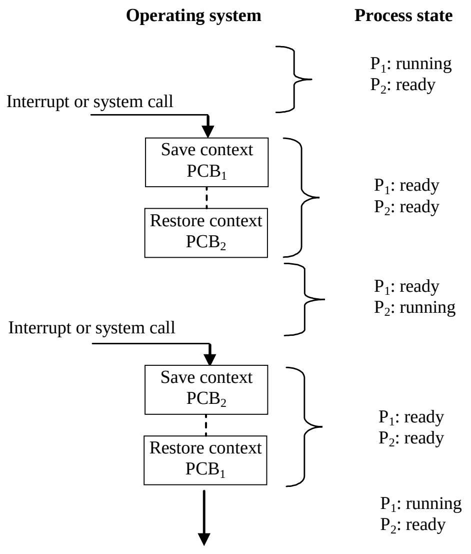


Fig. 4.5 *Context Switching between Processes P_1 and P_2*

Check Your Progress

1. What does a process control block contain?
2. Define the significance of process states.
3. How do we define the running to waiting process state transition?
4. What are the different types of queues in the system used for scheduling purposes?
5. What is meant by context switch in terms of processes?

4.4 OPERATIONS ON PROCESSES

There are innumerable operations that can be fulfilled on processes which encompasses creating, terminating, suspending, or resuming a process, and so on. To successfully execute these operations, the operating system provides run-time services (or system calls) for the process management. The user may invoke these system calls either directly by embedding the process supervisory calls in the user's program or indirectly by typing commands on the terminal which are translated by the system into system calls. In this section, we will discuss only about process creation and termination operations.

Process Creation

Whenever, an operating system is initialized on the system, a number of processes (system processes) are created automatically. Out of these, some involve user interaction (called **foreground processes**) while others are not related with any user but still perform some specific function (called **background processes**) to smooth completion of the booting task. In addition to system processes, new processes can be created afterward as well. Sometimes, a user process may need to create one or more processes during its execution. It can do the same by invoking the process creation system call (for example, `CreateProcess()` in Windows and `fork()` in UNIX which tells the an operating system to create a new process. This task of creating a new process on the request of some another process is called **process spawning**. The process that spawns a new process is called **parent** process whereas the spawned process is called the **child** process (or **sub** process). The newly created process can further create new processes thereby generating hierarchy of processes.

Whenever, a process creates a child process, there are chances of innumerable likelihoods that may arise depending on the operating system installed. Some of these likelihoods are as follows:

(i) Either the parent and child process may run concurrently (**asynchronous process creation**) or the parent process may wait until the child process completes its task and terminates (**synchronous process creation**).

(ii) The newly created process may be the duplicate of the parent process in which case it contains a copy of the address space of its parent. On the other hand, the child process may have a new program loaded into its address space.

(iii) The child process may be restricted to a subset of resources available to the parent process or the child process may obtain its resources directly from the operating system. In the former case, the resources being used by the parent process need to be divided or shared among its various child processes.

***Note**: Whenever, a process creates a new process, the PID of the child process is passed to the parent process*.

Process Termination

Depending upon the condition, a process may be terminated either normally or forcibly by some another process. Normal termination occurs when the process successfully executes the assigned task and invokes an appropriate system call (for example, `ExitProcess()` in Windows and `exit()` in UNIX) to tell the operating system that it is finished. Consequently, all the resources held by the process are deallocated, the process returns output data (if any) to its parent and finally, the process is removed from the memory by deleting its PCB from the process table.

***Note**: A process that no longer exists but still its PCB is not removed from the process table is known as a **zombie** process.*

Contrary to this, a process may cause abnormal termination of some another process. For this, the process invokes an appropriate system call (for example, `TerminateProcess()` in Windows and `kill()` in UNIX) that tells the operating system to kill some other process. Generally, the parent process can invoke such a system call to terminate its child process. As a general rule, this occurrence takes place because of the following reasons.

(i) Cascading termination in which the termination (whether normal or forced) of a process causes the termination of all its children. On some operating systems, a child process is not allowed to execute when its parent is being terminated. In such cases, the operating system initiates cascading termination.

(ii) The task that was being performed by the child process is not required.

(iii) The child process has used up the resources allocated to it more than that it was permitted.

4.5 INTERP ROCESS COMMUNICATION

The processes that coexist in the memory at the same time are called **concurrent processes**. The concurrent processes may either be independent or cooperating. The **independent** (also called **competitors**) processes, as the name implies, do not share any kind of information or data with each other. They just compete with each other for the resources like CPU, I/O devices, and so on that are required to accomplish their operations. The **cooperating** (also called **interacting**) processes, on the other hand, need to exchange data or information with each other. For this, they require some mechanism that allows them to communicate with each other. One such mechanism is Interprocess Communication (IPC)—a very useful facility provided by the operating system.

Two basic communication models for providing IPC are **shared memory systems** and **message passing systems**. In the former model, a fragment of memory is shared among the cooperating processes. Hence, if processes want to exchange data or information, it can do so by writing to and reading from this shared memory. However, in the latter model, the cooperating processes communicate by sending and receiving messages from each other. The communication using message passing is very time consuming as compared to shared memory. This is because the message passing system is implemented with the help of an operating system calls and thus, it requires a major involvement of kernel. On the other hand, in shared memory systems, system calls are used only to set up the shared memory area. Once the shared area is set up, no further kernel intervention is required.

Shared Memory Systems

In shared memory systems, the process that needs to communicate with other processes creates a shared memory segment in its independent address space. Other processes can communicate with this process by attaching its shared memory segment along with their address space. All the communicating processes can read or write data through this shared area. Note that these processes must be synchronized so that no two processes should be able to access the shared area simultaneously. Figure 4.6 shows a shared memory communication model.

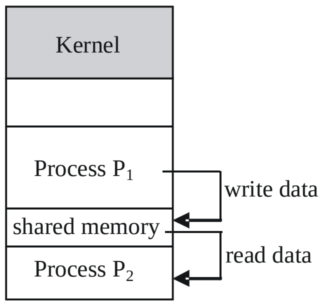


Fig. 4.6 *Shared Memory Communication Model*

To understand the concept of shared memory systems, consider a common example of cooperating processes known as **producer–consumer** problem. In this problem, there are two processes, one is producer that produces the items and other is consumer that consumes the items produced by the producer. These two processes need to run simultaneously thereby require communication with each other. One possible solution to this problem can be provided through shared memory. Both the producer and consumer processes are made to share a common **buffer** between them. The producer fills the buffer by placing the produced items in it and the consumer vacates the buffer by consuming these items.

The buffer shared between producer and consumer may be bounded or unbounded. In bounded buffer, the size of buffer is fixed. Therefore, the producer has to wait in case the buffer is full and consumer has to wait in case the buffer is empty. On the other hand, in unbounded buffer, there is no limit on the buffer size. Thus, only consumer has to wait in case there is no item to be consumed. However, producer need not wait and it may continuously produce items.

***Note**: In case of bounded buffer, the producer–consumer problem is also known as **bounded-buffer** problem.*

To implement the bounded buffer problem using shared memory, consider the shared buffer consists of N slots with each capable of storing an item. Further, assume that the buffer is implemented as a circular array having two pointers in and out. The pointer in points to the next free slot in the buffer and the pointer out points to the slot containing the next item to be consumed. Initially, both in and out are set to zero. The following code written in ‘C’ language illustrates the implementation of shared area.

```
#define size 100;     /* N=100 */
int buffer[size];
int in=0;
int out=0;
```

To implement the producer process, a local variable `item_produced` is used that stores the newly produced item. The producer produces an item, places it in the buffer at the position denoted by in, and updates the value of in. It continues to do so as long as buffer is not full. Once the buffer gets full, that is, when `(in + 1) % size == out`, it goes to the waiting state and remains in that state until some slot becomes free in the buffer (that is, until consumer removes some item from the buffer). The following code illustrates the implementation of the producer process:

```
int item_produced;
while(1)
{
    item_produced = produce_item();
          /* calling procedure to produce an item */
    while(((in + 1) % size) != out)
    {     /* check if buffer is not full */
       buffer[in] = item_produced;
          /* put item into the buffer */
       in = (in + 1) % size;
          /* update in to point to next free slot */
    }
}
```

Likewise, to put into the effect, the consumer process, a local variable `item_consumed` is used that stores the item to be consumed. The consumer removes an item from the position denoted by out in the buffer, updates the value of out, and consumes that item. It continues to do so as long as the buffer is not empty. Once the buffer gets empty of the contents, that is, when `in == out`, it goes to the waiting state and remains in that state until producer places some item in the buffer. The following code illustrates the implementation of the consumer process.

```
int  item_consumed;
while(1)
{
         while(in != out)      /* check if buffer is not
empty */
         {
          item_consumed = buffer[out];
                    /* remove item from the buffer */
          out = (out + 1) % size;  /*  update  out  to
point */
```

```
                              /* to the next item to be consumed
*/
        }
        consume_item(item_consumed);
             /* calling procedure to consume an item */
}
```

***Note**: For the sake of simplicity, we have assumed that the item in the buffer is of type integer, and the implementation of procedures for producing or consuming items is not shown here.*

This solution to bounded-buffer problem permits to have at most size-1 items in the buffer at the same time. In order to have size items in the buffer at the same time, we will need to develop a different solution. In addition, this solution does not address to how to implement the synchronization between producer and consumer.

Message Passing Systems

As mentioned earlier, the message passing system, uses two system calls, `send()` and `receive()`. The sender process (say, P_1) sends the message to the operating system by invoking the `send()` system call. The operating system stores this message in the buffer area until the `receive()` system call is invoked by the receiver process (say, P_2). After that the operating system delivers this message to P_2. In case there is no message available for P_2 when it invokes the `receive()` system call, the operating system blocks it until some message arrives for it. On the other hand, if a number of messages arrive for P_2, the operating system puts them in a queue and delivers them in FIFO order upon the invocation of `receive()` call (one for each process) by P_2. Figure 4.7 shows the message passing communication model.

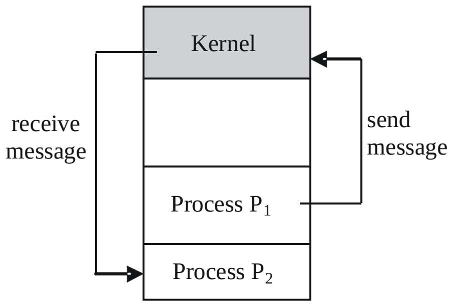


***Fig. 4.7** Message Passing Communication Model*

In message passing, it is not mandatory for the communicating processes to be made on the self same computer rather they can made on different computers connected via network (a distributed environment). Therefore, a communication relationship is set up between the two processes, whenever they want to communicate. At the physical level, the communication association may be brought about via shared variables or bus or the network, and so on. However, at the logical level, some features related with the implementation of communication link arise, which are discussed here.

Types of Communication

Processes may communicate with each other either directly or indirectly.

In **direct communication**, processes address each other by their PID assigned to them by the operating system. For example, if a process P_1 wants to send a message to process P_2, then the system calls `send()` and `receive()` will be defined as follows:

- `send(PID2, message)`
- `receive(PID1, message)`

Since, both sender and receiver process need to know each other's PID, this type of communication is known as symmetric direct communication. However, asymmetry in addressing can be represented by making only the sender process to address the receiver process by its PID but the receiver process need not know the PID of the sender process. In case of asymmetric direct communication, the calls `send()` and `receive()` will be defined as follows:

- `send(PID2, message)`
- `receive(id, message)`

Now, when an operating system delivers a message to process P_2 upon the invocation of a `receive()` call by it, the parameter id is replaced with the PID of the sender process.

In **indirect communication**, messages are sent and received via **mailbox** (also known as **port**)—a repository of interprocess messages. A mailbox, as the name implies, is just like a post box into which messages sent by the processes can be stored and removed by other processes. The different characteristics of a mailbox are as follows:

- Each mailbox has a unique ID assigned to it and the communication between the processes takes place through a number of mailboxes.
- The possessor of the mailbox is the one that has created it and only this process has the authority to receive messages from it. Nevertheless, other processes can only send messages to it. In other words, there can be multiple senders but a single recipient for a mailbox.
- The process that knows the ID of a mailbox can send messages to it.
- Besides a user process, the operating system may also own a mailbox. In this case, the operating system may allow the processes to create or delete a mailbox, send and receive messages via mailbox. The process that creates the mailbox becomes the owner of that mailbox and may receive messages through this mailbox. However, with time, other processes can also be made to receive messages through this mailbox by passing ownership to them.

The system calls to send a message to a mailbox (say, X) and receive a message from a mailbox will be defined as follows:

(i) `send(X, message)`

(ii) `receive(X, message)`

As stated earlier, it is compulsory that a communication link must exist between processes before the onset of the communication. The communication link exhibits different properties in direct and indirect communication, which are discussed in Table 4.1.

Table 4.1 *Comparison of Direct and Indirect Communication*

Direct Communication	Indirect Communication
• There exists only one link between each pair of communicating processes.	• There may be multiple links between each pair of communicating processes, where each link corresponds to exactly one mailbox.
• A link is associated with just two processes.	• A link may be associated with more than two processes.
• The link is established automatically between the communicating processes, provided the sender process knows the PID of the receiver process.	• The communication link can be established between two processes only if both the communicating processes share a mailbox with each other.

Message Synchronization

Messages can be sent or received either **synchronously** or **asynchronously**, also called **blocking** or **non-blocking**, respectively. Various design options for implementing `send()` and `receive()` calls are as follows:

- **Blocking Send**: If a process (say, P_1) invokes `send()` call to send a message to another process (say, P_2) or to a mailbox, the operating system blocks P_1 until the message is received by P_2 or by the mailbox.
- **Blocking Receive**: If there is no message available for P_2 when it invokes the `receive()` system call, the operating system blocks it until some message arrives for it.
- **Non-Blocking Send**: P_1 sends the message and continues to perform its operation without waiting for the message delivery by P_2 or by mailbox.
- **Non-Blocking Receive**: When P_2 invokes a `receive()` call, it either gets a valid message if some message is available for it or NULL if there is no message available for it.

Buffering

As discussed earlier, the messages sent by a process are temporarily set aside in a temporary queue (also called **buffer**) by the operating system before delivering them to the recipient. This buffer can be implemented in a variety of ways, which are as follows:

- **No Buffering**: The capacity of buffer is zero, that is, no messages may wait in the queue. This implies the sender process has to wait until the message is received by the receiver process.

- **Bounded Buffer**: The capacity of the buffer is fixed, say m, that is, at most m processes may wait in the queue at a time. When there are less than m messages waiting in the queue and a new message arrives, it is added to the queue. The sender process need not wait in the queue and it can resume its operation. However, if the queue is full, the sender process is blocked until some space is made available in the queue.
- **Unbounded Buffer**: The buffer has an unlimited capacity, that is, an infinite number of messages can be stored in the queue for execution. In this case, the sender process never gets blocked.

Check Your Progress

6. Define zombie process.
7. What are the two important communication models used for providing IPC?
8. State the difference between direct and indirect communication of processes.
9. Differentiate between the bounded and unbounded buffer.

4.6 ANSWERS TO CHECK YOUR PROGRESS

1. A process comprises of not only the program code (known as text section) but also a set of global variables (known as data section) and the Process Control Block (PCB). The process control block of a process contains some additional information about the process like a program counter to specify the current activity in progress, the contents of CPU's registers, a process stack to store temporary data like function parameters, local variables used in a function, return addresses, and so on.
2. Each process in the operating system tagged with a 'State' variable—an integer value that helps the operating system to settle conclusively what should be done with the active process and also gives the evidence of the nature of the current activity in a process. A process may be in one of the following states depending on the current activity of the process.
3. Running $\rightarrow$ Waiting: This transition may take place if the currently running process.

   (i) Needs to perform some I/O operation.

   (ii) Has to wait for a message or some action from another process.

   (iii) Requests for some other resource.

   In this case, the CPU gets relieved by the process and can be allocated to some another ready process.

4. Scheduling Queues

   For scheduling purposes, there exist different queues in the system that are as follows:

   - Job Queue: Each process enters the system for execution. As soon as they enter the execution phase, they get into a queue called job queue (Or input queue) on a mass storage device, such as hard disk.
   - Ready Queue: From the job queue, the processes which are ready for the throughput are shifted into the main memory, where they are kept into a queue called ready queue. In other words, the ready queue contains all those processes that are waiting for the CPU.
   - Device Queue: For each I/O device in the system, a separate queue called device queue is maintained. The process that needs to perform I/O during its execution is kept into the queue of that specific I/O device and waits there until it is served by the device.
5. Transferring the control of CPU from one process to another demands for saving the context of the currently running process and loading the context of another ready process. This mechanism of saving and restoring the context of the ongoing process is known as context switch. The portion of the process control block including the process state, memory management information and CPU scheduling information together constitutes the context (also called state information) of a process.
6. A process that no longer exists but still its PCB is not removed from the process table is known as a **zombie** process.
7. Two basic communication models for providing IPC are shared memory systems and message passing systems.
8. In direct communication, processes address each other by their PID assigned to them by the operating system. For example, if a process P_1 wants to send a message to process P_2, then the system calls send() and receive() will be defined as follows:

   - send(PID2, message)
   - receive(PID1, message)

   In indirect communication, messages are sent and received via mailbox (also known as port)—a repository of interprocess messages. A mailbox, as the name implies, is just like a post box into which messages sent by the processes can be stored and removed by other processes.
9. Bounded Buffer: The capacity of the buffer is fixed, say m, that is, at most m processes may wait in the queue at a time. When there are less than m messages waiting in the queue and a new message arrives, it is added to the queue. The sender process need not wait in the queue and it can resume its operation. However, if the queue is full, the sender process is blocked until some space is made available in the queue.

Unbounded Buffer: The buffer has an unlimited capacity, that is, an infinite number of messages can be stored in the queue for execution. In this case, the sender process never gets blocked.

4.7 SUMMARY

- A process is a program under execution or we can say an executing set of machine instructions. It can be either a system process executing the system's code or a user process executing the user's code.
- A process comprises not only the program code (known as text section) but also a set of global variables (known as data section) and the Process Control Block (PCB).
- The set of instructions, data and stack together form the address space of a process.
- The processes that involve more computation than I/O operations thereby demanding more use of CPU than I/O devices during their life time are called CPU-bound or computer-bound processes.
- The processes that involve a lot of I/O operations as compared to computation during their life time are called I/O-bound processes.
- Each process is labelled with a 'state' variable—an integer value that helps the operating system to decide what to do with the process. It indicates the nature of the current activity in a process.
- The various possible states for a process are new, ready, running, waiting and terminated.
- The change in state of a process is known as state transition of a process and is caused by the occurrence of some event in the system.
- To keep track of all the processes in the system, the operating system maintains a table called process table that includes an entry for each process. This entry is called Process Control Block (PCB).
- A process control block stores descriptive information pertaining to a process such as, its state, program counter, memory management information, information about its scheduling, allocated resources, accounting information, and so on, that is required to control the process.
- The procedure of determining the next process to be executed on the CPU is called process scheduling and the module of operating system that makes this decision is called the scheduler.
- As the processes enter the system for execution, they are kept into a queue called job queue (or input queue).

- From the job queue, the processes which are ready for the execution are brought into the main memory. In the main memory, these processes are kept into a queue called ready queue.
- For each I/O device in the system, a separate queue called device queue is maintained. The process that needs to perform I/O during its execution is kept into the queue of that specific I/O device and waits there until it is served by the device.
- The long-term scheduler also known as job scheduler or admission scheduler selects the next process to be executed from the job queue and loads it into the main memory for execution.
- The short-term scheduler also known as CPU scheduler or process scheduler selects a process from the ready queue and allocates CPU to it.
- The medium-term scheduler also known as swapper selects a process among the partially executed or unexecuted swapped-out processes and swaps it in the main memory.
- Transferring the control of CPU from one process to another demands for saving the context of the currently running process and loading the context of another ready process. This task of saving and restoring the context is known as context switch.
- The portion of the process control block including the process state, memory management information and CPU scheduling information together constitute the context (also called state information) of a process.
- A user process may create one or more processes during its execution by invoking the process creation system call.
- The task of creating a new process on the request of some another process is called process spawning. The process that spawns a new process is called parent process whereas the spawned process is called the child process.
- When a process is terminated, all the resources held by the process are deallocated, the process returns output data (if any) to its parent and finally, the process is removed from the memory by deleting its PCB from the process table.
- A process that no longer exists but still its PCB is not removed from the process table is known as a zombie process.
- The processes that coexist in the memory at some time are called concurrent processes. The concurrent processes may either be independent or cooperating.
- The independent (also called competitors) processes, as the name implies, do not share any kind of information or data with each other.

- The cooperating (also called interacting) processes, on the other hand, need to exchange data or information with each other.
- The cooperating processes require some mechanism to communicate with each other. One such mechanism is Interprocess Communication (IPC)—a facility provided by the operating system.
- Two basic communication models for providing IPC are shared memory systems and message passing systems.
- In shared memory systems, a part of memory is shared among the cooperating processes. The processes that need to exchange data or information can do so by writing to and reading from this shared memory.
- In message passing systems, cooperating processes communicate by sending and receiving messages from each other. The system calls, send () and receive () are used to send and receive messages, respectively.
- Processes may communicate with each other directly or indirectly. In the direct communication, processes address each other by their PID assigned to them by the operating system. In indirect communication, messages are sent and received via mailbox—a repository of interprocess messages.
- A mailbox, as the name implies, is just like a post box into which messages sent by the processes can be stored and removed by other processes.
- Messages can be sent or received either synchronously or asynchronously, also called blocking or non-blocking, respectively.

4.8 KEY WORDS

- **New state:** A process is said to be in a 'new' state if it is being created for the first time.
- **Ready state:** A process is said to be in a 'ready' state if it is ready for the execution and waiting for the CPU to be allocated to it.
- **Running state:** A process is said to be in a 'running' state if CPU has been allocated to it and it is being executed.
- **Waiting state:** A process is said to be in a 'waiting' state if it has been blocked by some event.
- **Bounded-buffer problem:** In case of bounded buffer, the producer-consumer problem is also known as bounded- buffer problem.

4.9 SELF-ASSESSMENT QUESTIONS AND EXERCISES

NOTES

Short-Answer Questions

1. Define PCB.
2. State the different states of transition of the process.
3. Distinguish between CPU-bound and I/O-bound processes.
4. List three important fields that are stored in a process control block.
5. What is the significance of process identifier?
6. Define the various types of schedulers in a complex operating system.
7. What are the reasons due to which context switch occurs?
8. What is context switching?
9. Define synchronous process creation.
10. What do you understand by interprocess communication?
11. State the definitions of message synchronization and buffering.

Long-Answer Questions

1. Discuss the various states of a process with the help of a diagram.
2. Describe the events under which state transitions between ready, running and waiting take place.
3. Distinguish among long-term scheduler, short-term scheduler and medium-term scheduler.
4. Describe the different models used for interprocess communication. Which one is better?
5. Elaborate the concept of processes, types of schedulers and operations carried by the processes.
6. Illustrate on the significance of scheduling queues and its transition- based classifications of processes.

4.10 FURTHER READINGS

Silberschatz, Abraham, Peter B. Galvin and Greg Gagne. 2008. *Operating System Concepts*, 8th Edition. New Jersey: John Wiley & Sons.

Tanenbaum, Andrew S. 2006. *Operating Systems Design and Implementation*, 3rd Edition. New Jersey: Prentice Hall.

Tanenbaum, Andrew S. 2001. *Modern Operating Systems*. New Jersey: Prentice Hall.

Deitel, Harvey M. 1984. *An Introduction to Operating Systems*. Boston (US): Addison-Wesley.

Stallings, William. 1995. *Operating Systems,* 2nd Edition. New Jersey: Prentice Hall.

Milenkovic, Milan. 1992. *Operating Systems: Concepts and Design*. New York: McGraw Hill Higher Education.

Mano, M. Morris. 1993. *Computer System Architecture*. New Jersey: Prentice Hall Inc.

UNIT 5 PROCESS SCHEDULING

NOTES

Structure

5.0 INTRODUCTION

Process scheduling (or CPU scheduling) is the procedure employed for deciding as to which of the ready processes, the CPU should be allocated. CPU scheduling plays a pivotal role in the basic framework of an operating system owing to the fact that CPU is the central unit of any computer system. No computer can operate properly in the absence of CPU. The algorithm used by the scheduler to carry out the selection of a process for effective execution is known as scheduling algorithm. There are umpteenth number of tasks at hand for CPU scheduling.

Each scheduling algorithm is responsible for regulating the resource utilization, overall system performance and quality of service provided to the user. Therefore, one has to reason out a number of criteria while fixing an algorithm for an achieving desired prerequisites on a particular system.

In this unit, you will study the concept of process scheduling, scheduling criteria and algorithms, multiple processor scheduling.

5.1 OBJECTIVES

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

- Understand the basic concepts of scheduling
- Discuss the criteria for scheduling
- Explain various scheduling algorithms
- Discuss scheduling for multiprocessor systems

5.2 PROCESS SCHEDULING CONCEPTS

Before we start discussing about the scheduling criteria and scheduling algorithms comprehensively, we will first take into account some comparatively important concepts of scheduling which are mentioned underneath.

Process Behaviour

The reaction and response of a process while it is in the execution phase highly influences CPU scheduling.. Virtually, there is a continuous interchange of processes between CPU (for processing) and I/O devices (for performing I/O) during the period of execution. The time period elapsed in processing before performing the next I/O operation is known as **CPU burst** and the time period elapsed in performing I/O before the next CPU burst is known as **I/O burst**. Generally, the process execution starts with a CPU burst, followed by an I/O burst, then again by a CPU burst and so on until the process terminates. Thus, we can say the process execution includes alternate cycles of CPU and I/O burst. Figure 5.1 shows the sequence of CPU and I/O bursts upon the execution of the following code segment written in C language.

```
i=1;                          /* CPU burst
*/
sum=0;
scanf(“%d”, &num);       /* I/O burst */
while (i <= 10 )              /* CPU burst
*/
{
     sum += num * i;
     i = i + 1;
```

Fig 5.1 *Alternate Cycles of CPU and I/O Bursts*

The length of CPU burst and the I/O burst varies from process to process depending on whether the process is CPU-bound or I/O-bound. If the process is CPU-bound, it will have longer CPU bursts as compared to I/O bursts and vice versa in case the process is I/O-bound. From the scheduling perspective, only the length of CPU burst is taken into consideration and not the length of I/O burst.

When to Schedule

An important facet of scheduling is to determine under what circumstances the scheduler should be initiated to make scheduling decisions. The following circumstances may require the scheduler to make scheduling decisions.

- When a process switches from running to waiting state. Such a situation crops up, in case, the process has to wait for I/O or the termination of its child process or some another reason. In such situations, the scheduler has to select some ready process for execution.

- When a process switches from running to ready state due to occurrence of an interrupt. In such situation, the scheduler may decide to run a process from the ready queue. If the interrupt was caused by some I/O device that has now completed its task, the scheduler may choose the process that was blocked waiting for the I/O.
- When a process switches from waiting state to ready state, for example, in case, the process has completed its I/O operation. In such situation, the scheduler may select either the process that has now come into the ready state or the current process may be continued.
- When a process terminates and exits the system. In this case, the scheduler has to select a process for execution from the set of ready processes.

Dispatcher

The main aim of short-term scheduler is to opt for a process to be performed next on the CPU but it cannot allocate CPU to the selected process. The basic resposibility of setting up the execution of the selected process on the CPU is performed by some other module of the operating system, called **dispatcher**. The dispatcher involves the following three steps to perform this function.

(i) The dispatcher performs context switching that is switching the CPU to another process. The kernel saves the context of currently running process and restores the saved state of the process selected by the CPU scheduler. In case, the process selected by the short-term scheduler is new, the kernel loads its context.

(ii) The system switches from the kernel mode to user mode as a user process is to be executed.

(iii) The execution of the user process selected by the CPU scheduler is started by transferring the control either to the instruction that was supposed to be executed at the time the process was interrupted or to the first instruction if the process is going to be executed for the first time after its creation.

5.3 SCHEDULING CRITERIA

The scheduler must consider the following parameters and optimization criterias in order to maximize the performance of the system. Following mentioned are some of the criterias that help to judge the performance.

- **Fairness**: Generally, CPU is busy performing varied tasks.Fairness is defined as the degree to which each process is getting an equal chance to execute. The scheduler must ensure that each process should get a fair share of CPU time. However, it may treat different categories of processes (batch, real-time, or interactive) in a different manner.

NOTES

- **CPU Utilization**: It is defined as the percentage of time the CPU is otherwise engaged in carrying out processes. To ensure better utilization, CPU must be kept as busy as possible, that is, there must be some process running everytime.
- **Balanced Utilization**: It is defined as the percentage of time all the system resources are busy. It considers not only the CPU utilization but the utilization of I/O devices, memory, and other resources also. To get more work done by the system, the CPU and I/O devices must be kept running simultaneously. For this, it is desirable to load a mixture of CPU-bound and I/O-bound processes in the memory.
- **Throughput**: It is defined as the total number of processes that a system can execute per unit of time. By and large, it depends on the average length of the processes to be executed. For the systems running long processes, throughput will be less as compared to the systems running short processes.
- **Turnaround Time**: It is defined as the relative amount of time that has rolled by from the time of initiation to the time of termination of a process. To put it differently, it is the difference between the time a process enters the system and the time it exits from the system. It includes all the time the process has spent waiting to enter into ready queue, within ready queue to get CPU, running on CPU, and in I/O queues. It is inversely proportional to throughput, that is, the more is the turnaround time, the less will be the throughput.
- **Waiting Time**: It is defined as the time used up by the process while waiting in the ready queue. However, it does not take into account the execution time or time consumed for I/O. In practice, waiting time is more accurately measured as compared to turnaround time.
- **Response Time**: It is defined as intervening time between the time the user initiates a request and the system starts responding to this request. For interactive systems, it is one of the best metric syetems employed to gauge the performance because in such systems, only the speed with which the system responds to user's request matters and not the time it takes to output the response.

The basic purpose of a CPU scheduling algorithm is that it should manage to make the best of fairness, CPU utilization, balanced utilization and throughput, and minimize turnaround, waiting and response time. Pratically speaking, no scheduling algorithm optimizes all the scheduling criteria. Thus the performance of an algorithm is evaluated on the basis of certain assumptions and average measures. For example, an algorithm that minimizes the average waiting time is considered as a good algorithm because this improves the overall efficiency of the program. However, in case of response time, minimizing the average is not a good criterion rather the variance in the response time of the processes should be minimized.

This is because it is not desirable to have a process with long response time as compared to other processes.

5.4 SCHEDULING ALGORITHMS

The performance of any system depends almost as much on proper software setup as it does on your computer's hardware peripherals. To maximize the functioning, a broad range of algorithms are used for the CPU scheduling. These scheduling algorithms fall into two categories, namely, *non-preemptive* and *preemptive*.

- **Non-Preemptive Scheduling Algorithms**: Once the CPU is allocated to a process, it cannot be taken back until the process voluntarily releases it (in case the process has to wait for I/O or some other event) or the process terminates. In other words, we can say the decision to schedule a process is made only when the currently running process either switches to the waiting state or terminates. In both cases, the CPU executes some other process from the set of ready processes.
- **Preemptive Scheduling Algorithms**: The CPU can be forcibly taken back from the currently running process before its completion and allocated to some other process. The preempted process is put back in the ready queue and resumes its execution when it is scheduled again. Thus, a process may be scheduled many times before its completion. In preemptive scheduling, the decision to schedule another process is made whenever an interrupt occurs causing the currently running process to switch to ready state or a process having higher priority than the currently running process is ready to execute.

Note: *A non-preemptive scheduling algorithm is also known as a* ***cooperative*** *or* ***voluntary*** *scheduling algorithm.*

First-Come First-Served Scheduling

First-Come First-Serve (FCFS) is one of the simplest scheduling algorithms. As the name implies, the processes are executed in the order of their arrival in the ready queue, which means the process that enters the ready queue first gets the CPU first. To put it differently, it means the process that comes in first is managed first and the processes that come in subsequently wait in the queue until the first is completed. FCFS is a non-preemptive scheduling algorithm. Therefore, once the CPU is allocated to the process, it retains the control of CPU until it blocks or terminates.

To implement FCFS scheduling, the implementation of ready queue is managed as a FIFO (First-In First-Out) queue. When the first process enters the ready queue, it immediately gets the CPU and starts executing. Meanwhile, other processes enter the system and are added to the end of queue by inserting their

PCBs in the queue. When the currently running process completes or blocks, the CPU is allocated to the process at the front of the queue and its PCB is removed from the queue. In case, a currently running process was blocked and later it comes to the ready state, its PCB is linked to the end of queue.

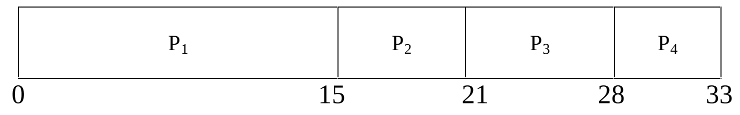
NOTES

Example 5.1: Consider four processes P_1, P_2, P_3, and P_4 with their arrival times and required CPU burst (in milliseconds) as shown in the following table.

Process	P_1	P_2	P_3	P_4
Arrival time	0	2	3	5
CPU burst (ms)	15	6	7	5

How will these processes be scheduled according to FCFS scheduling algorithm? Compute the average waiting time and average turnaround time.

Solution: The processes will be scheduled as depicted in the following Gantt chart.



Initially, P_1 enters the ready queue at $t = 0$ and CPU is allocated to it. While P_1 is executing, P_2, P_3 and P_4 enter the ready queue at $t = 2$, $t = 3$, and $t = 5$, respectively. When P_1 completes, CPU is allocated to P_2 as it has entered before P_3 and P_4. When P_2 completes, P_3 gets the CPU after which P_4 gets the CPU.

Waiting time for $P_1 = 0$ ms as P_1 starts immediately

Waiting time for $P_2 = (15 – 2) = 13$ ms as P_2 enters at $t = 2$ and starts at $t = 15$

Waiting time for $P_3 = (21 – 3) = 18$ ms as P_3 enters at $t = 3$ and starts at $t = 21$

Waiting time for $P_4 = (28 – 5) = 23$ ms as P_4 enters at $t = 5$ and starts at $t = 28$

Average waiting time = (0 + 13 + 18 + 23)/4 = 13.5 ms

Turnaround time for $P_1 = (15 – 0) = 15$ ms as P_1 enters at $t = 0$ and exits at $t = 15$

Turnaround time for $P_2 = (21 – 2) = 19$ ms as P_2 enters at $t = 2$ and exits at $t = 21$

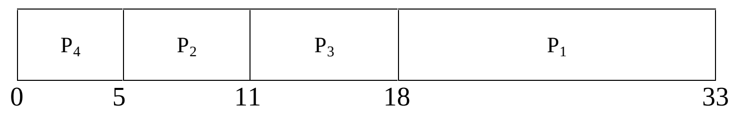
Turnaround time for $P_3 = (28 – 3) = 25$ ms as P_3 enters at $t = 3$ and exits at $t = 28$

Turnaround time for $P_4 = (33 – 5) = 28$ ms as P_4 enters at $t = 5$ and exits at $t = 33$

Average turnaround time = (15 + 19 + 25 + 28)/4 = 21.75 ms

NOTES

The performance of FCFS scheduling algorithm largely depends on the order of arrival of processes in the ready queue. That is, whether the processes having long CPU burst enter before those having short CPU burst or vice versa. To illustrate this, assume that the processes (shown in Example 2.1) enter the ready queue in the order P_4, P_2, P_3 and P_1. Now, the processes will be scheduled as shown in the following Gantt chart.



Average waiting time = (0 + (5 – 2) + (11 – 3) + (18 – 5))/4 = 6 ms

Average turnaround time = ((5 – 0) + (11 – 2) + (18 – 3) + (33 – 5))/4 = 14.25 ms

It is clear that the average waiting and turnaround time may be cut down marginally if the processes having shorter CPU burst execute before those having longer CPU burst.

Advantages

- It is easier to understand and implement as processes are to be added at the end and removed from the front of queue. This process help to manipulate data effectively as the processes cannot be accessed from the middle of queue which means it always proceeds in sequentially.
- It is well suited for batch systems where the longer time periods for each process are often acceptable.

Disadvantages

- The average waiting time is not minimal, that is, it varies from process to process. Therefore, this scheduling algorithm is never recommended where performance is a major issue.
- It reduces the CPU and I/O devices utilization under some circumstances. For example, assume that there is one long CPU-bound process and many short I/O-bound processes in the ready queue. Now, it may happen that while the CPU-bound process is executing, the I/O-bound processes complete their I/O and come to the ready queue for execution. There they have to wait for the CPU-bound process to release the CPU and the I/O devices also remain idle during this time. When the CPU-bound process needs to perform I/O, it comes to the device queue and the CPU is allocated to I/O-bound processes. As the I/O-bound processes require a little CPU burst, they execute quickly and come back to the device queue thereby leaving the CPU idle. Then the CPU-bound process enters the ready queue and is allocated the CPU which again makes the I/O processes waiting in ready queue at some point of time. This process takes place repeatedly until the CPU-bound process is done which results in low CPU and I/O devices utilization.

- It is not suitable for time sharing systems where it is desirable each process should get the equal amount of CPU time.

Shortest Job First Scheduling

NOTES

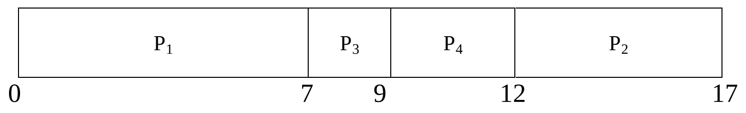
Shortest Job First (SJF), also known as **Shortest Process Next** (**SPN**) or **Shortest Request Next** (**SRN**), is a non-preemptive scheduling algorithm that schedules the processes according to the length of CPU burst they require. At any point of time, among all the ready processes waiting in the queue to be executed, the one having the shortest CPU burst is scheduled first. In other words, the process with the shortest execution is selected for the execution. Thus, a longer process has to wait until all the processes shorter than it have been executed. In case two processes have the same CPU burst, they are scheduled in the FCFS order.

Example 5.2: Consider four processes P_1, P_2, P_3 and P_4 with their arrival times and required CPU burst (in milliseconds) as shown in the following table.

Process	P_1	P_2	P_3	P_4
Arrival time	0	1	3	4
CPU burst (ms)	7	5	2	3

How will these processes be scheduled according to SJF scheduling algorithm? Compute the average waiting time and average turnaround time.

Solution: The processes will be scheduled as depicted in the following.



Initially, P_1 enters the ready queue at $t = 0$ and gets the CPU as there are no other processes in the queue. While it is executing, P_2, P_3 and P_4 enter the queue at $t = 1$, $t = 3$ and $t = 4$, respectively. When CPU becomes free, that is, at $t = 7$, it is allocated to P_3 because it is having the shortest CPU burst among the three processes. When P_3 completes, CPU is allocated first to P_4 and then to P_2.

Waiting time for P_1 = 0 ms as P_1 starts immediately

Waiting time for P_2 = (12 – 1) = 11 ms as P_2 enters at $t = 1$ and starts at $t = 12$

Waiting time for P_3 = (7 – 3) = 4 ms as P_3 enters at $t = 3$ and starts at $t = 7$

Waiting time for P_4 = (9 – 4) = 5 ms as P_4 enters at $t = 4$ and starts at $t = 9$

Average waiting time = (0 + 11 + 4 + 5)/4 = 5 ms

Turnaround time for P_1 = (7 – 0) = 7 ms as P_1 enters at $t = 0$ and exits at $t = 7$

Turnaround time for P_2 = (17 – 1) = 16 ms as P_2 enters at $t = 1$ and exits at $t = 17$

Turnaround time for P_3 = (9 – 3) = 6 ms as P_3 enters at t = 3 and exits at t = 9

Turnaround time for P_4 = (12 – 4) = 8 ms as P_4 enters at t = 4 and exits at t = 12

Average turnaround time = (7 + 16 + 6 + 8)/4 = 9.25 ms

Advantages

- It eliminates the variance in waiting and turnaround times., It is incontrovertibly superlative with respect to average waiting time if all processes are available at the same time due to the fact that short processes are given the highest priority in comparison to the long ones which results in immense decrement of the waiting time for short processes. Accordingly, the waiting time for long processes is increased. However, the diminuition in waiting time is relatively more than the increment in waiting time and thus, the average waiting time decreases.

Disadvantages

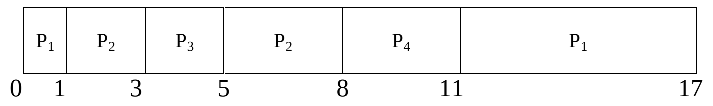
- It is quite intricate to implement because it needs to be familiarized with the exact length of CPU burst of processes in advance. Realistically, it is quite impractical to have the prior knowledge of required processing time of processes. Many systems expect users to provide estimates of CPU burst of processes where there are maximum feasibilities of errorneous ones.
- It favours the processes having short CPU burst because as long as the short processes continue to enter the ready queue, the long processes are not allowed to get the CPU. This results in **starvation** of long processes.

Shortest Remaining Time Next Scheduling

The Shortest Remaining Time Next (SRTN) also known as **Shortest Time To Go** (**STG**), is a preemptive version of the SJF scheduling algorithm. It takes into account the length of remaining CPU burst of the processes rather than the whole length in order to schedule them. Here also the scheduler always chooses the process for execution that has the shortest remaining processing time. While a process is being executed, the CPU can be taken back from it and assigned to some newly arrived process if the CPU burst of the new process is shorter than its remaining CPU burst. Notice that if at any point of time, the remaining CPU burst of two processes becomes equal; they are scheduled in the FCFS order.

Example 5.3: Consider the same set of processes, their arrival times and CPU burst as shown in Example 5.2. How will these processes be scheduled according to SRTN scheduling algorithm? Compute the average waiting time and average turnaround time.

Solution: The processes will be scheduled as depicted in the following Gantt chart.



NOTES

Initially, P_1 enters the ready queue at $t = 0$ and gets the CPU as there are no other processes in the queue. While it is executing, at time $t = 1$, P_2 with CPU burst of 5 ms enters the queue. At that time the remaining CPU burst of P_1 is 6 ms which is greater than that of P_2. Therefore, the CPU is taken back from P_1 and allocated to P_2. During execution of P_2, P_3 enters at $t = 3$ with a CPU burst of 2 ms. Again CPU is switched from P_2 to P_3 as the remaining CPU burst of P_2 at $t = 3$ is 3 ms which is greater than that of P_3. However, when at time $t = 4$, P_4 with CPU burst of 3 ms enters the queue, the CPU is not assigned to it because at that time the remaining CPU burst of currently running process (that is, P_3) is 1 ms which is shorter than that of P_4. When P_3 completes, there are three processes P_1 (6 ms), P_2 (3 ms) and P_4 (3 ms) in the queue. To break the tie between P_2 and P_4, the scheduler takes into consideration their arrival order and the CPU is allocated first to P_2, then to P_4 and finally, to P_1.

Waiting time for P_1 = (11– 1) = 10 ms as P_1 enters at $t = 0$, executes for 1 ms, preempts at $t = 1$, and then resumes at $t = 11$

Waiting time for P_2 = (5 – 2 – 1) = 2 ms as P_2 enters at $t = 1$, executes for 2 ms, preempts at $t = 3$, and then resumes at $t = 5$

Waiting time for P_3 = 0 ms as P_3 enters at $t = 3$, starts immediately and executes completely

Waiting time for P_4 = (8 – 4) = 4 ms as P_4 enters at $t = 4$, starts at $t = 8$ and executes completely

Average waiting time = (10 + 2 + 0 + 4)/4 = 4 ms

Turnaround time for P_1 = (17 – 0) = 17 ms as P_1 enters at $t = 0$ and exits at $t = 17$

Turnaround time for P_2 = (8 – 1) = 7 ms as P_2 enters at $t = 1$ and exits at $t = 8$

Turnaround time for P_3 = (5 – 3) = 2 ms as P_3 enters at $t = 3$ and exits at $t = 5$

Turnaround time for P_4 = (11 – 4) = 7 ms as P_4 enters at $t = 4$ and exits at $t = 11$

Average turnaround time = (17 + 7 + 2 + 7)/4 = 8.25 ms

Advantages

- A long process that is near to its completion may be favoured over the short processes entering the system. This results in an improvement in the turnaround time of the long process.

Disadvantages

- Like SJF, it also requires an estimate of the next CPU burst of a process in advance.
- Favouring a long process nearing its completion over the several short processes entering the system may impact the turnaround times of short processes waiting in the queue.
- It favours only those long processes that are just about to complete and not those who have just started their operation. Thus, starvation of long processes still may occur.

NOTES

Priority-Based Scheduling

As the name implies, in priority-based scheduling algorithm, each process is manually assigned a priority and the process that is assigned the higher priority process is scheduled before the lower priority process. At any point of time, the process having the highest priority among all the ready processes is scheduled first. In case, two processes are having the same priority, they are executed in the FCFS order.

The priority scheduling may be either preemptive or non-preemptive. The choice is made whenever a new process enters the ready queue while some process is executing. If the newly arrived process has the higher priority than the currently running process, the preemptive priority scheduling algorithm preempts the currently running process and allocates CPU to the new process. On the other hand, the non-preemptive scheduling algorithm allows the currently running process to complete its execution and the new process has to wait for the CPU.

Note: *Both SJF and SRTN are special cases of priority-based scheduling where priority of a process is equal to inverse of the next CPU burst. Lower is the CPU burst, higher will be the priority.*

A major design issue related with priority scheduling is how to compute priorities of the processes. The priority can be assigned to a process either internally defined by the system depending on the process's characteristics like memory usage, I/O frequency, usage cost, and so on, or externally defined by the user executing that process.

Example 5.4: Consider four processes P_1, P_2, P_3, and P_4 with their arrival times, required CPU burst (in milliseconds), and priorities as shown in the following table.

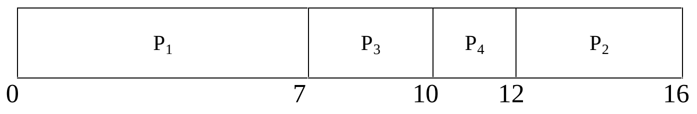
Process	P_1	P_2	P_3	P_4
Arrival time	0	1	3	4
CPU burst (ms)	7	4	3	2
Priority	4	3	1	2

Assuming that the lower priority number means the higher priority, how will these processes be scheduled according to non-preemptive as well as preemptive priority scheduling algorithm? Compute the average waiting time and average turnaround time in both cases.

NOTES

Solution: **Non-Preemptive Priority Scheduling Algorithm**

The processes will be scheduled as depicted in the following Gantt chart.



Initially, P_1 enters the ready queue at $t = 0$ and gets the CPU as there are no other processes in the queue. While it is executing, P_2, P_3, and P_4 enter the queue at $t = 1$, $t = 3$, and $t = 4$, respectively. When CPU becomes free, that is, at $t = 7$, it is allocated to P_3 because it is having the highest priority (that is, 1) among the three processes. When P_3 completes, CPU is allocated to the next lower priority process, that is, P_4 and finally, the lowest priority process P_2 is executed.

Waiting time for P_1 = 0 ms as P_1 starts immediately

Waiting time for P_2 = (12 – 1) = 11 ms as P_2 enters at $t = 1$ and starts at $t = 12$

Waiting time for P_3 = (7 – 3) = 4 ms as P_3 enters at $t = 3$ and starts at $t = 7$

Waiting time for P_4 = (10 – 4) = 6 ms as P_4 enters at $t = 4$ and starts at $t = 10$

Average waiting time = (0 + 11 + 4 + 6)/4 = 5.25 ms

Turnaround time for P_1 = (7 – 0) = 7 ms as P_1 enters at $t = 0$ and exits at $t = 7$

Turnaround time for P_2 = (16 – 1) = 15 ms as P_2 enters at $t = 1$ and exits at $t = 16$

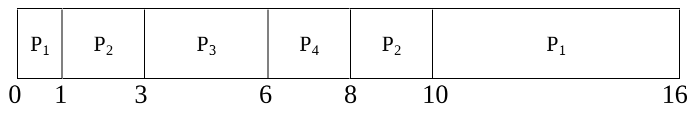
Turnaround time for P_3 = (10 – 3) = 7 ms as P_3 enters at $t = 3$ and exits at $t = 10$

Turnaround time for P_4 = (12 – 4) = 8 ms as P_4 enters at $t = 4$ and exits at $t = 12$

Average turnaround time = (7 + 15 + 7 + 8)/4 = 9.25 ms

Preemptive Priority Scheduling Algorithm

The processes will be scheduled as depicted in the following Gantt chart.



Initially, P_1 of priority 4 enters the ready queue at $t = 0$ and gets the CPU as there are no other processes in the queue. While it is executing, at time $t = 1$, P_2 of priority 3 greater than that of currently running process P_1, enters the queue. Therefore, P_1 is preempted (with remaining CPU burst of 6 ms) and the CPU is allocated to P_2. During execution of P_2, P_3 of priority 1 enters at $t = 3$. Again CPU is switched from P_2 (with remaining CPU burst of 2 ms) to P_3 as the priority of P_3 is greater than that of P_2. However, when at time $t = 4$, P_4 of priority 2 enters the queue, the CPU is not assigned to it because it has lower priority than currently running process P_3. When P_3 completes, there are three processes P_1, P_2, and P_4 in the ready queue having priorities 4, 3, and 2, respectively. The CPU is allocated first to P_4, then to P_2 and finally to P_1.

Waiting time for P_1 = (10 – 1) = 9 ms as P_1 enters at t = 0, executes for 1 ms, preempts at t = 1 and then resumes at t = 11

Waiting time for P_2 = (8 – 2 – 1) = 5 ms as P_2 enters at t = 1, executes for 2 ms, preempts at t = 3 and then resumes at t = 8

Waiting time for P_3 = 0 ms as P_3 enters at t = 3, starts immediately and executes completely

Waiting time for P_4 = (6 – 4) = 2 ms as P_4 enters at t = 4, starts at t = 6 and executes completely

Average waiting time = (9 + 5 + 0 + 2)/4 = 4 ms

Turnaround time for P_1 = (16 – 0) = 16 ms as P_1 enters at t = 0 and exits at t = 16

Turnaround time for P_2 = (10 – 1) = 9 ms as P_2 enters at t = 1 and exits at t = 10

Turnaround time for P_3 = (6 – 3) = 3 ms as P_3 enters at t = 3 and exits at t = 6

Turnaround time for P_4 = (8 – 4) = 4 ms as P_4 enters at t = 4 and exits at t = 8

Average turnaround time = (16 + 9 + 3 + 4)/4 = 8 ms

Advantages

- Important processes are always executed first without letting them to wait in the queue because of the least priority execution of less important processes.

Disadvantages

- It suffers from the problem of starvation of lower priority processes, because of the constant arrival of higher priority processes. It seldom gives chance to lower priority processes to acquire the CPU. One possible solution to this problem is **aging** which is a process of gradually increasing the priority of a low priority process with increase in its waiting time. If the priority of a low priority process is increased after each fixed time of interval, it is ensured that at some time it will become a highest priority process and will ultimately get executed.

Highest Response Ratio Next Scheduling

The Highest Response Ratio Next (HRN) scheduling is a non-preemptive scheduling algorithm that schedules the processes based on their response ratio. Whenever, CPU becomes available, the process having the highest value of response ratio among all the ready processes is scheduled next. The Response Ratio (RR) of a process in the queue is computed by using the following equation.

NOTES

$$\text{Response Ratio (RR)} = \frac{\text{Time since arrival + CPU burst}}{\text{CPU burst}}$$

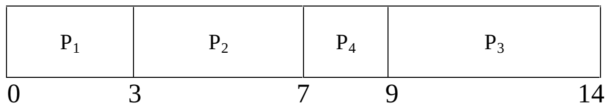
Initially, when a process enters, its response ratio is 1. It goes on increasing at the rate of (1/CPU burst) as the process's waiting time increases.

Example 5.5: Consider four processes P_1, P_2, P_3, and P_4 with their arrival times and required CPU burst (in milliseconds) as shown in the following table.

Process	P_1	P_2	P_3	P_4
Arrival time	0	2	3	4
CPU burst (ms)	3	4	5	2

How will these processes be scheduled according to HRN scheduling algorithm? Compute the average waiting time and average turnaround time.

Solution: The processes will be scheduled as depicted in the following Gantt chart.



Initially, P_1 enters the ready queue at $t = 0$ and CPU is allocated to it. By the time P_1 completes, P_2 and P_3 have arrived at $t = 2$ and $t = 3$, respectively. At $t = 3$, the response ratio of P_2 is $((3 – 2) + 4)/4 = 1.25$ and of P_3 is 1 as it has just arrived. Therefore P_2 is scheduled next. During execution of P_2, P_4 enters the queue at $t = 4$. When P_2 completes at $t = 7$, the response ratio of P_3 is $((7 – 3) + 5)/5 = 1.8$ and of P_4 is $((7 – 4) + 2)/2 = 2.5$. As P_4 has higher response ratio, the CPU is allocated to it and after its completion, P_3 is executed.

Waiting time for P_1 = 0 ms as P_1 starts immediately

Waiting time for P_2 = (3 – 2) = 1 ms as P_2 enters at $t = 2$ and starts at $t = 3$

Waiting time for P_3 = (9 – 3) = 6 ms as P_2 enters at $t = 3$ and starts at $t = 9$

Waiting time for P_4 = (7 – 4) = 3 ms as P_4 enters at $t = 4$ and starts at $t = 7$

Average waiting time = (0 + 1 + 6 + 3)/4 = 2.5 ms

Turnaround time for P_1 = (3 – 0) = 3 ms as P_1 enters at $t = 0$ and exits at $t = 3$

Turnaround time for P_2 = (7 – 2) = 5 ms as P_2 enters at $t = 2$ and exits at $t = 7$

Turnaround time for $P_3 = (14 - 3) = 11$ ms as P_3 enters at $t = 3$ and exits at $t = 14$

Turnaround time for $P_4 = (9 - 4) = 5$ ms as P_4 enters at $t = 4$ and exits at $t = 9$

Average turnaround time = (3 + 5 + 11 + 5)/4 = 6 ms

Advantages

- It favours short processes because with increase in waiting time, the response ratio of short processes increases speedily as compared to long processes. Thus, they are scheduled earlier than long processes.
- Unlike SJF, starvation does not occur since with increase in waiting time, the response ratio of long processes also increases and eventually they are scheduled.

Disadvantages

- Like SJF and SRTN, it also requires an estimate of the expected service time (CPU burst) of a process.

Round Robin Scheduling

Round Robin (RR) scheduling is one of the most widely used preemptive scheduling algorithms which considers all the processes as equally important and treats them in a favourable manner. Each process in the ready queue gets a fixed amount of CPU time (generally from 10 to 100 ms) known as **time slice** or **time quantum** for its execution. If the process does not execute completely till the end of time slice, it is preempted and the CPU is allocated to the next process in the ready queue. However, if the process blocks or terminates before the time slice expires, the CPU is switched to the next process in the ready queue at that moment only.

To implement the round robin scheduling algorithm, the ready queue is treated as a circular queue. All the processes arriving in the ready queue are put at the end of queue. The CPU is allocated to the first process in the queue, and the process executes until its time slice expires. If the CPU burst of the process being executed is less than one time quantum, the process itself releases the CPU and is deleted from the queue. The CPU is then allocated to the next process in the queue. However, if the process does not execute completely within the time slice, an interrupt occurs when the time slice expires. The currently running process is preempted, put back at the end of the queue and the CPU is allocated to the next process in the queue. The preempted process again gets the CPU after all the processes before it in the queue have been allocated their CPU time slice. The whole process continues until all the processes in queue have been executed.

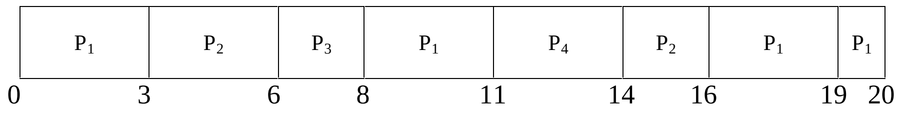
Example 5.6: Consider four processes P_1, P_2, P_3 and P_4 with their arrival times and required CPU burst (in milliseconds) as shown in the following table.

Process	P_1	P_2	P_3	P_4
Arrival time	0	1	3	4
CPU burst (ms)	10	5	2	3

NOTES

Assuming that the time slice is 3 ms, how will these processes be scheduled according to round robin scheduling algorithm? Compute the average waiting time and average turnaround time.

Solution: The processes will be scheduled as depicted in the following Gantt chart.



Initially, P_1 enters the ready queue at $t = 0$ and gets the CPU for 3 ms. While it executes, P_2 and P_3 enter the queue at $t = 1$ and $t = 3$, respectively. Since, P_1 does not execute within 3 ms, an interrupt occurs when the time slice gets over. P_1 is preempted (with remaining CPU burst of 7 ms), put back in the queue after P_3 because P_4 has not entered yet and the CPU is allocated to P_2. During execution of P_2, P_4 enters in the queue at $t = 4$ and put at the end of queue after P_1. When P_2 times out, it is preempted (with remaining CPU burst of 2 ms) and put back at the end of queue after P_4. The CPU is allocated to the next process in the queue, that is, to P_3 and it executes completely before the time slice expires. Thus, the CPU is allocated to the next process in the queue which is P_1. P_1 again executes for 3 ms, then preempted (with remaining CPU burst of 4 ms) and put back at the end of the queue after P_2 and the CPU is allocated to P_4. P_4 executes completely within the time slice and the CPU is allocated to next process in the queue, that is, P_2. As P_2 completes before the time out occurs, the CPU is switched to P_1 at $t = 16$ for another 3 ms. When the time slice expires, CPU is again allocated to P_1 as it is the only process in the queue.

Waiting time for $P_1 = (5 + 5) = 10$ ms as P_1 enters at $t = 0$, starts immediately, waits for $t = 3–8$ and then again waits for $t = 11–16$

Waiting time for $P_2 = (3 – 1 + 8) = 10$ ms as P_2 enters at $t = 1$, starts at $t = 3$, waits for $t = 6–14$ and then resumes at $t = 14$

Waiting time for $P_3 = (6 – 3) = 3$ ms as P_3 enters at $t = 3$, starts at $t = 6$ and executes completely

Waiting time for $P_4 = (11 – 4) = 7$ ms as P_4 enters at $t = 4$, starts at $t = 11$ and executes completely

Average waiting time = (10 + 10 + 3 + 7)/4 = 7.5 ms

Turnaround time for $P_1 = (20 – 0) = 20$ ms as P_1 enters at $t = 0$ and exits at $t = 20$

Turnaround time for $P_2 = (16 – 1) = 15$ ms as P_2 enters at $t = 1$ and exits at $t = 16$

Turnaround time for $P_3 = (8 – 3) = 5$ ms as P_3 enters at $t = 3$ and exits at $t = 8$

Turnaround time for $P_4 = (14 – 4) = 10$ ms as P_4 enters at $t = 4$ and exits at $t = 14$

Average turnaround time = (20 + 15 + 5 + 10)/4 = 12.5 ms

The performance of round robin scheduling is greatly affected by the size of the time quantum. If the time quantum is too small, a number of context switches occur which in turn increase the system overhead. The more time will be spent in performing context switching rather than executing the processes. On the other hand, if the time quantum is too large, the performance of round robin simply degrades to FCFS.

Note: *If the time quantum is too small, say 1 ms, the round robin scheduling is called* ***processor sharing****.*

Advantages

- It is efficient for time sharing systems where the CPU time is divided among the competing processes.
- It increases the fairness among the processes.

Disadvantages

- The processes (even the short processes) may take a long time to execute. This decreases the system throughput.
- It requires some extra hardware support such as a timer to cause interrupt after each time out.

Note: *Ideally, the size of time quantum should be such that 80% of the processes could complete their execution within one time quantum.*

Multilevel Queue Scheduling

The multilevel queue scheduling is designed especially for the environments where the processes can be categorized into different groups on the basis of their different response time requirements or different scheduling needs. One possible categorization may be based on whether the process is a system process, batch process or an interactive process (Refer Figure 5.2). Each group of processes is associated with a specific priority. The system processes, for example, may have the highest priority whereas the batch processes may have the least priority.

To implement multilevel scheduling algorithm, the ready queue is split up into as many separate queues as there are groups. Whenever, a new process enters, it is assigned permanently to one of the ready queues depending on its properties like memory requirements, type and priority. Each ready queue has its own scheduling algorithm. For example, for batch processes, FCFS scheduling algorithm may be used, and for interactive processes, one may use the round robin scheduling algorithm. In addition, the processes in higher priority queues are executed before those in lower priority queues. This implies no batch process can run unless all the system processes and interactive processes have been executed completely. Moreover, if a process enters into a higher priority queue while a process in lower priority queue is executing, then the lower priority process would be preempted in order to allocate the CPU to the higher priority process.

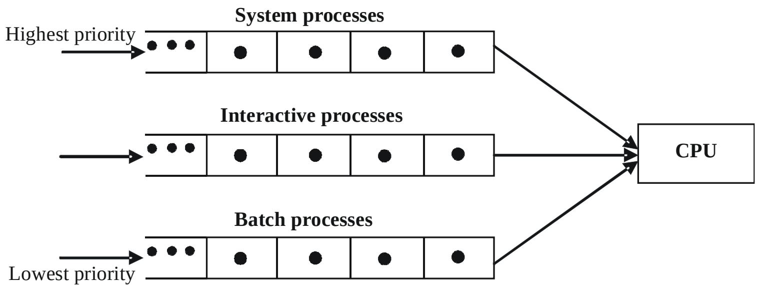


Fig. 5.2 *Multilevel Queue Scheduling*

Advantages

- Processes are permanently assigned to their respective queues and do not move between queues. This results in low scheduling overhead.

Disadvantages

- The processes in lower priority queues may have to starve for CPU in case processes are continuously arriving in higher priority queues. One possible way to prevent starvation is to time slice among the queues. Each queue gets a certain share of CPU time which it schedules among the processes in it. Note that the time slice of different priority queues may differ.

5.4.1 Multilevel Feedback Queue Scheduling

The multilevel feedback queue scheduling also known as **multilevel adaptive scheduling** is an improved version of multilevel queue scheduling algorithm. In this scheduling algorithm, processes are not permanently assigned to queues; instead they are allowed to move between the queues. The decision to move a process between queues is based on the time taken by it in execution so far and its waiting time. If a process uses too much CPU time, it is moved to a lower priority queue. Similarly, a process that has been waiting for too long in a lower priority queue is moved to a higher priority queue in order to avoid starvation.

To understand this algorithm, consider a multilevel feedback queue scheduler (Refer Figure 5.3) with three queues, namely, Q_1, Q_2 and Q_3. Further, assume that the queues Q_1 and Q_2 employ round robin scheduling algorithm with time quantum of 5 ms and 10 ms, respectively while in queue Q_3, the processes are scheduled in FCFS order. The scheduler first executes all processes in Q_1. When Q_1 is empty, the scheduler executes the processes in Q_2. Finally, when both Q_1 and Q_2 are empty, the processes in Q_3 are executed. While executing processes in Q_2, if a new process arrives in Q_1, the currently executing process is preempted and the new process starts executing. Similarly, a process arriving in Q_2 preempts a process executing in Q_3. Initially, when a process enters into ready queue; it is placed in Q_1 where it is allocated the CPU for 5 ms. If the process finishes its execution within 5 ms, it exits from the queue. Otherwise, it is preempted and placed at the end of Q_2. Here, it is allocated the CPU for 10 ms (if Q_1 is empty) and still if it does not finish, it is preempted and placed at the end of Q_3.

NOTES

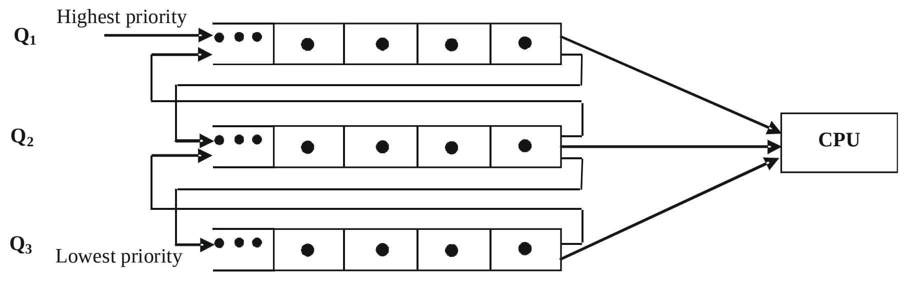


Fig. 5.3 *Multilevel Feedback Queue Scheduling*

Example 5.7: Consider four processes P_1, P_2, P_3 and P_4 with their arrival times and required CPU burst (in milliseconds) as shown in the following table.

Process	P_1	P_2	P_3	P_4
Arrival time	0	12	25	32
CPU burst (ms)	25	18	4	10

Assume that there are three ready queues Q_1, Q_2 and Q_3. The CPU time slice for Q_1 and Q_2 is 5 ms and 10 ms, respectively and in Q_3, processes are scheduled on FCFS basis. How will these processes be scheduled according to multilevel feedback queue scheduling algorithm? Compute the average waiting time and average turnaround time.

Solution: The processes will be scheduled as depicted in the following Gantt chart.

P_1	P_1	P_2	P_1	P_3	P_2	P_4	P_1	P_2	P_4
Q_1	Q_2	Q_1	Q_2	Q_1	Q_2	Q_1	Q_2	Q_2	Q_2

0 5 12 17 25 29 32 37 42 52 57

Initially, P_1 enters the system at $t = 0$, placed in Q_1 and allocated the CPU for 5 ms. Since, it does not execute completely, it is moved to Q_2 at $t = 5$. Now Q_1 is empty so the scheduler picks up the process from the head of Q_2. Since, P_1 is the only process in Q_2, it is again allocated the CPU for 10 ms. But during its execution, P_2 enters Q_1 at $t = 12$, therefore P_1 is preempted and P_2 starts executing. At $t = 17$, P_2 is moved to Q_2 and placed after P_1. The CPU is allocated to the first process in Q_2, that is, P_1. While P_1 is executing, P_3 enters Q_1 at $t = 25$ so P_1 is preempted, placed after P_2 in Q_2 and P_3 starts executing. As P_3 executes completely within time slice, the scheduler picks up the first process in Q_2 which is P_2 at $t = 29$. While P_2 is executing, P_4 enters Q_1 at $t = 32$ because of which P_2 is preempted and placed after P_1 in Q_2. The CPU is assigned to P_4 for 5 ms and at $t = 37$, P_4 is moved to Q_2 and placed after P_2. At the same time, the CPU is allocated to P_1 (first process in Q_2). When it completes at $t = 42$, the next process in Q_2 which is P_2, starts executing. When it completes, the last process in Q_2, that is, P_4 is executed.

Waiting time for P_1 = (5 + 12) = 17 ms as P_1 first waits for t = 12–17 and then again waits for t = 25–37

Waiting time for P_2 = (12+10) = 22 ms as P_2 first waits for t = 17–29 and then again waits for t = 32–42

Waiting time for P_3 = 0 ms as P_3 enters at t = 25, starts immediately and executes completely

Waiting time for P_4= (52 – 37) = 15 ms as P_4 waits for t = 37–52

Average waiting time = (17 + 22 + 0 + 15)/4 = 13.5 ms

Turnaround time for P_1 = (42 – 0) = 42 ms as P_1 enters at t = 0 and exits at t = 42

Turnaround time for P_2 = (52 – 12) = 40 ms as P_2 enters at t = 12 and exits at t = 52

Turnaround time for P_3 = (29 – 25) = 4 ms as P_3 enters at t = 25 and exits at t = 29

Turnaround time for P_4 = (57 – 32) = 25 ms as P_4 enters at t = 32 and exits at t = 57

Average turnaround time = (42 + 40 + 4 + 25)/4 = 27.75 ms

Advantages

- It is fair to I/O-bound (short) processes as these processes are not required to wait for an inordinate amount of time. Hence, are executed quickly.
- It prevents starvation by moving a lower priority process to a higher priority queue if it has been waiting for too long a period..

Disadvantages

- It is the most complex and cryptic scheduling algorithm.
- Moving the processes between queues causes a number of context switches which results in an increased overhead.
- The turnaround time for long processes may increase significantly.

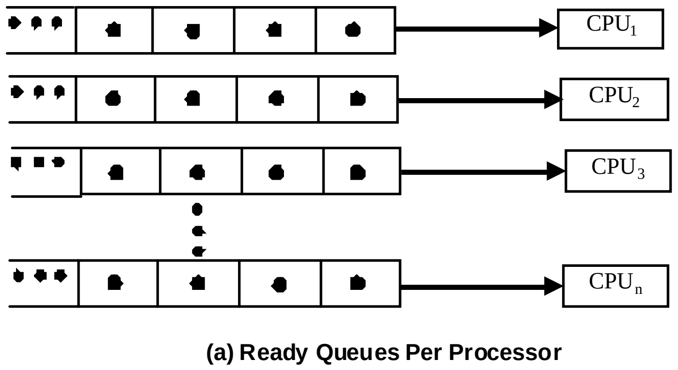
5.5 MULTIPLE PROCESSOR SCHEDULING

So far, we have discussed the scheduling of a single processor among a number of processes in the queue. In case of having more than one processor, different scheduling mechanisms need to be incorporated. In this section, we will concentrate on **homogeneous** multiprocessor systems which mean the systems in which all processors are identical in terms of their functionality, and any process in the queue can be assigned to any available processor.

The scheduling criteria for multiprocessor scheduling are same as that for single processor scheduling. But there are also some new considerations which are now discussed.

Implementation of Ready Queue

In multiprocessor systems, the ready queue can be implemented in two ways. Either there may be a separate ready queue for each processor (Refer Figure 5.4(a)) or there may be a single shared ready queue for all the processors (Refer Figure 5.4(b)). In the former case, it may happen that at any moment the ready queue of one processor is empty while the other processor is very busy in executing processes. To prevent this situation, the latter approach is preferred in which all the processes enter into one queue and scheduled on any available processor.



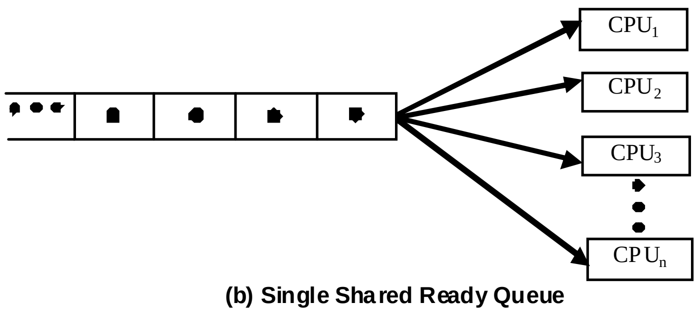


Fig. 5.4 *Implementation of Ready Queue in Multiprocessor Systems*

Scheduling Approaches

The next issue is how to schedule the processes from the ready queue to multiple processors. For this, one of the following scheduling approaches may be used.

- **Symmetric Multiprocessing (SMP)**: In this approach, each processor is self-scheduling. For each processor, the scheduler selects a process for execution from the ready queue. Since, multiple processors need to access common data structure, this approach necessitates synchronization among multiple processors. This is required so that no two processors could select the same process and no process is lost from the ready queue.

- **Asymmetric Multiprocessing**: This approach is based on the master–slave structure among the processors. The responsibility of making scheduling decisions, I/O processing and other system activities is limited to only one processor (called **master**), and other processors (called **slaves**) simply execute the user's code. Whenever some processor becomes available, the master processor examines the ready queue and selects a process for it. This approach is easier to implement than symmetric multiprocessing as only one processor has access to the system data structures. But at the same time, this approach is inefficient because a number of processes may block on the master processor.

Load Balancing

On SMP systems having a private ready queue for each processor, it might happen at a certain moment of time that one or more processors are sitting idle while others are overloaded with a number of processes waiting for them. Thus, in order to achieve the better utilization of multiple processors, load balancing is required which means to keep the workload evenly distributed among multiple processors. There are two techniques to perform load balancing, namely, *push migration* and *pull migration*.

In **push migration** technique, the load is balanced by periodically checking the load of each processor and shifting the processes from the ready queues of overloaded processors to that of less overloaded or idle processors. On the other hand, in **pull migration** technique, the idle processor itself pulls a waiting process from a busy processor.

Note: *Load balancing is often unnecessary on SMP systems with a single shared ready queue.*

Processor Affinity

Processor affinity means an effort to make a process to run on the same processor it was executed last time. Whenever a process executes on a processor, the data most recently accessed by it is kept in the cache memory of that processor. Next time if the process is run on the same processor, then most if its memory accesses are satisfied in the cache memory only and as a result the process execution speeds up. However, if the process is run on some different processor next time, the cache of the older processor becomes invalid and the cache of the new processor is to be re-populated. As a result, the process execution is delayed. Thus, an attempt should be made by the operating system to run a process on the same processor each time instead of migrating it to some another processor.

When an operating system tries to make a process to run on the same processor but does not guarantee to always do so, it is referred to as **soft affinity**. On the other hand, when an operating system provides system calls that force a process to run on the same processor, it is referred to as **hard affinity**. In soft affinity, there is a possibility of process migration from one processor to another whereas in hard affinity, the process is never migrated to some another processor.

Check Your Progress

1. Define process scheduling.
2. What happens if a process needs to perform some I/O operation during its execution?
3. Define process spawning.
4. What is the difference between symmetric and asymmetric direct communication?
5. What is the function of a dispatcher?
6. What is the difference between the non-preemptive and preemptive scheduling algorithms?
7. How are priorities assigned to processes in a priority scheduling algorithm?
8. How is the response ratio of a process computed?
9. What should be the ideal size of time quantum?
10. How does multilevel scheduling result in low scheduling overhead?
11. How does multilevel feedback queue scheduling provide an improvement over multilevel queue scheduling?
12. What is the main difference between soft affinity and hard affinity?

5.6 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. The procedure of determining the next process to be executed on the CPU is called process scheduling.
2. If the process needs to perform some I/O operation during its execution, it is removed from the ready queue and put into the appropriate device queue. After the process completes its I/O operation and is ready for the execution, it is switched from the device queue to ready queue.
3. The task of creating a new process on the request of some another process is called process spawning.
4. In symmetric direct communication, both sender and receiver process need to know each other's PID. On the other hand, in asymmetric direct communication, only the sender process needs to know PID of the receiver process but the receiver process need not know the PID of the sender process.
5. The dispatcher is a module of the operating system whose function is to set up the execution of the process selected by the short-term scheduler on the CPU.
6. In non-preemptive scheduling algorithms, once the CPU is allocated to a process, it cannot be taken back until the process voluntarily releases it or

the process terminates. On the other hand, in preemptive scheduling algorithms, the CPU can be forcibly taken back from the currently running process before its completion and allocated to some other process.

7. The priority can be assigned to a process either internally defined by the system depending on the process's characteristics like memory usage, I/O frequency, usage cost, and so on, or externally defined by the user executing that process.
8. The Response Ratio (RR) of a process in the queue is computed by using the following equation.

$$\text{Response Ratio (RR)} = \frac{\text{Time since arrival} + \text{CPU burst}}{\text{CPU burst}}$$

9. Ideally, the size of time quantum should be such that 80% of the processes could complete their execution within one time quantum.
10. The multilevel scheduling results in low scheduling overhead as the processes are permanently assigned to their respective queues and do not move between queues.
11. In multilevel feedback scheduling algorithm, processes are not permanently assigned to queues; instead they are allowed to move between the queues.
12. In soft affinity, there is a possibility of process migration from one processor to another whereas in hard affinity, the process is never migrated to some another processor.

5.7 SUMMARY

- The algorithm used by the scheduler to carry out the selection of a process for execution is known as scheduling algorithm.
- The time period elapsed in processing before performing the next I/O operation is known as CPU burst.
- The time interval in performing I/O before the next CPU burst is known as I/O burst.
- Dispatcher is the module of the operating system that performs the function of setting up the execution of the selected process on the CPU.
- For scheduling purposes, the scheduler may consider some performance measures and optimization criteria which include fairness, CPU utilization, balanced utilization, throughput, waiting time, turnaround time and response time.
- A broad range of algorithms are used for the effective CPU scheduling. These scheduling algorithms fall into two categories, namely, non-preemptive and preemptive.

- In non-preemptive scheduling algorithms, once the CPU is allocated to a process, it cannot be taken back until the process voluntarily releases it or the process terminates.
- In preemptive scheduling algorithms, the CPU can be forcibly taken back from the currently running process before its completion and allocated to some other process.
- FCFS is one of the simplest non-preemptive scheduling algorithms in which the processes are executed in the order of their arrival in the ready queue.
- The shortest job first also known as shortest process next or shortest request next is a non-preemptive scheduling algorithm that schedules the processes according to the length of CPU burst they require.
- The shortest remaining time next also known as shortest time to go is a preemptive version of the SJF scheduling algorithm. It takes into account the length of remaining CPU burst of the processes rather than the whole length in order to schedule them.
- In priority-based scheduling algorithm, each process is assigned a priority and the higher priority processes are scheduled before the lower priority processes.
- The highest response ratio next scheduling is a non-preemptive scheduling algorithm that schedules the processes according to their response ratio. Whenever, CPU becomes available, the process having the highest value of response ratio among all the ready processes is scheduled next.
- The round robin scheduling is one of the most widely used preemptive scheduling algorithms in which each process in the ready queue gets a fixed amount of CPU time (generally from 10 to 100 ms) known as time slice or time quantum for its execution.
- The multilevel queue scheduling is designed for the environments where the processes can be categorized into different groups on the basis of their different response time requirements or different scheduling needs.
- The multilevel feedback queue scheduling also known as multilevel adaptive scheduling is an improved version of multilevel queue scheduling algorithm. In this scheduling algorithm, processes are not permanently assigned to queues; instead they are allowed to move between the queues.
- In multiprocessor systems, the ready queue can be implemented in two ways. Either there may be a separate ready queue for each processor or there may be a single shared ready queue for all the processors.
- In symmetric multiprocessing scheduling approach, each processor is self-scheduling. For each processor, the scheduler selects a process for execution from the ready queue.
- In asymmetric multiprocessing scheduling approach, the responsibility of making scheduling decisions, I/O processing and other system activities is

up to only one processor (called master), and other processors (called slaves) simply execute the user's code.

- In order to achieve the better utilization of multiple processors, load balancing is required which means to keep the workload evenly distributed among multiple processors. There are two techniques to perform load balancing, namely, push migration and pull migration.
- In push migration technique, the load is balanced by periodically checking the load of each processor and shifting the processes from the ready queues of overloaded processors to that of less overloaded or idle processors.
- In pull migration technique, the idle processor itself pulls a waiting process from a busy processor.
- Processor affinity means an effort to make a process to run on the same processor it was executed last time.
- When an operating system tries to make a process to run on the same processor but does not guarantee to always do so, it is referred to as soft affinity.
- When an operating system provides system calls that force a process to run on the same processor, it is referred to as hard affinity.

5.8 KEY WORDS

- **Process scheduling:** Procedure of determining the next process to be executed on the CPU.
- **Swapping:** Task of temporarily switching a process in and out of main memory.
- **Process spawning:** Task of creating a new process on the request of some another process.
- **Concurrent processes:** Processes that coexist in the memory at the same time.
- **CPU burst:** Time period elapsed in processing before performing the next I/O operation.
- **I/O burst:** Time period elapsed in performing I/O before the next CPU burst.
- **Time slice or time quantum:** Fixed amount of CPU time (generally from 10 to 100 ms) each process in the ready queue gets for its execution.

5.9 SELF-ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. Distinguish between non-preemptive and preemptive scheduling algorithms.

2. Define throughput, turnaround time, waiting time and response time.
3. List the situations that may require the scheduler to make scheduling decisions.
4. Write short notes on the following:
   (a) Load balancing
   (b) Difference between multilevel queue and multilevel feedback queue scheduling
   (c) Soft affinity vs. hard affinity
   (d) Dispatcher
   (e) Scheduling approaches for multiprocessor scheduling

Long-Answer Questions

1. Compare the ways of process communication in message passing systems.
2. Consider the indirect communication method where mailboxes are used.
   (i) Suppose a process P wants to wait for two messages, one from mailbox M and other from mailbox N.
   (ii) Suppose P wants to wait for one message from mailbox M or from mailbox N (or from both).

   What will be the sequence of execution of `send()` and `receive()` calls in both cases?
3. Explain the relation (if any) between the following pairs of scheduling algorithms:
   (i) Round robin and FCFS
   (ii) Multilevel feedback queue and FCFS
   (iii) SJF and SRTN
   (iv) SRTN and priority-based
4. Which non-preemptive scheduling algorithms suffer from starvation and under what conditions?
5. Consider five processes P_1, P_2, P_3, P_4 and P_5 with their arrival times, required CPU burst (in milliseconds), and priorities as shown in the following table.

Process	P_1	P_2	P_3	P_4	P_5
Arrival time	0	1	3	4	5
CPU burst (ms)	10	6	3	2	5
Priority	4	3	1	2	3

Assume that the lower the number, higher the priority. Compute the average waiting time and average turnaround time of the processes for each of the following scheduling algorithms. Also determine which of the following scheduling algorithms result in minimum waiting time.

(i) FCFS

(ii) SJF

(iii) HRN

(iv) Non-preemptive priority-based

6. Consider the same set of processes as shown in Question 6. Compute the average waiting time and average turnaround time of processes for each of the following scheduling algorithms.
   - SRTN
   - Preemptive priority-based
   - Round robin (if CPU time slice is 2 ms)

   Compare the performance of these scheduling algorithms with each other.
7. Which of the following scheduling algorithms favour I/O-bound processes and how?
   - Multilevel feedback queue
   - SJF
   - HRN
8. Consider a scheduling algorithm that prefers to schedule those processes first which have consumed the least amount of CPU time. How will this algorithm treat the I/O-bound and CPU-bound processes? Is there any chance of starvation?

5.10 FURTHER READINGS

Silberschatz, Abraham, Peter B. Galvin and Greg Gagne. 2008. *Operating System Concepts*, 8th Edition. New Jersey: John Wiley & Sons.

Tanenbaum, Andrew S. 2006. *Operating Systems Design and Implementation*, 3rd Edition. New Jersey: Prentice Hall.

Tanenbaum, Andrew S. 2001. *Modern Operating Systems*. New Jersey: Prentice Hall.

Deitel, Harvey M. 1984. *An Introduction to Operating Systems*. Boston (US): Addison-Wesley.

Stallings, William. 1995. *Operating Systems*, 2nd Edition. New Jersey: Prentice Hall.

Milenkovic, Milan. 1992. *Operating Systems: Concepts and Design*. New York: McGraw Hill Higher Education.

Mano, M. Morris. 1993. *Computer System Architecture*. New Jersey: Prentice Hall Inc.

BLOCK - III SYNCHRONIZATION

NOTES

UNIT 6 PROCESS SYNCHRONIZATION

Structure

6.0 INTRODUCTION

Operating systems that facilitate multiprogramming allow multiple processes to execute concurrently in a system even with a single processor. These processes may share information with each other through shared memory locations or shared files. A process that shares information with other processes is called a cooperating process. If cooperating processes are not executed in a systematic manner, there are possibilities of data inconsistency. Keeping these possibilities in mind, several mechanisms have been formulated to ascertain data consistency at various phases of execution with concurrent execution of cooperating processes.

In this unit, you will study the concepts of process synchronization, critical section problem, synchronization hardware, semaphores, classical problems of synchronization and monitors.

6.1 OBJECTIVES

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

- Analyse the need for synchronization among processes
- Describe the critical section problem

NOTES

- Explain the Peterson solution for a critical section problem
- Explain the bakery algorithm
- Discuss the hardware-supported solutions for the critical section problem
- Understand the concept and functions of semaphores
- Discuss various classical synchronization problems and their solutions
- Understand the role and function of monitors

6.2 SYNCHRONIZATION

Afore mentioned, the major drawback of unordered execution of cooperating processes is data inconsistency. Thus, to comprehend the concept, let us consider two cooperating processes P_1 and P_2 that update the balance of an account in a bank. The code segment for the processes is given in Table 6.1.

Table 6.1 *Code Segment for Processes P_1 and P_2*

Process P_1	Process P_2

`Read Balance`
`Balance = Balance + 1000`
	`Read Balance`
	`Balance = Balance – 400`

Suppose that the balance is initially 5000, then after the execution of both P_1 and P_2, it should be 5600. The correct result is achieved if P_1 and P_2 execute one by one in any order either P_1 followed by P_2 or P_2 followed by P_1. However, if the instructions of P_1 and P_2 are interleaved arbitrarily, the balance may not be 5600 after the execution of both P_1 and P_2. One possible interleaving sequence for the execution of instructions of P_1 and P_2 is given in Table 6.2.

Table 6.2 *Possible Interleaved Sequence*

Process P_1	Process P_2	Balance
`Read Balance`		5000
	`Read Balance`	5000
`Balance = Balance + 1000`		6000
	`Balance = Balance – 400`	4600

The above interleaved sequence results in an inconsistent balance, that is, 4600. If the order of last two instructions is interchanged, the balance would be 6000 (again, inconsistent). Note that a situation where several processes sharing some data execute concurrently and the result of execution depends on the order in which the shared data is accessed by the processes is called **race condition**.

To avoid race conditions or such inconsistent situations, some form of synchronization among the processes is required which ensures that only one process is manipulating the shared data at a time. One common way to synchronize the processes is signaling, in which the process generates signals to allow other processes to manipulate the shared data.

6.3 CRITICAL SECTION PROBLEM

Critical section is the portion of code of a process in which it accesses or changes the shared data. It is important for the system to ensure that the execution of critical sections by the cooperating processes is mutually exclusive. It means that no two processes are allowed to execute in their critical sections at one time. The **critical section** problem is to design a protocol that the processes can use to cooperate. Each process must request permission to enter its critical section and signal the entrance by setting the values of some variables. The process does this in the code just before the critical section. That part of code is called the **entry section**. After executing the critical section, the process again sets some variables to signal the exit from the critical section. The portion of code in which the process does this is called the **exit section**. A solution to critical section problem must satisfy the mutual exclusion requirement in addition to the following two requirements.

- **Progress**: Suppose a process P_1 is executing in its critical section, then all other processes that wish to enter their critical sections have to wait. When P_1 finishes its execution in critical section, a decision has to be settled as to which process will enter its subsequent critical section. In the decision, it is the basic principle that only the waiting processes will participate, and the decision should be made in a stipulated period of time. Principally, a process that has to exit from its critical section cannot hold back other processes from entering their critical sections.
- **Bounded Waiting**: A process wishing to enter its critical section cannot be detained for an unspecified period of time. There is always an upper limit on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before the permission is acknowledged.

  Countless workings have been reinforced over a period of time to overcome the problem of critical section. Fundamentally, these workings include software solutions, hardware-supported solutions, operating system primitives, and programming language constructs. In this section, we present two-process and multiple-process software solutions. Hardware-supported solutions, operating system primitives (semaphores) and programming language constructs (Monitors).

Peterson's Algorithm: Two-Process Solution

Peterson advocated an algorithm to solve the critical section problem for two processes. The algorithm lets the two processes `Pi` and `Pj` to share the following two variables:

```
int turn;
boolean flag[2];
```

The value of variable turn is initialized to either `i` or `j` and both the elements of array flag are initialized to false. The general structure for the code segment of process, say `Pi`, is as follows:

```
do
{
 flag[i] = true;
 turn = j;
 while(flag[j] && turn==j)              Entry section
           doNothing();

//critical section

 flag[i] = false;               Exit section

//remaining code

}while(1);
```

When any process, suppose `Pi`, wishes to enter its critical section, it first sets `flag[i]` to `true` and the value of `turn` to other number, that is, `j`. It, then, verifies the following two conditions:

1. Whether `flag[j]` is `true`
2. Whether `turn` equals `j`

If any of these conditions is false, the process `Pi` enters its critical section, otherwise, it waits. In case, only `Pi` wishes to enter the critical section, the first condition remains false. The process `Pi` then executes in its critical section, and after that resets the `flag[i]` to `false`, indicating that `Pi` is not in its critical section.

Now, consider the case when both `Pi` and `Pj` wish to enter their critical sections at the same time. In this case, both the elements of the `flag` will be set to `true` and the value of `turn` will be set to `i` and `j` one by one (by `Pi` and `Pj`) but only one retains. Now, the first condition is `true`, thus, the value of `turn` decides which process enters its critical section first. The other process has to wait. It implies mutual exclusion is preserved.

To verify that the algorithm also satisfies the other two requirements, observe that a process `Pi` can be prevented from entering its critical section if `flag[j]` is `true` and `turn` is `j`. If `Pj` does not wish to enter its critical section, then `Pi` found `flag[j]` as `false` and can enter its critical section. However, when both processes wish to enter their critical section at the same time the variable `turn` plays its role and allowed one process to enter its critical section. Suppose `turn` is `j`, then `Pj` is allowed first and `Pi` is stuck in the loop. Now, when `Pj` exits from its critical section, it sets `flag[j]` to `false` to indicate that it is not in its critical section now. This allows `Pi` to enter its critical section. It means `Pi` enters its critical section after at most one entry by `Pj`, satisfying both progress and bounded-waiting requirements.

Bakery Algorithm: Multiple-Process Solution

Lamport conceived the idea of **bakery algorithm**, targeted specifically to resolve the critical section problem for `N` processes. The algorithm lets the processes to share the following two variables:

NOTES

```
boolean choosing[N];
int number[N];
```

All the elements of the arrays, that is, `choosing` and `number` are initialized to `false` and `0`, respectively.

The algorithm assigns a number to each process and serves the process with the lowest number first. The algorithm cannot ensure that two processes do not receive the same number. Thus, if two processes, say `Pi` and `Pj`, receive the same number, then `Pi` is served first if `i<j`. The general structure for the code segment of process, say `Pi`, is as follows:

```
do
{
    choosing[i] = true;
    number[i] = MAX(number) + 1;
    choosing[i] = false;
    for(j=0;j<N;j++)
    {
         while(choosing[j])
              doNothing();
         while(number[j]!=0 && number[j]<number[i] ||
         number[j]==number[i] && j<i)
              doNothing();
    }
```

Entry section

```
//critical section

    number[i] = 0;
```

Exit section

```
//remaining code

}while(1);
```

***Note**: For simplicity, the notation* `MAX(number)` *is used to retrieve the maximum element in the array* `number`.

To verify that mutual exclusion is preserved, suppose a process `P0` is executing in its critical section and another process, say `P1`, attempts to enter the critical section. For `j=0`, the process `P1` is not blocked in the first `while` loop because `P0` had set `choosing[0]` to `false` in the entry section. Nonetheless, in the latter case `while` loop for `j=0`, `P1` finds the following:

- `number[j]!=0`, since `P0` is executing in the critical section after setting it to a nonzero number in the entry section
- `number[j]<number[i]`, since `P1` is assigned a number after `P0`. Though, `P1` may be assigned the same number as that of `P0` but in that case, `P1` finds `j<i` because 0<1.

NOTES

Since, the eventual result in the second while loop is true, `P1` is blocked in the while loop until `P0` finishes execution in its critical section, thus, preserving the mutual exclusion requirement. The algorithm not only preserves the mutual exclusion requirement, but also the progress and bounded-waiting requirements. To verify these requirements, observe that if two or more processes are waiting to enter the critical sections, then the process that had come first is allowed to enter the critical section first. The conditions in the second while statement ensure this. To be more precise,it means that the processes are served on the First-Come, First-Serve or FCFS basis, and no process is delayed because of starvation.

6.4 SYNCHRONIZATION HARDWARE

Hardware-supported solutions developed for the critical section problem make maximum use of hardware instructions available on many systems, thus, are effective and efficient.

A system with a single-processor, executes only one process at a given moment. The other processes can gain control of processor through interrupts. Therefore, to decipher any problem in the critical section, it must be ascertained that interrupt should not occur while a process is executing in its critical section. A process can achieve this by disabling interrupts before entering in its critical section. Note that the process must enable the interrupts immediately after finishing execution in its critical section.

Although this method appears to be quite simple and straightforward, but it has certain disadvantages. First, it is only feasible in a single-processor environment only because disabling interrupts in a multiprocessor environment takes a long time as message is transmitted to all the processors. This message passing delays processes from entering into their critical sections, thus, affecting the input output ratio of the system. Second, it may impact the scheduling goals, since the processor cannot be preempted from a process executing in its critical section.

Due to the aforesaid disadvantages, many systems provide special hardware instructions to solve the critical section problem. One special instruction is the `TestAndSet` instruction which can be defined as follows:

```
boolean TestAndSet (boolean &lock)
{
    if(lock)
       return lock;
    else
    {
       lock = true;
       return false;
    }
}
```

NOTES

A salient trait of the `TestAndSet` instruction is that it executes as an atomic action. This clearly implies that if two `TestAndSet` instructions are executed contemporaneously (each on an individual CPU) in a multiprocessor system, then one must complete the execution procedure before another one starts on another processor.

On systems that support the `TestAndSet` instruction, the mutual exclusion can be implemented by allowing the processes to share a Boolean variable, say `lock`, initialized to `false`. The general structure for the code segment of process, say `Pi`, is as follows:

```
do
{

  while(TestAndSet(lock))        ← Entry section
        doNothing();

  //critical section

  lock = false;                  ← Exit section

  //remaining code

}while(1);
```

The algorithm is easy and simple to understand. Any process that wishes to enter its critical section executes the `TestAndSet` instruction and passes the value of `lock` as a parameter to it. If the value of `lock` is `false` (means no process is in its critical section), the `TestAndSet` instruction sets the `lock` to `true` and returns `false`, which breaks the `while` loop and allows the process to enter its critical section. However, if the value of `lock` is `true`, the `TestAndSet` instruction returns `true`, thus, blocking the process in the loop. The algorithm satisfies the mutual exclusion requirement, but does not satisfy the bounded-waiting requirement.

Another special hardware instruction is the `Swap` instruction that operates on two Boolean variables. Like the `TestAndSet` instruction, the `Swap` instruction also executes as an atomic action. This instruction can be defined as follows:

```
void Swap(boolean &a, boolean &b)
{
boolean temp = a;
a = b;
b = temp;
}
```

On systems that support the `Swap` instruction, the mutual exclusion can be implemented by allowing the processes to share a Boolean variable, say `lock`, initialized to `false`. In addition to this, each process uses a local Boolean variable, say `key`. The general structure for the code segment of process, say `Pi`, is as follows:

NOTES

```
do
{

key = true;
while (key==true)
      Swap(lock, key);
```

Entry section

```
//critical section

lock = false;
```

Exit section

```
//remaining code

}while(1);
```

The algorithm is quite simple and uncomplicated for users to interpret and get the desired task executed. Initially, the value of `lock` is `false`, so the first process, say `Pi`, when executes the `Swap` instruction sets the `key` to `false` and `lock` to `true`. The `false` value for `key` allows the process to enter its critical section. Any other process, say `Pj`, that attempts to enter its critical section finds that the `lock` is `true` and when it is swapped with `key`, the `key` remains `true`. The `true` value for `key` blocks the process `Pj` in the `while` loop until the `lock` becomes `false`. Note that the `lock` becomes `false` when `Pi` exits from the critical section. This algorithm also fulfils the expectations of only the mutual exclusion requirement and the bounded-waiting requirements are not fulfiled.

To meet all the requirements of the solution for critical section problem, another algorithm is developed that uses the `TestAndSet` instruction. The algorithm lets the processes to share the following two variables:

```
boolean lock;
boolean waiting[N];
```

The variable `lock` and all the elements of array `waiting` are initialized to `false`. Each process also has a local Boolean variable, say `key`. The general structure for the code segment of process, say `Pi`, is as follows:

```
do
{

waiting[i] = true;
key = true;
while(waiting[i] && key)
      key = TestAndSet(lock);
waiting[i] = false;
```

Entry section

```
//critical section

j = (i + 1) % N;
while(j!=i && waiting[next_ts]==false)
      j = (j + 1) mod N;
if (j==i)
      lock = false;
else
      waiting[j] = false;
```

Exit section

```
//remaining code

}while(1);
```

To verify that the mutual exclusion requirement is met, suppose a process `Pi` attempts to enter its critical section. It first sets the `waiting[i]` and `key` to `true`, and then reaches the `while` loop in the entry section. If `Pi` is the first process attempting to enter its critical section, it finds that both the conditions in the `while` loop are `true`. Then it executes the `TestAndSet` instruction which sets the `lock` to `true` and returns `false`, since `lock` is initially `false`. The returned value, that is `false`, is assigned to `key`, which allows the process `Pi` to exit from the loop and enter its critical section after resetting the `waiting[i]` to `false`.

Now, the value of `lock` is `true`, thus, any other process, say `Pj`, that attempts to enter its critical section when executes the `TestAndSet` instruction sets the `key` to `true` and is blocked in the `while` loop until either `key` or `waiting[j]` becomes `false`. Note that neither `key` nor `waiting[j]` becomes `false` until `Pi` is in its critical section. This maintains the mutual exclusion requirement.

To verify the progress requirement, observe in the exit section that `Pi` sets either `lock` or `waiting[j]` to `false`. Setting `lock` (on which the value of `key` depends) or `waiting[j]` to `false` allows any other waiting process to enter its critical section.

The algorithm also satisfies the bounded-waiting requirement. To verify this, observe that when any process, say `Pi`, exits from its critical section, it scans the `waiting` array in the cyclic order (`i+1`, `i+2`, `...`, `N-1`, `0`, `1`, …, `i-1`) to locate the first process, say `Pj`, with `waiting[j]` equal to `true`. If no such process is found, `Pi` sets `lock` to `false`, so that any other process that now attempts to enter its critical section need not to wait. On the contrary, if such a process is found, it enters its critical section next, since `Pi` sets `waiting[j]` to `false`. In this way, each process gets its turn to enter its critical section after a maximum of `N-1` processes.

Check Your Progress

1. Define the term race condition.
2. What are the major requirements that must be followed by a solution to overcome a critical section problem?
3. What is the general structure for the code segment of a process under the bakery algorithm?
4. A critical section problem can be solved with disabling interrupts. What are the disadvantages of this method as a solution to a critical section problem?
5. State a test and set instruction.

6.5 SEMAPHORES

NOTES

In 1965, Dijkstra suggested the use of an abstract data type called a **semaphore** intended for controlling synchronization. The primary intention of employing a semaphore `S` (an integer variable) is to provide a general-purpose solution to critical section problem. In this proposal, two standard atomic operations are defined on `S`, namely, `wait` and `signal`, and after initialization, `S` is accessed only through these two operations. The definition of `wait` and `signal` operation in pseudocode is as follows:

```
wait(S)
{
    while(S<=0)
        doNothing();
    S—;
}

signal(S)
{
    S++;
}
```

The solution of critical section problem for `N` processes is implemented by allowing the processes to share a semaphore `S`, which is initialized to `1`. The general structure for the code segment of process, say `Pi`, is as follows:

```
do
{
 wait(S);       ← Entry section

//critical section

 signal(S);     ← Exit section

//remaining code

}while(1);
```

Here, it is important to note that all the solutions presented until now for the critical section problem, including the solution using semaphore, require busy waiting. This entails that if a process is executing in its critical section, all the other processes that attempt to enter their respective critical sections must loop continuously in the entry section. Executing a loop continuously wastes CPU cycles, and is considered to be a crucial lacuna in multiprogramming systems with one processor.

To master the problem of busy waiitng, the definition of semaphore is modified to hold an integer value and a list of processes, and the `wait` and `signal` operations are also modified. In the modified `wait` operation, when a process finds that the value of the semaphore is negative, it blocks itself in lieu of busy waiting. Blocking a process means it is inserted in the queue associated with the semaphore and the state of the process is switched to the waiting state. The `signal` operation is modified to remove a process, if any, from the queue associated with the semaphore and restart it. The modified definition of semaphore, the `wait` operation, and the `signal` operation is as follows:

```
struct semaphore
{
    int value;
    struct process *queue;
};

void wait(semaphore S)
{
    S.value—;
    if(S.value<0)
    {
       insert this process to queue associated with S
       block(); //suspend this process
    }
}

void signal(semaphore S)
{
    S.value++;
    if(S.value<=0)
    {
       remove a process P from the queue associated with
S
       wakeup(P); //resume the execution of blocked process
P
    }
}
```

***Note**: The* `block()` *operation and* `wakeup()` *operation are provided by the operating system as basic system calls.*

An important requirement is that both the `wait` and `signal` operations must be treated as atomic instructions. It means that no two processes can execute `wait` and `signal` operations on the same semaphore simultaneously. We can view this as a critical section problem, where the critical section consists of `wait` and `signal` operations. This problem is not a big issue and can be worked out by making use of either of the aforementioned solutions.

In this way, though, we have not completely eliminated the busy waiting but limited the busy waiting to only the critical sections consisting of `wait` and `signal` operations. Since these two operations are very short, busy waiting seldom takes place and even if it takes place, the duration is very short.

The semaphore presented above is known as **counting semaphore** or **general semaphore**, since its integer value can range over an unrestricted domain. Another type of semaphore is **binary semaphore** whose integer value can range only between 0 and 1. Binary semaphore is more straightforward to put into effect than general semaphore. The `wait` and `signal` operations for a binary semaphore `S`, initialized to 1, are as follows:

```
void wait(semaphore S)
{
    if(S.value==1)
       S.value = 0;
    else
       insert this process to queue associated with S
       block(); //suspend this process
}

void signal(semaphore S)
{
    if(emptyqueue()) //check if queue is empty
       S.value = 1;
    else
remove a process P from the queue associated with S
       wakeup(P); //resume the execution of blocked process
P
}
```

6.6 CLASSICAL PROBLEMS OF SYNCHRONIZATION

So far, the importance and correct utilization of semaphore to find a successful resolution for critical section problem was elucidated. Because of its versatile nature, semaphore also authenticates synchronization and hence, is also used to work out varying problems pertaining to synchronization. In this section, we present some classical problems of synchronization and using semaphores for synchronization purposes as a solutions of these problems. Note that these problems of synchronization are used for testing almost all the newly proposed synchronization scheme.

Bounded-Buffer Problem

The bounded-buffer problem is considered a feasible solution to the problem that was apprised by using shared memory. It allows at most `size-1` items to be in the buffer at the same time. One possible solution to eliminate this inadequacy is to have an integer variable `count`, initialized to 0, to keep track of the number of items in the buffer. The producer process increments the `count` every time it adds a new item to the buffer, and the consumer process decrements the `count` every time it removes an item from the buffer. The modified code of the producer and consumer processes is as follows:

```
//producer process
while(1)
{
    item_produced = produce_item();
    while(count == size)
       doNothing();
    buffer[in] = item_produced;
    in = (in + 1) % size;
    count++;
}

//consumer process
while(1)
{
    while(count == 0)
       doNothing();
    item_consumed = buffer[out];
    out = (out + 1) % size;
    count—;
    consume_item(item_consumed);
}
```

The producer process first determines whether the value of `count` is equal to size or not. If it is equal, the producer waits, since the buffer is full. Otherwise, it adds an item to the buffer and increments the `count`. Correspondingly, the consumer process first determines whether the value of `count` is `0` or not. If it is so, the consumer waits, since the buffer is empty. Otherwise, it removes an item from the buffer and decrements the `count`.

Though, both producer and consumer processes are correct when employed individually, but concurrent execution of these processes may result into the race condition. To understand this, suppose the statement `count++` is internally implemented as follows:

```
register1 = count
register1 = register1 + 1
count = register1
```

Here, `register1` is a local CPU register. In this implementation, the value of `count` is first read into a local CPU register. Then, the value in the register is incremented by one, which is finally assigned back to the variable `count`. Similarly, suppose the statement `count--` is internally implemented as follows:

```
register2 = count
register2 = register2 – 1
count = register2
```

Further, suppose the value of `count` is currently 2, and the producer process reads this value in `register1` and then increments the value in `register1`. The value in `register1` becomes `3`. However, before the producer process assigns back the incremented value to `count`, the scheduler decides to temporarily suspend it and start running the consumer process. The consumer process reads the value of `count` (which is still `2`) in `register2` and then decrements it. The value in `register2` becomes `1`.

Now, the order in which `count` is updated by the producer and consumer processes decides the final value of `count`. It means, if producer first and consumer secondly updates `count`, its value becomes `1`. On the other hand, if consumer first and producer secondly updates `count`, its value becomes `3`. However, the only correct value of `count` is `2`, which is now cannot be produced. The incorrect result is generated because access to the variable `count` is unconstrained and both the processes manipulate it concurrently.

To keep the practicability of occurrence of race condition at bay, we present a solution to the bounded-buffer problem using semaphores. The biggest advantage of this solution using semaphores is that it not only avoids the occurrence of race condition but also allows to have `size` items in the buffer at the same time, thus, eliminating the shortcomings of the solutions using shared memory. The following three semaphores are used in this solution.

- The `mutex` semaphore, initialized to `1`, is used to provide the producer and consumer processes the mutually exclusive access to the buffer. This semaphore ensures that only one process, either producer or consumer, is accessing the buffer and the associated variables at a time.
- The `full` semaphore, initialized to `0`, is used to count the number of full buffers. This semaphore ensures that the producer stops executing items when the buffer is full.

- The `empty` semaphore, initialized to the value of `size`, is used to count the number of empty buffers. This semaphore ensures that consumer stops executing when the buffer is empty.

The general structure for the code segment of producer process and consumer process is as follows:

```
//structure of a producer process
do
{
    item_produced = produce_item();
    wait(empty);
    wait(mutex);
    buffer[in] = item_produced;
    in = (in + 1) % size;
    signal(mutex);
    signal(full);
}while(1);

//structure of consumer process
do
{
    wait(full);
    wait(mutex);
    item_consumed = buffer[out];
    out = (out + 1) % size;
    signal(mutex);
    signal(empty);
    consume_item(item_consumed);
}while(1);
```

The Readers-Writers Problem

Concurrently executing processes that are sharing a data object, such as a file or a variable, fall into two groups: readers and writers. The processes in the **readers** group want only to read the contents of the shared object, whereas, the processes in **writers** group want to update (read and write) the value of shared object. There is no problem if multiple readers access the shared object simultaneously. However, if a writer and some other process (either a reader or a writer) access the shared object simultaneously, data may become inconsistent.

To ensure that such a problem does not arise, we must guarantee that when a writer is accessing the shared object, no reader or writer accesses that shared object. This synchronization problem is termed as **readers–writers** problem, and it has many variations. The first readers–writers problem (the simplest one) requires the following.

- All readers and writers should wait if a writer is accessing the shared object. It means writers should get mutually exclusive access to the shared object.
- Readers should not wait unless a writer is accessing the shared object. It means if a reader is currently reading the shared object and a writer and a reader request, then writer should wait, but reader should not wait just because a writer is waiting.

  To develop the viable solution to the first readers–writers problem, the readers are allowed to share two semaphores read and write, both initialized to 1, and an integer variable count, initialized to 0. The writers share the semaphore write with the readers. The functions of read and write semaphores and count variable are as follows:
- The `count` variable is used to count the number of readers currently reading the shared object. The `count` is updated each time a reader enters or exits the critical section.
- The `read` semaphore is used to provide mutual-exclusion to readers when `count` is being updated.
- The `write` semaphore is used to provide mutual-exclusion to writers. It is accessed by all the writers and only the first or last reader that enters or exits its critical section.

The general structure for the code segment of a reader process and a writer process is as follows:

```
//structure of a reader process
wait(read); //mutual-exclusion for readers before updating
 //count
count++;
if(count==1) //if it is the first reader
    wait(write);
signal(read);
...
reading contents of shared object
...
wait(read);
count—;
if(count==0) //if it is the only reader
       signal(write);
signal(read);

//structure of a writer process
```

```
wait(write); //mutual-exclusion for writers
...
updating the shared object
...
signal(write);
```

NOTES

The Dining Philosophers Problem

To understand the dining philosophers problem, consider five philosophers sitting around a circular table. There is a bowl of rice in the centre of the table and five chopsticks—one in between each pair of philosophers (Refer Figure 6.1).

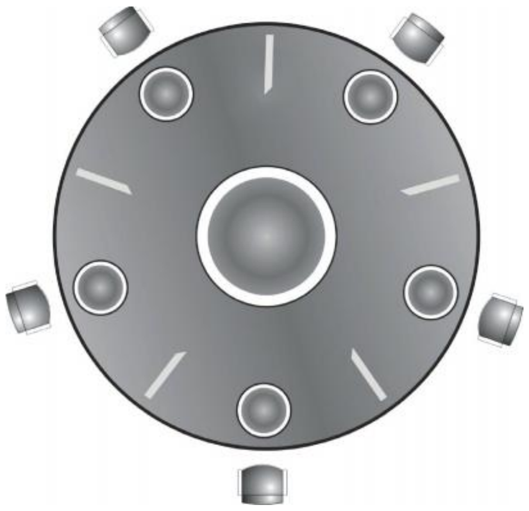


Fig. 6.1 *Situation in Dining Philosophers*

Initially, all the philosophers are in the thinking phase and while thinking, they make sure that they do not interact with each other. As time passes by, philosophers might feel hungry. When a philosopher feels hungry, he attempts to pick up the two chopsticks kept in close proximity to him (that are in between him and his left and his right philosophers). If the philosophers on his left and right are not eating, he successfully gets the two chopsticks. With the two chopsticks in his hand, he starts eating. After he finishes eating, the chopsticks are positioned back on the table and the philosopher begins to think again. On the contrary, if the philosopher on his left or right is already eating, then fails to grab the two chopsticks at the same time, and thus, has to wait. Here it is necessary to note that this situation is identical to the one that occurs in the system to allocate resources among several processes. Each process should get required resources to finish its task without being deadlocked and starved.

A solution to this problem is to represent each chopstick as a semaphore, and philosophers must grab or release chopsticks by executing `wait` operation or `signal` operation, respectively, on the appropriate semaphores. We use an array `chopstick` of size 5 where each element is initialized to `1`. The general structure for the code segment of philosopher `i` is as follows:

```
do
{
...
thinking
...

wait(chopstick[i]);
wait(chopstick[(i+1)%5];
...
eating
...

signal(chopstick[i]);
signal(chopstick[(i+1)%5];
...
thinking
...

}while(1);
```

This solution is simple and ensures that no two neighbors are eating at the same time. However, the solution is not free from deadlock. Suppose all the philosophers attempt to grab the chopsticks simultaneously and grab one chopstick successfully. In this case, all the elements of chopstick will be 0. Thus, when each philosopher attempts to grab the second chopstick, he will go in waiting state forever.

A simple solution to avoid this deadlock is to ensure that a philosopher either picks up both chopsticks or no chopstick at all. It means he must pick chopsticks in a critical section. A deadlock free solution to dining-philosophers problem is presented in the next section with the use of monitors.

6.7 MONITORS

A monitor is a programming language construct which is also used to provide mutually exclusive access to critical sections. The programmer defines monitor type which consists of declaration of shared data (or variables), procedures or functions that access these variables and initialization code. The general syntax of declaring a monitor type is as follows:

```
monitor <monitor-name>
{
//shared data (or variable) declarations
data type <variable-name>;
...
//function (or procedure) declarations

return_type <function-name>(parameters)
{
//body of function
}
.
.
.
monitor-name()
{
    //initialization code
}
}
```

The variables defined inside a monitor can only be accessed by the functions defined within the monitor, and it is not feasible for any process to access these variables.. Thus, if any process has to access these variables, it is only possible through the execution of the functions defined inside the monitor. Further, the monitor construct checks that only one process may be executing within the monitor at a given moment. But if a process is executing within the monitor, then other requesting processes are blocked and placed on an entry queue.

Though, monitor construct ensures mutual exclusion for processes, but sometimes programmer may find them insufficient to represent some synchronization schemes. For such situations, programmer needs to define his own synchronization mechanisms. He can define his own mechanisms by defining variables of `condition` type on which only two operations can be invoked: `wait` and `signal`. Suppose, programmer defines a variable `C` of `condition` type, then execution of the operation `C.wait()` by a process, say `Pi`, suspends the execution of `Pi`, and places it on a queue associated with the `condition` variable `C`. On the other hand, the execution of the operation `C.signal()` by a process, say `Pi`, resumes the execution of exactly one suspended process `Pj`, if any. It means that the execution of the `signal` operation by `Pi` allows other suspended process `Pj` to execute within the monitor. However, only one process is allowed to execute within the monitor at one time. Thus, monitor construct prevents `Pj` from resuming until `Pi` is executing in the monitor. There are following possibilities to handle this situation.

NOTES

- The process `Pi` must be suspended to allow `Pj` to resume and wait until `Pj` leaves the monitor.
- The process `Pj` must remain suspended until `Pi` leaves the monitor.
- The process `Pi` must execute the `signal` operation as its last statement in the monitor so that `Pj` can resume immediately.

Now, we are in a situation to use the monitor to develop a deadlock-free solution to dining philosophers problem. The following monitor controls the distribution of chopsticks to philosophers.

```
monitor diningPhilosophers
{
enum {thinking, hungry, eating} state[5];
condition self[5];
void getChopsticks(int i)
{
       int left, right;
       state[i] = hungry;
       left = (i+4)%5;
       right = (i+1)%5;
       if((state[left]==eating) || (state[right]==eating))
          self[i].wait();
       else
          state[i] = eating;
}
void putDownChopsticks(int i)
{
       int left, right;
state[i] = thinking;
left = (i+4)%5;
right = (i+1)%5;
       verifyAndAllow(left);
       verifyAndAllow(right);
}
void verifyAndAllow(int i)
{
int left, right;
left = (i+4)%5;
right = (i+1)%5;
if(state[i]==hungry)
{
if((state[left]!=eating) && (state[right]!=eating))
{
```

```
            state[i] = eating;
   self[i].signal();
   }
   }
       }
   void initial()
   {
   int i;
   for(i=0; i<5; i++)
          state[i] = thinking;
   }
   }
```

NOTES

Each philosopher that feels hungry must invoke the `getchopsticks()` operation before start eating and after eating is finished, he must invoke `putDownchopsticks()` operations and then may start thinking. Thus, the general structure for the code segment philosopher `i` is as follows:

```
...
diningPhilosophers.getChopsticks(i);
eating
diningPhilosophers.putDownChopsticks(i);
...
```

The `getChopsticks()` operation changes the state of philosopher process from thinking to hungry and then verifies whether philosopher on his left or right is in eating state. If either philosopher is in eating state, then the philosopher process is suspended and its state remains hungry. Otherwise, the state of philosopher process is changed to eating.

After eating is finished, each philosopher invokes `putDownChopsticks()` operation before start thinking. This operation changes the state of philosopher process to thinking and then invoke `verifyAndAllow()` operation for philosophers on his left and right side (one by one). The `verifyAndAllow()` operation verifies whether the philosopher feels hungry, and if so then allows him to eat in case philosophers on his left and right side are not eating.

Check Your Progress

6. Define the term semaphore.
7. What is busy waiting?
8. What are the requirements of first readers-writers problem?
9. Give the general structure for the code segment of a reader process.
10. What is a monitor?

6.8 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. A situation where several processes sharing some data execute concurrently and the result of execution depends on the order in which the shared data is accessed by the processes is called race condition.
2. Bounded waiting, progress, mutual exclusion are the major requirements that must be met by a solution to overcome a critical section problem.
3. The general structure for the code segment of a process, say `Pi` is as follows:

```
do
{
 choosing[i] = true;
 number[i] = MAX(number) + 1;
 choosing[i] = false;
 for(j=0;j<N;j++)
 {
      while(choosing[j])
           doNothing();                              ← Entry section
      while(number[j]!=0 && number[j]<number[i] ||
      number[j]==number[i] && j<i)
           doNothing();
 }

//critical section

 number[i] = 0;   ← Exit section

//remaining code
```

4. First, disabling interrupts is feasible in a single-processor environment only because doing this in a multiprocessor environment takes time as message is passed to all the processors. This message passing delays processes from entering into their critical sections, thus, decreasing the system efficiency. Second, it may affect the scheduling goals, since the processor cannot be preempted from a process executing in its critical section.
5. The `TestAndSet` instruction can be defined as follows:

```
boolean TestAndSet (boolean &lock)
{
   if(lock)
      return lock;
   else
   {
      lock = true;
      return false;
   }
}
```

6. A semaphore `S` is an integer variable which is used to provide a general-purpose solution to critical section problem. Two standard atomic operations are defined on `S`, namely, `wait` and `signal` and after initialization, `S` is accessed only through these two operations.
7. If a process is executing in its critical section, then all other processes that attempt to enter their critical sections loop continuously in the entry section. The time that the process spends executing a loop continuously is busy waiting. Executing a loop continuously wastes CPU cycles, and is considered a major problem in multiprogramming systems with one processor.
8. The first readers–writers problem (the simplest one) requires the following:
   - All readers and writers should wait if a writer is accessing the shared object. It means writers should get mutually exclusive access to the shared object.
   - Readers should not wait unless a writer is accessing the shared object. It means if a reader is currently reading the shared object and a writer and a reader request, then writer should wait, but reader should not wait just because of a writer is waiting.
9. The general structure for the code segment of a reader process is as follows:

```
    //structure of a reader process
    wait(read);  //mutual-exclusion  for  readers  before
updating
    //count
    count++;
    if(count==1) //if it is the first reader
       wait(write);
    signal(read);
    ...
    reading contents of shared object
    ...
    wait(read);
    count—;
    if(count==0) //if it is the only reader
          signal(write);
    signal(read);
```

10. A monitor is a programming language construct which is also used to provide mutually exclusive access to critical sections. The programmer defines monitor type which consists of declaration of shared data (or variables), procedures or functions that access these variables, and initialization code.

6.9 SUMMARY

- If cooperating processes are not executed in an ordered manner, data inconsistency may occur.
- A situation where several processes sharing some data execute concurrently and the result of the execution depends on the order in which the shared data is accessed by the processes is called race condition.
- To avoid race conditions, some form of synchronization among the processes is required which ensures that only one process is manipulating the share data at a time.
- Critical section is the portion of code of a process in which it accesses or changes the shared data. No two processes are allowed to execute in their critical sections at one time.
- The critical section problem is to design a protocol that the processes can use to cooperate.
- A solution to critical section problem must me the mutual exclusion, progress, and bounded-waiting requirements.
- Peterson proposed an algorithm to solve the critical section problem for two processes.
- Lamport proposed an algorithm, known as bakery algorithm, to solve the critical section problem for N processes.
- The hardware-supported solutions developed for the critical section problem that make use of hardware instruction available on many systems, thus, are effective and efficient.
- On a system with single-processor, the critical section problem can be solved by disabling interrupts, but this solution is not feasible in multiprocessor environment.
- Many systems provide special hardware instructions to solve the critical section problem.
- One special instruction is the `TestAndSet` instruction. An important characteristic of this instruction is that it executes as an atomic action.
- Another special hardware instruction is the Swap instruction that operates on two Boolean variables. It also executes as an atomic action.
- To meet all the requirement of the solution for critical section problem, an algorithm is developed that uses the `TestAndSet` instruction.
- A semaphore `S` is an integer variable which is used to provide a general-purpose solution to critical section problem.
- Two standard atomic operations are defined on `S`, namely, wait and signal, and after initialization, `S` is accessed only through these two operations.

- The semaphore whose integer value can range over an unrestricted domain is known as counting semaphore or general semaphore. Another type of semaphore whose integer value can range only between `0` and `1` is known as binary semaphore.
- Semaphore can also be used to solve various synchronization problems. Some classical problems of synchronization include bounded-buffer problem, readers–writers problem, and dining philosophers problem. These problems of synchronization are used for testing almost all the newly proposed synchronization scheme.
- A monitor is a programming language construct which is also used to provide mutually exclusive access to critical sections.
- Monitor construct ensures that only one process may be executing within the monitor at a time.
- Programmer can define his own synchronization mechanisms by defining variables of condition type on which only two operations can be invoked: wait and signal.
- Monitor is used to develop a deadlock-free solution to dining philosophers problem.

6.10 KEY WORDS

- **Race condition:** A situation where several processes sharing some data execute concurrently and the result of the execution depends on the order in which the shared data is accessed by the processes
- **Semaphore:** An integer variable which is used to provide a general-purpose solution to critical section problem
- **Monitor**: A programming language construct which is also used to provide mutually exclusive access to critical sections

6.11 SELF-ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. Define the process synchronization.
2. What do you understand by bounded waiting in terms of critical section problem?
3. State the Bakery algorithm of multiple- process solution.
4. How do we define the `wait` and `signal` operation in the pseudocode of semaphores?

5. Define `mutex`, `full` and `empty` semaphores.
6. What are basic requirements for readers-writers problem?
7. What are the possibilities of handling the situation occurred by monitors?
8. Write short notes on the following:
   (a) Semaphore
   (b) Swap instruction
   (c) Entry and exit section
   (d) Critical section
9. What is busy waiting? How is semaphore used to overcome the busy waiting problem?

Long-Answer Questions

1. Explain with examples why some form of synchronization among processes is required.
2. Define the critical section problem. Explain all the requirements that a solution to a critical section problem must meet.
3. Explain the bakery algorithm to solve a critical section problem.
4. State a `TestAndSet` instruction. Also write the algorithm that uses the `TestAndSet` instruction to solve the critical section problem and meets all the requirements of the solution for such a problem.
5. Explain the use of semaphores in developing a solution to a bounded-buffer problem.
6. Describe the dining philosophers problem. Suggest a solution to the dining-philosopher problem with the use of monitors.

6.12 FURTHER READINGS

Silberschatz, Abraham, Peter B. Galvin and Greg Gagne. 2008. *Operating System Concepts*, 8th Edition. New Jersey: John Wiley & Sons.

Tanenbaum, Andrew S. 2006. *Operating Systems Design and Implementation*, 3rd Edition. New Jersey: Prentice Hall.

Tanenbaum, Andrew S. 2001. *Modern Operating Systems*. New Jersey: Prentice Hall.

Deitel, Harvey M. 1984. *An Introduction to Operating Systems*. Boston (US): Addison-Wesley.

Stallings, William. 1995. *Operating Systems*, 2nd Edition. New Jersey: Prentice Hall.

Milenkovic, Milan. 1992. *Operating Systems: Concepts and Design*. New York: McGraw Hill Higher Education.

Mano, M. Morris. 1993. *Computer System Architecture*. New Jersey: Prentice Hall Inc.

UNIT 7 DEADLOCKS

NOTES

Structure

7.0 INTRODUCTION

A deadlock occurs when every process in a set of processes is in a simultaneous wait state and each of them is waiting for the release of a resource held exclusively by one of the waiting processes in the set. Deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process. Deadlocks may occur on a single system or across several machines. Consider an example when two trains are coming toward each other on the same track and there is only one track, none of the trains can move once they are in front of each other. A similar situation occurs in operating systems when there are two or more processes that hold some resources and wait for resources held by other(s). It is very important to handle a deadlock before it can occur. So, the system checks each transaction before it is executed to make sure it does not lead to deadlock. If there is even a slight chance that a transaction may lead to deadlock in the future, it is never allowed to execute. None of the processes can proceed until at least one of the waiting processes releases the acquired resource.

In this unit, you will study the basic concepts of deadlocks, deadlocks characterization and methods for handling deadlocks.

7.1 OBJECTIVES

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

- Understand the concept of a system model
- Discuss the features that characterize a deadlock
- Describe the different methods of handling deadlock

7.2 SYSTEM MODELS

NOTES

A system consists of various types of resources, like input/output devices, memory space, processors, disks, and so on. For some resource types, several instances may be available. For example, a system may have two printers. When several instances of a resource type are available, any one of them can be used to satisfy the request for that resource type.

A process may need multiple resource types to accomplish its task. However, to use any resource type, it must follow some steps which are as follows:

(i) Request for the required resource

(ii) Use the allocated resource

(iii) Release the resource after completing the task

If the requested resource is not available, the requesting process enters a waiting state until it acquires the resource. Consider a system with a printer and a disk drive and two processes P_1 and P_2 are executing simultaneously on this system. During execution, the process P_1 requests for the printer and process P_2 requests for the disk drive and both the requests are granted. Further, the process P_2 requests for the printer held by process P_1 and process P_1 requests for the disk drive held by the process P_2. Here, both processes will enter a waiting state. Since each process is waiting for the release of resource held by other, they will remain in waiting state forever. This situation is called **deadlock**.

7.3 DEADLOCK CHARACTERIZATION

Before discussing the methods to handle a deadlock, we will discuss the conditions that cause a deadlock and how a deadlock can be depicted using resource allocation graph.

Deadlock Conditions

A deadlock occurs when all the following four conditions are satisfied at any given point of time.

(i) **Mutual Exclusion**: Only one process can acquire a given resource at any point of time. Any other process requesting for that resource has to wait for earlier process to release it.

(ii) **Hold and Wait**: Process holding a resource allocated to it and waiting to acquire another resource held by other process.

(iii) **No Preemption**: Resource allocated to a process cannot be forcibly revoked by the system, it can only be released voluntarily by the process holding it.

(iv) Circular Wait: A set of processes waiting for allocation of the resources held by other processes forms a circular chain in which each process is waiting for the resource held by its successor process in chain.

In the absence of any one of these conditions, deadlock will not occur. We will discuss these conditions in detail in subsequent sections and see how they can be prevented.

Resource Allocation Graph

A deadlock can be depicted with the help of a directed graph known as **resource allocation graph**. The graph consists of two different types of nodes, namely, *processes* and *resources*. Processes are depicted as circles and resources as squares. A directed arc from a process to a resource is known as **request edge** and indicates that the process has requested for the resource and is waiting for it to be allocated. Whereas a directed arc from a resource to a process is known as **assignment edge** and indicates that the resource has been allocated to the process. For example, consider the resource allocation graph shown in Figure 7.1. Here, the process P_1 is holding resource R_2 and requesting for the resource $R_{1,}$ which in turn is held by the process P_2. The process P_2 is requesting for the resource R_2 held by the process $P\text{-}_1$, this means there is a deadlock.

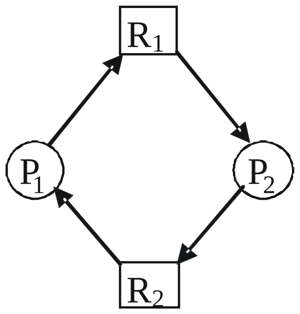


Fig. 7.1 *Resource Allocation Graph*

It can be observed that this graph forms a cycle ($P_1 \rightarrow R_1 \rightarrow {}_2 \rightarrow R_2 \rightarrow P_1$). A cycle in the resource allocation graph indicates that there is deadlock and all the processes forming the part of the cycle are deadlocked. If there is no cycle in a graph, there is no deadlock. In this example, there is only one instance of each resource type. However, there can be multiple instances of a resource type. The resource allocation graph for two instances (R_{21} and R_{22}) of resource type R_2 is shown in Figure 7.2.

NOTES

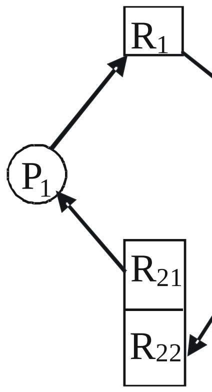


Fig. 7.2 *Resource Allocation Graph for Multiple Instances of a Resource Type*

This resource allocation graph has the following indications:

(i) Process P_1 is waiting for the allocation of resource R_1 held by the process P_2.

(ii) Process P_2 is waiting for the allocation of instance (R_{22}) of resource type R_2.

(iii) Process P_1 is holding an instance (R_{21}) of resource type R_2.

It can be observed that the graph forms a cycle but still processes are not deadlocked. The process P_2 can acquire the second instance (R_{22}) of the resource type R_2 and completes its execution. After completing the execution, it can release the resource R_1 that can be used by the process P_1. Since, no process is in waiting state, there is no deadlock.

From this discussion, it is clear that if each resource type has exactly one instance, cycle in resource allocation graph indicates a deadlock. If each resource type has several instances, cycle in resource allocation graph does not necessarily imply a deadlock. Thus, it can be concluded that if a graph contains no cycle, the set of processes are not deadlocked; however, if there is a cycle then deadlock may exist.

7.4 HANDLING DEADLOCKS

A deadlock can be handled in four different ways which are as follows:

- Prevent the deadlock from occurring.
- Adopt methods for avoiding the deadlock.
- Allow the deadlock to occur, detect it and recover from it.
- Ignore the deadlock.

Deadlock prevention or **deadlock avoidance techniques** can be used to ensure that deadlocks never occur in a system. If any of these two techniques is not used, a deadlock may occur. In this case, an algorithm can be provided for detecting the deadlock and then using the algorithm to recover the system from the deadlock.

One or the other method must be provided to either prevent the deadlock from occurrence or detect the deadlock and take an appropriate action if a deadlock has occurred. However, if in a system, deadlock occurs less frequently, say, once in two years, then it is better to ignore the deadlocks instead of adopting expensive techniques for deadlock prevention, deadlock avoidance, or deadlock detection and recovery.

Check Your Progress

1. Define the system model.
2. What are the steps performed by a process to use any resource type?
3. List the conditions necessary for a deadlock to occur.
4. Which of the following is not associated with the resource allocation graph to depict a deadlock?

   (a) Request edge

   (b) Claim edge

   (c) Assignment edge

   (d) None of these
5. List the various ways to handle a deadlock.

7.5 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. A system consists of various types of resources, like input/output devices, memory space, processors, disks, and so on. For some resource types, several instances may be available. For example, a system may have two printers. When several instances of a resource type are available, any one of them can be used to satisfy the request for that resource type.
2. To use any resource, following steps must be performed:

   (i) Request for the required resource.

   (ii) Use the allocated resource.

   (iii) Release the resource after completing the task.
3. A deadlock occurs when all the following four conditions are satisfied:

   (i) Mutual exclusion

   (ii) Hold and wait

(iii) No preemption

(iv) Circular wait

4. Claim Edge

NOTES

5. A deadlock can be handled in four different ways which are as follows:
   - (i) Prevent the deadlock from occurring.
   - (ii) Adopt methods for avoiding the deadlock.
   - (iii) Allow the deadlock to occur, detect it and recover from it.
   - (iv) Ignore the deadlock.

7.6 SUMMARY

- Deadlock occurs when every process in a set of processes are in a simultaneous wait state and each of them is waiting for the release of a resource held exclusively by one of the waiting processes in the set.
- A system consists of various types of resources, like input/output devices, memory space, processors, disks, and so on. For some resource types, several instances may be available. When several instances of a resource type are available, any one of them can be used to satisfy the request for that resource type.
- Four necessary conditions for a deadlock are mutual exclusion, hold and wait, no preemption and circular wait.
- A deadlock can be depicted with the help of a directed graph known as resource allocation graph.
- In case there are multiple instances of a resource type in a system, the deadlock cannot be avoided using resource allocation graph algorithm. An algorithm known as banker's algorithm is used in such a case.
- If each resource type has exactly one instance, cycle in resource allocation graph indicates a deadlock. If each resource type has several instances, cycle in resource allocation graph does not necessarily imply a deadlock.
- Deadlock prevention or deadlock avoidance techniques can be used to ensure that deadlocks never occur in a system.

7.7 KEY WORDS

- **Hold and wait:** Process holding a resource allocated to it and waiting to acquire another resource held by other process.
- **No preemption:** Resource allocated to a process cannot be forcibly revoked by the system, it can only be released voluntarily by the process holding it.

- **Resource allocation graph:** A deadlock can be depicted with the help of a directed graph known as resource allocation graph.
- **Request edge:** A directed arc from a process to a resource is known as request edge.

NOTES

7.8 SELF-ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. What do you understand by the system and write the steps to invoke its resources?
2. Define mutual exclusion principle.
3. What is meant by circular wait in the deadlock characterization?
4. How do we define the significance of assignment edge?
5. State the different ways of handling any deadlock.

Long- Answer Questions

1. Explain the significance of system models in process coordination.
2. Briefly illustrate the deadlock conditions giving suitable examples.
3. Elaborate on the indications required to explain any resource allocation graph for multiple instances of a resource type.
4. Discuss the importance of deadlock prevention and briefly describe the deadlock avoidance techniques.

7.9 FURTHER READINGS

Silberschatz, Abraham, Peter B. Galvin and Greg Gagne. 2008. *Operating System Concepts*, 8th Edition. New Jersey: John Wiley & Sons.

Tanenbaum, Andrew S. 2006. *Operating Systems Design and Implementation*, 3rd Edition. New Jersey: Prentice Hall.

Tanenbaum, Andrew S. 2001. *Modern Operating Systems*. New Jersey: Prentice Hall.

Deitel, Harvey M. 1984. *An Introduction to Operating Systems*. Boston (US): Addison-Wesley.

Stallings, William. 1995. *Operating Systems*, 2nd Edition. New Jersey: Prentice Hall.

Milenkovic, Milan. 1992. *Operating Systems: Concepts and Design*. New York: McGraw Hill Higher Education.

Mano, M. Morris. 1993. *Computer System Architecture*. New Jersey: Prentice Hall Inc.

NOTES

UNIT 8 DEADLOCK PREVENTION

NOTES

Structure

8.0 INTRODUCTION

A deadlock can be detected by a resource scheduler as it keeps track of all the resources that are allocated to different processes. After a deadlock is detected, it can be resolved using the following methods, such as all the processes that are involved in the deadlock are terminated which is not a good approach as all the progress made by the processes is destroyed. Resources can be pre-empted from some processes and given to others till the deadlock is resolved. Deadlock Prevention is very important to prevent a deadlock before it can occur. So, the system checks each transaction before it is executed to make sure it does not lead to deadlock. If there is even a slight chance that a transaction may lead to deadlock in the future, it is never allowed to execute. Deadlock Avoidance is better to avoid a deadlock rather than take measures after the deadlock has occurred. The wait for graph can be used for deadlock avoidance. This is however only useful for smaller databases as it can get quite complex in larger databases. Real-time operating systems use Deadlock recovery. Killing all the process involved in the deadlock. Killing process one by one. After killing each process check for deadlock again keep repeating the process till system recover from deadlock.

In this unit, you will study the concepts of prevention, avoidance, detection and recovery of deadlock.

8.1 OBJECTIVES

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

- Explain how a deadlock can be prevented by eliminating the conditions of a deadlock

- Understand the concept of safe and unsafe state
- Discuss various deadlock avoidance algorithms
- Explain the different deadlock detection methodologies
- Describe the ways to recover from a deadlock

8.2 DEADLOCK PREVENTION

As stated earlier, a deadlock occurs when all of the four conditions are satisfied at any point of time. The deadlock can be prevented by not allowing all four conditions to be satisfied simultaneously; that is, by making sure that at least one of the four conditions does not hold. Now, let us analyse all four conditions one by one and see how their occurrence can be prevented.

Eliminating Mutual Exclusion

The mutual exclusion property does not hold for the resources that are sharable. For example, a file opened in read-only mode can be shared among various processes. Hence, processes will never have to wait for the sharable resources. However, there are certain resources which can never be shared, like printer can work for only one process at a time. It cannot print data being sent as output from more than one process simultaneously. Hence, the condition of mutual exclusion cannot be eliminated in case of all the resources.

Eliminating Hold and Wait Condition

This condition can be eliminated by not allowing any process to request for a resource until it releases the resources held by it, which is impractical as process may require the resources simultaneously. Another way to prevent hold and wait condition is by allocating all the required resources to the process before starting the execution of that process. The disadvantage associated with it is that a process may not know in advance about the resources that will be required during its execution. Even if it knows in advance, it may unnecessarily hold the resources which may be required at the end of its execution. Thus, the resources are not utilized optimally.

Eliminating No Preemption

The elimination of this condition means a process can release the resource held by it. If a process requests for a resource held by some other process then instead of making it wait, all the resources currently held by this process can be preempted. The process will be restarted only when it is allocated the requested as well as the preempted resources. Note that only those resources can be preempted whose current working state can be saved and can be later restored. For example, the resources like printer and disk drives cannot be preempted.

NOTES

Eliminating Circular-Wait Condition

The circular-wait condition can be eliminated by assigning a priority number to each available resource and a process can request resources only in increasing order. Whenever, a process requests for a resource, the priority number of the required resource is compared with the priority numbers of the resources already held by it. If the priority number of a requested resource is greater than that of all the currently held resources, the request is granted. If the priority number of a requested resource is less than that of the currently held resources, all the resources with greater priority number must be released first, before acquiring the new resource.

8.3 DEADLOCK AVOIDANCE

A deadlock can be prevented by eliminating any one of the four necessary conditions of the deadlock. Preventing deadlock using this method results in the inefficient use of resources. Thus, instead of preventing deadlock, it can be avoided by never allowing allocation of a resource to a process if it leads to a deadlock. This can be achieved when some additional information is available about how the processes are going to request for resources in future. Information can be in the form of the number of resources of each type that will be requested by a process and in which order. On the basis of the amount of information available, different algorithms can be used for deadlock avoidance.

One of the simplest algorithms requires each process to declare the maximum number of resources (of each type) required by it during its course of execution. This information is used to construct an algorithm that will prevent the system from entering a state of deadlock. This deadlock avoidance algorithm continuously examines the state of resource allocation ensuring that circular-wait condition never exists in a system. The state of resource allocation can be either safe or unsafe.

Safe and Unsafe State of Resource Allocation

A state is said to be **safe** if allocation of resources to processes does not lead to the deadlock. More precisely, a system is in safe state only if there is a safe sequence. A **safe sequence** is a sequence of process execution such that each and every process executes till its completion. For example, consider a sequence of processes (P_1, P_2, P_3, ..., P_n) forming a safe sequence. In this sequence, first the process P_1 will be executed till its completion, and then P_2 will be executed till its completion, and so on. The number of resources required by any process can be allocated either from the available resources or from the resources held by previously executing process. When a process completes its execution, it releases all the resources held by it which then can be utilized by the next process in a sequence. That is, the request for the resources by the process P_n can be satisfied either from the available resources or from the resources held by the process P_m, where $m<n$.

Since this sequence of process execution is safe, the system following it is in the safe state. If no such sequence of process execution exists then the state of the system is said to be **unsafe** (Refer Figure 8.1).

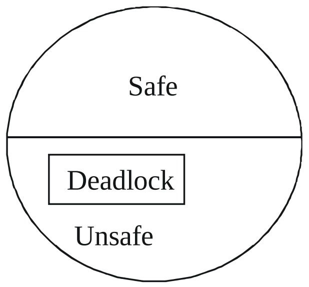


Fig. 8.1 *Relationship between Safe State, Unsafe State and Deadlock*

Consider a system in which three processes P_1, P_2 and P_3 are executing and there are 10 instances of a resource type. The maximum number of resources required by each process, the number of resources already allocated and the total number of available resources are shown in Figure 8.2.

	Maximum	Currently allocated
P_1	9	3
P_2		2
P_3	7	2

Available resources = 3

(a) Initial State

	Maximum	Currently allocated
P_1	9	3
P_2	4	4
P_3	7	2

Available resources = 1

(b) Resource Allocation to Process P_2

	Maximum	Currently allocated
P_1	9	3
P_2	0	0
P_3	7	2

Available resources = 5

(c) State after Completion of Process P_2

	Maximum required	Currently allocated
P_1	9	3
P_2	0	0
P_3	7	7

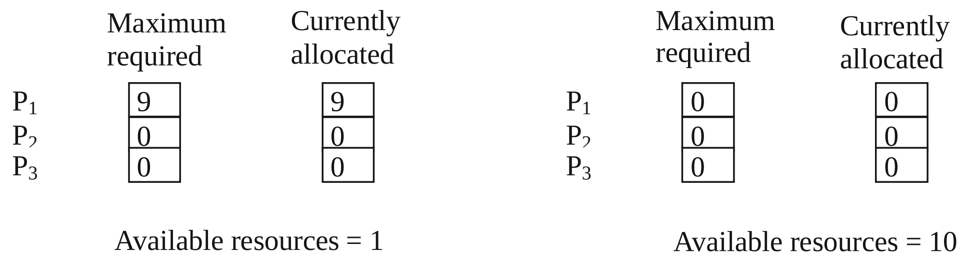
Available resources = 0

(d) Resource Allocation to Process P_3

	Maximum required	Currently allocated
P_1	9	3
P_2	0	0
P_3	0	0

Available resources = 7

(e) State after Completion of Process P_3



(f) Resource Allocation to Process P_1 **(g) State after Completion of Process P_1**

Fig. 8.2 *Safe Sequence of Execution of Processes*

On the basis of available information, it can be easily observed that the resource requirements of the process P_2 can be easily satisfied. Therefore, resources are allocated to the process P_2 and it is allowed to execute till its completion. After the execution of the process P_2, all the resources held by it are released. The number of the resources now available are not enough to be allocated to the process P_1, whereas, they are sufficient for the process P_3. Therefore, resources are allocated to the process P_3 and it is allowed to execute till its completion. The number of resources available after the execution of process P_3 can now easily be allocated to the process P_1. Hence, the execution of the processes in the sequence P_2, P_3, P_1 is safe.

Now consider a sequence P_2, P_1, P_3. In this sequence, after the execution of the process P_2, the number of available resources is 5, and is allocated to the process P_1. Even after the allocation of all the available resources, the process P_1 is still short of one resource for its complete execution. As a result, the process P_1 enters a waiting state and waits for the process P_3 to release the resource held by it, which in turn is waiting for the remaining resources to be allocated for its complete execution. Now the processes P_1 and P_3 are waiting for each other to release the resources, leading to a deadlock. Hence, this sequence of process execution is unsafe.

Note that a safe state is a deadlock-free state, whereas all unsafe states may or may not result in a deadlock. That is, an unsafe state may lead to a deadlock but not always.

Resource Allocation Graph Algorithm

As discussed earlier, resource allocation graph consists of two types of edges: request edge and assignment edge. In addition to these edges, another edge known as **claim edge** can also be introduced in this graph, which helps in avoiding the deadlock. A claim edge from a process to the resource indicates that the process will request for that resource in near future. This edge is represented to be same as that of request edge but with dotted line. Whenever the process actually requests for that resource, the claim edge is converted to the request edge. Also, whenever a resource is released by any process, corresponding assignment edge is converted back to the claim edge. The pre-requisite of this representation is that all the claim edges related to a process must be depicted in the graph before the process starts executing. However, a claim edge can be added at the later stage only if all the edges related to that process are claim edges.

Whenever, the process requests for a resource, the claim edge is converted to request edge only if converting the corresponding request edge to assignment edge does not lead to the formation of a cycle in a graph, as cycle in a graph indicates the deadlock. For example, consider the resource allocation graph shown in Figure 8.3, the claim edge from process P_1 to the resource R_1 cannot be converted to the request edge as it will lead to the formation of cycle in the graph.

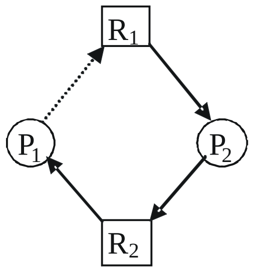


Fig. 8.3 *Resource Allocation Graph with Claim Edges*

Banker's Algorithm

In case there are multiple instances of a resource type in a system, the deadlock cannot be avoided using resource allocation graph algorithm. This is because the presence of cycle in the resource allocation graph for multiple resources does not always imply the deadlock. In such cases, an algorithm known as **Banker's algorithm** is used. In this algorithm, any process entering the system must inform the maximum number of resources (less than the total number of available resources) required during its execution. If allocating this much number of resources to the process leaves the system in a safe state only than the resources are allocated. On the other hand, if allocation of resources leaves the system in unsafe state then the resources are not allocated and the process is made to wait for some other processes to release enough resources. To implement the banker's algorithm certain data structures are required, which help in determining whether the system is in safe state or not. These data structures are as follows:

1. **Available Resources, `A`:** A vector of size `q` stores information about the number of resources available of each type.
2. **Maximum, `M`:** A matrix of order `pxq` stores information about the maximum number of resources of each type required by each process (`p` number of processes). That is, `M[i][j]` indicates the maximum number of resources of type `j` required by the process `i`.

3. **Current Allocation, C:** A matrix of order pxq stores information about the number of resources of each type allocated to each process. That is, C[i][j] indicates the number of resources of type j currently held by the process i.
4. **Required, R:** A matrix of order pxq stores information about the remaining number of resources of each type required by each process. That is, R[i][j] indicates the remaining number of resources of type j required by the process i. Note that this vector can be obtained by M–C, that is, R[i][j]=M[i][j]-C[i][j].

The values of these data structures keep on changing during the execution of processes. Note that the condition A<=B holds for the vectors A and B of size p, if and only if A[i] <=B[i] for all i=1, 2, 3, ..., p. For example, if A={2,1} and B={3,4}, then A<=B.

Safety Algorithm

This algorithm us used to determine whether a system is in safe state.

To understand the algorithm for determining whether a system is in safe state, consider a vector Complete of size p. Following are the steps of the algorithm.

1. Initialize Complete[i]=False for all i=1, 2, 3, ..., p. Complete[i]=False indicates that the ith process is still not completed.
2. Search for an i, such that Complete[i]=False and (R<=A) that is, resources required by this process is less than the available resources. If no such process exists, then go to Step 4.
3. Allocate the required resources and let the process finish its execution and set Complete[i]=True for that process. Go to Step 2.
4. If Complete[i]=True for all i, then the system is in safe state. Otherwise, it indicates that there exist a process for which Complete[i]=False and resources required by it are more than the available resources. Hence, it is in unending waiting state leading to an unsafe state.

Resource-Request Algorithm

Once it is confirmed that system is in safe state, an algorithm called **resource-request** algorithm is used for determining whether the request by a process can be satisfied or not. To understand this algorithm, let Req be a matrix of the order pxq, indicating the number of resources of each type requested by each process at any given point of time. That is, Req[i][j] indicates the number of resources of jth type requested by the ith process at any given point of time. Following are the steps of this algorithm:

1. If `Req[i][j]<= R[i][j]`, go to Step 2, otherwise an error occurs as process is requesting for more resources than the maximum number of resources required by it.
2. If `Req[i][j] <=A[i][j],` go to Step 3, otherwise the process `Pi` must wait until the required resources are available.
3. Allocate the resources are allocated and make the following changes in the data structures.

   ```
   A = A – Req
   C = C + Req
   R = R - Req
   ```

For example, consider a system with three processes (`P1`, `P2` and `P3`) and three resource types (`X, Y` and `Z`). There are 10 instances of resource type `X`, 5 of `Y` and 7 of `Z`. The matrix `M` for maximum number of resources required by the process, matrix `C` for the number of resources currently allocated to each process and vector A for maximum available resources are shown in Figure 8.4.

$$\text{M}\quad \begin{array}{c|c|c|c|} & X & Y & Z \\ \hline P_1 & 7 & 5 & 6 \\ \hline P_2 & 5 & 2 & 2 \\ \hline P_3 & 9 & 0 & 2 \\ \hline \end{array} \qquad \text{C}\quad \begin{array}{c|c|c|c|} & X & Y & Z \\ \hline P_1 & 0 & 1 & 0 \\ \hline P_2 & 2 & 0 & 0 \\ \hline P_3 & 3 & 0 & 2 \\ \hline \end{array}$$

(a) Maximum Matrix *(b) Current Allocation Matrix*

$$\text{A}\quad \begin{array}{|c|c|c|} X & Y & Z \\ \hline 5 & 4 & 5 \\ \hline \end{array} \qquad \text{R}\quad \begin{array}{c|c|c|c|} & X & Y & Z \\ \hline P_1 & 7 & 4 & 6 \\ \hline P_2 & 3 & 2 & 2 \\ \hline P_3 & 6 & 0 & 0 \\ \hline \end{array}$$

(c) Available Resources vector *(d) Required Matrix*

Fig. 8.4 *Initial State of System*

Now, the matrix `R` representing the number of remaining resources required by each process can be obtained by the formula `M–C`, which is shown in Figure 8.4.

It can be observed that currently the system is in safe state and safe sequence of execution of processes is (`P2`, `P3`, `P1`). Now suppose that process `P2` requests one more resource of each type, that is, the request vector for process `P2` is (1,1,1). First, it is checked whether this request vector is less than or equal to its corresponding required vector (3,2,2). If the process has requested for less number of resources than the declared maximum number of resources of each type by it at initial stage, then it is checked whether these much number of resources of each type are available. If it is then it is assumed that the request is granted, and the changes will be made in the corresponding matrices shown in Figure 8.5.

NOTES

$$\text{M}\quad\begin{array}{c|c|c|c|}
 & X & Y & Z \\ \hline
P_1 & 7 & 5 & 6 \\ \hline
P_2 & 5 & 2 & 2 \\ \hline
P_3 & 9 & 0 & 2 \\ \hline
\end{array}
\qquad
\text{C}\quad\begin{array}{c|c|c|c|}
 & X & Y & Z \\ \hline
P_1 & 0 & 1 & 0 \\ \hline
P_2 & 3 & 1 & 1 \\ \hline
P_3 & 3 & 0 & 2 \\ \hline
\end{array}$$

(a) Maximum Matrix *(b) Current Allocation Matrix*

$$\text{A}\quad\begin{array}{|c|c|c|}
X & Y & Z \\ \hline
4 & 3 & 4 \\ \hline
\end{array}
\qquad
\text{R}\quad\begin{array}{c|c|c|c|}
 & X & Y & Z \\ \hline
P_1 & 7 & 4 & 6 \\ \hline
P_2 & 2 & 1 & 1 \\ \hline
P_3 & 6 & 0 & 0 \\ \hline
\end{array}$$

(c) Available Resources vector *(d) Required Matrix*

Fig. 8.5 *State after Granting Request of P2*

This new state of system must be checked whether it is safe. For this, an algorithm is executed to check the safe state of the system and it is determined that the sequence (P2, P3, P1) is a safe sequence. Thus, the request of process P2 is granted immediately.

Consider another state of a system shown in Figure 8.6. Now, a request for (1, 2, 2) from process P2 arrives. If this request is granted, the resulting state is unsafe. This is because after the complete execution of process P2, the resultant vector A is (5,4,5). Clearly, the resource requirement of processes P1 and P3 cannot be satisfied. Thus, even though the system has resources, request cannot be granted.

$$\text{M}\quad\begin{array}{c|c|c|c|}
 & X & Y & Z \\ \hline
P_1 & 7 & 5 & 6 \\ \hline
P_2 & 5 & 2 & 2 \\ \hline
P_3 & 9 & 0 & 2 \\ \hline
\end{array}
\qquad
\text{C}\quad\begin{array}{c|c|c|c|}
 & X & Y & Z \\ \hline
P_1 & 2 & 1 & 0 \\ \hline
P_2 & 4 & 0 & 0 \\ \hline
P_3 & 3 & 0 & 2 \\ \hline
\end{array}$$

(a) Maximum Matrix *(b) Current Allocation Matrix*

$$\text{A}\quad\begin{array}{|c|c|c|}
X & Y & Z \\ \hline
1 & 4 & 5 \\ \hline
\end{array}
\qquad
\text{R}\quad\begin{array}{c|c|c|c|}
 & X & Y & Z \\ \hline
P_1 & 5 & 4 & 6 \\ \hline
P_2 & 1 & 2 & 2 \\ \hline
P_3 & 6 & 0 & 0 \\ \hline
\end{array}$$

Fig. 8.6 *Example of Unsafe State*

Check Your Progress

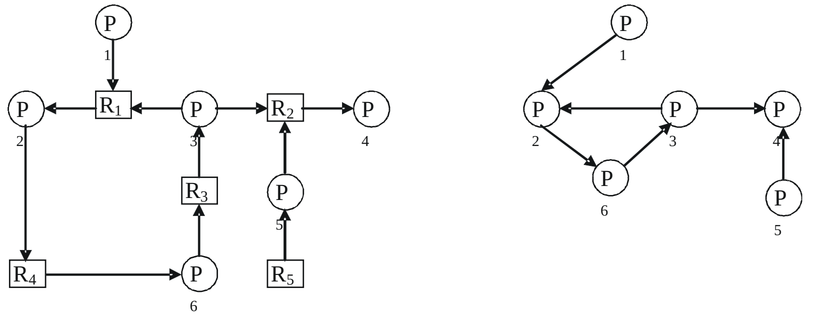
1. State the process of eliminating mutual exclusion.
2. Define safe sequence.
3. Mention different ways by which a system can recover from a deadlock.
4. What do you mean by claim edge in resource allocation graph algorithm?
5. How do we define Banker's algorithm of resources?
6. Write steps to process the resource-request algorithm?

8.4 DEADLOCK DETECTION

There is a possibility of deadlock if neither the deadlock prevention nor deadlock avoidance method is applied in a system. In such a situation, an algorithm must be provided for detecting the occurrence of deadlock in a system. Once the deadlock is detected, a methodology must be provided for the recovery of the system from the deadlock. In this section, we will discuss some of the ways by which deadlock can be detected.

Single Instance of Each Resource Type

When only single resource of each type is available, the deadlock can be detected by using variation of resource allocation graph. In this variation, the nodes representing resources and corresponding edges are removed. This new variation of resource allocation graph is known as **wait-for graph**, which shows the dependency of a process on another process for the resource allocation. For example, an edge from the process `Pi` to `Pj` indicates that the process `Pi` is waiting for the process `Pj` to release the resources required by it. If there are two edges `Pn`→`Ri` and `Ri`→`Pm` in resource allocation graph, then the corresponding edge in the wait-for graph will be `Pn`→`Pm` indicating that the process `Pn` is waiting for the process `Pm` for the release of the resources. A resource allocation graph involving 6 processes and 5 resources is shown in Figure 8.7(a). The corresponding wait-for graph is shown in Figure 8.7(b).



(a) Resource Allocation Graph *(b) Wait-for Graph*

Fig. 8.7 *Converting Resource Allocation Graph to Wait-For Graph*

If there exists a cycle in wait-for graph, there is a deadlock in the system, and the processes forming the part of cycle are blocked in the deadlock. In wait-for graph (Refer Figure 8.6), the processes `P2`, `P3` and `P6` form the cycle and hence are blocked in the deadlock. To take appropriate action to recover from this situation, an algorithm needs to be called periodically to detect existence of cycle in wait-for graph.

Multiple Instances of a Resource Type

When multiple instances of a resource type exist, the wait-for graph becomes inefficient to detect the deadlock in the system. For such system, another algorithm which uses certain data structures similar to the ones used in banker's algorithm is applied. The data structures used are as follows:

- **(i) Available Resources, A**: A vector of size `q` stores information about the number of available resources of each type.
- **(ii) Current Allocation, C**: A matrix of order `pxq` stores information about the number of resources of each type allocated to each process. That is, `C[i][j]`, indicates the number of resources of type `j` currently held by the process `i`.
- **(iii) Request, Req**: A matrix of order `pxq` stores information about the number of resources of each type currently requested by each process. That is, `R[i][j]`, indicates the number of resources of type `j` currently requested by the process `i`.

To understand the working of deadlock detection algorithm, consider a vector `Complete` of size `p.` Following are the steps to detect the deadlock.

1. Initialize `Complete[i]=False` for all `i`=1, 2, 3, ..., `p`. `Complete[i]=False` indicates that the `i`th process is still not completed.
2. Search for an `i`, such that `Complete[i]=False` and (`Req<=A`), that is, resources currently requested by this process is less than the available resources. If no such process exists, then go to Step 4.
3. Allocate the requested resources and let the process finish its execution and set `Complete[i]=True` for that process. Go to Step 2.
4. If `Complete[i]=False` for some `i`, then the system is in the state of deadlock and the `i`th process is deadlocked.

8.5 DEADLOCK RECOVERY

Once the system has detected deadlock in the system, a method is needed to recover the system from the deadlock and continue with the processing. Three different ways in which system can be recovered are—terminate one or more process to break the circular-wait condition, preempt the resources from the processes involved in the deadlock and roll back the processes to the previous checkpoint.

Terminating the Processes

There are two methods that can be used for terminating the processes to recover from the deadlock. These two methods are as follows:

- **Terminating one process at a time until the circular-wait condition is eliminated**. It involves an over head of invoking a deadlock detection algorithm after terminating each process to detect whether circular-wait condition is eliminated or not, that is, whether any processes are still deadlocked.

- **Terminating all processes involved in the deadlock**. This method will definitely ensure the recovery of a system from the deadlock. The disadvantage of this method is that many processes may have executed for a long time; close to their completion. As a result, the computations performed till the time of termination are discarded.

  In both the cases, all the resources which were acquired by the processes being terminated are returned to the system. While terminating any process, it must be ensured that it does not leave any part of the system in an inconsistent state. For example, a process might be in the middle of updating a disk file and termination of such a process may leave that file in an inconsistent state. Similarly, a printer might be in the middle of printing some document. In this case when system is recovered from the deadlock, the system must reset the printer to a correct state.

  In case of partial termination, while selecting the process to be terminated, the choice of processes must be such that it incurs minimum cost to the system. The factors which can effect the selection of a process for termination are as follows:
- Number of remaining resources required by it to complete its task.
- Number of processes required to be terminated.
- Number and type of resources held by the process.
- Duration of time for which process has already been executed.
- Priority of the process.

Preempting the Resources

An alternative method to recover system from the state of deadlock is to preempt the resources from the processes one by one and allocate them to other processes until the circular-wait condition is eliminated. The steps involved in the preemption of resources from the process are as follows:

1. **Select a Process for Preemption**: The choice of resources and processes must be such that they incur minimum cost to the system. All the factors mentioned earlier must be considered while making choice.
2. **Roll Back of the Process**: After preempting the resources, the corresponding process must be rolled backed properly so that it does not leave the system in an inconsistent state. Since resources are preempted from the process, it cannot continue with the normal execution, hence must be brought to some safe state from where it can be restarted later. In case no such safe state can be achieved, the process must be totally rolled backed. However, partial rollback is always preferred over total rollback.
3. **Prevent Starvation**: In case the selection of a process is based on the cost factor, it is quiet possible that same process is selected repeatedly for the rollback leading to the situation of starvation. This can be avoided by including the number of rollbacks of a given process in the cost factor.

Check Your Progress

7. What do you understand by wait-for-graph?
8. State the data structures involved in banker's algorithm for multiple instances of a resource type.
9. What are the two methods that can be used for terminating the processes to recover from the deadlock?
10. Write the major steps required in the preemption of resources from the process.

8.6 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. The mutual exclusion property does not hold for the resources that are sharable. For example, a file opened in read-only mode can be shared among various processes. Hence, processes will never have to wait for the sharable resources. However, there are certain resources which can never be shared, like printer can work for only one process at a time. It cannot print data being sent as output from more than one process simultaneously. Hence, the condition of mutual exclusion cannot be eliminated in case of all the resources.
2. A safe sequence is a sequence of process execution such that each and every process executes till its completion. For example, consider a sequence of processes (P_1, P_2, P_3, ..., P_n) forming a safe sequence. In this sequence, first the process P_1 will be executed till its completion, and then P_2 will be executed till its completion, and so on.
3. The three different ways in which system can be recovered are as follows, terminate one or more process to break the circular-wait condition, pre-empt the resources from the processes involved in the deadlock and roll back the processes to the previous checkpoint.
4. Resource allocation graph consists of two types of edges: request edge and assignment edge. In addition to these edges, another edge known as claim edge can also be introduced in this graph, which helps in avoiding the deadlock. A claim edge from a process to the resource indicates that the process will request for that resource in near future. This edge is represented to be same as that of request edge but with dotted line.
5. In this algorithm, any process entering the system must inform the maximum number of resources (less than the total number of available resources) required during its execution. If allocating this much number of resources to the process leaves the system in a safe state only than the resources are

allocated. On the other hand, if allocation of resources leaves the system in unsafe state then the resources are not allocated and the process is made to wait for some other processes to release enough resources. To implement the banker's algorithm certain data structures are required, which help in determining whether the system is in safe state or not.

6. Following are the steps of this algorithm:

   1. If `Req[i][j]<= R[i][j]`, go to Step 2, otherwise an error occurs as process is requesting for more resources than the maximum number of resources required by it.

   2. If `Req[i][j] <=A[i][j],` go to Step 3, otherwise the process `Pi` must wait until the required resources are available.

   3. Allocate the resources are allocated and make the following changes in the data structures.

   ```
   A = A – Req
   C = C + Req
   R = R - Req
   ```

7. When only single resource of each type is available, the deadlock can be detected by using variation of resource allocation graph. In this variation, the nodes representing resources and corresponding edges are removed. This new variation of resource allocation graph is known as wait-for graph, which shows the dependency of a process on another process for the resource allocation.

8. When multiple instances of a resource type exist, the wait-for graph becomes inefficient to detect the deadlock in the system. For such system, another algorithm which uses certain data structures similar to the ones used in banker's algorithm is applied. The data structures used are as follows:

   (i) Available Resources, A: A vector of size `q` stores information about the number of available resources of each type.

   (ii) Current Allocation, C: A matrix of order `pxq` stores information about the number of resources of each type allocated to each process. That is, `C[i][j]`, indicates the number of resources of type `j` currently held by the process `i`.

   (iii) Request, Req: A matrix of order `pxq` stores information about the number of resources of each type currently requested by each process. That is, `R[i][j]`, indicates the number of resources of type `j` currently requested by the process `i`.

9. There are two methods that can be used for terminating the processes to recover from the deadlock. These two methods are as follows:

   - Terminating one process at a time until the circular-wait condition is eliminated. It involves an over head of invoking a deadlock detection algorithm after terminating each process to detect whether circular-wait condition is eliminated or not, that is, whether any processes are still deadlocked.

- Terminating all processes involved in the deadlock. This method will definitely ensure the recovery of a system from the deadlock. The disadvantage of this method is that many processes may have executed for a long time; close to their completion.

10. The steps involved in the preemption of resources from the process are as follows:
    1. Select a Process for Preemption: The choice of resources and processes must be such that they incur minimum cost to the system. All the factors mentioned earlier must be considered while making choice.
    2. Roll Back of the Process: After preempting the resources, the corresponding process must be rolled backed properly so that it does not leave the system in an inconsistent state. Since resources are preempted from the process, it cannot continue with the normal execution, hence must be brought to some safe state from where it can be restarted later. In case no such safe state can be achieved, the process must be totally rolled backed. However, partial rollback is always preferred over total rollback.
    3. Prevent Starvation: In case the selection of a process is based on the cost factor, it is quiet possible that same process is selected repeatedly for the rollback leading to the situation of starvation. This can be avoided by including the number of rollbacks of a given process in the cost factor.

8.7 SUMMARY

- A deadlock can be depicted with the help of a directed graph known as resource allocation graph.
- In case there are multiple instances of a resource type in a system, the deadlock cannot be avoided using resource allocation graph algorithm. An algorithm known as banker's algorithm is used in such a case.
- If each resource type has exactly one instance, cycle in resource allocation graph indicates a deadlock. If each resource type has several instances, cycle in resource allocation graph does not necessarily imply a deadlock.
- Deadlock prevention or deadlock avoidance techniques can be used to ensure that deadlocks never occur in a system.
- A deadlock can be prevented by not allowing all four conditions to be satisfied simultaneously, that is, by making sure that at least one of the four conditions does not hold.
- A deadlock can be avoided by never allowing allocation of a resource to a process if it leads to a deadlock. This can be achieved when some additional information is available about how the processes are going to request for resources in future.

- A state is said to be safe if allocation of resources to processes does not lead to the deadlock. More precisely, a system is in safe state only if there is a safe sequence. A safe sequence is a sequence of process execution such that each and every process executes till its completion. If no such sequence of process execution exists then the state of the system is said to be unsafe.
- There is a possibility of deadlock if neither deadlock prevention nor deadlock avoidance method is applied in a system. In such a situation, an algorithm must be provided for detecting the occurrence of deadlock in a system.
- When only single resource of each type is available, the deadlock can be detected by using variation of resource allocation graph known as wait-for graph.
- When multiple instances of a resource type exist, the wait-for graph becomes inefficient in detecting the deadlock in the system. For such system, another algorithm which uses certain data structures similar to the ones used in banker's algorithm is applied.
- Once the deadlock is detected, a methodology must be provided for the recovery of the system from the deadlock.
- The three different ways in which system can be recovered are—terminate one or more process to break the circular-wait condition, preempt the resources from the processes involved in the deadlock and roll back the processes to the previous checkpoint.

8.8 KEY WORDS

- **Deadlock**: It occurs when every process in a set of processes is in a simultaneous wait state and each of them is waiting for the release of a resource held exclusively by one of the waiting processes in the set
- **Safe state:** A state in which the allocation of resources to processes does not lead to a deadlock; a system is in safe state only if there is a safe sequence.
- **Safe sequence:** A sequence of process execution such that each and every process executes till its completion. If no such sequence of process execution exists then the state of the system is said to be unsafe

8.9 SELF-ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. State the process of eliminating circular-wait condition.
2. What is meant by safe sequence in resource allocation?

3. What do you understand by the terms ‘safe state’ and ‘unsafe state’?
4. State the data structures involved in banker’s algorithm.
5. What is the significance of resource-request algorithm?
6. Briefly explain the initial state of system along with the suitable examples.
7. What are required steps to detect the deadlock?
8. Define the factors which effect the selection of a process for termination.
9. How do we define the method of pre-empting resources?

Long-Answer Questions

1. Explain the four conditions necessary for the deadlock.
2. Consider a system having three instances of a resource type and two processes. Each process needs two resources to complete its execution. Can deadlock occur? Explain in brief.
3. Consider a system is in an unsafe state. Illustrate how the processes can complete their execution without entering a deadlock state.
4. Consider a system has six instances of a resource type and m processes. For which values of m, deadlock will not occur?
5. Consider a system consisting of four processes and a single resource. The current state of the system is given here.

Maximum matrix	Current allocation matrix
3	1
2	1
9	3
7	2

For this state to be safe, what should be the minimum number of instances of this resource?

6. Consider the following state of a system.

M

	X	Y	Z
P_1	5	4	3
P_2	3	0	6
P_3	7	5	1

C

	X	Y	Z
P_1	3	1	1
P_2	1	0	4
P_3	3	2	0

A

X	Y	Z
3	2	3

Answer the following questions using the Banker's algorithm:

(i) What is the content of the matrix Required?

(ii) Is the system in a safe state?

(iii) If a request from a process P2 arrives for (1,0,2) can the request be granted immediately.

8.10 FURTHER READINGS

Silberschatz, Abraham, Peter B. Galvin and Greg Gagne. 2008. *Operating System Concepts*, 8th Edition. New Jersey: John Wiley & Sons.

Tanenbaum, Andrew S. 2006. *Operating Systems Design and Implementation*, 3rd Edition. New Jersey: Prentice Hall.

Tanenbaum, Andrew S. 2001. *Modern Operating Systems*. New Jersey: Prentice Hall.

Deitel, Harvey M. 1984. *An Introduction to Operating Systems*. Boston (US): Addison-Wesley.

Stallings, William. 1995. *Operating Systems*, 2nd Edition. New Jersey: Prentice Hall.

Milenkovic, Milan. 1992. *Operating Systems: Concepts and Design*. New York: McGraw Hill Higher Education.

Mano, M. Morris. 1993. *Computer System Architecture*. New Jersey: Prentice Hall Inc.

BLOCK - IV
MEMORY MANAGEMENT

NOTES

UNIT 9 MEMORY MANAGEMENT STRATEGIES

Structure

9.0 INTRODUCTION

Most systems allow multiple processes to reside in the main memory at the same time to increase CPU utilization. It is the job of the memory manager, a part of the operating system, to manage memory between these processes in an efficient way. For this, it keeps track of which part of the memory is occupied and which part is free, allocates and deallocates memory to processes, whenever required, and so on. Moreover, it provides a protection mechanism to protect the memory allocated to each process from being accessed by other processes. For managing the memory, the memory manager may use a strategy from any number of available memory management strategies that are discussed in this unit.

All these strategies require the entire process to be in main memory before their execution. Thus, the size of the process is limited to the size of the physical memory. To overcome this limitation, a memory management scheme called **overlaying** can be used, which allows a process to execute irrespective of the size of the system's having insufficient physical memory. The programmer splits a program into smaller parts called **overlays** in such a way that no two overlays are required to be in main memory at the same time. An overlay is loaded into memory only when it is needed. Initially, overlay 0 would run. When it is completed, it would call another overlay, and so on until the process terminates. These overlays reside on the disk and are swapped in and out of memory dynamically as needed, thereby reducing the amount of memory needed by the process. Despite all these advantages, one major disadvantage of this technique is that it requires major involvement of the programmer. Moreover, splitting a program into smaller parts is quite time consuming.

This resulted in the formulation of another memory management technique known as virtual memory. Virtual memory gives the illusion that the system has a much larger memory than is actually available. The combined size of code, data and stack may exceed the amount of physical memory. Virtual memory frees programs from the constraints of physical memory limitation. The virtual memory can be implemented by demand paging or demand segmentation. Of these two ways, demand paging is commonly used as it is easier to implement.

9.1 OBJECTIVES

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

- Differentiate between logical and physical addresses
- Understand the basic concept of address binding
- Discuss the basic memory management strategies
- Explain the memory management scheme, called paging
- Understand how the memory management scheme, segmentation, is used to overcome the drawbacks of paging
- Describe a memory management scheme that is a combination of segmentation and paging (called segmentation with paging)
- Understand the process of swapping
- Understand the concept of virtual memory
- Understand how the virtual memory technique works by using demand paging

- Explain the copy-on-write process
- Discuss the various page replacement algorithms

NOTES

9.2 PRELIMINARIES

Every byte in the memory has a specific address that may range from 0 to some maximum value as defined by the hardware. This address is known as **physical address**. Whenever, a program is brought into the main memory for execution, it occupies certain number of memory locations. The set of all physical addresses used by the program is known as **physical address space**. However, before a program can be executed, it must be compiled to produce the machine code. A program is compiled to run starting from some fixed address and accordingly all the variables and procedures used in the source program are assigned some specific addresses known as **logical addresses**. Thus, in machine code, all references to data or code are made by specifying the logical addresses and not by the variable or procedure names, and so on. The range of addresses that user programs can use is system-defined and the set of all logical addresses used by a user program is known as its **logical address space**.

When a user program is brought into main memory for execution, its logical addresses must be mapped to physical addresses. This mapping from addresses associated with a program to memory addresses is known as **address binding**. The address binding can take place at one of the following times:

- **Compile Time**: Address binding takes place at compile time if it is known which addresses the program will occupy in the main memory at that time. In this case, the program generates **absolute code** at the compile time only, that is, logical addresses are same as that of physical addresses.
- **Load Time**: Address binding occurs at load time if it is not known at compile time which addresses the program will occupy in the main memory. In this case, the program generates **relocatable code** at the compile time which is then converted into the absolute code at the load time.
- **Run Time**: The address binding occurs at run time if the process is supposed to move from one memory segment to other during its execution. In this case also, the program generates relocatable code at compile time which is then converted into the absolute code at the run time.

Note*: The run time address binding is performed by the hardware device known as* ***Memory-Management Unit*** *(****MMU****).*

9.3 MEMORY MANAGEMENT STRATEGIES

To improve utilization of the CPU and the speed of the computer's response to its users, the system keeps several processes in memory, that is, several processes share memory. Due to the sharing of memory, there is need of memory management.

There are various strategies that are used to manage memory. All these strategies allocate memory to the processes using either of following two approaches.

- Contiguous memory allocation.
- Non-contiguous memory allocation.

Both these approaches are discussed in detail in the subsequent sections.

9.4 CONTIGUOUS MEMORY ALLOCATION

In contiguous memory allocation, each process is allocated a single contiguous part of the memory. The different memory management schemes that are based on this approach are *single partition* and *multiple partitions*.

9.4.1 Single Partition

One of the simplest ways to manage memory is to partition the main memory into two parts. One of them is permanently allocated to an operating system while the other part is allocated to the user process (Refer Figure 9.1). In this figure, an operating system is in lower part of the memory. However, it is not essential that operating system must reside at the bottom of the memory; it can reside at the upper part of the memory also. In order to provide a contiguous area for the user process, it usually resides at one extreme end of the memory. The factor that decides the location of the operating system in the memory is the location of the interrupt vector. An operating system is placed at the same end of the memory where the interrupt vector is located.

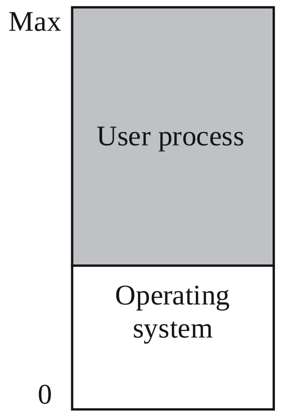


Fig. 9.1 *Memory Having Single Partition*

In this scheme, only one process can be executed at a time. Whenever, a process is to be executed, the operating system loads it into the main memory for execution. After termination of that process, the operating system waits for another process. When another process arrives, the operating system loads it into the main memory, thus overwriting the first one.

This scheme is easy to implement. Generally, the operating system needs to keep track of the first and the last location allocated to the user processes. However, in this case, the first location is immediately following the operating system and the last location is determined by the capacity of the memory. It needs no hardware support except for protecting the operating system from the user process.

***Note**: The memory management scheme having single partition is used by single process microcomputer operating systems, such as CP/M and PC DOS.*

9.4.2 Multiple Partitions

A single partition scheme restricts the system to have only one process in memory at a time that reduces utilization of the CPU as well as of memory. Thus, monoprogramming systems are rarely used. Most of the systems used today support multiprogramming which allows multiple processes to reside in the memory at the same time. The simple way to achieve multiprogramming is to divide main memory into a number of partitions which may be of fixed or variable size.

Fixed Partitions

In this technique, each partition is of fixed size and can contain only one process. There are two alternatives for this technique—equal-sized partition and unequal-sized partition (Refer Figure 9.2). First, consider the case of equal-sized partitions where any process can be loaded into any partition. Whenever, a partition is free, a process whose size is less than or equal to the partition size is selected from the input queue and loaded into this partition. When the process terminates, the partition becomes free to be allocated to another process. The implementation of this method does not require much effort since all partitions are of same size. The operating system is required to keep track of only the partition occupied by each process. For this, it maintains a table that keeps either the starting address of each process or the partition number occupied by each process.

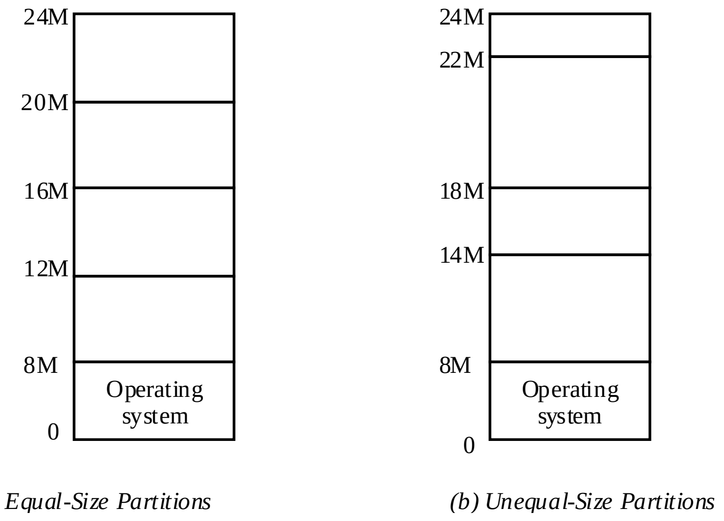


***Fig. 9.2** Memory having Multiple Fixed Partitions*

NOTES

There is one problem with this method that is the memory utilization is not efficient. Any process regardless of how small it is, occupies an entire partition which leads to the wastage of memory within the partition. This phenomenon which results in the wastage of memory within the partition is called **internal fragmentation**. For example, loading a process of size 4M-*n* bytes into a partition of size 4M (where, M stands for Megabytes) would result in a wasted space of *n* bytes within the partition.

This problem cannot be resolved completely but can be reduced to some extent by using unequal-sized partition method where a separate input queue is maintained for each partition. Whenever a process arrives, it is placed into the input queue of the smallest partition large enough to hold it. When this partition becomes free, it is allocated to the process. According to Figure 9.2(b), if a process of size 5M arrives, it will be accommodated in the partition of size 6M. In this case also, some memory is wasted, that is, internal fragmentation still exists, but less than that of equal-sized partition method.

With this method, there may be possibility that the input queue for a large partition is empty but the queue for small partition is full (Refer Figure 9.3(a)). That is, the small jobs have to wait to be loaded into memory, though a large amount of memory is free. To prevent this, a single input queue can be maintained (Refer Figure 9.3(b)). Whenever, a partition becomes free, the process that fits in it can be chosen from the input queue using some scheduling algorithm.

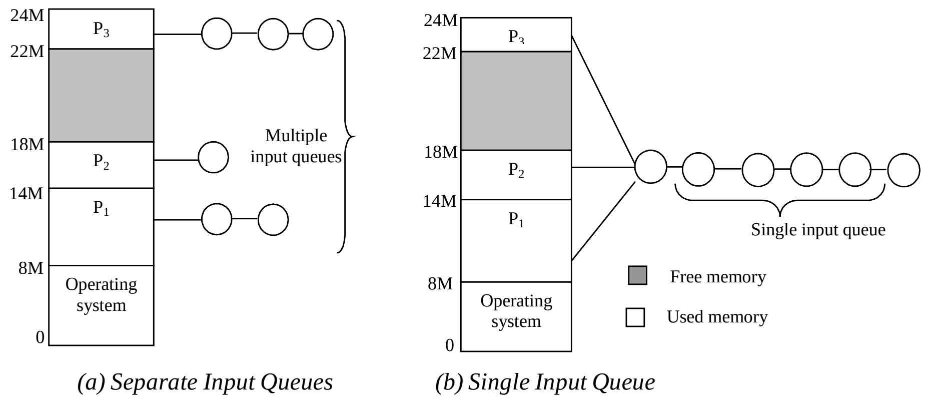


Fig. 9.3 *Memory Allocation in Fixed Partitions*

The fixed-partitioning technique is easy to implement and requires less overhead but has some disadvantages which are as follows:

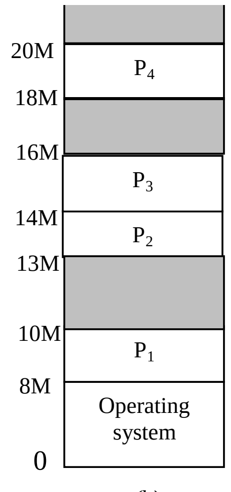
- The number of processes in memory depends on the number of partitions. Thus, the degree of multiprogramming is limited.
- The memory cannot be used efficiently in case of small-sized processes.

Note: *The technique having fixed-partitions is no longer in use.*

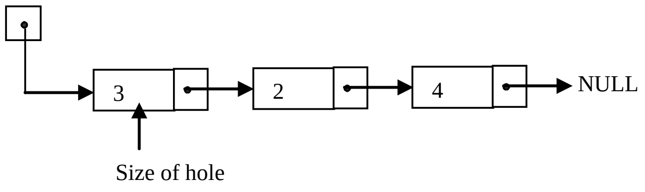
Variable Partitions

To overcome the disadvantages of fixed-partitions technique, a technique called Multiprogramming with a Variable number of Tasks (MVT) is used. It is the generalization of the fixed partitions technique in which the partitions can vary in number and size. In this technique, the amount of memory allocated is exactly the amount of memory a process requires. To implement this, the table maintained by the operating system stores both the starting address and ending address of each process.

Initially, when there is no process in the memory, the whole memory is available for allocation and is considered as a single large partition of available memory (a **hole**). Whenever a process requests for the memory, the hole large enough to accommodate that process is allocated. The rest of the memory is available to other processes. As soon as the process terminates, the memory occupied by it is deallocated and can be used for other processes. Thus, at a given point of time, some parts of memory may be in use while others may be free (Refer Figure 9.4(a)). Now to make further allocations, the memory manager must keep track of the free space in memory. For this, the memory manager maintains a free-storage list that keeps track of the unused part (holes of variable sizes) of memory. The free-storage list is implemented as a linked list where each node contains the size of the hole and the address of the next available hole (Refer Figure 9.4 (b)).



(b)



(b)

Fig. 9.4 *Memory Map and Free-Storage List*

In general, at a certain point of time, there will be a set of holes of various sizes dispersed in the memory. As a result, there may be possibility that the total available memory is large enough to accommodate the waiting process. However, it cannot be utilized as it is scattered. This wastage of the memory space is called **external fragmentation** (also known as **checkerboarding**) since, the wasted memory is not a part of any partition. For example, if a request for a partition of size 5M arrives, it cannot be granted because no single partition is available that is large enough to satisfy the request (Refer Figure 9.4). However, the combined free space can definitely satisfy the request.

To get rid of this problem, it is desirable to relocate (or shuffle) some or all portions of the memory in order to place all the free holes together at one end of memory to make one large hole. This technique of reforming the storage is termed as **compaction**. Compaction results in the memory partitioned into two contiguous blocks—one of used memory and another of free memory. Figure 9.5 shows the memory map after performing compaction. Compaction may take place at the moment any node frees some memory or when a request for allocating memory fails provided the combined free space is enough to satisfy the request. Since it is expensive in terms of CPU time, it is rarely used.

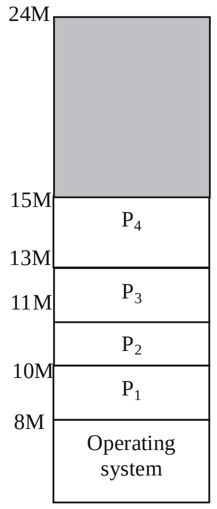


Fig. 9.5 *Memory After Compaction*

Partition Selection Algorithms

Whenever a process arrives and there are various holes large enough to accommodate it, the operating system may use one of the following algorithms to select a partition for the process.

- **First Fit**: In this algorithm, the operating system scans the free-storage list and allocates the first hole that is large enough to accommodate that process. This algorithm is fast because search is little as compared to other algorithms.
- **Best Fit**: In this algorithm, the operating system scans the free-storage list and allocates the smallest hole whose size is larger than or equal to the size of the process. Unlike first fit algorithm, it allocates a partition that is close to the size required for that process. It is slower than the first fit algorithm as it has to search the entire list every time. Moreover, it leads to more wastage of memory as it results in the smallest leftover holes that cannot be used to satisfy the memory allocation request.

NOTES

- **Worst Fit**: In this algorithm, the operating system scans the free-storage list and allocates the largest hole to that process. Unlike best fit, this algorithm results in the largest leftover holes. However, simulation indicates that worst fit allocation is not very effective in reducing the wastage of memory.

***Note**: First fit and best fit are among the most popular algorithms for dynamic memory allocation.*

Example 9.1: Consider the memory map given in Figure 9.4, how would each of the first fit, best fit and worst fit algorithms allocate memory to a process P of size 2M.

Solution: According to the different algorithms, memory will be allocated to the process P as shown in Figure 9.6.

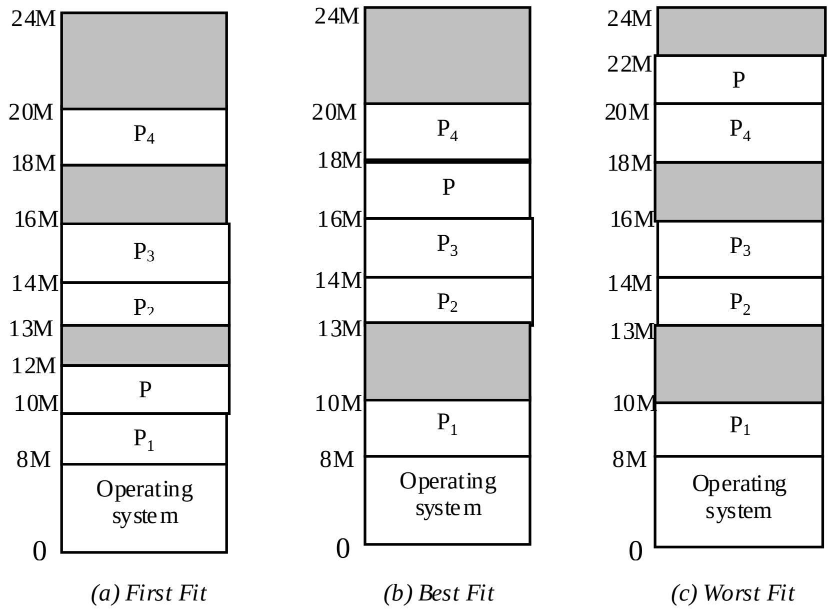


***Fig. 9.6** Memory Allocation Using Different Algorithms*

9.4.3 Relocation and Protection

In multiprogramming environment, multiple processes are executed due to which two problems can arise which are relocation and protection.

Relocation

From the earlier discussion, it is clear that different processes run at different partitions. Now, suppose a process contains an instruction to call a procedure at absolute address 5. If this process is loaded into a partition at address 10M, this instruction will jump to absolute address 5 which is inside the operating system. Instead, it is required to call memory address (10M+5). Similarly, if the process is loaded into some other partition, say at address 20M, then it should call at memory address 20M+5. This problem is known as **relocation problem**.

This relocation problem can be solved by equipping the system with a hardware register called **relocation register** which contains the starting address of the partition into which the process is to be loaded. Whenever, an address is generated during the execution of a process, the memory management unit adds the content of the relocation register to the address resulting in physical memory address.

Example 9.2: Consider the logical address of an instruction in a program is 7632 and the content of relocation register is 2500. To which location in the memory will this address be mapped?

Solution: Here, Logical address = 7632

Content of relocation register = 2500

Since, Physical address = Logical address + Content of relocation register

Physical address = 7632 + 2500 = 10,132

Thus, the logical address 7632 will be mapped to the location 10,132 in memory.

Protection

Using relocation register, the problem of relocation can be solved but there is a possibility that a user process may access the memory address of other processes or the operating system. To protect the operating system from being accessed by other processes and the processes from one another, another hardware register called **limit register** is used. This register holds the range of logical addresses. Each logical address of a program is checked against this register to ensure that it does not attempt to access the memory address outside the allocated partition. Figure 9.7 shows relocation and protection mechanism using relocation and limit register.

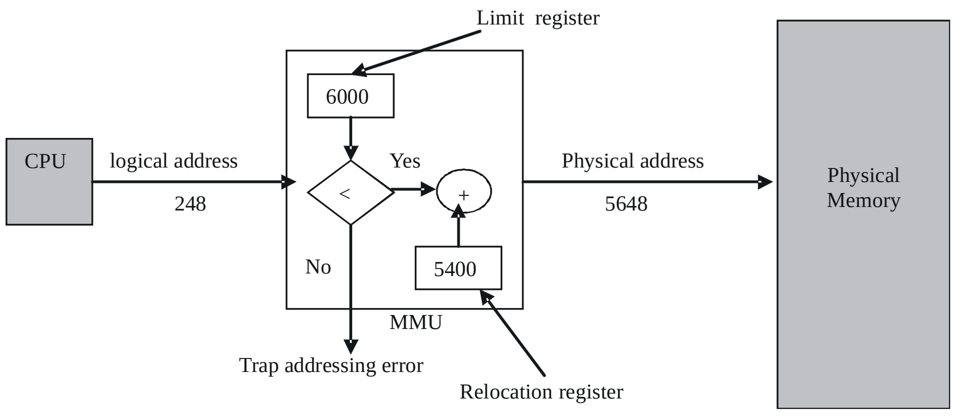


***Fig. 9.7** Relocation and Protection using Relocation and Limit Register*

Check Your Progress

1. Differentiate between physical and logical address space.
2. How do you improve the usage and the speed of the computer's response?
3. What do you understand by internal fragmentation?
4. Define checker boarding.
5. How do we define different partition search algorithms?

NOTES

9.5 NONCONTIGUOUS MEMORY ALLOCATION

In noncontiguous allocation approach, parts of a single process can occupy noncontiguous physical addresses. In this section, we will discuss memory management schemes based on noncontiguous allocation.

9.5.1 Paging

In paging, the physical memory is divided into fixed-sized blocks called **page frames** and logical memory is also divided into fixed-size blocks called **pages** which are of same size as that of page frames. When a process is to be executed, its pages can be loaded into any unallocated frames (not necessarily contiguous) from the disk. Figure 9.8 shows two processes A and B with all their pages loaded into the memory. In this figure, the page size is of 4KB. However, some systems support even larger page sizes such as 8KB, 4MB, and so on. Nowadays, the pages sizes between 4KB and 8KB are used.

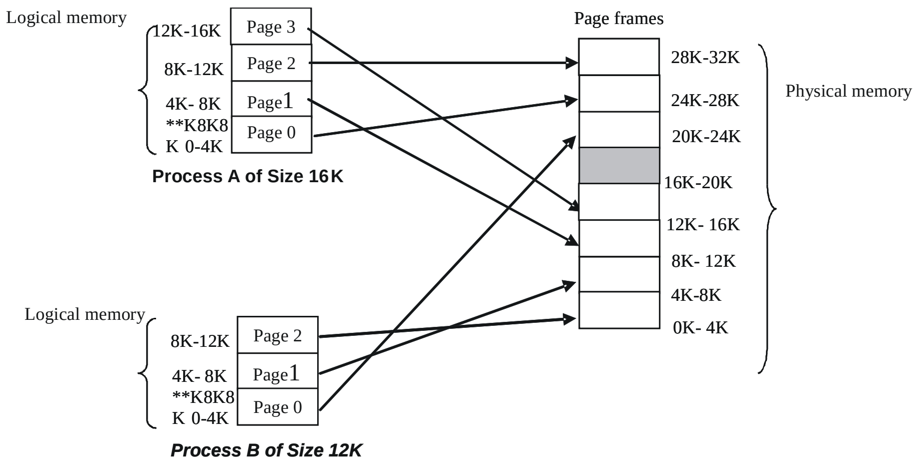


Fig. 9.8 *Concept of Paging*

***Note**: In real systems, the page size can vary from 512 bytes to 64 KB.*

NOTES

When the CPU generates a logical address, it is divided into two parts: A page number (p) [high-order bits] and A page offset (d) [low-order bits] where d specifies the address of the instruction within the page p. Since the logical address is a power of 2, the page size is always chosen as a power of 2 so that the logical address can be converted easily into page number and page offset. To understand this, consider the size of logical address space is 2^m. Now, if we choose a page size of 2^n (bytes or words), then *n* bits will specify the page offset and *m n* bits will specify the page number.

Example 9.3: Consider a system that generates logical address of 16 bits and page size is 4KB. How many bits would specify the page number and page offset?

Solution: Here, the logical address is of 16 bits, that is, the size of logical address space is 2^{16},

Page size is 4KB, that is, 4×1024 bytes = 2^{12} bytes

Thus, the page offset will be of 12 bits and page number will be of (16 12) = 4 bits.

Now let us see how a logical address is translated into a physical address. In paging, address translation is performed using a mapping table, called **page table**. The operating system maintains a page table for each process to keep track of which page frame is allocated to which page. It stores the frame number allocated to each page and the page number is used as index to the page table (Refer Figure 9.9).

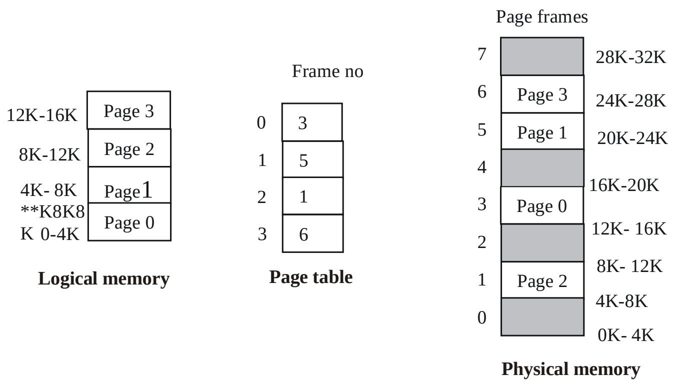


Fig. 9.9 *Page Table*

When CPU generates a logical address, that address is sent to MMU. The MMU uses the page number to find the corresponding page frame number in the page table. That page frame number is attached to the high-order end of the page offset to form the physical address that is sent to the memory. The mechanism of translation of logical address into physical address is shown in Figure 9.10.

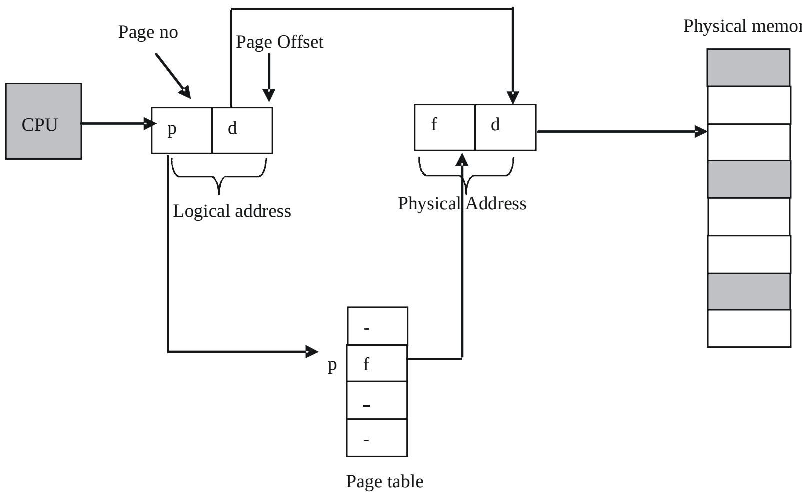


Fig. 9.10 *Address Translation in Paging*

NOTES

Note*: Since both the page and page frames are of same size, the offset within them are identical, and need not be mapped.*

Example 9.4: Consider a paged memory system with eight pages of 8KB page size each and 16 page frames in memory. Using the following page table, compute the physical address for the logical address 18325.

7	1010
6	0100
5	0000
4	0111
3	1101
2	1011
1	1110
0	0101

Page Table

Solution: Since, total number of pages = 8, that is, 2^3 and each page size = 8KB, that is, 2^{13} bytes, the logical address will be of 16 bits. Out of these 16 bits, the three high-end order bits represent the page number and the 13 low-end order bits represent offset within the page. In addition, there are 16, that is, 2^4 page frames in memory, thus, the physical address will be of 17 bits.

Given logical address = 18325 which is equivalent to 0100011110010101. In this address, page number = 010, that is, 2 and page offset = 0011110010101. From the page table it is clear that the page number 2 is in page frame 1011.

Therefore, the physical address = 10110011110010101, which is equivalent to 92053.

Advantages

- Since the memory allocated is always in fixed unit, any free frame can be allocated to a process. Thus, there is no external fragmentation.

Disadvantages

- There may be some internal fragmentation. To illustrate this, consider a page size is of 4KB and a process requires memory of 8195 bytes, that is 2 page + 3 bytes. In this case, for only 3 bytes, an entire frame is wasted resulting in internal fragmentation.

Hardware Implementation of Page Table

A page table can be implemented in several ways. The simplest way is to use registers to store the page table entries indexed by page number. Though this method is faster and does not require any memory reference, its disadvantage is that it is not feasible in case of large page table as registers are expensive. Moreover, at every context switch, the page table needs to be changed which in turn requires all the registers to be reloaded. This degrades the performance.

Another way is to keep the entire page table in main memory and the pointer to page table stored in a register called **Page-Table Base Register** (**PTBR**). Using this method, page table can be changed by reloading only one register, thus reduces context switch time to a great extent. The disadvantage of this scheme is that it requires two memory references to access a memory location; first, to access page table using PTBR to find the page frame number, and second, to access the desired memory location. Thus, memory accessing is slowed down by a factor of two.

To overcome this problem, the system can be equipped with a special hardware device known as **Translation Look-Aside Buffer** (**TLB**) (or **associative memory**). The TLB is inside MMU and contains a limited number of page table entries. When CPU generates a logical address and presents it to the MMU, it is compared with the page numbers present in the TLB. If a match is found in TLB (called **TLB hit**), the corresponding page frame number is used to access the physical memory. In case a match is not found in TLB (called **TLB miss**), memory is referenced for the page table. Further, this page number and the corresponding frame number are added to the TLB so that next time if this page is required, it can be referenced quickly. Since the size of TLB is limited so when it is full, one entry needs to be replaced. Figure 9.11 shows the mechanism of paging using TLB.

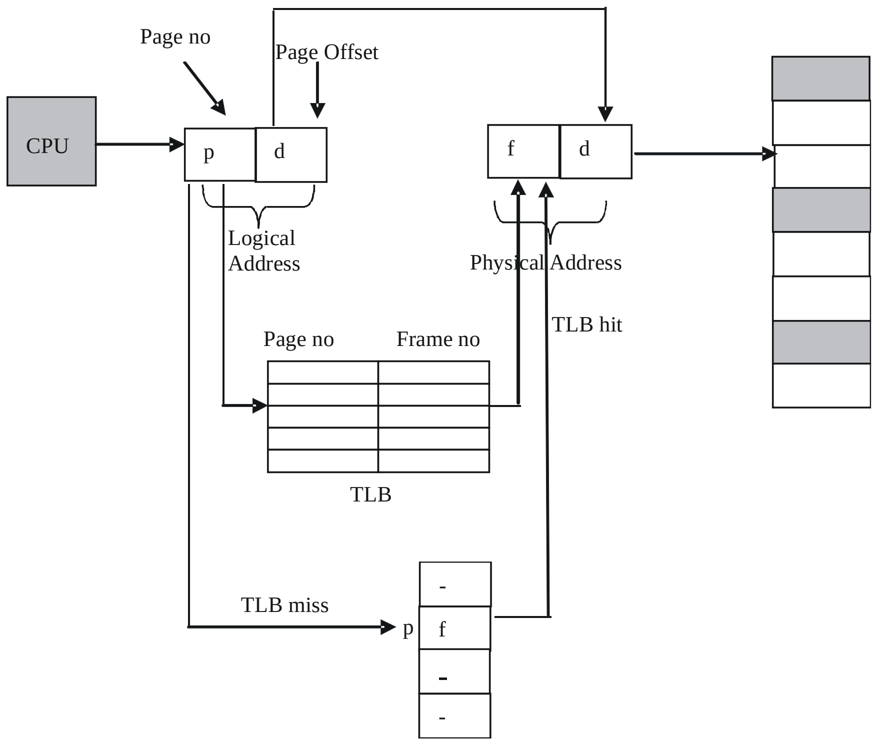


Fig. 9.11 *Paging with TLB*

NOTES

TLB can contain entries for more than one process at the same time, so there is a possibility that two processes map the same page number to different frames. To resolve this ambiguity, a Process Identifier (PID) can be added with each entry of TLB. For each memory access, the PID present in the TLB is matched with the value in a special register that holds the PID of the currently executing process. If it matches, the page number is searched to find the page frame number; otherwise it is treated as a TLB miss.

9.5.2 Segmentation

A user views a program as a collection of segments such as main program, routines, variables, and so on. All of these segments are variable in size and their size varies during execution. Each segment is identified by a name (or segment number) and the elements within a segment are identified by their offset from the starting of the segment. Figure 9.12 shows the user view of a program.

NOTES

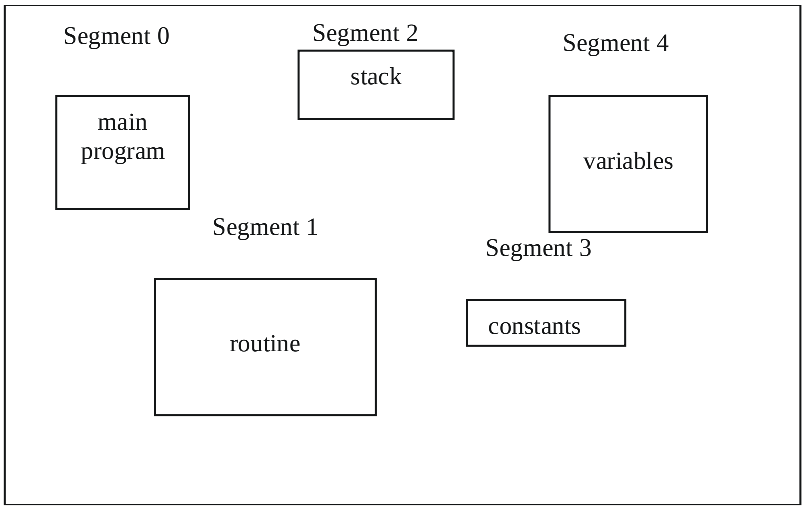
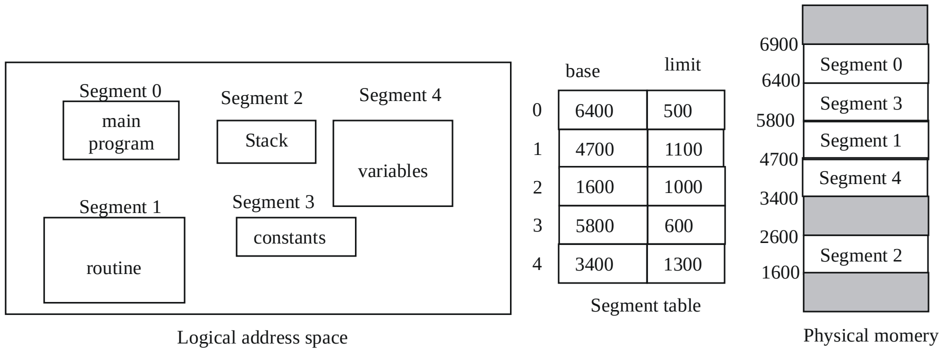


Fig. 9.12 *User View of a Program*

Segmentation is a memory management scheme that implements the user view of a program. In this scheme, the entire logical address space is considered as a collection of segments with each segment having a number and a length. The length of a segment may range from 0 to some maximum value as specified by the hardware and may also change during the execution. The user specifies each logical address consisting of a segment number (s) and an offset (d). This differentiates segmentation from paging in which the division of logical address into page number and page offset is performed by the hardware.

To keep track of each segment, a **segment table** is maintained by the operating system (Refer Figure 9.13). Each entry in the segment table consists of two fields: segment base and segment limit. The **segment base** specifies the starting address of the segment in physical memory and the **segment limit** specifies the length of the segment. The segment number is used as an index to the segment table.



	base	limit
0	6400	500
1	4700	1100
2	1600	1000
3	5800	600
4	3400	1300

Fig. 9.13 *Segment Table*

When CPU generates a logical address, that address is sent to MMU. The MMU uses the segment number of logical address as an index to the segment table. The offset is compared with the segment limit and if it is greater, invalid-address error is generated. Otherwise, the offset is added to the segment base to form the physical address that is sent to the memory. Figure 9.14 shows the hardware to translate logical address into physical address in segmentation.

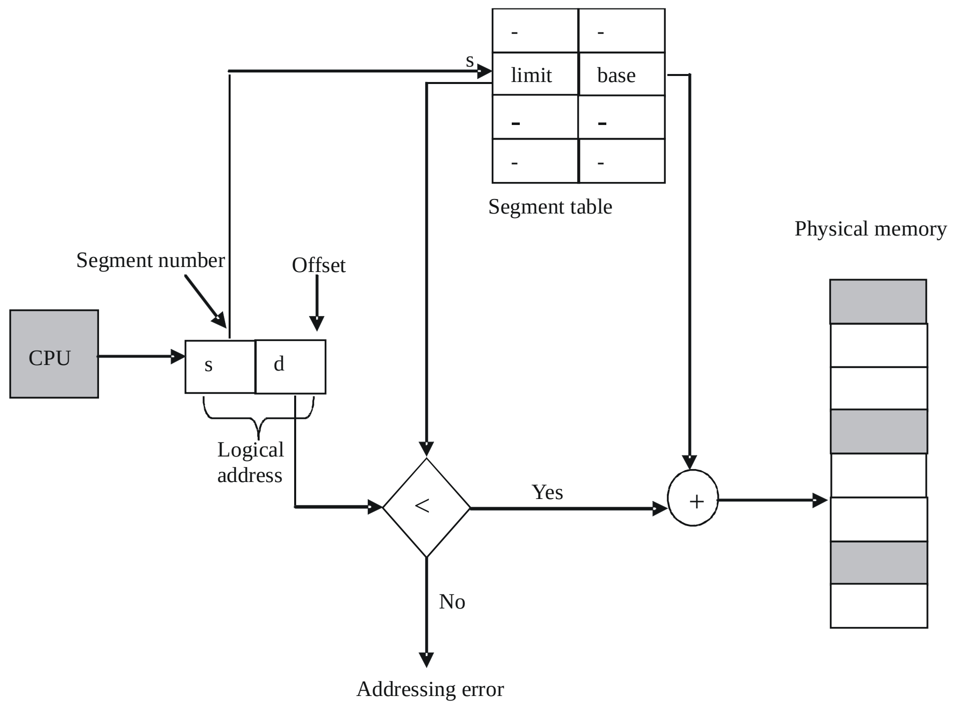


Fig. 9.14 *Segmentation Hardware*

Advantages

- Since a segment contains one type of object, each segment can have different types of protection. For example, a procedure can be specified as execute only whereas a char type array can be specified as read only.
- It allows sharing of data or code between several processes. For example, a common function or shared library can be shared between various processes. Instead of having them in address space of every process, they can be put in one segment and that segment can be shared.

NOTES

Example 9.5: Using the following segment table, compute the physical address for the logical address consisting of segment and offset as given below.

	Base	**Limit**
0	5432	350
1	115	100
2	2200	780
3	4235	1100
4	1650	400

Segment Table

(a) Segment 2 and offset 247
(b) Segment 4 and offset 439

Solution:

(a) Here, offset = 247 and segment is 2

It is clear from the segment table that limit of segment 2 = 780 and segment base = 2200

Since, the offset is less than the segment limit, physical address is computed as

physical address = offset + segment base

= 247 + 2200 = 2447

(b) Here, offset = 439 and segment is 4

It is clear from the segment table that limit of segment 4 = 400 and segment base = 1650

Since, the offset is greater than the segment limit, invalid-address error is generated.

9.5.3 Segmentation with Paging

The idea behind the segmentation with paging is to combine the advantages of both paging, (such as uniform page size) and segmentation, (such as protection and sharing) together into a single scheme. In this scheme, each segment is divided into a number of pages. To keep track of these pages, a page table is maintained for each segment. The segment offset in the logical address (comprising segment number and offset) is further divided into a page number and a page offset. Each entry of segment table contains the segment base, segment limit and one more entry that contains the address of the segment's page table.

The logical address consists of three parts: Segment number (s), Page number (p) and Page offset (d). Whenever, address translation is to be performed, firstly, the MMU uses the segment number as an index to segment table to find the address of page table. Then the page number of logical address is attached to the high-order end of the page table address and used as an index to page table to find the page table entry. Finally, the physical address is formed by attaching the frame number obtained from the page table entry to the high-order end of the page offset. Figure 9.15 shows the address translation in segmentation with paging scheme.

NOTES

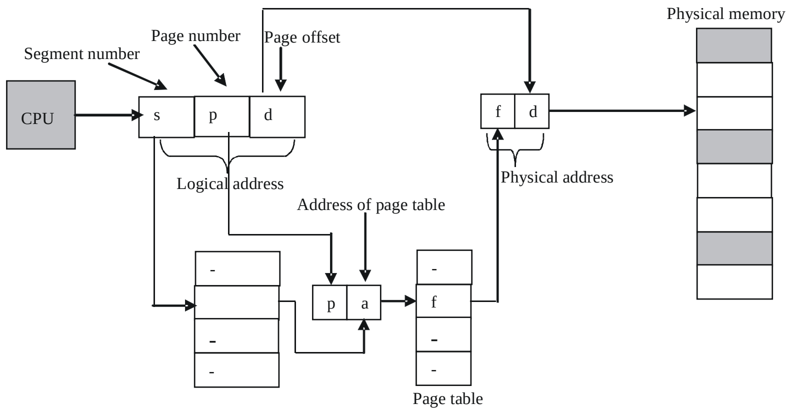


Fig. 9.15 *Segmentation with Paging*

9.6 SWAPPING

In multiprogramming, a memory management scheme called swapping can be used to increase the CPU utilization. The process of bringing a process to memory and after running for a while, temporarily copying it to disk is known as **swapping**. Figure 9.16 shows the swapping process. The decision of the process to be swapped in and swapped out is made by the CPU scheduler. For example, consider a multiprogramming environment with priority based scheduling algorithm. When a process of high-priority enters the input queue, a process of low priority is swapped out so that the process of high priority can be loaded and executed. On the termination of this process, the process of low priority is swapped back in the memory to continue its execution.

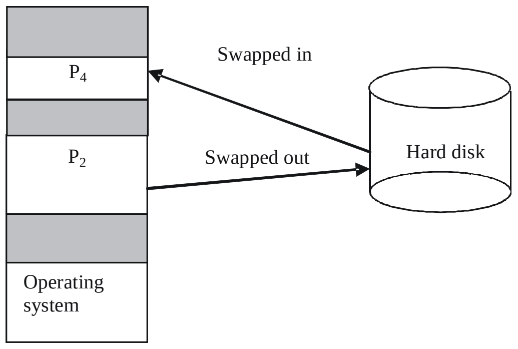


Fig. 9.16 *Swapping*

NOTES

Check Your Progress

6. State the formulation for the size of logical address space and page size.
7. Write the advantages and disadvantages of paging?
8. How do you define each entry in the table segmentation?
9. Define the three major parts of logical address.
10. What is meant by swapping?

9.7 VIRTUAL MEMORY

Virtual memory is a technique that allows execution of a program that is bigger than the physical memory of the computer system. In this technique, the operating system loads only those parts of program in memory that are currently needed for the execution of the process. The rest part is kept on the disk and is loaded into the memory only when needed. For example, a 64M program can run on a 32M system by loading the 32M in the memory at an instant; the parts of the program are swapped between the memory and the disk as needed (Refer Figure 9.17).

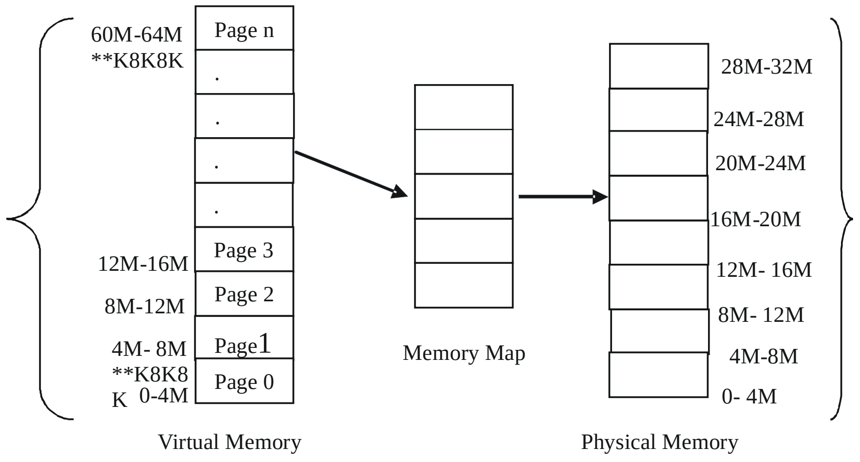


Fig. 9.17 *Virtual Memory and Physical Memory*

***Note**: In virtual memory systems, the logical address is referred to as **virtual address** and logical address space is referred to as **virtual address space**.*

9.8 VIRTUAL MEMORY MANAGEMENT: DEMAND PAGING

Demand paging is a system in which a page is loaded into the memory only when it is needed during program execution. Pages that are never accessed are never loaded into the memory.

A demand paging system combines the features of paging with swapping. To facilitate swapping, the entire virtual address space of a process is stored contiguously on a secondary storage device (usually, a disk). Whenever, a process is to be executed, an area on secondary storage device is allocated to it on which its pages are copied. The area is known as **swap space** of the process. During the execution of a process, whenever a page is required, it is loaded into the main memory from the swap space. Similarly, when a process is to be removed from main memory, it is written back into the swap space if it has been modified.

Other than swap space, some form of hardware support is also needed to differentiate the pages that are in memory from the ones on disk. For this, only an additional bit **valid** is maintained in each page table entry to indicate whether the page is in memory. If a page is valid (that is, it exists in the virtual address space of the process) and in memory, the associated valid bit is set to 1, otherwise it is set to 0. Figure 9.18 shows the page table in demand paging system.

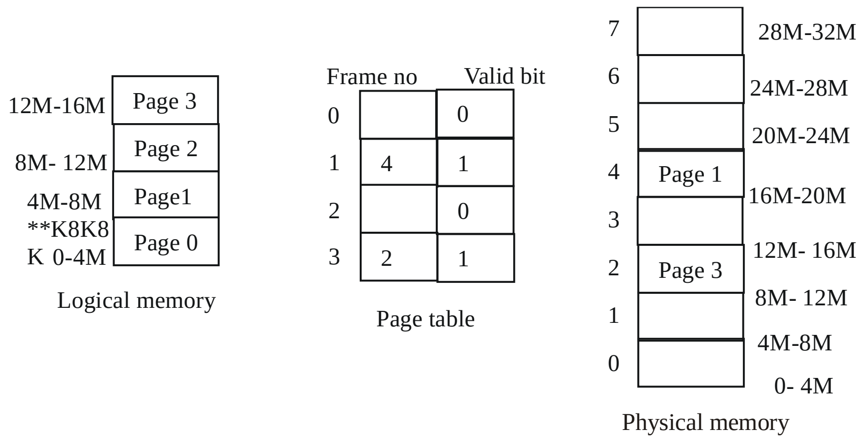


Fig. 9.18 *Page Table in Demand Paging System*

Whenever, a process requests for a page, the virtual address is sent to MMU. The MMU checks the valid bit in the page table entry of that page. If the valid bit is 1 (that is, the requested page is in memory), it is accessed as in paging. Otherwise, the MMU raises an interrupt called **page fault** or a **missing page interrupt** and the control is passed to the page fault routine in the operating system.

To handle the page fault, the page fault routine first of all checks whether the virtual address for the desired page is valid from its PCB stored in the process table. If it is invalid, it terminates the process giving error. Otherwise, it takes the following steps.

1. Locates for a free page frame in memory and allocates it to the process.
2. Swaps the desired page into this allocated page frame.
3. Updates the process table and page table to indicate that the page is in memory.

After performing these steps, the CPU restarts from the instruction that it left off due to the page fault (Refer Figure 9.19).

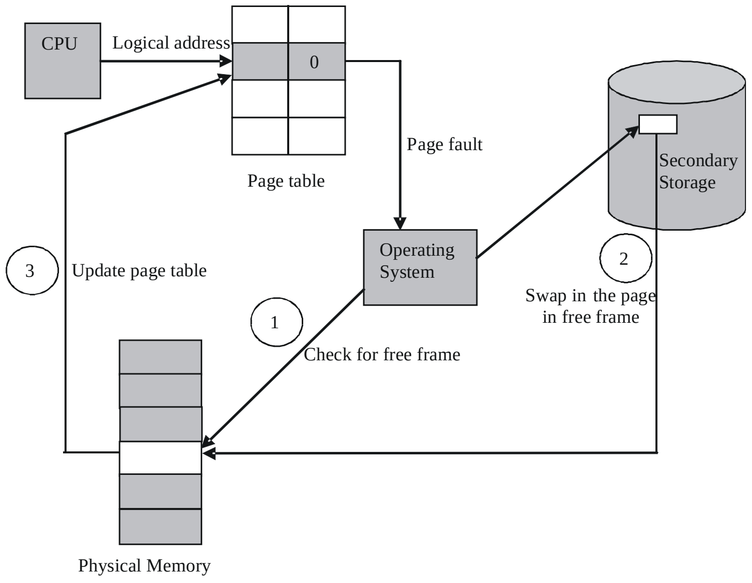


Fig. 9.19 *Handling a Page Fault*

***Note**: In demand paging, the process of loading a page in memory is known as **page-in** operation instead of swap-in. It is because the whole process is not loaded; only some pages are loaded into the memory*.

Advantages

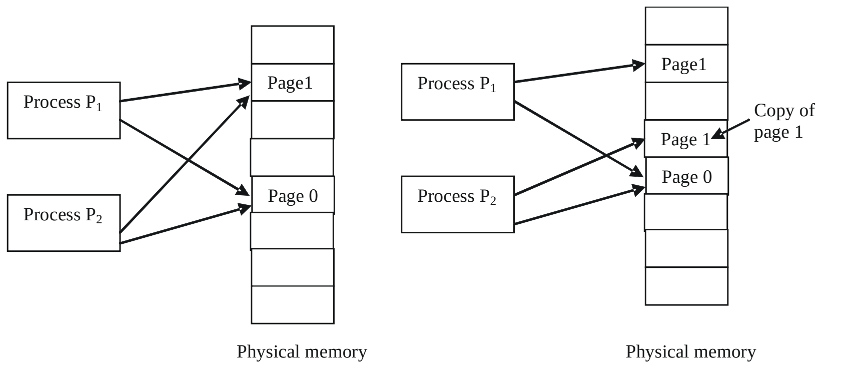
- It reduces the swap time since only the required pages are swapped in instead of the whole process.
- It increases the degree of multiprogramming by reducing the amount of physical memory required for a process.
- It minimizes the initial disk overhead as not all pages are to be read initially.
- It does not need extra hardware support.

9.9 COPY-ON-WRITE

A process may need to create several processes during its execution and the parent and child processes have their own distinct address spaces. If the newly created process is the duplicate of the parent process, it will contain same pages in its address space as that of parent. However, if the newly created process need to load another program in its memory space, immediately after creation, then the copying of parent's address space may be unnecessary. To avoid copying, a technique made available by virtual memory called **copy-on-write** can be employed.

In this technique, initially, parent and child processes are allowed to share the same pages and these shared pages are marked as copy-on-write pages. Now, if either process attempts to write on a shared page, a copy of that page is created for that process. Note that only the pages that can be modified (for example, pages containing data) are marked as copy-on-write pages while the pages that cannot be modified (for example, pages containing executable code) are shared between parent and child processes.

To implement this, a bit is associated in page table entry of each shared page to indicate that it is a copy-on-write page (Refer Figure 9.20). Whenever, either process say, child process tries to modify a page, the operating system creates a copy of that page, maps it to the address space of the child process and turns off the copy-on-write bit. Now, the child process can modify the copied page without affecting the page of the parent process. Thus, copy-on-write technique makes the process creation faster and conserves the memory.



(a) Before Process P_2 Modifies Page 1 *(b) After Process P_2 Modifies Page 1*

Fig 9.20 *Implementing Copy-On-Write*

Note*: A number of operating systems including Windows XP, Linux and Solaris support the use of copy-on-write technique*.

9.10 PAGE REPLACEMENT

As stated earlier, when page fault occurs, page fault routine locates for a free page frame in memory and allocates it to the process. However, there is a possibility that the memory is full, that is, no free frame is available for allocation. In that case, the operating system has to evict a page from the memory to make space for the desired page to be swapped in. The page to be evicted will be written on the disk depending on whether it has been modified or not. If the page has been modified while in the memory, it is rewritten to the disk; otherwise no rewrite is needed.

To keep track whether the page has been modified, a **Modified** (**M**) bit (also known as **dirty** bit) is added to each page table entry. This bit indicates whether the page has been modified. When a page is first loaded into the memory, this bit is cleared. It is set by the hardware when any word or byte in the page is

NOTES

written into. At the time of page replacement, if dirty bit for a selected page is cleared, it implies that the page has not been modified since it was loaded into the memory. The page frame is written back to the swap space only if dirty bit is set.

The system can select a page frame at random and replace it by the new page. However, if the replaced page is frequently accessed, then another page fault would occur when the replaced page is accessed again resulting in degradation of system performance. Thus, there must be some policy to select a page to be evicted. For this, there are various page replacement algorithms. These replacement algorithms can be evaluated by determining the number of page faults using a reference string. A **reference string** is an ordered list of memory references made by a process. It can be generated by a random-number generator or recording the actual memory references made by an executing program. To illustrate the page replacement algorithms, consider the reference string as shown in Figure 9.21 for the memory with three page frames. For simplicity, instead of actual memory references, we have considered only the page numbers.

5	0	5	3	5	2	5	0	1	0	7	3

Fig 9.21 *Structure of the Reference String*

9.10.1 First-In First-Out Page Replacement

The First-In, First-Out (FIFO) is the simplest page replacement algorithm. As the name suggests, the first page loaded into the memory is the first page to be replaced. That is, the page is replaced in the order in which it is loaded into the memory.

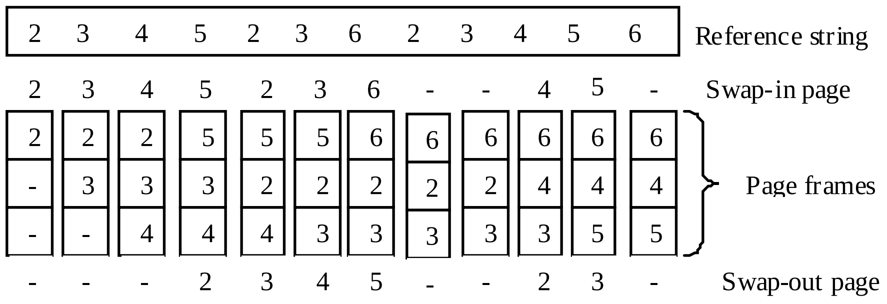
To illustrate the FIFO replacement algorithm, consider our example reference string shown in Figure 9.22. Assuming that initially, all the three frames are empty, the first two references made to page 5 and 0 and cause page faults. As a result, they are swapped in memory. The third reference made to page 5 does not cause page fault as it is already in memory. The next reference made to page 3 causes a page fault and that page is brought in memory. The reference to page 2 causes a page fault which results in the replacement of page 5 as it is the oldest page. Now, the oldest page is 0, so reference made to page 5 will replace page 0. This process continues until all the pages of reference string are accessed. It is clear from Figure 9.22 that there are nine page faults.

5	0	5	3	5	2	5	0	1	0	7	3	Reference string
5	0	-	3	-	2	5	0	1	-	7	3	Swap-in page
5	5	5	5	5	2	2	2	1	1	1	1	Page frames
-	0	0	0	0	0	5	5	5	5	7	7	
-	-	-	3	3	3	3	0	0	0	0	3	
-	-	-	-	-	5	0	3	2	-	5	0	Swap-out page

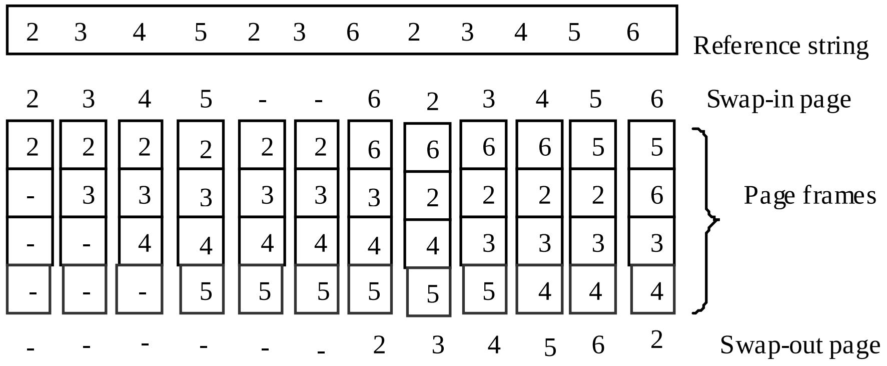
Fig 9.22 *FIFO Replacement Algorithm*

To implement this algorithm, each page table entry includes the time (called **swap-in** time) when the page was swapped in the memory. When a page is to be replaced, the page with the earliest swap-in time is replaced. Alternatively, a FIFO queue can be created to keep track of all the pages in the memory with the earliest one at the front and the recent at the rear of the queue. At the time of page fault, the page at the front of the queue is removed and the newly arrived page is added to the rear of the queue.

The FIFO page replacement algorithm is easier to implement as compared to all other replacement algorithms. However, it is rarely used as it is not very efficient. Since it does not consider the pattern of the usage of a page, a frequently used page frame may be replaced resulting in more page faults. Moreover, it suffers from **Belady's anomaly**—a situation in which increasing the number of page frames would result in more page faults. To illustrate this, consider the reference string containing five pages, numbered from 2 to 6 (Refer Figure 9.23). From this figure, it is clear that with three page frames, a total of nine page faults occur. On the other hand, with four page frames, a total of ten page faults occur.



(a) FIFO with Three Page Frames



(b) FIFO with Four Page Frames

Fig. 9.23 *Belady's Anomaly*

9.10.2 Optimal Page Replacement

The Optimal Page Replacement (OPT) algorithm is the best possible page replacement algorithm. The basic idea behind this algorithm is that whenever a page fault occurs, some pages are in memory; out of these pages, one will be referenced at the next instruction while other pages may not be referenced until the execution of certain number of instructions. In case of page replacement, the page that is referenced in last will be replaced. That is, the page to be referenced in the most distant future is replaced. For this, each page can be labelled in the memory with the number of instructions to be executed before that page is referenced for the first time. The page with the highest label is replaced from the memory.

To illustrate this algorithm, consider our example reference string (Refer Figure 9.21). Like FIFO, the first two references made to pages 5 and 0 cause page faults. As a result, they are swapped into the memory. The third reference made to page 5 does not cause page fault as it is already in memory. The reference made to page 3 causes a page fault and thus is swapped into memory. However, the reference made to page 2 replaces page 3 because page 3 is required at the last instruction whereas pages 5 and 0 are required at the next instruction. The page faults and the pages swapped-in and swapped-out for all the page references are shown in Figure 9.24. This algorithm causes seven page faults.

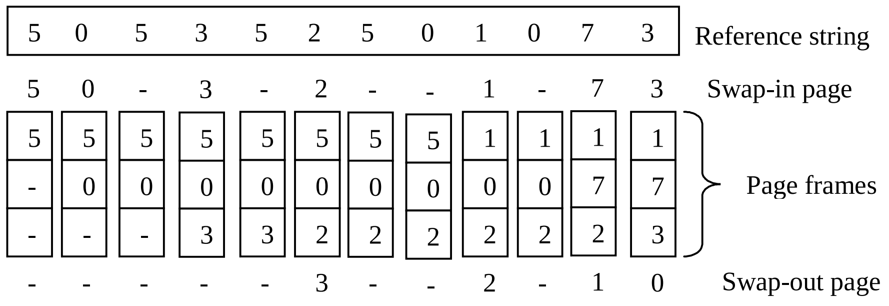


Fig 9.24 *Optimal Page Replacement Algorithm*

The advantage of this algorithm is that it causes the lowest number of page faults as compared to other algorithms. The disadvantage of this algorithm is that its implementation requires prior knowledge of which page will be referenced next. Though this algorithm is not used in systems practically, it is used as the basis for comparing performance of other algorithms.

Note*: To implement OPT, a program can be executed on a simulator and all the page references are recorded. Using the page reference records obtained during first run, it can be implemented at second run.*

9.10.3 Least Recently Used Page Replacement

The Least Recently Used (LRU) algorithm is an approximation to the optimal algorithm. Unlike optimal algorithm, it uses the recent past behaviour of the program to predict the near future. It is based on the assumption that the page that has been used in the last few instructions will probably be referenced in the next few instructions. Thus, it replaces the page that has not been referenced for the longest time.

Consider our example reference string (Refer Figure 9.21). As a result of this algorithm, the page faults and the pages swapped-in and swapped-out for all the page references are shown in Figure 9.25. Upto five references, page faults are same as that of optimal algorithm. When a reference is made to page 2, page 0 is replaced as it was least recently used. However, after page 5, it is being used again leading to a page fault. Regardless of this, the number of page faults is eight which is less than in case of FIFO.

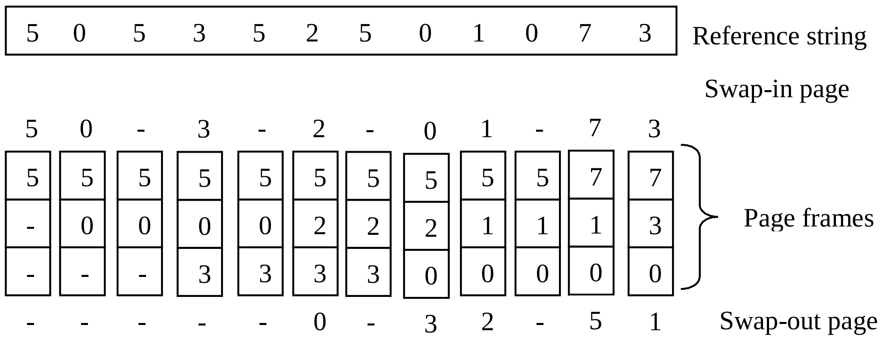


Fig 9.25 *LRU Page Replacement Algorithm*

One way to implement LRU is maintaining a linked list of all the pages in the memory; the most recently used page is at the head and the least recently used page is at the tail of the list. Whenever a page is to be replaced, it is deleted from the tail of the linked list and the new page is inserted at the head of the linked list. The problem with this implementation is that it requires updating the list at every page reference despite of whether page fault occurs or not. It is because whenever a page in memory is referenced, being the most recent page, it is removed from its current position and inserted at the head of the linked list. This results in extra overhead.

Alternatively, the hardware can be equipped with a counter. This counter is incremented by one after each instruction. The page table has a field to store the value of the counter. Whenever, a page is referenced, the current value of the counter is copied to that field in the page table entry for that page. Whenever, a page is to be replaced, this algorithm searches the page table for the entry having the lowest counter value (means the least recently used page) and replaces that page.

Clearly, it has less page faults as compared to FIFO algorithm. Moreover, it does not suffer from the Belady's anomaly. Thus, it is better than FIFO algorithm and is used in many systems. The disadvantage of this algorithm is that it is time consuming when implemented using linked list. Otherwise, it needs extra hardware support for its implementation.

Note: Both the optimal and LRU algorithms belong to a special class of page replacement algorithms called ***stack algorithms*** *that never exhibit Belady's anomaly.*

9.10.4 The Second Chance Page Replacement

NOTES

The second chance page replacement algorithm (sometimes also referred to as **clock** algorithm) is a refinement over FIFO algorithm. It replaces the page that is both the oldest as well as unused instead of the oldest page that may be heavily used. To keep track of the usage of the page, it uses the **Reference** bit (**R**) which is associated with each page. This bit indicates whether the reference has been made to the page while it is in memory. It is set whenever a page is accessed for either reading or writing. If this bit is clear for a page that means this page is not being used.

Whenever, a page is to be replaced, this algorithm uses the FIFO algorithm to find the oldest page and inspects its `reference` bit. If this bit is clear, the page is both the oldest and unused and thus, replaced. Otherwise, the second chance is given to this page and the `reference` bit of this page is cleared and its load time is set to the current time. Then the algorithm moves to the next oldest page using FIFO algorithm. This process continues until a page is found whose `reference` bit is clear. If the `reference` bit of all the pages is set (that is, all the pages are referenced), then this algorithm will proceed as pure FIFO.

This algorithm is implemented using a circular linked list and a pointer that points to the next victim page. Whenever a page is to be replaced, the list is traversed until a page whose reference bit is clear is found. While traversing, the `reference` bit of each examined page is cleared. When a page whose `reference` bit is clear is found, that page is replaced with the new page and pointer is advanced to the next page. For example, consider our example reference string with `reference` bits shown in Figure 9.26. The algorithm starts with the page 5, say at time `t` = 18. Since, the `reference` bit of this page is set, its `reference` bit is cleared and time is reset to the current system time as though it has just arrived in the memory. The pointer is advanced to the next page that is page 0. The `reference` bit of this page is clear, so it is replaced by the new page. The pointer is advanced to the page 3 which will be the starting point for next invocation of this algorithm.

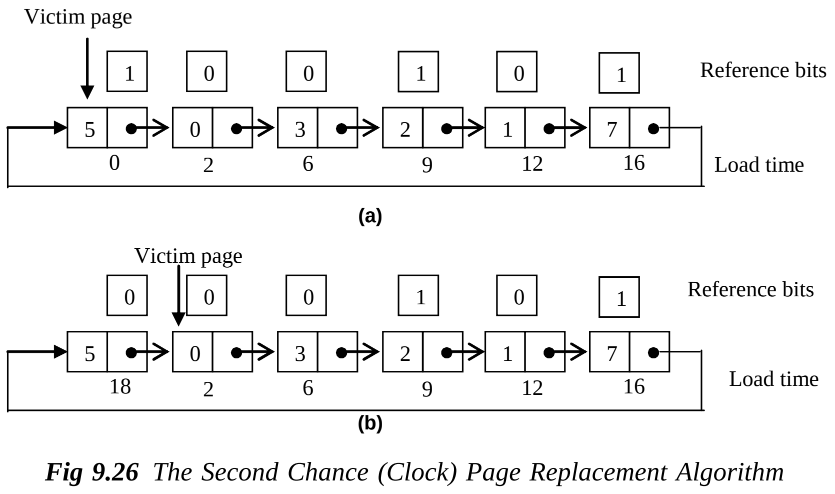


Fig 9.26 *The Second Chance (Clock) Page Replacement Algorithm*

9.10.5 Counting-Based Page Replacement Algorithm

NOTES

Other than the page replacement algorithms discussed earlier, there are several other algorithms. Some of them keep record of how often each page has been referenced by associating a counter with each page. Initially, the value of this counter is 0, which is incremented every time when a reference to that page is made. That is, the counter counts the number of references that have been made to each page. A page with the highest value of the counter is heavily used while for a page with the lowest value of counter, there are following interpretations:

- That page is least frequently used.
- That page has been just brought in and is yet to be used

The algorithm based on the first interpretation is known as **Least Frequently Used** (**LFU**) page-replacement algorithm. In this algorithm, when a page is to be replaced, the page with lowest value of counter is chosen for replacement. Clearly, the page that is heavily used is not replaced. The problem with this algorithm arises when there is a page that was used heavily initially, but afterwards never used again. For example, in a multipass compiler, some pages are used heavily during pass 1; after that pass, they may not be required. Still, these pages will not be replaced as they have high value of counter. Thus, this algorithm may replace useful pages instead of pages that are not in use. The algorithm that is based on the second interpretation is called the **Most Frequently Used** (**MFU**) page-replacement algorithm.

Both these algorithms are not commonly used, as their implementation is expensive. Moreover, they do not approximate the OPT page replacement algorithm.

Check Your Progress

11. What is the significance of virtual memory?
12. How do we handle the page fault?
13. What are the advantages of virtual memory management?
14. Define the usage of copy-on-write technique.
15. What do you understand by reference string in page replacement technique?
16. Explain the difference between the FIFO, LRU and Optimal page replacement algorithms.
17. Define LFU and MFU page replacement algorithms.

9.11 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. Whenever, a program is brought into the main memory for execution, it occupies certain number of memory locations. The set of all physical addresses used by the program is known as physical address space.

   The range of addresses that user programs can use is system-defined and the set of all logical addresses used by a user program is known as its logical address space.
2. To improve utilization of the CPU and the speed of the computer's response to its users, the system keeps several processes in memory, that is, several processes share memory. Due to the sharing of memory, there is need of memory management. There are various strategies that are used to manage memory. All these strategies allocate memory to the processes using either of following two approaches.
   - Contiguous memory allocation.
   - Non-contiguous memory allocation.
3. Any process regardless of how small it is, occupies an entire partition which leads to the wastage of memory within the partition. This phenomenon which results in the wastage of memory within the partition is called internal fragmentation. For example, loading a process of size 4M-*n* bytes into a partition of size 4M (where, M stands for Megabytes) would result in a wasted space of *n* bytes within the partition.
4. In general, at a certain point of time, there will be a set of holes of various sizes dispersed in the memory. As a result, there may be possibility that the total available memory is large enough to accommodate the waiting process. However, it cannot be utilized as it is scattered. This wastage of the memory space is called external fragmentation (also known as checkerboarding) since, the wasted memory is not a part of any partition.
5. Whenever a process arrives and there are various holes large enough to accommodate it, the operating system may use one of the following algorithms to select a partition for the process.
   - First Fit: In this algorithm, the operating system scans the free-storage list and allocates the first hole that is large enough to accommodate that process. This algorithm is fast because search is little as compared to other algorithms.
   - Best Fit: In this algorithm, the operating system scans the free-storage list and allocates the smallest hole whose size is larger than or equal to the size of the process. Unlike first fit algorithm, it allocates a partition that is close to the size required for that process. It is slower than the first fit algorithm as it has to search the entire list every time. Moreover,

NOTES

it leads to more wastage of memory as it results in the smallest leftover holes that cannot be used to satisfy the memory allocation request.

- Worst Fit: In this algorithm, the operating system scans the free-storage list and allocates the largest hole to that process. Unlike best fit, this algorithm results in the largest leftover holes. However, simulation indicates that worst fit allocation is not very effective in reducing the wastage of memory.

6. To understand this, consider the size of logical address space is 2^m. Now, if we choose a page size of 2^n (bytes or words), then *n* bits will specify the page offset and *m n* bits will specify the page number.
7. Advantages
   - Since the memory allocated is always in fixed unit, any free frame can be allocated to a process. Thus, there is no external fragmentation.

   Disadvantages

   - There may be some internal fragmentation. To illustrate this, consider a page size is of 4KB and a process requires memory of 8195 bytes, that is 2 page + 3 bytes. In this case, for only 3 bytes, an entire frame is wasted resulting in internal fragmentation.
8. Each entry in the segment table consists of two fields: segment base and segment limit. The segment base specifies the starting address of the segment in physical memory and the segment limit specifies the length of the segment. The segment number is used as an index to the segment table.
9. The logical address consists of three parts: Segment number (s), Page number (p) and Page offset (d).
10. The process of bringing a process to memory and after running for a while, temporarily copying it to disk is known as swapping.
11. Virtual memory is a technique that allows execution of a program that is bigger than the physical memory of the computer system. In this technique, the operating system loads only those parts of program in memory that are currently needed for the execution of the process. The rest part is kept on the disk and is loaded into the memory only when needed. For example, a 64M program can run on a 32M system by loading the 32M in the memory at an instant; the parts of the program are swapped between the memory and the disk as needed.
12. Whenever, a process requests for a page, the virtual address is sent to MMU. The MMU checks the valid bit in the page table entry of that page. If the valid bit is 1 (that is, the requested page is in memory), it is accessed as in paging. Otherwise, the MMU raises an interrupt called page fault or a missing page interrupt and the control is passed to the page fault routine in the operating system.

13. Advantages
    - It reduces the swap time since only the required pages are swapped in instead of the whole process.
    - It increases the degree of multiprogramming by reducing the amount of physical memory required for a process.
    - It minimizes the initial disk overhead as not all pages are to be read initially.
    - It does not need extra hardware support.
14. To avoid copying, a technique made available by virtual memory called copy-on-write can be employed. In this technique, initially, parent and child processes are allowed to share the same pages and these shared pages are marked as copy-on-write pages. Now, if either process attempts to write on a shared page, a copy of that page is created for that process.
15. A reference string is an ordered list of memory references made by a process. It can be generated by a random-number generator or recording the actual memory references made by an executing program.
16. The First-In, First-Out (FIFO) is the simplest page replacement algorithm. As the name suggests, the first page loaded into the memory is the first page to be replaced. That is, the page is replaced in the order in which it is loaded into the memory.

    The Optimal Page Replacement (OPT) algorithm is the best possible page replacement algorithm. The basic idea behind this algorithm is that whenever a page fault occurs, some pages are in memory; out of these pages, one will be referenced at the next instruction while other pages may not be referenced until the execution of certain number of instructions.

    The Least Recently Used (LRU) algorithm is an approximation to the optimal algorithm. Unlike optimal algorithm, it uses the recent past behaviour of the program to predict the near future. It is based on the assumption that the page that has been used in the last few instructions will probably be referenced in the next few instructions. Thus, it replaces the page that has not been referenced for the longest time.
17. A page with the highest value of the counter is heavily used while for a page with the lowest value of counter, there are following interpretations:

    • That page is least frequently used.

    • That page has been just brought in and is yet to be used

    The algorithm based on the first interpretation is known as Least Frequently Used (LFU) page-replacement algorithm. In this algorithm, when a page is to be replaced, the page with lowest value of counter is chosen for replacement. Clearly, the page that is heavily used is not replaced.

The algorithm that is based on the second interpretation is called the Most Frequently Used (MFU) page-replacement algorithm.

Both these algorithms are not commonly used, as their implementation is expensive. Moreover, they do not approximate the OPT page replacement algorithm.

NOTES

9.12 SUMMARY

- Many systems allow multiple processes to reside in the main memory at the same time, to increase the CPU utilization. It is the job of memory manager, a part of the operating system, to manage memory between these processes in an efficient way.
- Every byte in the memory has a specific address that may range from 0 to some maximum value as defined by the hardware. This address is known as physical address.
- A program is compiled to run starting from some fixed address and accordingly all the variables and procedures used in the source program are assigned some specific address known as logical address.
- The mapping from addresses associated with a program to memory addresses is known as address binding. The addresses binding can take place at compilation time, load time or run time.
- For managing the memory, the memory manager may use one strategy from a number of available memory management strategies.
- All the memory management strategies allocate memory to the processes using either of two approaches: contiguous memory allocation or non-contiguous memory allocation.
- In contiguous memory allocation, each process is allocated a single contiguous part of the memory. The different memory management schemes that are based on this approach are single partition and multiple partitions.
- In the single partition technique, main memory is partitioned into two parts. One of them is permanently allocated to the operating system while the other part is allocated to the user process.
- There are two alternatives for multiple partition technique—equal-sized partitions and unequal-sized partitions.
- In equal-sized partitions technique, any process can be loaded into any partition. Regardless of how small a process is, the process occupies an entire partition which leads to the wastage of memory within the partition. This phenomenon, which results in the wastage of memory within the partition, is called internal fragmentation.

NOTES

- In unequal-sized partition, whenever a process arrives, it is placed into the input queue of the smallest partition large enough to hold it. When this partition becomes free, it is allocated to the process.
- MVT (Multiprogramming with a Variable number of Tasks) is the generalization of the fixed partitions technique in which the partitions can vary in number and size. In this technique, the amount of memory allocated is exactly the amount of memory a process requires.
- In MVT, the wastage of the memory space is called external fragmentation (also known as checkerboarding) since the wasted memory is not a part of any partition.
- Whenever a process arrives and there are various holes large enough to accommodate it, the operating system may use one of the algorithms to select a partition for the process: first fit, best fit and worst fit.
- In multiprogramming environment, multiple processes are executed due to which two problems can arise which are relocation and protection.
- The relocation problem can be solved by equipping the system with a hardware register called relocation register which contains the starting address of the partition into which the process is to be loaded.
- To protect the operating system from access by other processes and the processes from one another, another hardware register called limit register is used.
- In noncontiguous allocation approach, parts of a single process can occupy noncontiguous physical addresses.
- Paging and segmentation are the memory management techniques based on the noncontiguous allocation approach.
- In paging, the physical memory is divided into fixed-sized blocks called page frames and logical memory is also divided into fixed-size blocks called pages which are of same size as that of page frames. The address translation is performed using a mapping table, called page table.
- Segmentation is a memory management scheme that implements the user view of a program. In this scheme, the entire logical address space is considered as a collection of segments with each segment having a number and a length. To keep track of each segment, a segment table is maintained by the operating system.
- The idea behind the segmentation with paging is to combine the advantages of both paging, (such as uniform page size) and segmentation, (such as protection and sharing) together into a single scheme. In this scheme, each segment is divided into number of pages. To keep track of these pages, a page table is maintained for each segment.
- A memory management scheme called swapping can be used to increase the CPU utilization. The process of bringing a process to memory and after running for a while, temporarily copying it to disk is known as swapping.

- Overlaying allows a process to execute irrespective of the system having insufficient physical memory. The programmer splits a program into smaller parts called overlays in such a way that no two overlays are required to be in main memory at the same time. An overlay is loaded into memory only when it is needed.
- Virtual memory is a technique that allows execution of a program that is bigger than the physical memory of the computer system. It can be implemented by demand paging or demand segmentation.
- In demand paging, a page is loaded into the memory only when it is needed during program execution. Pages that are never accessed are never loaded into the memory.
- Whenever a process requests for a page and that page is not in memory then MMU raises an interrupt called page fault or a missing page interrupt.
- A reference string is an ordered list of memory references made by a process.
- A technique made available by virtual memory called copy-on-write makes the process creation faster and conserves memory.
- The First-In, First-Out (FIFO) is the simplest page replacement algorithm. As the name suggests, the first page loaded into the memory is the first page is to be replaced.
- The Optimal Page Replacement (OPT) algorithm is the best possible page replacement algorithm in which the page to be referenced in the most distant future is replaced.
- The Least Recently Used (LRU) algorithm is an approximation to the optimal algorithm in which the page that has not been referenced for the longest time is replaced.
- The second chance page replacement algorithm (sometimes also referred to as clock algorithm) is a refinement over FIFO algorithm which replaces the page that is both the oldest as well as unused instead of the oldest page that may be heavily used.
- The Least Frequently Used (LFU) algorithm replaces the page that is least frequently used.
- The Most Frequently Used (MFU) algorithm replaces the page that has been just brought in and is yet to be used.

9.13 KEY WORDS

- **Address binding:** It is the process of mapping addresses associated with a program to memory addresses
- **Paging**: A process in which the physical memory is divided into fixed-sized blocks called page frames and logical memory is also divided into fixed-

size blocks called pages which are the same size as the page frames. Address translation is done using a mapping table, called page table

- **Segmentation**: A memory management scheme that implements the user view of a program, in which the entire logical address space is considered as a collection of segments with each segment having a number and a length
- **Segmentation with paging**: It is a combination of the advantages of both paging, (such as uniform page size) and segmentation, (such as protection and sharing) together into a single scheme. In this scheme, each segment is divided into number of pages. To keep track of these pages, a page table is maintained for each segment
- **Swapping**: A memory management scheme that can be used to increase CPU utilization. The process of bringing a process to memory and after running for a while, temporarily copying it to disk is known as swapping
- **Overlaying:** A technique that allows a process to be executed even when the system does not have sufficient physical memory
- **Virtual memory:** A technique that allows execution of a program that is bigger than the physical memory of the computer system
- **Demand paging:** A system in which a page is loaded into the memory only when it is needed during program execution. Pages that are never accessed are never loaded into the memory
- **Copy-on-write:** A technique made available by virtual memory that makes process creation faster and conserves memory

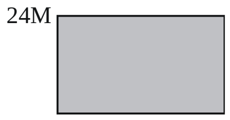
9.14 SELF-ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. What is address binding? When does it take place?
2. Name the different memory management strategies.
3. Write short notes on:
   (a) Segmentation
   (b) Swapping
   (c) Paging table
4. What is the basic concept of virtual memory?
5. When does a page fault occur? Mention the steps that are taken to handle a page fault.
6. What is demand paging? List some of its advantages.
7. State the hardware support required for demand paging.

Long-Answer Questions

1. Consider the following memory map with a number of variable size partitions.



   Assume that initially, all the partitions are empty. How would each of the first fit, best fit and the worst fit partition selection algorithms allocate memory to the following processes arriving one after another?

   (a) P_1 of size 2M

   (b) P_2 of size 2.9M

   (c) P_3 of size 1.4M

   (d) P_4 of size 5.4M

   Does any of the algorithms result in process waiting because of insufficient memory available? Also determine the algorithms that will lead to the most efficient use of memory.

2. Which of the following memory management schemes suffer from internal or external fragmentation?

   (a) Multiple fixed-partition

   (b) Multiple variable-partition

   (c) Paging

   (d) Segmentation

3. Consider a paged memory system with 16 pages of 2048 bytes each in logical memory and 32 frames in physical memory. How many bits will each of the following comprise?

   (a) Logical address

   (b) Page number

   (c) Page offset

   (d) Physical address

4. Consider a paged memory system with 2^{16} bytes of physical memory, 256 pages of logical address space, and a page size of 2^{10} bytes, how many bytes are in a page frame?

5. Can a process on a paged memory system access memory allocated to some other process? Why or why not?

6. What are the two major differences between segmentation and paging? Explain giving suitable example.

7. How does the segmentation scheme allow different processes to share data or code? Discuss in brief.

NOTES

8. Using the following segment table, compute the physical address for the logical address consisting of segment and offset as given below.

	Base	Limit
0	5432	350
1	115	100
2	2200	780
3	4235	1100
4	1650	400

Segment Table

   (a) Segment 0 and offset 193
   (b) Segment 2 and offset 546
   (c) Segment 3 and offset 1265
9. What is the idea behind combining segmentation with paging? When is it useful?
10. The copy-on-write technique makes the creation of a process faster and conserves memory. Explain in brief.
11. What is Belady's anomaly? Does the LRU replacement algorithm suffer from this anomaly? Justify your answer with an example.
12. Consider the following reference string consisting of 6 pages from 0 to 7.

0	1	2	3	1	0	4	5	1	0	1	2	6	5	2	1	0	1

   Determine how many page faults would occur in case of (a) FIFO replacement, (b) Optimal replacement and (c) LRU replacement assuming one, two, three and four frames.

9.15 FURTHER READINGS

Silberschatz, Abraham, Peter B. Galvin and Greg Gagne. 2008. *Operating System Concepts*, 8th Edition. New Jersey: John Wiley & Sons.

Tanenbaum, Andrew S. 2006. *Operating Systems Design and Implementation*, 3rd Edition. New Jersey: Prentice Hall.

Tanenbaum, Andrew S. 2001. *Modern Operating Systems*. New Jersey: Prentice Hall.

Deitel, Harvey M. 1984. *An Introduction to Operating Systems*. Boston (US): Addison-Wesley.

Stallings, William. 1995. *Operating Systems,* 2nd Edition. New Jersey: Prentice Hall.

Milenkovic, Milan. 1992. *Operating Systems: Concepts and Design*. New York: McGraw Hill Higher Education.

Mano, M. Morris. 1993. *Computer System Architecture*. New Jersey: Prentice Hall Inc.

BLOCK - V
FILE SYSTEM

NOTES

UNIT 10 FILE SYSTEM, ACCESS METHODS AND DIRECTORY

Structure

10.0 INTRODUCTION

The operating system abstracts from the physical properties of its storage devices and defines a logical storage unit known as a file. This allows user to directly access the data (on physical devices) without knowing where the data is actually stored. A file is a collection of related data stored as a named unit on the secondary storage. It can store different types of data, like text, graphic, database, executable code, sound, videos, etc. and on the basis of the data, a file can be categorized as a data file, graphic file, database file, executable file, sound file, video file, etc. File operations are the functions that can be performed on a file. An operating system handles the file operations through the use of system calls. The various operations that can be performed on a file are create, write, read, seek, delete, open, append, rename and close a file. The operating system can handle a file in a reasonable way only if it recognizes and supports that file type. The file structure refers to the internal structure of the file, that is, how a file is internally stored in the system.

The file structure refers to the internal structure of the file, that is, how a file is internally stored in the system. When the information in the file is accessed in order, one record after the other, it is called sequential access. To overcome this problem, the data on the disk is stored as blocks of data with index numbers which helps to read and write data on the disk in any order known as random or direct access.

NOTES

To manage this data, the disk is divided into one or more partitions also known as volumes and each partition contains information about the files stored in it. This information is stored in a directory also known as device directory.

In this unit, you will study the concepts of file, access methods and directory.

10.1 OBJECTIVES

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

- Introduce the basic concepts of files
- Define the file attributes and file operations
- Discuss about the various access methods, such as sequential and direct access method
- Explain the definition and functions of having a directory

10.2 FILE CONCEPT

As stated in the introduction, a system stores data on various storage devices. The operating system, however, for the convenience of use of data on these devices provides a uniform logical view of the data storage to the users. The operating system abstracts from the physical properties of its storage devices and defines a logical storage unit known as a **file**. This allows user to directly access the data (on physical devices) without knowing where the data is actually stored.

A file is a collection of related data stored as a named unit on the secondary storage. It can store different types of data, like text, graphic, database, executable code, sound, videos, etc. and on the basis of the data, a file can be categorized as a data file, graphic file, database file, executable file, sound file, video file, etc.

Moreover, the structure of a file is based on the type of the file. For example, a graphic file is an organized collection of pixels, a database file is a collection of tables and records, and a batch file is a collection of commands.

Note: *From a users' view, it is not possible to write data directly to a storage device until it is within a file.*

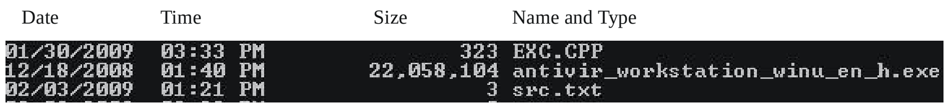
10.2.1 File Attributes

A file in a system is identified by its name. The file name helps a user to locate a specific file in the system. Different operating systems follow different file naming conventions. However, most operating systems accept a file name as a string of characters, or numbers or some special symbols as well. For instance, names, such as `alice`, `tom`, `3546`, `!hello` and `table2-1` are all valid file names. Note that some operating systems distinguish the upper and lower case characters in the file names. For instance, in UNIX the file names `Alice, alice, ALICE` refer to three different files whereas, in DOS and Windows they refer to the same file.

Apart from the file name, some additional information (also known as **file attributes**) is also associated with each file. This information helps the file system to manage a file within the system. The file attributes related to a file may vary in different operating systems. Some of the common file attributes are as follows.

- **Name**: Helps to identify and locate a file in the system.
- **Size**: Stores information about the current size of the file (in bytes, words, or blocks).
- **Type**: Helps the operating system to recognize and use the recommended program to open a particular file type. For instance, to open an mpeg (multimedia) file, operating system uses a media player.
- **Identifier**: A unique tag, usually a number that helps the file system to recognize the file within the file system.
- **Location**: A pointer that stores location information of the device and location of the file on that device.
- **Date and Time**: Stores information related to a file, such as, creation, last modification and last use. Such information may be useful in case of protection, security and monitoring, etc.
- **Protection**: Stores information about the access permissions (read, write, execute) of different users. For example, it may specify who can access the file and which operations can be performed on a file by a user.

Figure 10.1 shows the list of some attributes that MS DOS attaches to a file.



```
01/30/2009  03:33 PM               323 EXC.CPP
12/18/2008  01:40 PM        22,058,104 antivir_workstation_winu_en_h.exe
02/03/2009  01:21 PM                 3 src.txt
```

Fig. 10.1 *File Attributes of MS DOS*

The information related to a file is stored as a directory entry in the directory structure. The directory entry includes the file's name and the unique identifier. The identifier in turn locates the other file attributes.

10.2.2 File Operations

File operations are the functions that can be performed on a file. An operating system handles the file operations through the use of system calls. The various operations that can be performed on a file are create, write, read, seek, delete, open, append, rename and close a file.

- **Create a File**: To bring a file into existence, the `create` system call is used. When this system call is used, the operating system searches for the free space in the file system and allocates it to the file. In addition, the operating system makes a directory entry to record the name, location and other information about the file.

- **Open-File**: To open a file, the `open` system call is used which accepts the file name and the access-mode (read, write, execute) as parameters and returns a pointer to the entry in the **open-file table** (a table in the main memory that stores information about the files that are opened at a particular time). The operating system searches the directory entry table for the file name and checks if the access permission in directory entry matches the request. If that access-mode is allowed, it then copies the directory entry of the file to the open-file table.
- **Write to a File**: To store data into a file, the `write` system call is used which accepts the file name and the data to be written to the file as parameters. The operating system searches the directory entry to locate the file and writes data to the specified position in the file and also updates the write pointer to the location where next write operation is to take place.
- **Read a File**: To retrieve data from a file, the `read` system call is used which accepts the file name, amount of data to be read and a read pointer to point to the position from where the data is to be read as parameters. The operating system searches the specified file using the directory entry, performs the read operation and updates the pointer to the new location. Note that since a process may be only reading or writing a file at a time, a single pointer called **current position pointer** can be used for both reading and writing. Every time a read or write operation is performed, this pointer must be updated.
- **Seek File**: To position the pointer to a specific position in a file, the `seek` system call is used. Once the pointer is positioned, data can be read from and written to that position.
- **Close File**: When all the operations on a file are completed, it must be closed using the `close` system call. The operating system searches and erases the file entry from the open-file table to make space for the new file entries. Some systems automatically close a file when the process that has opened the file terminates.
- **Delete File**: When a file is not required, the `delete` system call is used. The operating system searches the file name in the directory listing. Having found the associated entry, it releases all space allocated to the file (that can be reused by other files) by erasing its corresponding directory entry.
- **Append File**: To add data at the end of an existing file, `append` system call is used. This system call works similar to the `write` system call, except that it positions the pointer to the end of file and then performs the write operation.
- **Rename File**: To change the name of an existing file, `rename` system call is used. This system call changes the existing entry for the file name in the directory to a new file name.

10.2.3 File Types

As stated in the above sections, files can be of different types. The operating system can handle a file in a reasonable way only if it recognizes and supports that file type. A user request to open an executable file with a text editor will produce garbage if the operating system has not been told that it is an executable file.

NOTES

The most common technique to implement a file type is by providing extension to a file. The file name is divided into two parts, with the two parts separated by a period ('.') symbol, where the first part is the **name** and the second part after the period is the **file extension**. A file extension is generally one to three characters long, it indicates the *type* of the file and the operations (read, write, execute) that can be performed on that file. For example, in the file name `Itlesl.doc`, `Itlesl` is the name and `.doc` is the file extension. The extension `.doc` indicates that `Itlesl.doc` is a document file and should be opened with an editor. Similarly, a file with `.exe` or `.com` extension is an executable file. Table 10.1 lists various file types, extension and their meaning.

Table 10.1 *File Types and Extensions*

File type	Extension	Meaning
Archive	`arc, zip, tar`	Related files compressed and grouped together into single file for storage
Batch	`bat, sh`	An executable file stores a series of commands that can be executed with a single command
Backup file	`bak, bkf`	Stores a copy of the data on the disk, used for recovering system crash
Executable	`exe, com, bin`	Used to run various programs on a computer
Library	`lib, a, so, dll,`	Stores libraries of routines for programmers
Image	`bmp, jpeg, gif, jfif, dib`	Stores images and graphics
Multimedia	`mpeg, mp2, mpa, mpe`	Stores audio and video information
Object	`obj, o`	Machine language file, precompiled, used for generating output
System file	`inf, ini, drv`	Stores system information for loading and managing different applications
Text	`txt, doc`	Stores textual data, documents
Word processor	`wp, txt, rrf, doc`	Stores various word processor formats

File extensions help the operating system to know about the application program that has created the file. For instance, the file with .txt extension will be opened with a text editor and the file with .mp3 extension will be opened with a music player supporting the .mp3 files. Note that the operating system automatically opens the application program (for the known file types) whenever a user double clicks the file icon.

Some operating systems, such as UNIX, support the use of double extension to a file name. For example, the file name `file1.c.z` is a valid file name, where `.c` means that `file1` is a C language file and `.z` means the file is compressed using some zip program. A file extension can be system defined or user defined.

Another way to implement the file type is the use of **Magic Number**. A magic number is a sequence of bits, placed at the starting of a file to indicate roughly the type of file. The UNIX system makes use of magic number to recognize the file type. However, not all its files have magic numbers. UNIX system allows file-name-extension hints to help its user determine the type of contents of the file, it.

Check Your Progress

1. Define the concept of file.
2. State the common file attributes.
3. What are the file operations that can be performed on a file?
4. Name the most common technique to implement any file type.

10.3 ACCESS METHODS

The information stored in the file can be accessed in one of the two ways: sequential access or direct access.

Sequential Access

When the information in the file is accessed in order, one record after the other, it is called **sequential access**. It is the easiest file access method. Compilers, multimedia applications, sound files and editors are the most common examples of the programs using sequential access.

The most frequent and common operations performed on a file are read and write operations. In case of read operation, the record at the location pointed by the file pointer is read and the file pointer is then advanced to the next record. Similarly, in case of write operation, the record is written to the end of the file and pointer is advanced to the end of new record.

Direct Access

With the advent of disks as a storage media, large amount of data can be stored on it. Sequential access of this data would be very lengthy and slow process. To overcome this problem, the data on the disk is stored as blocks of data with index numbers which helps to read and write data on the disk in any order (known as **random** or **direct access**).

Under direct access, a file is viewed as a sequence of blocks (or records) which are numbered. The records of a file can be read or written in any order using this number. For instance, it is possible to read block 20, then write block 4, and then read block 13. The block number is a number given by the user. This number is relative to the beginning of the file. This relative number internally has an actual absolute disk address. For example, the record number 10 can have the actual address 12546 and block number 11 can have the actual address 3450. The relative address is internally mapped to the absolute disk address by the file system. The user gives relative block number for accessing the data without knowing the actual disk address. Depending on the system, this relative number starts with either 0 or 1 for a file.

In direct access, the system calls for read and write operations are modified to include the block number as a parameter. For instance, to perform the read or write operation on a file, user gives `read n` or `write n` (`n` is the block number) rather than `read next` or `write next` system calls used in sequential access.

Most applications with large databases require direct access method for immediate access to large amounts of information. For example, in a railway reservation system, if a customer requests to check the status for reservation of the ticket, the system must be able to access the record of that customer directly without having the need to access all other customers' records.

Note that an operating system may support either sequential access or direct access, or both for accessing the files. Some systems require a file to be defined as sequential or direct when it is created, so that it can be accessed in the way it is declared.

10.4 DIRECTORY

As stated earlier, a computer stores numerous data on disk. To manage this data, the disk is divided into one or more partitions (also known as **volumes**) and each partition contains information about the files stored in it. This information is stored in a **directory** (also known as **device directory**). Figure 10.3 shows different file-system organization.

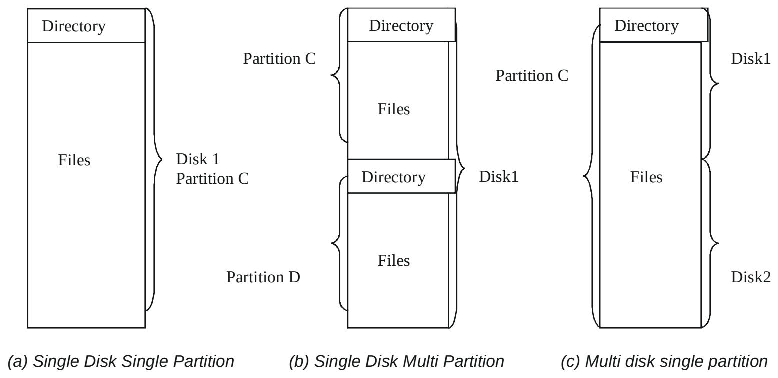


Fig. 10.3 *Various File System Organization Schemes*

Note: It is possible to have more than one operating system on a single disk, where a user can boot any of the operating system according to the need.

Different operations that can be performed on a directory are as follows.

- **Create a File**: New files can be created and added to a directory by adding a directory entry in it.
- **Search a File**: Whenever a file is required to be searched, its corresponding entry is searched in the directory.
- **List a Directory**: All the files along with their contents in the directory entry are listed.
- **Rename a File**: A file can be renamed. A user might need to rename the file with the change in its content. When a file is renamed its position within the directory may also change.
- **Delete a File**: When a file is no longer required, it can be deleted from the directory.
- **Traverse the File System**: Every directory and every file within a directory structure can be accessed.

Note: A directory is a flat file that stores information about files and subdirectories.

There are various schemes to define the structure of a directory. The most commonly used schemes are as follows.

- Single-level directory
- Two-level directory
- Hierarchical directory

All these schemes are discussed in subsequent section.

Single-Level Directory

Single-level directory is the simplest directory structure. There is only one directory that holds all the files. Sometimes this directory is referred to as **root directory.** Figure 10.4 shows a single-level directory structure having five files. In this figure, box represents directory and circles represent files.

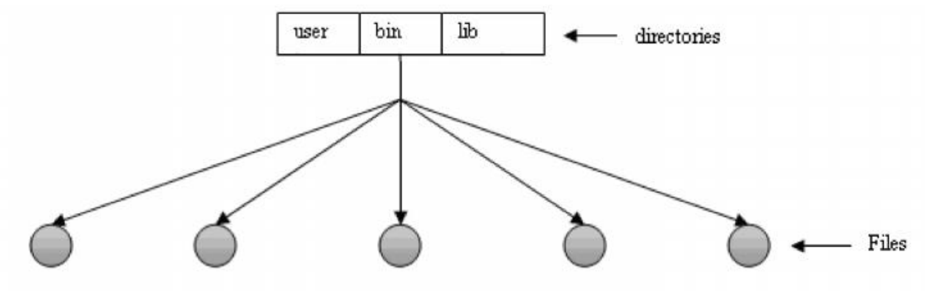


Fig. 10.4 *Single-Level Directory Structure*

The main drawback of this system is that no two files can have the same name. For instance, if one user (say, `jojo`) creates a file with name `file1` and then another user (say, `abc`) also creates a file with the same name, the file created by the user `abc` will overwrite the file created by the user `jojo`. Thus, all the files must have unique names in a single-level directory structure. With the increase in the number of files and users on a system, it becomes very difficult to have unique names for all the files.

Two-Level Directory

In a two-level directory structure, a separate directory known as **User File Directory (UFD)** is created for each user. Whenever, a new UFD is created, an entry is added to the **Master File Directory** (**MFD**) which is at the highest level in this structure (Refer Figure 10.5). When a user refers to a particular file, first, the MFD is searched for the UFD entry of that user and then the file is searched in the UFD.

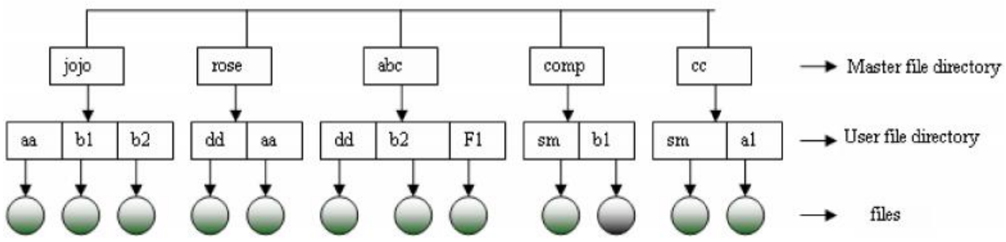


***Fig. 10.5** Two-Level Directory Structure*

Unlike, single-level directory structure, only the file names should be unique in a two-level directory. That is, there may be files with same name in different directories. Thus, there will not be the problem of name-collision in this directory structure but there is one disadvantage that the users in this directory structure are not allowed to access files of other users. If a user wants to access a file of other user, he needs special permissions from the administrator. In addition, to access other users' file, the user must know the correct **path name** (which includes the user name and the file name). Note that different systems use different syntax for file naming in directories. For instance, in MS DOS, to access the file in the `sm` directory, the user gives `//comp/sm,` where `//` refers to root, `comp` is the user name, `sm` is the directory.

In some situations, a user might need to access files other than their own file. One such situation might occur with system files. The user might want to use system programs like compilers, assemblers, loaders, or other utility programs. In such a case, to copy all the files in every user directory would require a lot of space and thus, would not be feasible. One possible solution to this is to make a special user directory and copy system files into it. Now, whenever a filename is given, it is first searched in the local UFD, and if not found there then the file is searched in the special user directory that contains system files.

NOTES

Hierarchical Directory

The hierarchical directory, also known as **tree of directory** or **tree-structured directory**, allows users to have subdirectories under their directories, thus making the file system more logical and organized for the user. For instance, a user may have directory `furniture`, which stores files related to types of furniture say `wooden`, `steel`, `cane`, etc. Further, he wants to define a subdirectory which states the kind of furniture available under each type say `sofa`, `bed`, `table`, `chair`, etc. Under this system, the user has the flexibility to define, group and organize directories and sub-directories according to his requirements.

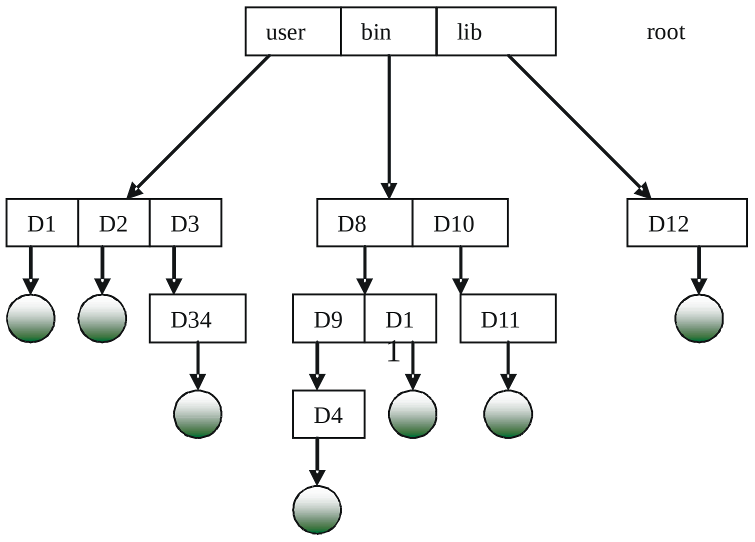


Fig. 10.6 *Hierarchical Directory Structure*

The hierarchical directory structure has the root directory at the highest level, which is the **parent** directory for all directories and subdirectories. The root directory generally consists of system library files. All files or directories at the lower levels are called **child** directories and a directory with no files or subdirectory is called a **leaf**. Every file in the system has a unique path name. A path name is the path from the root, through all the sub-directories, to a specified file. Figure 10.6 shows the hierarchical directory structure having different levels of directories, subdirectories and related files.

The user under hierarchical directory system can access files of other users in addition to his own files. To access the files the user can specify either absolute path name or relative path name. The **absolute path name** begins at the root and follows a path down to the specified file or using the **relative path name** that defines a path from the current working directory. For instance, to access a file under directory `D1`, using absolute path name, the user will give the path `\\bin\D8\D1\filename`. On the other hand, if the user's current working directory is `\\bin\D8`, the relative path name will be `D1\filename`.

In this structure, the major concern is the deletion of the files. If a directory is empty it can be deleted simply; however, if the directory contains subdirectories and files, they need to be handled first. Some systems, for example, MS DOS requires a directory to be completely empty before the delete operation can be performed on it. The user will have to delete all the files, subdirectories, files in subdirectories before performing the delete operation on a directory. Whereas, other systems, for example, UNIX is flexible as it allows user to delete a complete directory structure containing files and subdirectory with a single `rm` command. Though it is easy for a user to handle delete operation on directory under the UNIX system, but it increases the chances of accidental deletion of files.

Note: MS DOS, WINDOWS, and UNIX are some of the examples of systems using hierarchical directory structure.

Check Your Progress

5. Define the terms, such as sequential access and direct access.
6. List down the different operations that can be performed on a directory.
7. Define root directory.
8. What do you understand by hierarchical directory?
9. How do we define a child, leaf and parent directory?

10.5 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. The operating system, however, for the convenience of use of data on these devices provides a uniform logical view of the data storage to the users. The operating system abstracts from the physical properties of its storage devices and defines a logical storage unit known as a file.
2. Some of the common file attributes are as follows.
   - Name: Helps to identify and locate a file in the system.
   - Size: Stores information about the current size of the file (in bytes, words, or blocks).
   - Type: Helps the operating system to recognize and use the recommended program to open a particular file type. For instance, to open an mpeg (multimedia) file, operating system uses a media player.
   - Identifier: A unique tag, usually a number that helps the file system to recognize the file within the file system.
   - Location: A pointer that stores location information of the device and location of the file on that device.

- Date and Time: Stores information related to a file, such as, creation, last modification and last use. Such information may be useful in case of protection, security and monitoring, etc.
- Protection: Stores information about the access permissions (read, write, execute) of different users. For example, it may specify who can access the file and which operations can be performed on a file by a user.

3. File operations are the functions that can be performed on a file. An operating system handles the file operations through the use of system calls. The various operations that can be performed on a file are create, write, read, seek, delete, open, append, rename and close a file.

   Open-File: To open a file, the open system call is used which accepts the file name and the access-mode (read, write, execute) as parameters and returns a pointer to the entry in the open-file table (a table in the main memory that stores information about the files that are opened at a particular time). The operating system searches the directory entry table for the file name and checks if the access permission in directory entry matches the request. If that access-mode is allowed, it then copies the directory entry of the file to the open-file table.
4. The most common technique to implement a file type is by providing extension to a file. The file name is divided into two parts, with the two parts separated by a period ('.') symbol, where the first part is the name and the second part after the period is the file extension. A file extension is generally one to three characters long, it indicates the *type* of the file and the operations (read, write, execute) that can be performed on that file.
5. When the information in the file is accessed in order, one record after the other, it is called sequential access. It is the easiest file access method. Compilers, multimedia applications, sound files and editors are the most common examples of the programs using sequential access.

   With the advent of disks as a storage media, large amount of data can be stored on it. Sequential access of this data would be very lengthy and slow process. To overcome this problem, the data on the disk is stored as blocks of data with index numbers which helps to read and write data on the disk in any order (known as random or direct access).
6. Different operations that can be performed on a directory are as follows.
   - Create a File: New files can be created and added to a directory by adding a directory entry in it.
   - Search a File: Whenever a file is required to be searched, its corresponding entry is searched in the directory.
   - List a Directory: All the files along with their contents in the directory entry are listed.

- Rename a File: A file can be renamed. A user might need to rename the file with the change in its content. When a file is renamed its position within the directory may also change.
- Delete a File: When a file is no longer required, it can be deleted from the directory.
- Traverse the File System: Every directory and every file within a directory structure can be accessed.

7. Single-level directory is the simplest directory structure. There is only one directory that holds all the files. Sometimes this directory is referred to as root directory.
8. The hierarchical directory, also known as tree of directory or tree-structured directory, allows users to have subdirectories under their directories, thus making the file system more logical and organized for the user.
9. The hierarchical directory structure has the root directory at the highest level, which is the parent directory for all directories and subdirectories. The root directory generally consists of system library files. All files or directories at the lower levels are called child directories and a directory with no files or subdirectory is called a leaf. Every file in the system has a unique path name.

10.6 SUMMARY

- The operating system, however, for the convenience of use of data on these devices provides a uniform logical view of the data storage to the users. The operating system abstracts from the physical properties of its storage devices and defines a logical storage unit known as a file.
- A file is a collection of related data stored as a named unit on the secondary storage. It can store different types of data, like text, graphic, database, executable code, sound, videos, etc. and on the basis of the data, a file can be categorized as a data file, graphic file, database file, executable file, sound file, video file, etc.
- Apart from the file name, some additional information (also known as file attributes) is also associated with each file. This information helps the file system to manage a file within the system. The file attributes related to a file may vary in different operating systems.
- The information related to a file is stored as a directory entry in the directory structure. The directory entry includes the file’s name and the unique identifier. The identifier in turn locates the other file attributes.
- File operations are the functions that can be performed on a file. An operating system handles the file operations through the use of system calls. The various

operations that can be performed on a file are create, write, read, seek, delete, open, append, rename and close a file.

Open-File: To open a file, the open system call is used which accepts the file name and the access-mode (read, write, execute) as parameters and returns a pointer to the entry in the open-file table (a table in the main memory that stores information about the files that are opened at a particular time). The operating system searches the directory entry table for the file name and checks if the access permission in directory entry matches the request. If that access-mode is allowed, it then copies the directory entry of the file to the open-file table.

- Since a process may be only reading or writing a file at a time, a single pointer called current position pointer can be used for both reading and writing. Every time a read or write operation is performed, this pointer must be updated.
- The most common technique to implement a file type is by providing extension to a file. The file name is divided into two parts, with the two parts separated by a period ('.') symbol, where the first part is the name and the second part after the period is the file extension. A file extension is generally one to three characters long, it indicates the *type* of the file and the operations (read, write, execute) that can be performed on that file.
- The information stored in the file can be accessed in one of the two ways: sequential access or direct access.
- To manage this data, the disk is divided into one or more partitions (also known as volumes) and each partition contains information about the files stored in it. This information is stored in a directory (also known as device directory).
- There are various schemes to define the structure of a directory. The most commonly used schemes are as follows.
  - Single-level directory
  - Two-level directory
  - Hierarchical directory
- In a two-level directory structure, a separate directory known as User File Directory (UFD) is created for each user. Whenever, a new UFD is created, an entry is added to the Master File Directory (MFD) which is at the highest level in this structure. When a user refers to a particular file, first, the MFD is searched for the UFD entry of that user and then the file is searched in the UFD.
- The hierarchical directory structure has the root directory at the highest level, which is the parent directory for all directories and subdirectories.

The root directory generally consists of system library files. All files or directories at the lower levels are called child directories and a directory with no files or subdirectory is called a leaf. Every file in the system has a unique path name.

NOTES

10.7 KEY WORDS

- **File:** A file is a collection of related data stored as a named unit on the secondary storage.
- **File system:** Primarily responsible for the management and organization of various files in a system.
- **File operations:** Functions that can be performed on a file.
- **Identifier:** A unique tag, usually a number that helps the file system to recognize the file within the file system.
- **Append file:** To add data at the end of an existing file, append system call is used.
- **Sequential access:** A type of access mode when the information in a file is accessed in order, one record after the other.
- **Direct access:** When a file is viewed as a sequence of blocks (or records) which are numbered and can be read or written in any order using this number.

10.8 SELF-ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. What do you understand by the concept of file?
2. State the attributes of file in a system.
3. How do we seek any file in an operating system?
4. What is the significance of current position pointer?
5. Why do we use magic number in the implementation of any file type?
6. What is a record sequence?
7. What is the difference between sequential and direct access methods?
8. What is definition of user file directory?
9. State few examples of systems using hierarchical directory structure.

Long-Answer Questions

1. Describe the concept of file, attributes of files, file types and file structure.
2. Elaborate on the difference between byte, record and tree sequence.
3. Explain briefly about the access methods of an operating systems.
4. What is directory? Explain its functions and different operations.
5. Discuss briefly about the architecture of the single-level directory, two-level directory and hierarchical directory.

10.9 FURTHER READINGS

Silberschatz, Abraham, Peter B. Galvin and Greg Gagne. 2008. *Operating System Concepts*, 8th Edition. New Jersey: John Wiley & Sons.

Tanenbaum, Andrew S. 2006. *Operating Systems Design and Implementation*, 3rd Edition. New Jersey: Prentice Hall.

Tanenbaum, Andrew S. 2001. *Modern Operating Systems*. New Jersey: Prentice Hall.

Deitel, Harvey M. 1984. *An Introduction to Operating Systems*. Boston (US): Addison-Wesley.

Stallings, William. 1995. *Operating Systems,* 2nd Edition. New Jersey: Prentice Hall.

Milenkovic, Milan. 1992. *Operating Systems: Concepts and Design*. New York: McGraw Hill Higher Education.

Mano, M. Morris. 1993. *Computer System Architecture*. New Jersey: Prentice Hall Inc.

UNIT 11 FILE STRUCTURE

NOTES

Structure

11.0 INTRODUCTION

The file structure refers to the internal structure of the file, that is, how a file is internally stored in the system. In this file structure, each file is made up of a sequence of 8-bit bytes having no fixed structure. The operating system does not attach any meaning to the file. It is the responsibility of the application program to include code to interpret the input file into an appropriate structure. In this file structure, a file consists of a sequence of fixed-length records where arbitrary number of records can be read from or written to a file. In this file structure, a file consists of a tree of disk blocks where each block holds a number of records of varied lengths. Each record contains a key field at a fixed position. A file system is a combination of methods and data structures which is also useful for any operating system to keep track of files on a disk or partition and states the way the files are organized on the disk. To attach a partition or device to the directory hierarchy you must mount its associated device file. To do this, a mount point has to be created. This is simply a directory where the device will be attached. This directory exists on a previously mounted device (with the exception of the root directory (/) which is a special case) and is empty. If the directory is not empty, then the files in the directory will not be visible while the device is mounted but will reappear after the device has been disconnected or uncounted.

In computer system file sharing is known as enabling access to digitally stored information that may be file having some program or a text document, electronic book or audio/video type multimedia files. If a document in form of file is distributed by other means, it is also a type of file sharing. A file system can be damaged due to various reasons such as, a system breakdown, theft, fire, lightning or any other extreme condition that is unavoidable and uncertain. It is very difficult to restore the data back in such conditions.

In this unit, your will study the basic concepts of file structure, file system mounting, file sharing and protection of files.

11.1 OBJECTIVES

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

- Describe the basics concepts of file structure
- Understand the concept and functions of file system mounting
- Discuss about the process of file sharing
- Analyse the advantages and disadvantages of protecting file system

11.2 DEFINITION OF FILE STRUCTURE

The file structure refers to the internal structure of the file, that is, how a file is internally stored in the system. The most common file structures recognized and enforced by different operating systems are as follows:

- **Byte Sequence**: In this file structure, each file is made up of a sequence of 8-bit bytes (Refer Figure 11.1(a)) having no fixed structure. The operating system does not attach any meaning to the file. It is the responsibility of the application program to include code to interpret the input file into an appropriate structure. This type of file structure provides flexibility to the user programs as they can store any type of data in the files and name these files in any way as per their convenience. UNIX operating systems support this type of file structure.
- **Record Sequence**: In this file structure, a file consists of a sequence of fixed-length records where arbitrary number of records can be read from or written to a file. The records cannot be inserted or deleted in the middle of a file. In this system, the read operation returns one record and the write operation appends or overwrites one record. CP/M operating system supports this type of scheme.

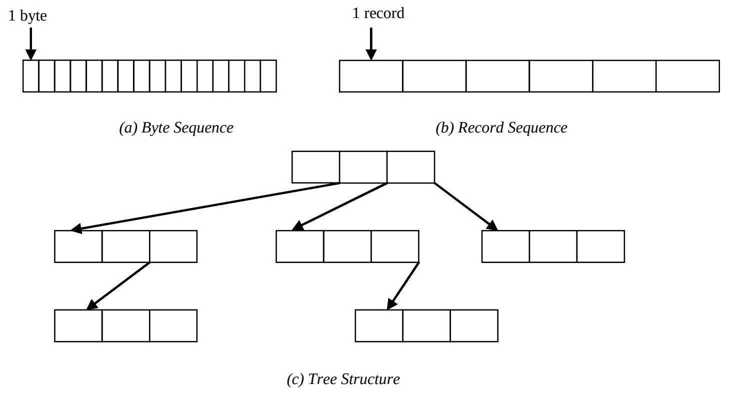


Fig. 11.1 *File Structures*

- **Tree Structure**: In this file structure, a file consists of a tree of disk blocks where each block holds a number of records of varied lengths. Each record contains a key field at a fixed position. The records are searched on key value and new records can be inserted anywhere in the file structure. This type of file structure is used on mainframe system where, it is called **ISAM** (**Indexed Sequential Access Method**).

  Regardless of the file structure used, all disk I/O take place in terms of blocks (physical records) where all blocks are of equal sizes and the size of a block is generally determined by the size of the sector. Since the disk space to a file is allocated in a number of blocks, some portion of the last block in a file is generally wasted. For instance, if each block is of 512 bytes, then a file of 3150 bytes would be allocated seven blocks, and the last 434 bytes will be wasted. The wastage of bytes to keep everything in units of blocks (instead of bytes) is internal fragmentation. Note that all file systems face internal fragmentation and with larger block sizes, there is more internal fragmentation.

11.3 FILE SYSTEM MOUNTING

A file system is a combination of methods and data structures which is also useful for any operating system to keep track of files on a disk or partition and states the way the files are organized on the disk. We can refer to a file system as a partition or disk that can be used to store files or the type of the file system.

It is important to differentiate the disk or partition and the file system which it contains. It should also be noted that many programs operate directly on the raw sectors of a disk or partition which also includes the program that create a file system. If a file system is already there it will be destroyed or corrupted. Most programs operate on a file system and therefore will not work on a partition that does not one or have has one of the wrong type.

The file system is a process which starts much before a partition or disk can be used as a file system, prior to this the bookkeeping data structures need to be written to the disk.

The general structure of most UNIX file systems is similar, although the exact details vary quite a bit. The central concepts of a file system includes the following:

- Superblock
- Inode
- Data block
- Directory block
- Indirection block

The superblock contains information about the file system as a whole, such as its size (the exact information here depends on the file system).

An inode contains all the information about a file, except its name. The name is stored in the directory together with the number of the inode. A directory entry consists of a filename and the number of the inode which represents the file. The inode contains the numbers of the data blocks that are used to store the data in the file.

There is space only for a few data block numbers in the inode, however, and if more are needed, more space for pointers to the data blocks is allocated dynamically.

These dynamically allocated blocks are indirect blocks; the name indicates that in order to find the data block, one has to find its number in the indirect block first.

Creating File Systems

A file system is either the device file associated with the partition or device or is the directory where the file system is mounted.

The basic requirement to mount a partition or to use a partition is that the file system must first be installed on it by using ext2. This is also a process of creating Inodes and data blocks.

This process of formatting the partition as similar to MS DOS `format` command in Windows. In Linux, the command to create a file system is called `mkfs`.

The command is issued in the following way:

```
mkfs  [-c] [ -t fstype ]  file system [ blocks ]
```

where:

`-c` forces a check for bad blocks

`-t fstype` specifies the file system type

The shorthand for most file system types.

The following is the example syntax:

```
     mkfs -t ext2 /dev/fd0   # Make a ext2 file system
on        a floppy
```

`mkfs -t ext2` can also be written as `mke2fs` or `mkfs.ext2` and `mkfs -t vfat` or `mkfs -t msdos` can also be written as `mkfs.vfat`, `mkfs.msdos` or `mkdosfs`

Note: Remember that creating a file system on a device with an existing file system will cause all data on the old file system to be erased.

Mounting and Unmounting

To attach a partition or device to the directory hierarchy you must mount its associated device file. To do this, a mount point has to be created. This is simply a directory where the device will be attached. This directory exists on a previously mounted device (with the exception of the root directory (/) which is a special case) and is empty. If the directory is not empty, then the files in the directory will not be visible while the device is mounted but will reappear after the device has been disconnected (or unmounted).

To mount a device, use the `mount` command:

```
mount [switches] device_file mount_point
```

In certain devices, `mount` will detect what type of file system exists on the device; however it is common to use mount in the following form:

```
mount [switches] -t file_system_type device_file
mount_point
```

Generally, only the root user can use the `mount` command mainly due to the fact that the device files are owned by root. For example, to mount the first partition on the second hard drive of the `/usr` directory and assuming it contained the ext2 file system, you would enter the command:

```
mount -t ext2 /dev/hdb1 /usr
```

A common device that is mounted is the floppy drive. A floppy disk generally contains the File Allocation Table (FAT), also known as `msdos` file system (but not always) and is mounted with the command:

```
mount -t msdos /dev/fd0 /mnt
```

Note that the floppy disk was mounted under the `/mnt` directory. This is because the `/mnt` directory is the usual place to temporarily mount devices.

To see what devices are currently mounted, simply type the command `mount`. Typing it on the system reveals:

```
/dev/hda3 on / type ext2 (rw)
/dev/hda1 on /dos type msdos (rw)
none on /proc type proc (rw)
/dev/cdrom on /cdrom type iso9660 (ro)
/dev/fd0 on /mnt type msdos (rw)
```

If we examine these commands, each line tells us what device file is mounted, where it is mounted, what file system type each partition is and how it is mounted (`ro` = read only, `rw` = read/write). Note the unusual entry on line. This is a special 'virtual' file system used by Linux systems to store information about the kernel, processes and current resource usages. It is actually part of the system's memory; in other words, the kernel sets aside an area of memory in which it stores information about the system. This area is mounted onto the file system so user programs can access this information.

The information in the proc file system can also be used to see what file systems are mounted by issuing the command:

NOTES

```
$ cat /proc/mounts
   /dev/root / ext2 rw 0 0
 proc /proc proc rw 0 0
/dev/hda1 /dos msdos rw 0 0
/dev/cdrom /cdrom iso9660 ro 0 0
/dev/fd0 /mnt msdos rw 0 0
```

To release a device and disconnect it from the file system, the `umount` command is used. It is issued in the following form:

```
umount device_file
```

or

```
umount mount_point
```

To release the floppy disk, you would issue the command:

```
umount /mnt
```

or

```
umount /dev/fd0
```

The point to note here is that either you should be the root user or a user with some special privileges to do this. You cannot unmount a `device/mount` point that is in use by a user (the user's current working directory is within the mount point) or is in use by a process. Nor can you unmount `devices/mount` points which in turn have devices mounted to them. This all leads to question, how does the system know which devices to mount when the OS boots?

As in UNIX, in Linux also there is a file which governs the behaviour of mounting devices at boot time. In Linux, this file is `/etc/fstab`.

So the next question is, what is in the file? An example line from the `fstab` file uses the following format:

```
   device_file mount_point file_system_type
 mount_options     [n] [n]
```

The first three fields are self explanatory; the fourth field, `mount_options` defines how the device will be mounted, which includes information of access mode `ro/rw`, execute permissions and other information. The fifth and sixth fields are used by the system utilities `dump` and `fsck,` respectively.

11.4 FILE SHARING AND LOCKING

Sharing means things available for more than one person. If an item is shared between many persons, it should be available to each of them. In computer system file sharing is known as enabling access to digitally stored information that may be file having some program or a text document, electronic book or audio/video type multimedia files. If a document in form of file is distributed by other means, it is also a type of file sharing. There are manual methods of sharing files by copying on some removable medium such as floppy diskettes or CD-ROM of a flash derive, from one system and then putting the same on other system that reads such files.

Such a method of sharing requires physical movement of the medium on which files are stored. Use of computer networks is one of the most efficient and quick means of file sharing. This requires installation of a centralized computer that acts as file server on the computer networks. The largest network, World Wide Web contains hyperlinked documents and clicking on this leads one to the required page. There are various types of networking, centralized as well as distributed, for sharing files. P2P (peer-to-peer) networking is also used to share files that is one most popular option sharing of file sharing on the Internet. Software that connects to a P2P network can be used for sharing files that reside on others computers known as peers. Once connected, desired file(s) can easily be downloaded, directly from other computer that is peer on the network.

Computer networking is done with the main objective of sharing information. Since information resides on computer in digital form, as a file, this information sharing is sharing of files. Many networks of computers, and also those of smaller networks, were created for sharing files. In year 1979, USENET was created. This network was initially based on UNIX to UNIX Communication Protocol (UUCP) enabling dial-up connections and was transported across Internet using specialized protocol in client-server architecture in Network News Transfer Protocol (NNTP). Main objective was for exchanging text based messages, but attachments were also possible that could be encoded for distribution subscribers who participate in the newsgroups of USENET.

Napster which is centralized but unstructured P2P system was released in the year 1999 in the month of June. Napster requires a central server for indexing as well as locating a peer. Napster claims to be the first P2P system for file sharing.

In the year 2000, Gnutella (March), Freenet (July) and eDonkey2000 (September) were released. Gnutella was first decentralized network for file sharing. In this network there was no central point and all connecting software had equal status. Thus, the problem of system going out of order if central point failed was not a problem. Freenet was first anonymity network. The eDonkey2000 was based on client and server software.

Software Kazaa and Poisoned, for Mac OS, were released in the year 2001. FastTrack network was having distributed architecture. More traffic was assigned to 'supernodes' for increasing routing efficiency. This network was encrypted and proprietary.

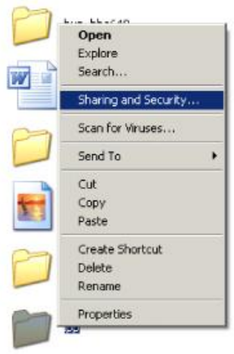
Sharing of Files/Folders on Windows

Windows operating system like any other platform provided software for file sharing.

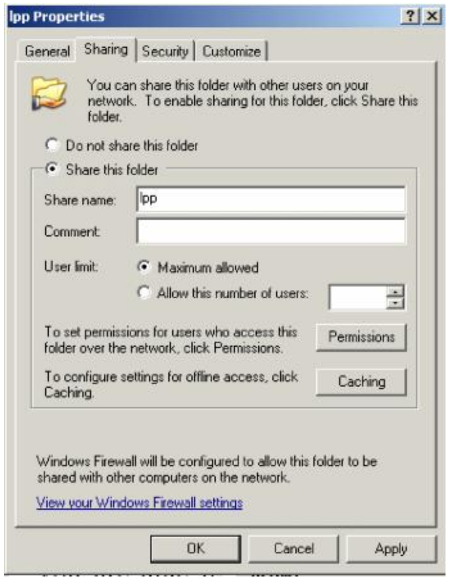
NOTES

Windows XP

Users of XP can share folder by making a right on the folder and then selecting **Sharing and Security** option.



After selecting **Sharing and Security** option from the dropdown menu a window is displayed that gives option for sharing folder. You find an option **Share this folder.** Check the radio button to enable sharing option. There are other tabs too for adding security features and putting a limit on number of users. See the screenshot given below:



lpp Properties

General | Sharing | Security | Customize

You can share this folder with other users on your network. To enable sharing for this folder, click Share this folder.

( ) Do not share this folder
(●) Share this folder

Share name: lpp

Comment:

User limit: (●) Maximum allowed
( ) Allow this number of users:

To set permissions for users who access this folder over the network, click Permissions. [Permissions]

To configure settings for offline access, click Caching. [Caching]

Windows Firewall will be configured to allow this folder to be shared with other computers on the network.

View your Windows Firewall settings

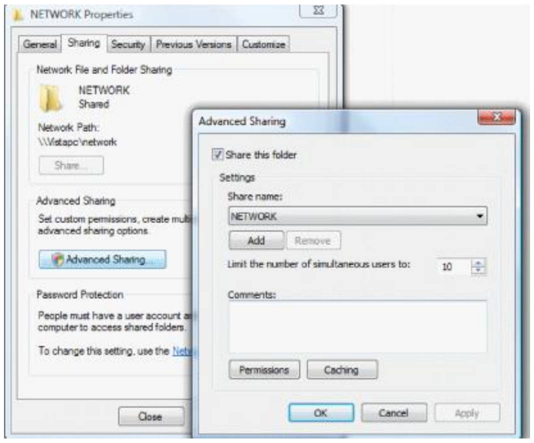
[OK] [Cancel] [Apply]

Select option as needed and then click apply. There will be a change in the folder icon and a hand is added.

Windows Vista

Sharing in Windows Vista is not much different. Make a right click on the folder and select **Share**. Select **Everybody** in the combo box and then click the **Add** button. The next step is changing permission level to the **Contributor** or **Co-owner**. This gives user who may access files and have permission to make changes in the shared files. The button **Share** shows exact network share for accessing shared folder. Here, in this example, it is **\\VISTAPC\NETWORK**. The link '**Show me all the network shares on this computer'** displays all previously folders that are shared. When you click the **Done** button, folder icon changes and makes it look that it is a shared folder.

Permissions as well as name of shared folder may be edited by making a right click on **Properties**, then taking tab **Sharing**, and **Advanced settings** button. To use advanced settings as default, sharing wizard should be disabled. For this, choose **Organize, Folder and Search Options**, take the tab **View** and then disable. You may take the option **'Use Sharing Wizard'.** This is recommended**.** A folder may be with various names. After addition of a new network name, there is a need of setting permissions too.


NETWORK Properties

General | Sharing | Security | Previous Versions | Customize

Network File and Folder Sharing

NETWORK
Shared

Network Path:
\\Vistapc\network

Share...

Advanced Sharing

Set custom permissions, create mult
advanced sharing options.

Advanced Sharing...

Password Protection

People must have a user account a
computer to access shared folders.

To change this setting, use the Netw

Close

Advanced Sharing

[✓] Share this folder

Settings

Share name:
NETWORK

Add | Remove

Limit the number of simultaneous users to: 10

Comments:

Permissions | Caching

OK | Cancel | Apply

Windows Vista provides multiple levels of security for file sharing. This may be changed in Network and Sharing Center that is available in Control Panel. At least, **File sharing** option should be turned on and if needed take the option **'Password protected sharing'.** The option **Public folder sharing** can also be activated. For using public folder files that are to be shared has to be placed in this folder.

If a firewall is set, it may block requests for accessing shared files.

Opening Shared Files/Folders

First ensure that all computers in the network are in the same workgroup. This can be done by going to **Control Panel Þ System** then selecting the tab **Computername** then **Change** button. It is good to use a name for the workgroup that is simple and short. Default name is MSHOME or WORKGROUP).

File Locking

File locking is a mechanism for restricting access to a file on computer by permitting only one process or user to access it at a time. In a shared access a user may just read the file but may not make any change in it. In such cases access by more than one user is permitted till there is no writing in the file. But if user is allowed to write in the file and changes are made in the file, simultaneous access can not be allowed. In such cases, restriction is put in access so that when one user is accessing the file and carrying out modification in it, access by other users at that time has to be restricted. This is done by a process known as file locking. Purpose of locking a file is to prevent *interceding update* scenario.

The Interceding Update Problem

If there are two processes named A and B such that both access same record in a file and following situation is faced:

(i) Process A, accesses a file to read a customer record that contains account information of a customer such as account balance and phone number.

(ii) Process B, subsequently reads same file and accesses same record and thus, has its own copy.

(iii) Process A, makes some changes in account balance in its copy of customer record with writing the record back to this file.

(iv) Process B, still has the original stale value for account balance in its own copy of customer record and updates phone number of the customer, subsequently writing customer record back to file.

(v) Process B, then writes its stale account balance to file, and this causes changes made by process A to be lost.

This problem can be prevented by file locking that enforces serialization of update processes for a given file. Most of the operating systems provide support for record locking. This means that individual records within a given file can be locked.

File locking is used in maintaining a database where access is serialized for the whole physical file in a database. This prevents other processes from accessing this file. But this can be more efficient than individual locking of a many regions in the file by removal of the overhead needed for acquisition and release of each lock.

Enough care should be exercised while using locks as poor use of suck locks may result in deadlock.

UNIX

In UNIX, open files or programs are not locked automatically. Different mechanisms for file locking are available in different versions UNIX. Many operating systems provide support for more than one type for maintaining compatibility. Two most common of these are `fcntl` and `flock`. There are some types of locks that are configured as mandatory file locks are advisory under UNIX, by default. Thus, cooperating processes may use locks for coordinating access to a file amongst them. But programs can ignore locks and file can still be accessed.

There are two types of locks; shared and exclusive. Using `fcntl`, different locks can be applied on different sections in a file, or it may be applied to entire file. Shared locks are acquired by huge number of processes at the same time, but when exclusive lock is applied, it is by one process, at one time and does not coexist in case of a shared lock. For acquiring a shared lock, a process has to wait until no process holds any exclusive locks. For acquiring an exclusive lock, a process has to wait until no process holds either kind of lock.

Shared locks are also known as 'read locks' whereas exclusive locks are termed 'write locks'. As already told, locks in UNIX is enforced, it is advisory. Thus a database may have 'shared writes' and 'exclusive writes'. Changing a field in place may be done under shared access, whereas exclusive access is required for rewriting and garbage-collection.

File locks are not based on file name, but these are based on **inode**. UNIX permits multiple names for referring to the same file. Thus, a combination of **inode** and non-mandatory locking provides great flexibility when file is accessed by multiple processes. The approach of cooperative locking may give rise to problems when one process is writing to a file ignoring locks set by other processes. For this reason, some UNIX variants and UNIX-like OS provide support for *mandatory locking* also.

Both `flock` and `fcntl` puzzle programmers that handle other operating systems, at times. Mandatory locks does not affect unlink function. Working of `flock` on network filesystems like NFS, depends on implementation. Prior to Linux 2.6.12 `flock` accesses NFS files that acts only locally. Kernel 2.6.12 and alter implement `flock` on NFS files by use of POSIX byte range locks. These locks becomes visible to other NFS clients implementing `fcntl`()/POSIX locks.

Locks are upgraded or downgraded by process before release of old lock or before application of new lock. If an application has downgraded an exclusive lock to a shared one and another application has been blocked and it waits for an exclusive lock, the latter application will have exclusive lock and but first application would be locked out.

All `fcntl` locks that has association with a file for a given process are removed when *any* file descriptor for that file is closed by that process, even if a lock was never requested for that file descriptor. A `fcntl` lock is not inherited by any child process. The `fcntl` close semantics are especially troublesome for applications making call to subroutine libraries for accessing files.

Linux

Linux 2.4 and above added notification on external changes to files with a mechanism known as `dnotify` using *F_NOTIFY* parameter of `fcntl`. This mechanism has been replaced another mechanism known as superior `inotify` that is superior and it was incorporated in Linux 2.6.13. Linux also provides *mandatory locking* using special "`mount(8)-o mand`" parameter for filesystem mounting. This is, however, used rarely.

Lock Files

NOTES

A system having similarity to file locking is normally used by shell scripts and programs for creating *lock files*. Lock files are files whose contents are of no relevance as its only purpose is to signal that some resource have been locked. Most of the time, a lock file is thought to be the best approach in case the resource that is to be controlled, is not a regular file at all and hence methods of locking files does not apply.

Care has to be taken in using lock files in ensuring that operations are atomic; either it is done fully or not done. For obtaining a lock, process should verify whether the lock file exists or not. If it does not exit then it should be created and other processes should be prevented from creating it during this period. There are different methods for achieving this. One is taking advantage of a system call or by creating lock file with a temporary name and subsequently moving it into place.

Certain Mozilla products make use of such file resource lock mechanism and use a temporary file having `parent.lock` as its name.

Unlocker Software

It is a utility for determining which process is using a file and if a user attempts to delete it, display is made of a list of processes with options on things that can be done to the processes. It may be kill task, unlock etc. It also gives a list of file function options such as rename, move, delete, etc.

Check Your Progress

1. Define the file structure.
2. What is meant by byte and record sequence file structure?
3. How do you define indexed sequential access method?
4. What are the central concepts required in a file system?
5. Write the command used for mounting any device?
6. State the command to release a device or any floppy disk and disconnect it from the file system.

11.5 FILE PROTECTION

The information stored in a system requires to be protected from the physical damage and unauthorized access. A file system can be damaged due to various reasons such as, a system breakdown, theft, fire, lightning or any other extreme condition that is unavoidable and uncertain. It is very difficult to restore the data back in such conditions. In some cases, when the physical damage is irreversible, the data can be lost permanently. Though, physical damage to a system is unavoidable, measures can be taken to safeguard and protect the data.

In a single-user system, protection can be provided by storing a copy of information on the disk to the disk itself, or to some other removable storage media such as, magnetic tapes and compact disc. If the original data on the disk is accidentally erased or overwritten, or becomes inaccessible because of its malfunctioning, the backup copy can be used to restore the lost or damaged data. Apart, from protecting the files from physical damage, the files in a system also need a protection mechanism to control improper access.

Types of Access

In a single-user system or in a system where users are not allowed to access the files of other users, there is no need for a protection mechanism. However, in a multi-user system where one user can access files of other users and the system is prone to improper access, a protection mechanism is a necessity. The access rights define the type of operation that a user is allowed to perform on a file. The different access rights that can be assigned to a particular user for a particular a file are as follows.

- Read: Allow only reading the file.
- Write: Allow writing or rewriting the file.
- Execute: Allow running the program or application.
- Append: Allow writing new information at the end of the file.
- Copy: Allow creating a new copy of the file.
- Rename: Allow renaming a file.
- Edit: Allow adding and deleting information from the file.
- Delete: Allow deleting the file and releasing the space.

There are many protection mechanisms, each having some advantages and disadvantages. However, the kind of protection mechanism used depends on the need and size of the organization. A smaller organization needs a protection mechanism different from the protection mechanism of a larger organization with large number of people accessing the files.

Access Control

To protect the files from improper accesses, the access control mechanism can follow either of the two approaches.

- **Password**: A password can be assigned to each file and only a user knowing the password can access the file. This scheme protects the file from unauthorized access. The main drawback of this approach is the large number of passwords which are practically very difficult to remember (for each files separately). However, if only one password is used for accessing all the files, then all the files become easily accessible if somebody knows the password. To balance the number of passwords in a system, some systems follow a scheme, where a user can associate a password with a subdirectory. This scheme allows a user to access all the files under a subdirectory with a single password. This scheme is also not very much safe. To overcome the drawbacks of these schemes, the protection must be provided at a more detailed level by using multiple passwords.

- **Access Control List**: In this approach, access to a file is provided on the basis of identity of the user. An **Access-Control List (ACL)** is associated with each file and directory, it stores user names and the type of access allowed to each user. When a user tries to access a file, the ACL is searched for that particular file. If that user is listed for the requested access, the access is allowed. Otherwise, the user is denied access to the file. This system of access control is effective but, in case when all the users want to read a file, the ACL for that particular file should list all users with read permission. The main drawback of this system is that, making such a list would be a tedious job when number of users is not known. Moreover, the list need to be dynamic in nature as the number of users will keep on changing, thus resulting in complicated space management.

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

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

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

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

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

Check Your Progress

7. Define the file sharing and locking concept in any file system.
8. What are the different access rights that can be assigned to a particular user for a particular file?
9. What do you understand by access control list?

11.6 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. The file structure refers to the internal structure of the file, that is, how a file is internally stored in the system.
2. In this file structure, each file is made up of a sequence of 8-bit bytes having no fixed structure.

   In this file structure, a file consists of a sequence of fixed-length records where arbitrary number of records can be read from or written to a file.
3. In this file structure, a file consists of a tree of disk blocks where each block holds a number of records of varied lengths. Each record contains a key field at a fixed position. The records are searched on key value and new records can be inserted anywhere in the file structure. This type of file structure is used on mainframe system where, it is called ISAM (Indexed Sequential Access Method).
4. The general structure of most UNIX file systems is similar, although the exact details vary quite a bit. The central concepts of a file system includes the following:
   - Superblock
   - Inode
   - Data block
   - Directory block
   - Indirection block
5. To mount a device, use the `mount` command:

```
mount [switches] device_file mount_point
```

NOTES

6. To release a device and disconnect it from the file system, the `umount` command is used. It is issued in the following form:

```
umount device_file
```

   or

```
umount mount_point
```

   To release the floppy disk, you would issue the command:

```
umount /mnt
```

   or

```
umount /dev/fd0
```

7. In computer system file sharing is known as enabling access to digitally stored information that may be file having some program or a text document, electronic book or audio/video type multimedia files. If a document in form of file is distributed by other means, it is also a type of file sharing. There are manual methods of sharing files by copying on some removable medium such as floppy diskettes or CD-ROM of a flash derive, from one system and then putting the same on other system that reads such files.

   A system having similarity to file locking is normally used by shell scripts and programs for creating *lock files*. Lock files are files whose contents are of no relevance as its only purpose is to signal that some resource have been locked. Most of the time, a lock file is thought to be the best approach in case the resource that is to be controlled, is not a regular file at all and hence methods of locking files does not apply.

8. The different access rights that can be assigned to a particular user for a particular a file are as follows.
   - **Read**: Allow only reading the file.
   - **Write**: Allow writing or rewriting the file.
   - **Execute**: Allow running the program or application.
   - **Append**: Allow writing new information at the end of the file.
   - **Copy**: Allow creating a new copy of the file.
   - **Rename**: Allow renaming a file.
   - **Edit**: Allow adding and deleting information from the file.
   - **Delete**: Allow deleting the file and releasing the space.

9. An **Access-Control List (ACL)** is associated with each file and directory, it stores user names and the type of access allowed to each user. When a user tries to access a file, the ACL is searched for that particular file. If that user is listed for the requested access, the access is allowed. Otherwise, the user is denied access to the file. This system of access control is effective but, in case when all the users want to read a file, the ACL for that particular file should list all users with read permission.

11.7 SUMMARY

NOTES

- The file structure refers to the internal structure of the file, that is, how a file is internally stored in the system.
- In this file structure, each file is made up of a sequence of 8-bit bytes having no fixed structure.
- In this file structure, a file consists of a sequence of fixed-length records where arbitrary number of records can be read from or written to a file.
- In this file structure, a file consists of a tree of disk blocks where each block holds a number of records of varied lengths. Each record contains a key field at a fixed position. The records are searched on key value and new records can be inserted anywhere in the file structure. This type of file structure is used on mainframe system where, it is called ISAM (Indexed Sequential Access Method).
- A file system is a combination of methods and data structures which is also useful for any operating system to keep track of files on a disk or partition and states the way the files are organized on the disk. We can refer to a file system as a partition or disk that can be used to store files or the type of the file system.
- The general structure of most UNIX file systems is similar, although the exact details vary quite a bit. The central concepts of a file system includes the following:
  - Superblock
  - Inode
  - Data block
  - Directory block
  - Indirection block
- A file system is either the device file associated with the partition or device or is the directory where the file system is mounted.
- To attach a partition or device to the directory hierarchy you must mount its associated device file. To do this, a mount point has to be created. This is simply a directory where the device will be attached. This directory exists on a previously mounted device (with the exception of the root directory (/) which is a special case) and is empty. If the directory is not empty, then the files in the directory will not be visible while the device is mounted but will reappear after the device has been disconnected (or unmounted).

NOTES

- A common device that is mounted is the floppy drive. A floppy disk generally contains the File Allocation Table (FAT), also known as `msdos` file system (but not always) and is mounted with the command:

  ```
  mount -t msdos /dev/fd0 /mnt
  ```

- In computer system file sharing is known as enabling access to digitally stored information that may be file having some program or a text document, electronic book or audio/video type multimedia files. If a document in form of file is distributed by other means, it is also a type of file sharing. There are manual methods of sharing files by copying on some removable medium such as floppy diskettes or CD-ROM of a flash derive, from one system and then putting the same on other system that reads such files.
- Napster which is centralized but unstructured P2P system was released in the year 1999 in the month of June. Napster requires a central server for indexing as well as locating a peer. Napster claims to be the first P2P system for file sharing.
- In such cases, restriction is put in access so that when one user is accessing the file and carrying out modification in it, access by other users at that time has to be restricted. This is done by a process known as file locking. Purpose of locking a file is to prevent *interceding update* scenario.
- A system having similarity to file locking is normally used by shell scripts and programs for creating *lock files*. Lock files are files whose contents are of no relevance as its only purpose is to signal that some resource have been locked. Most of the time, a lock file is thought to be the best approach in case the resource that is to be controlled, is not a regular file at all and hence methods of locking files does not apply.
- It is a utility for determining which process is using a file and if a user attempts to delete it, display is made of a list of processes with options on things that can be done to the processes. It may be kill task, unlock etc. It also gives a list of file function options such as rename, move, delete, etc.

11.8 KEY WORDS

- **Superblock:** The Superblock contains information about the file system as a whole, such as its size.
- **Inode:** An inode contains all the information about the file, except its name.
- **File Allocation Table (FAT):** It is also known as msdos file system and mounted with the command.

- **File locking:** It is a mechanism for restricting access to a file on computer by permitting only one process or user to access it at a time.
- **Append:** Allow writing new information at the end of the file.
- **Group:** A set of users who need similar access permission for sharing the file is a group, or work group.

NOTES

11.9 SELF-ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. What do you understand by byte sequence?
2. Define the significance of ISAM in tree sequence file structure.
3. Why is it important to differentiate the disk or partition and file system?
4. How do we create file systems?
5. Define mounting and unmounting of file systems.
6. What is meant by file sharing on Windows Vista?
7. Define the term file locking.
8. What are the different types of access in file protection where the information is to be stored?
9. What is a group and universe in terms of access control?

Long-Answer Questions

1. Elaborate the concept of file structure, types and advantages of having file structure.
2. Illustrate on the significance of superblock, data block, indirection block and Inode in file system mounting.
3. Write short notes on each of the following:

   (a) File sharing on Windows.

   (b) Opening shared files/folders.
4. Describe how a file locking and state the situation faced in the interceding update problem.
5. Explain the computing of file locking in Linux, UNIX and other unlocker software's.
6. Discuss the file protection and the importance of studying access control in it along with the help of a diagram.

11.10 FURTHER READINGS

NOTES

Silberschatz, Abraham, Peter B. Galvin and Greg Gagne. 2008. *Operating System Concepts*, 8th Edition. New Jersey: John Wiley & Sons.

Tanenbaum, Andrew S. 2006. *Operating Systems Design and Implementation*, 3rd Edition. New Jersey: Prentice Hall.

Tanenbaum, Andrew S. 2001. *Modern Operating Systems*. New Jersey: Prentice Hall.

Deitel, Harvey M. 1984. *An Introduction to Operating Systems*. Boston (US): Addison-Wesley.

Stallings, William. 1995. *Operating Systems*, 2nd Edition. New Jersey: Prentice Hall.

Milenkovic, Milan. 1992. *Operating Systems: Concepts and Design*. New York: McGraw Hill Higher Education.

Mano, M. Morris. 1993. *Computer System Architecture*. New Jersey: Prentice Hall Inc.

UNIT 12 IMPLEMENTING FILE SYSTEMS

NOTES

Structure

12.0 INTRODUCTION

Every operating system imposes a file system that helps to organize, manage and retrieve data on the disk. The file system resides permanently on the disk. The design of the file system involves two key issues. The first issue includes defining a file and its attributes, operations that can be performed on a file, and the directory structure. The second issue includes creating data structures and algorithms for mapping the logical file system onto the secondary storage devices. There are several on-disk and in-memory structures that are used to implement a file system. Depending on the operating system and the file system, these may vary, but the general principles remain the same.

In this unit, you study the basics concept of file system structure and methods of implementing the file system.

12.1 OBJECTIVES

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

- Introduce the basic concepts of file system structure
- Define the file system implementation
- Discuss about the various blocks of on-disk structures, such as boot control block and partition control block
- Explain the definition and functions of in-memory structures

12.2 FILE SYSTEM STRUCTURE

Every operating system imposes a file system that helps to organize, manage and retrieve data on the disk. The file system resides permanently on the disk. The design of the file system involves two key issues. The first issue includes defining a file and its attributes, operations that can be performed on a file, and the directory structure. The second issue includes creating data structures and algorithms for mapping the logical file system onto the secondary storage devices.

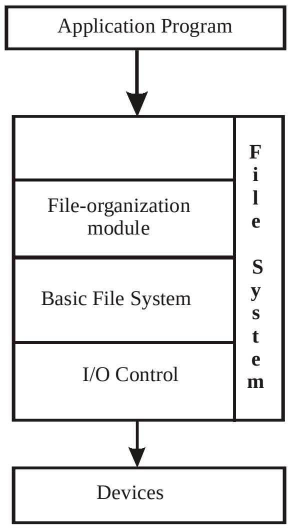


Fig. 12.1 *File System Layers*

Figure 12.1 shows that the file system is made up of different layers, where each layer represents a level. Each level uses the features of the lower levels to create new features that are used by higher levels. When the user interacts with the file system (through system commands), the I/O request occurs on a device. The **device driver** in turn generates an interrupt to transfer information between the disk drive and the main memory. High-level commands make the input to device driver, and its output is low-level hardware specific instructions. The device driver also writes specific bit pattern to special locations in the I/O controller's memory which tells the controller about the device location to act upon and the actions to be taken.

The next component in the file system is the **basic file system** that issues generic (general) commands to the appropriate device driver to read and write physical blocks on the disk. The physical blocks are referred to by the numeric disk address (for example, drive 1, cylinder 43, track 4, sector 16).

Next level component of the file system is the **file-organization module** which organizes the files. It knows the physical block address (the actual address) and logical block address (the relative address), allocation method, and location of a file. Using this information, it translates the logical address into physical address and helps basic file system to transfer files. The other function of the file organization module is to keep track of the free space and provide this space for allocation when needed.

The **logical file system** at the next level manages all the information about a file except the actual data (content of the file). It manages the directory structure and provides the necessary information to the file-organization module when the file name is given. It maintains file structure using the **File Control Block (FCB)** that stores information about a file such as ownership, permissions, and location of the file content. Protection and security is also taken care by the logical file system.

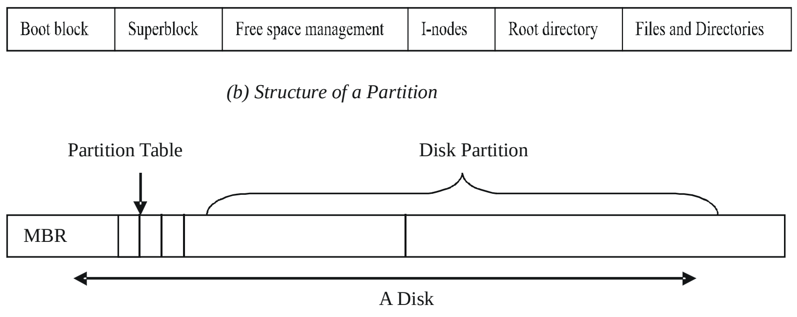
Apart from the physical disk drive, there are other removable devices also, such as CD-ROM, floppy disk, pen drives and other storage devices attached to a system. Each of these devices has a standard file system structure imposed by its manufactures. For instance, most CD-ROMs are written in *High Sierra* format, which is a standard format agreed upon by CD-ROM manufacturers. The standard file system for the removable media makes them interoperable and portable for use on different systems. Apart from the file systems for removable devices each operating system has one (or more) disk-based file system. UNIX system uses the UNIX File System (UFS) as a base. Windows NT supports disk file system formats, such as, FAT, FAT32, and NTFS (or Windows NT file system), along with CD-ROM, DVD, and floppy-disk file system formats.

12.3 FILE SYSTEM IMPLEMENTATION

There are several **on-disk** and **in-memory** structures that are used to implement a file system. Depending on the operating system and the file system, these may vary, but the general principles remain the same. Many of the structures that are used by most of the operating system are discussed here. The on-disk structures include:

- **Boot Control Block**: It contains enough information that the system needs to boot the operating system from that partition. Though, not all the partitions in a disk contain a bootable operating system, every partition starts with a boot block. Having one block in each partition reserved for a boot block is a good idea because any partition can have operating system in the future. If the partition does not contain any operating system this block can be empty. In UNIX File System (UFS), this block is called the **boot block** and in Windows (NTFS), it is called the **partition boot sector**.
- **Partition Control Block**: It is a table that stores key information related to partition, number and size of blocks in the partition, free block count and free block pointers, and FCB pointers and free FCB count. In UFS this is called **superblock**, in NTFS, it is **Master File Table**.

Further, each partition has a directory structure, with root directory at the top. The directory structure helps to manage and organize the files in the file system. A new FCB is allocated when creating a new file that stores information such as file permissions, ownership, size, and location of the data blocks. In UFS this is called the **i-node** (an array of data structures, one for each file). In NTFS, this information is kept within the Master File Table, which uses a relational database structure, where each row stores information about a file. Figure 12.2 shows an example of file system layout based on UNIX file system.



(b) Structure of a Partition

(a) Structure of a Disk

***Fig. 12.2** File System Layout*

The in-memory structure helps in improving performance of the file system. The in-memory structures include:

- **In-Memory Partition Table**: It stores information about each mounted partition.
- **In-Memory Directory Structure**: It stores the information about the recently accessed directories.
- **System-Wide Open-File Table**: It contains a copy of the FCB of each open file and a count of the number of processes that have the file open.
- **Per-Process Open-File Table**: It contains a pointer to the corresponding entry in the system-wide open-file table along with some related information.

When the open call passes a file name to the file system, the directory structure is searched for the given file name. When the file is found, its FCB is copied into the system-wide open-file table, and the count is incremented.

Once the system-wide open-file table is updated, an entry is made in the per-process open-file table. This entry includes a pointer to the appropriate entry in the system-wide open-file table, a pointer to the position in the file where the next read or write will occur, and the mode in which the file is open. The open call returns a pointer to the appropriate entry in the per-process file system table. This pointer is than used to perform all the operations as long as the file is open. It means that until a file is closed, all the operations are carried out on the open-file table.

When a process closes the file, the corresponding entry from the per-process open-file table is removed and the open count of the system-wide entry is decremented. The value in the count indicates the number of users who have opened the file currently. Thus, when this value becomes 0, the updated file information is copied back to the disk-based structures and the entry is removed from the system-wide open-file table.

Check Your Progress

1. What does the design of file system involves?
2. State the function of a basic file system.
3. Define the file control block of the file system.
4. What is the difference between boot block and partition boot sector?
5. Define i-node.
6. What does the in-memory structure of any file system includes?

12.4 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. The design of the file system involves two key issues. The first issue includes defining a file and its attributes, operations that can be performed on a file, and the directory structure. The second issue includes creating data structures and algorithms for mapping the logical file system onto the secondary storage devices.
2. In the file system is the **basic file system** that issues generic (general) commands to the appropriate device driver to read and write physical blocks on the disk. The physical blocks are referred to by the numeric disk address (for example, drive 1, cylinder 43, track 4, sector 16).
3. The **File Control Block (FCB)** that stores information about a file such as ownership, permissions, and location of the file content. Protection and security is also taken care by the logical file system.
4. **Boot Control Block**: It contains enough information that the system needs to boot the operating system from that partition. Though, not all the partitions in a disk contain a bootable operating system, every partition starts with a boot block. Having one block in each partition reserved for a boot block is a good idea because any partition can have operating system in the future. If the partition does not contain any operating system this block can be empty. In UNIX File System (UFS), this block is called the **boot block** and in Windows (NTFS), it is called the **partition boot sector**.
5. The directory structure helps to manage and organize the files in the file system. A new FCB is allocated when creating a new file that stores information such as file permissions, ownership, size, and location of the data blocks. In UFS this is called the **i-node** (an array of data structures, one for each file). In NTFS, this information is kept within the Master File Table, which uses a relational database structure, where each row stores information about a file.

6. The in-memory structure helps in improving performance of the file system. The in-memory structures include:
   - **In-Memory Partition Table**: It stores information about each mounted partition.
   - **In-Memory Directory Structure**: It stores the information about the recently accessed directories.
   - **System-Wide Open-File Table**: It contains a copy of the FCB of each open file and a count of the number of processes that have the file open.
   - **Per-Process Open-File Table**: It contains a pointer to the corresponding entry in the system-wide open-file table along with some related information.

12.5 SUMMARY

- The design of the file system involves two key issues. The first issue includes defining a file and its attributes, operations that can be performed on a file, and the directory structure. The second issue includes creating data structures and algorithms for mapping the logical file system onto the secondary storage devices.
- In the file system is the basic file system that issues generic (general) commands to the appropriate device driver to read and write physical blocks on the disk. The physical blocks are referred to by the numeric disk address (for example, drive 1, cylinder 43, track 4, sector 16).
- The logical file system at the next level manages all the information about a file except the actual data (content of the file). It manages the directory structure and provides the necessary information to the file-organization module when the file name is given.
- The File Control Block (FCB) that stores information about a file such as ownership, permissions, and location of the file content. Protection and security is also taken care by the logical file system.
- There are several on-disk and in-memory structures that are used to implement a file system. Depending on the operating system and the file system, these may vary, but the general principles remain the same.
- Boot Control Block: It contains enough information that the system needs to boot the operating system from that partition. Though, not all the partitions in a disk contain a bootable operating system, every partition starts with a boot block. Having one block in each partition reserved for a boot block is a good idea because any partition can have operating system in the future. If the partition does not contain any operating system this block can be empty. In UNIX File System (UFS), this block is called the boot block and in Windows (NTFS), it is called the partition boot sector.

- Partition Control Block: It is a table that stores key information related to partition, number and size of blocks in the partition, free block count and free block pointers, and FCB pointers and free FCB count. In UFS this is called superblock, in NTFS, it is Master File Table.
- The directory structure helps to manage and organize the files in the file system. A new FCB is allocated when creating a new file that stores information such as file permissions, ownership, size, and location of the data blocks. In UFS this is called the i-node (an array of data structures, one for each file). In NTFS, this information is kept within the Master File Table, which uses a relational database structure, where each row stores information about a file.
- The in-memory structure helps in improving performance of the file system. The in-memory structures include:
  - o In-Memory Partition Table: It stores information about each mounted partition.
  - o In-Memory Directory Structure: It stores the information about the recently accessed directories.
  - o System-Wide Open-File Table: It contains a copy of the FCB of each open file and a count of the number of processes that have the file open.
  - o Per-Process Open-File Table: It contains a pointer to the corresponding entry in the system-wide open-file table along with some related information.
- When a process closes the file, the corresponding entry from the per-process open-file table is removed and the open count of the system-wide entry is decremented. The value in the count indicates the number of users who have opened the file currently. Thus, when this value becomes 0, the updated file information is copied back to the disk-based structures and the entry is removed from the system-wide open-file table.

12.6 KEY WORDS

- **File Control Block (FCB):** It maintains file structure using the File Control Block (FCB) that stores information about a file such as ownership, permissions, and location of the file content.
- **Partition Control Block:** It is a table that stores key information related to partition, number and size of blocks in the partition, free block count and free block pointers, and FCB pointers and free FCB count.
- **System-Wide Open-File Table:** It contains a copy of the FCB of each open file and a count of the number of processes that have the file open.
- **Per-Process Open-File Table:** It contains a pointer to the corresponding entry in the system-wide open-file table along with some related information.

12.7 SELF-ASSESSMENT QUESTIONS AND EXERCISES

NOTES

Short-Answer Questions

1. What do you understand by key issues in file system structure?
2. State the role of a device driver in layers of file system.
3. How do we differentiate between the basic and logical file system?
4. What is the significance of *High Sierra* format in terms of manufacturers under standard file system structure?
5. Why do we use superblock in any file?
6. What is the difference between in-memory partition and system-wide open-file table?
7. What happens once the system-wide open-file table is updated?

Long-Answer Questions

1. Describe the concept of file system structure.
2. Explain briefly about the file organization module.
3. Elaborate on the difference between boot control block and partition control block in file system implementation.
4. What is master file table? Explain the difference between on-disk and in-memory file system implementation structure.

12.8 FURTHER READINGS

Silberschatz, Abraham, Peter B. Galvin and Greg Gagne. 2008. *Operating System Concepts*, 8th Edition. New Jersey: John Wiley & Sons.

Tanenbaum, Andrew S. 2006. *Operating Systems Design and Implementation*, 3rd Edition. New Jersey: Prentice Hall.

Tanenbaum, Andrew S. 2001. *Modern Operating Systems*. New Jersey: Prentice Hall.

Deitel, Harvey M. 1984. *An Introduction to Operating Systems*. Boston (US): Addison-Wesley.

Stallings, William. 1995. *Operating Systems*, 2nd Edition. New Jersey: Prentice Hall.

Milenkovic, Milan. 1992. *Operating Systems: Concepts and Design*. New York: McGraw Hill Higher Education.

Mano, M. Morris. 1993. *Computer System Architecture*. New Jersey: Prentice Hall Inc.

UNIT 13 DIRECTORY IMPLEMENTATION

NOTES

Structure

13.0 INTRODUCTION

The most commonly used directory-management algorithms are linear list and hash table. The linear list method organizes a directory as a collection of fixed size entries, where each entry contains a (fixed-length) file name, a fixed structure to store the file attributes, and pointers to the data blocks. While implementing directories, the search time required to locate a directory entry corresponding to a file is always a major concern. To considerably reduce the search time, a more complex data structure known as a hash table along with the linear list of directory entry is used. An important function of the file system is to manage the space on the secondary storage. It includes keeping track of the number of disk blocks allocated to files and the free blocks available for allocation.

Whenever a new file is created, it is allocated some space from the available free space on the disk. The free space can be either the space on the disk that is never used for allocation or the space left by the deleted files. The file system maintains a free-space list that indicates the free blocks on the disk.

In this unit, you will study the concepts of directory implementation, allocation methods and free space management.

13.1 OBJECTIVES

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

- Explain how a directory gets implemented
- Understand the concept of hash table

- Discuss various allocation methods, such as contiguous allocation, linked allocation and indexed allocation
- Describe the significance of free space management process

NOTES

13.2 INTRODUCTION TO DIRECTORY IMPLEMENTATION

The efficiency, performance and reliability of a file system are directly related to the directory-management and directory-allocation algorithms selected for a file system. The most commonly used directory-management algorithms are linear list and hash table.

Linear List

The linear list method organizes a directory as a collection of fixed size entries, where each entry contains a (fixed-length) file name, a fixed structure to store the file attributes, and pointers to the data blocks (Refer Figure 13.1). The linear list method stores all file attributes compiled as a single directory entry and uses a linear search to search a directory entry from the list of entries. It means to search an entry in the list of entries; each entry (starting from the first entry in the directory) is examined one by one until the desired entry is found. This is simple to program; however, with extremely large directories the search becomes very slow, which is a major disadvantage of this method. MS DOS and Windows operating system use this approach for implementing directories.

Directory

Itles1	attributes
language	attributes
magazine	attributes
sports	attributes
computer	attributes

Fig 13.1 *Example of Linear List Directory Entry (in MS DOS and Windows)*

Some systems follow a variation of linear list method for recording the file information. For example, in UNIX, the directory entry consists of two fields: the file name and the **I-node** (**index-node**) number. The i-node number contains the disk address of the I-node structure that stores the file attributes and the address of the file's data blocks (Refer Figure 13.2). With this approach, the size of the directory entry is very small. This approach has certain advantages over the linear list of directory.

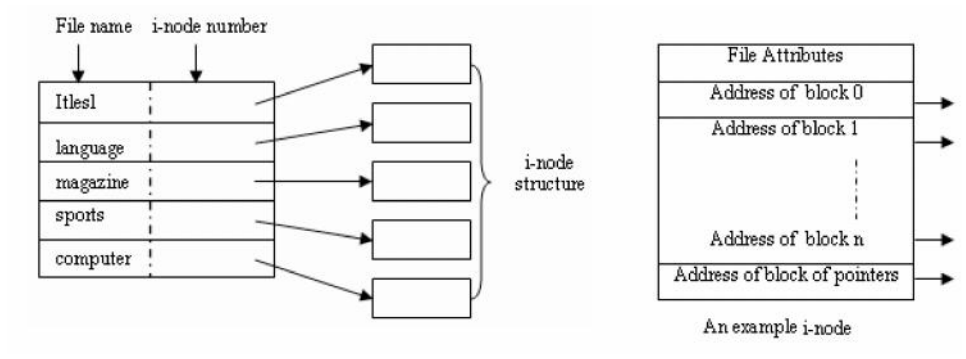


Fig. 13.2 *Example of Linear List Directory Entry (in UNIX)*

In both cases, when the user sends a request to create a new file, the directory is searched to check whether any other file has the same name or not. If no other file has the same name, the memory will be allocated and an entry for the same would be added at the end of the directory. To delete a file, the directory is searched for the file name and if the file is found, the space allocated to it is released. The delete operation results in free space that can be reused. To reuse this space, it can be marked with a used–unused bit, a special name can be assigned to it, such as all-zeros or it can be linked to a list of free directory entries.

When performing the file operations, one thing is common that is searching the directory for a particular file. The search technique applied greatly influences the time taken to make the search and in turn the performance and efficiency of the file system. As discussed, with long directories, a linear search becomes very slow and takes O(n) comparisons to locate a given entry, where n is the number of all entries in a directory. To decrease the search time, the list can be sorted and a binary search can be applied. Applying binary search reduces the average search time but keeping the list sorted is a bit difficult and time-consuming, as directory entries have to be moved with every creation and deletion of file.

Hash Table

While implementing directories, the search time required to locate a directory entry corresponding to a file is always a major concern. To considerably reduce the search time, a more complex data structure known as a **hash table** along with the linear list of directory entry is used.

A **hash table** is a data structure, with 0 to n–1 table entries, where n is the total number of entries in the table (Refer Figure 13.3). It uses a hash function to compute a hash value (a number between 0 to n–1) based on the file name. For instance, file name is converted into integers from 0 to n–1 and this number is divided by n to get the remainder. Then, the table entry corresponding to the hash value (value of remainder) is checked. If the entry space is unused, then a pointer to the file entry (in the directory) is placed there. However, if the entry is already in use, we can say a **collision** has occurred. In such a situation, a linked list of entries that hash to the same value is created and the table entry is made to point to the header of the linked list.

NOTES

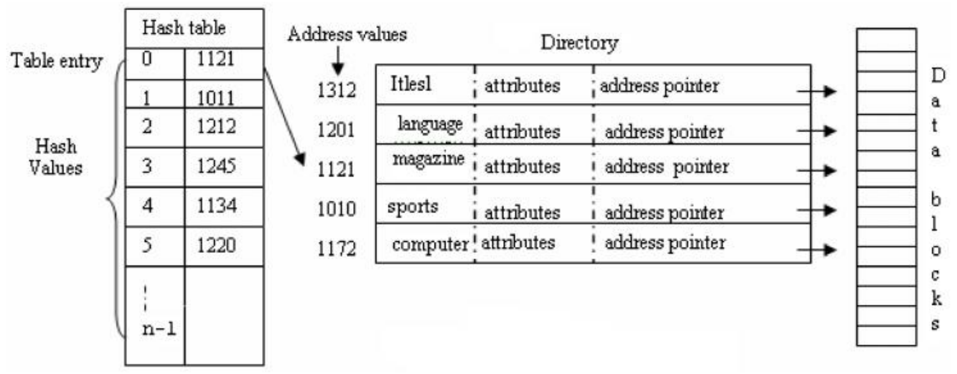

Hash table
Table entry
Hash Values
0 1121
1 1011
2 1212
3 1245
4 1134
5 1220
n-1
Address values
Directory
1312 Itles1 attributes address pointer
1201 language attributes address pointer
1121 magazine attributes address pointer
1010 sports attributes address pointer
1172 computer attributes address pointer
Data blocks

Fig. 13.3 *Hash Table Linked to a Directory*

We follow the same process to search a file. The file name is hashed to locate the hash table entry and then all the entries on the chain are checked to see if the file name exists. If the name is not found in the chain, the file is not present in the directory.

Despite all the advantages the main disadvantage of this approach is the creation of long overflow chains if the hash function is not distributing the values uniformly. Now, in order to search an entry, a linear search in the long overflow chain may be required, which increases the access time. In addition, the administration of hash table becomes a complex process. Some system copies the directory entries for frequently accessed files in the cache memory. This saves the time required to re-read information from the disk, thus, making the file access fast.

13.3 ALLOCATION METHODS

An important function of the file system is to manage the space on the secondary storage. It includes keeping track of the number of disk blocks allocated to files and the free blocks available for allocation.

The two main issues related to disk space allocation are:

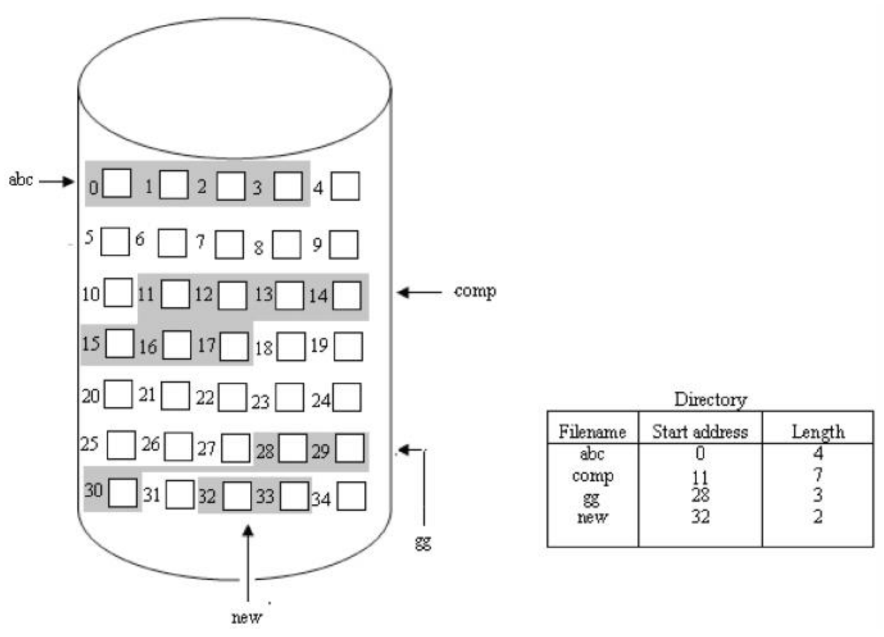
- Optimum utilization of the available disk space.
- Fast accessing of files.

The widely used methods for allocation of disk space are, contiguous, linked, and indexed. For discussing these different allocation strategies, a file is considered to be a sequence of blocks and all I/O operations on a disk occur in terms of blocks.

Contiguous Allocation

NOTES

In contiguous allocation, each file is allocated contiguous blocks on the disk, that is, one after the other (Refer Figure 13.4). Assuming only one job is accessing the disk, once the first block, say b, is accessed, accessing block b+1 requires no head movement normally. Head movement is required only when the head is currently at the last sector of a cylinder and moves to the first sector of the next cylinder; the head movement is only one track. Therefore, number of seeks, and thus, seek time in accessing contiguously allocated files is minimal. This improves the overall file system performance.



Directory

Filename	Start address	Length
abc	0	4
comp	11	7
gg	28	3
new	32	2

Fig. 13.4 *Example of Contiguous Allocation*

It is relatively simple to implement the file system using contiguous allocation method. The directory entry for each file contains the file name, the disk address of the first block, and the total size of the file.

Contiguous allocation supports both sequential and direct access to a file. For sequential access, the file system remembers the disk address of the last block referenced, and when required, reads the next block. For direct access to block b of a file that starts at location L, the block L+b can be accessed immediately.

Contiguous allocation has a significant problem of external fragmentation. Initially, the disk is free, and each new file can be allocated contiguous blocks starting from the block where the previous file ends. When a file is deleted, it leaves behind some free blocks in the disk. This is not a problem until we have contiguous blocks to allocate to a new file at the end of disk. However, with time, the disk will become full, and at that time the free blocks are fragmented throughout the disk. One solution to this problem is **compaction**, which involves moving the blocks on the disk to make all free space into one contiguous space. Compaction is quite time consuming as it may take hours to compact a large hard disk that uses contiguous allocation. Moreover, normal operations are not permitted generally during compaction.

An alternative to expensive compaction is to reuse the space. For this, we need to maintain a list of holes (an unallocated segment of contiguous blocks). In addition, we must know the final size of a file at the time of its creation so that a sufficiently large hole can be allocated to it. However, determining the file size in advance is generally difficult and allocating either too little or too more space to a file is a problem. If we allocate more space than what it needs, we end up wasting precious memory. On the other hand, if we allocate too little space than what is needed, we may not extend the file, since the blocks on both sides of the file may be allocated to some other files. One possibility to extend the space is to terminate the user program and then the user must restart it with more space. However, restarting the user program repeatedly may again be expensive. Alternatively, system may find the larger hole, copy the contents of the file to the new space, and then release the previous space. This can be done repeatedly until required space is available contiguously in the disk. Moreover, the user program need not be restarted and the user is also not informed about this. However, the task is again time-consuming.

Linked Allocation

The file size generally tends to change (grow and shrink) over time. The contiguous allocation of such files results in the several problems. Linked allocation method overcomes all the problems of contiguous allocation method.

In the linked allocation method, each file is stored as a linked list of disk blocks. The disk blocks are generally scattered throughout the disk, and each disk block stores the address of the next block. The directory entry contains the file name and the address of the first and last blocks of the file (Refer Figure 13.5).

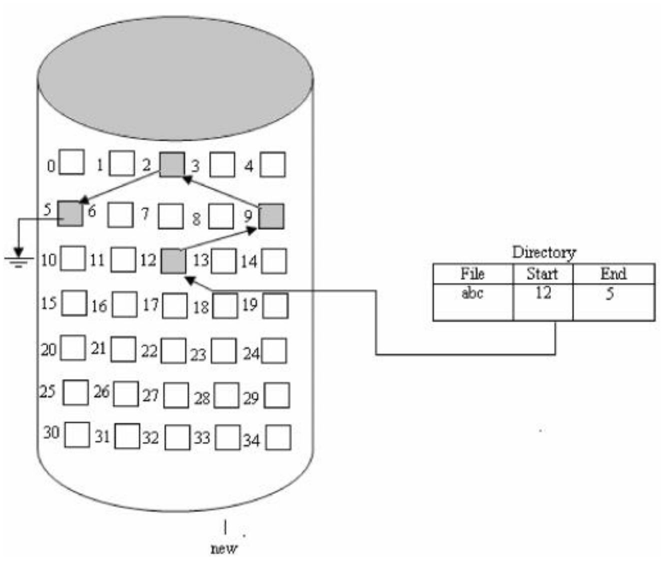


Fig. 13.5 Example of Linked Allocation

Figure 13.5 shows the linked allocation for a file. A total of four disk blocks are allocated to the file. The directory entry indicates that the file starts at block 12. It then continues at block 9, block 2, and finally ends at block 5.

NOTES

The simplicity and straightforwardness of this method makes it easy to implement. The linked allocation results in optimum utilization of disk space as even a single free block between the used blocks can be linked and allocated to a file. This method does not come across with the problem of external fragmentation, thus, compaction is never required.

The main disadvantages of using linked allocation are the slow access speed, disk space utilization by pointers, and low-reliability of the system. As this method provides only sequential access to files, therefore, to find out the *n*th block of a file, the search starts at the beginning of the file and follows the pointer until the *n*th block is found. For a very large file, the average turn around time is high.

In linked allocation, maintaining pointers in each block requires some disk space. The total disk space required by all the pointers in a file becomes substantial, which results in more space required by each file. The space required to store pointers can otherwise be used to store the information. To overcome this problem, contiguous blocks are grouped together as a **cluster**, and allocation to files takes place as clusters rather than blocks. Clusters allocated to a file are then linked together. Having a pointer per cluster rather than per block reduces the total space needed by all the pointers. This approach also improves the disk throughput as fewer disk seeks are required. However, this approach may increase internal fragmentation because having a partially full cluster wastes more space than having a partially full block.

The linked allocation is also not very reliable. Since disk blocks are linked together by pointers, a single damaged pointer may prevent us from accessing the file blocks that follow the damaged link. Some operating systems deal with this problem by creating special files for storing redundant copies of pointers. One copy of file is placed in main memory to provide faster access to disk blocks. Other redundant pointers files help in safer recovery.

Indexed Allocation

There is one thing common to both linked and indexed allocation, that is, non-contiguous allocation of disk blocks to the files. However, they follow different approaches to access the information on the disk. Linked allocation supports sequential access, whereas, indexed allocation supports sequential as well as direct access.

NOTES

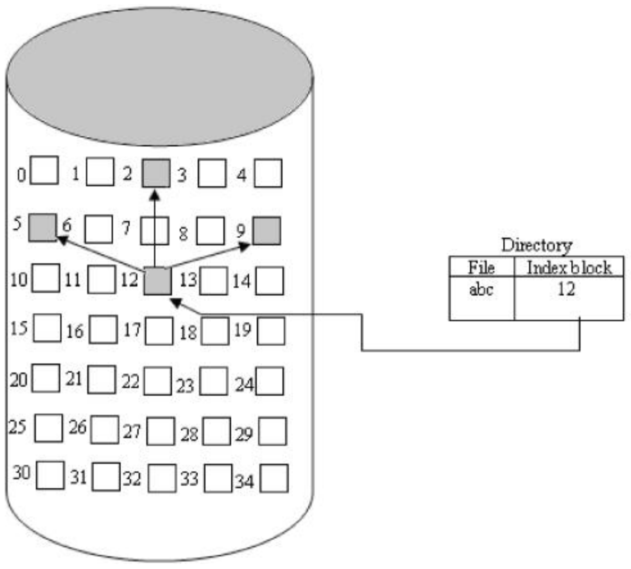


Fig. 13.6 *Example of Indexed Allocation*

In indexed allocation, the blocks of a file are scattered all over the disk in the same manner as they are in linked allocation. However, here the pointers to the blocks are brought together at one location known as the **index block**. Each file has an index block (Refer Figure 13.6), which is an array of disk-block pointers (addresses). The *k*th entry in the index block points to the *k*th disk block of the file. To read the *k*th disk block of a file, the pointer in the *k*th index block entry is used to find and read the desired block. The index block serves the same purpose as a page map table does in the paged memory systems.

The main advantage of indexed allocation is the absence of external fragmentation, since, any free blocks on the disk may be allocated to fulfill a demand for more space. Moreover, the index can be used to access the blocks in a random manner.

When compared to linked allocation, the pointer overhead in indexed allocation is comparatively more. This is because with linked allocation, a file of only two blocks uses a total of 8 bytes for storing pointers (assuming each pointer require 4 bytes of space). However, with indexed allocation, the system must allocate one block (512 bytes) of disk space for storing pointers. This results in wastage of 504 bytes of the index block as only 8 bytes are used for storing the two pointers.

Clearly, deciding the size of index block is a major issue because too large block may result in wastage of memory and too small index block limits the size of largest file in the system. If 4 bytes are used to store a pointer to a block, then a block of size 512 bytes can store up to 128 pointers, thus, the largest file in that system can have 65536 bytes (512 × 128) of information. However, we may have a file which exceeds the size limit of 65536 bytes. To solve this problem,

multi-level indexes, with two, three, or four levels of indexes may be used. The two level indexes, with 128 × 128 addressing is capable of supporting file sizes up to 8 MB and the three level indexes with 128 × 128 × 128 addressing can support file size of up to 1 GB.

The performance of the file system can be greatly enhanced by placing the frequently accessed index blocks in cache memory. This reduces the number of disk accesses required to retrieve the address of the target block.

13.4 FREE SPACE MANAGEMENT

Whenever a new file is created, it is allocated some space from the available free space on the disk. The free space can be either the space on the disk that is never used for allocation or the space left by the deleted files. The file system maintains a **free-space list** that indicates the free blocks on the disk. To create a file, the free-space list is searched for the required amount of space, and the space is then allocated to the new file. The newly allocated space is removed from the free-space list. Similarly, when a file is deleted, its space is added to the free-space list. The various methods used to implement free-space list are bit vector, linked list, grouping and counting.

Bit Vector

Bit vector also known as **bit map** is widely used to keep track of the free blocks on a disk. To track all the free and used blocks on a disk with total *n* blocks, a bit map having *n* bits is required. Each bit in a bit map represents a disk block where, a 0 in a bit represents an allocated block and a 1 in a bit represent a free block. Figure 13.7 shows bit map representation of a disk.

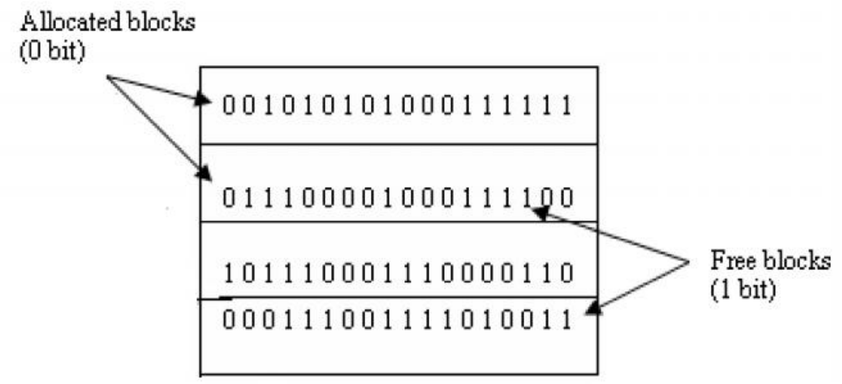


Fig. 13.7 *Bit Map Representation*

The bit map method for free-space list implementation is very simple. For instance, if a file requires four free blocks using contiguous allocation method, free blocks 12, 13, 14, and 15 (the first four free blocks on the disk that are adjacent to each other) may be allocated. However, for the same file using linked or indexed allocation, the file system may use free blocks 2, 4, 6, and 8 for allocation to the file.

The bit map is usually kept in main memory to optimize the search for free blocks. However, for systems with larger disks, keeping the complete bit map in main memory becomes difficult. For a 2 GB disk with 512-byte blocks, a bit map of 512 KB would be needed.

Linked List

The linked list method for free-space management creates a linked list of all the free blocks on the disk. A pointer to the first free block is kept in a special location on the disk and is cached in the memory. This first block contains a pointer to next free block, which contains a pointer to next free block, and so on. Figure 13.8 shows the linked list implementation of free blocks, where block 2 is the first free block on the disk, which points to block 4, which points to block 5, which points to block 8, which points to block 9, and so on.

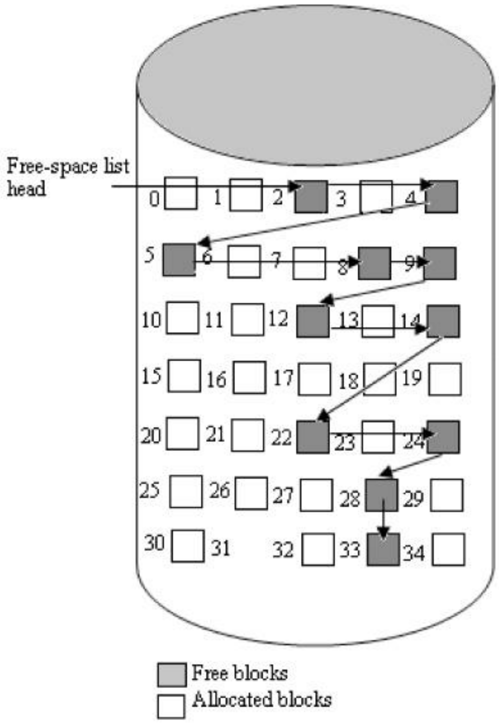


Fig. 13.8 *Free Space Management through Linked List*

Linked list implementation for managing free-space list requires additional space. This is because a single entry in linked list requires more disk space to store a pointer as compared to 1 bit in bit map method. In addition, traversing the free-list requires substantial I/O operations as we have to read each and every block, which takes a lot of time.

Grouping

Grouping is a modification to the free-list approach in the sense that instead of having a pointer in each free block to the next free block, we have pointers for first n free blocks in the first free block. The first n–1 blocks are then actually free. The nth block contains the address of next n free blocks, and so on. A major advantage of this approach is that the addresses of many free disk blocks can be found with only one disk access.

Counting

When contiguous or clustering approach is used, creation or deletion of a file allocates or de-allocates multiple contiguous blocks. Therefore, instead of having addresses of all the free blocks, as in grouping, we can have a pointer to the first free block and a count of contiguous free blocks that follow the first free block. With this approach, the size of each entry in the free-space list increases because an entry now consists of a disk address and a count, rather than just a disk address. However, the overall list will be shorter, as count is usually greater than 1.

Check Your Progress

1. How the linear list method does organizes a directory?
2. What happens to the directory when the user sends the request to create, delete or to reuse the space in any file?
3. Define the term hash table.
4. State the two main issues related to disk space allocation.
5. Define the contiguous, linked and indexed allocation in methods of directory allocation.
6. What do you understand by free-space list?

13.5 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. The efficiency, performance and reliability of a file system are directly related to the directory-management and directory-allocation algorithms selected for a file system. The most commonly used directory-management algorithms are linear list and hash table.
2. In both cases, when the user sends a request to create a new file, the directory is searched to check whether any other file has the same name or not. If no other file has the same name, the memory will be allocated and an entry for the same would be added at the end of the directory. To delete a file, the directory is searched for the file name and if the file is found, the space

allocated to it is released. The delete operation results in free space that can be reused. To reuse this space, it can be marked with a used–unused bit, a special name can be assigned to it, such as all-zeros or it can be linked to a list of free directory entries.

3. A hash table is a data structure, with 0 to n–1 table entries, where n is the total number of entries in the table. It uses a hash function to compute a hash value (a number between 0 to n–1) based on the file name. For instance, file name is converted into integers from 0 to n–1 and this number is divided by n to get the remainder.
4. The two main issues related to disk space allocation are:
   - Optimum utilization of the available disk space.
   - Fast accessing of files.
5. Contiguous allocation supports both sequential and direct access to a file. For sequential access, the file system remembers the disk address of the last block referenced, and when required, reads the next block. For direct access to block b of a file that starts at location L, the block L+b can be accessed immediately.

   In the linked allocation method, each file is stored as a linked list of disk blocks. The disk blocks are generally scattered throughout the disk, and each disk block stores the address of the next block. The directory entry contains the file name and the address of the first and last blocks of the file.

   In indexed allocation, the blocks of a file are scattered all over the disk in the same manner as they are in linked allocation. However, here the pointers to the blocks are brought together at one location known as the index block. Each file has an index block (Refer Figure 13.6), which is an array of disk-block pointers (addresses). The *k*th entry in the index block points to the *k*th disk block of the file.
6. The file system maintains a free-space list that indicates the free blocks on the disk. To create a file, the free-space list is searched for the required amount of space, and the space is then allocated to the new file. The newly allocated space is removed from the free-space list. Similarly, when a file is deleted, its space is added to the free-space list. The various methods used to implement free-space list are bit vector, linked list, grouping and counting.

   Bit vector also known as bit map is widely used to keep track of the free blocks on a disk. To track all the free and used blocks on a disk with total *n* blocks, a bit map having *n* bits is required. Each bit in a bit map represents a disk block where, a 0 in a bit represents an allocated block and a 1 in a bit represent a free block.

13.6 SUMMARY

NOTES

- The efficiency, performance and reliability of a file system are directly related to the directory-management and directory-allocation algorithms selected for a file system. The most commonly used directory-management algorithms are linear list and hash table.
- The linear list method organizes a directory as a collection of fixed size entries, where each entry contains a (fixed-length) file name, a fixed structure to store the file attributes, and pointers to the data blocks.
- In both cases, when the user sends a request to create a new file, the directory is searched to check whether any other file has the same name or not. If no other file has the same name, the memory will be allocated and an entry for the same would be added at the end of the directory. To delete a file, the directory is searched for the file name and if the file is found, the space allocated to it is released. The delete operation results in free space that can be reused. To reuse this space, it can be marked with a used–unused bit, a special name can be assigned to it, such as all-zeros or it can be linked to a list of free directory entries.
- A hash table is a data structure, with 0 to n–1 table entries, where n is the total number of entries in the table. It uses a hash function to compute a hash value (a number between 0 to n–1) based on the file name. For instance, file name is converted into integers from 0 to n–1 and this number is divided by n to get the remainder.
- An important function of the file system is to manage the space on the secondary storage. It includes keeping track of the number of disk blocks allocated to files and the free blocks available for allocation.
- The two main issues related to disk space allocation are:
  - Optimum utilization of the available disk space.
  - Fast accessing of files.
- In contiguous allocation, each file is allocated contiguous blocks on the disk, that is, one after the other.
- In the linked allocation method, each file is stored as a linked list of disk blocks. The disk blocks are generally scattered throughout the disk, and each disk block stores the address of the next block. The directory entry contains the file name and the address of the first and last blocks of the file.

- In indexed allocation, the blocks of a file are scattered all over the disk in the same manner as they are in linked allocation. However, here the pointers to the blocks are brought together at one location known as the index block. Each file has an index block (Refer Figure 13.6), which is an array of disk-block pointers (addresses). The *k*th entry in the index block points to the *k*th disk block of the file.
- The file system maintains a free-space list that indicates the free blocks on the disk. To create a file, the free-space list is searched for the required amount of space, and the space is then allocated to the new file. The newly allocated space is removed from the free-space list. Similarly, when a file is deleted, its space is added to the free-space list. The various methods used to implement free-space list are bit vector, linked list, grouping and counting.
- Bit vector also known as bit map is widely used to keep track of the free blocks on a disk. To track all the free and used blocks on a disk with total *n* blocks, a bit map having *n* bits is required. Each bit in a bit map represents a disk block where, a 0 in a bit represents an allocated block and a 1 in a bit represent a free block.
- The linked list method for free-space management creates a linked list of all the free blocks on the disk. A pointer to the first free block is kept in a special location on the disk and is cached in the memory.
- Grouping is a modification to the free-list approach in the sense that instead of having a pointer in each free block to the next free block, we have pointers for first n free blocks in the first free block. The first n–1 blocks are then actually free.
- When contiguous or clustering approach is used, creation or deletion of a file allocates or de-allocates multiple contiguous blocks. Therefore, instead of having addresses of all the free blocks, as in grouping, we can have a pointer to the first free block and a count of contiguous free blocks that follow the first free block.

13.7 KEY WORDS

- **Hash table:** A complex data structure with 0 to n-1 table entries, where n is the total number of entries. A hash table is used along with the linear list of directory to reduce search time.
- **Free-space list:** A list that indicates the free blocks on a disk.
- **Bit vector:** Bit vector also known as bit map is widely used to keep track of the free blocks on a disk.

- **Grouping:** Grouping is a modification to the free-list approach in the sense that instead of having a pointer in each free block to the next free block, we have pointers for first n free blocks in the first free block.

NOTES

13.8 SELF-ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. What are the commonly used directory-management algorithms?
2. Define index-node.
3. What is the significance of linking hash table to any directory?
4. How do we differentiate between the contiguous allocation and linked allocation method?
5. State the definition of bit vector, linked list, grouping and counting.

Long-Answer Questions

1. Explain the significance of directory implementation and why do we study different directory-management algorithms.
2. Briefly illustrate the functioning of various allocation methods giving suitable examples.
3. Elaborate on the method of free- space management.
4. Discuss the importance of grouping and counting in terms of the free-list approach.

13.9 FURTHER READINGS

Silberschatz, Abraham, Peter B. Galvin and Greg Gagne. 2008. *Operating System Concepts*, 8th Edition. New Jersey: John Wiley & Sons.

Tanenbaum, Andrew S. 2006. *Operating Systems Design and Implementation*, 3rd Edition. New Jersey: Prentice Hall.

Tanenbaum, Andrew S. 2001. *Modern Operating Systems*. New Jersey: Prentice Hall.

Deitel, Harvey M. 1984. *An Introduction to Operating Systems*. Boston (US): Addison-Wesley.

Stallings, William. 1995. *Operating Systems*, 2nd Edition. New Jersey: Prentice Hall.

Milenkovic, Milan. 1992. *Operating Systems: Concepts and Design*. New York: McGraw Hill Higher Education.

Mano, M. Morris. 1993. *Computer System Architecture*. New Jersey: Prentice Hall Inc.

NOTES

UNIT 14 SECONDARY STORAGE STRUCTURE

NOTES

Structure

14.0 INTRODUCTION

In computing, mass storage refers to the storage of large amounts of data in a persisting and machine-readable fashion. In general the term is uses as large in relation to contemporaneous disk drives, but it has been used large in relation to RAM as for example with floppy disks. A mass storage controller communicates between a mass storage device and a computer system. It acts as a driver to allow the systems to exchange information and commands with each other. Disks are used to store bulk data as secondary storage for modern computers. A hard disk has a device controller which translates numerical addresses into head movements. The device controller gets the instructions for execution from the operating system. Disk Management is a Microsoft Windows utility first introduced in Windows XP as a replacement for the f-disk command. It enables users to view and manage the disk drives installed in their computer and the partitions associated with those drives.

Disk scheduling is done by operating systems to schedule I/O requests arriving for the disk. Disk scheduling is also known as I/O scheduling. The most widely known algorithm for scheduling the request is SSTF or LOOK algorithm. The main goal of this and most of other disk scheduling algorithm is to reduce the number of disk head movement thereby reducing the seek time. To achieve this, we should have a fast access time and large disk bandwidth.

In this unit, you will study the overview of mass storage structure, disk management, disk structure, disk attachment and disk scheduling algorithms.

14.1 OBJECTIVES

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

- Analyse the overview of mass storage structure
- Describe the disk management
- Explain the disk structure
- Define the term disk attachment
- Understand the concept and functions of disk scheduling and its algorithms required

14.2 OVERVIEW OF MASS STORAGE STRUCTURE

The most common properties used for characterizing and evaluating the storage unit of the computer system are expressed below:

1. **Storage Capacity:** Represents the size of the memory. It is the amount of data that can be stored in the storage unit. Primary storage units have less storage capacity as compared to secondary storage units. The capacity of internal memory and main memory can be expressed in terms of the number of words or bytes.The capacity of external or secondary storage, on the other hand, is measured in terms of bytes.
2. **Storage Cost:** Cost is another key factor that is of prime concern in a memory system . It is usually expressed per bit. It is obvious that lower costs are desirable. It is worth noting that with the increase in the access time for memories , the cost decreases.
3. **Access Time:** The time required to locate and retrieve the data from the storage unit. It is dependent on the physical characteristics and access mode used for that device. Primary storage units have faster access time as compared to secondary storage units.
4. **Access Mode:** Memory comprises various locations. Access mode is the mode in which information is accessed from the memory. The user can access memory devices in any of the following ways:
   (a) **Random Access Memory (RAM):** This refers to the mode in which any memory location can be accessed in any order in the same amount of time. Ferrite and semiconductor memories, which usually constitute the primary storage or main memory, are of this nature.
   (b) **Sequential Access:** Memories that can be accessed only in a pre-defined sequence are sequential access memories. Here, since sequencing through other locations precedes the arrival at a desired location, the access time varies according to the location. Information

on a sequential device can be retrieved in the same sequence in which it was stored. Songs stored on a cassette, that can be accessed only one by one, is an example of sequential access. Typically, magnetic tapes are sequential access memory.

(c) **Direct Access:** Sometimes, the information is neither accessed randomly nor in sequence but something in between. In this type of access, a separate read/write head exists for each track, and on a track , you can access the information serially. This semi-random mode of access exists in magnetic disks.

NOTES

5. **Permanence of Storage:** If the storage unit can retain the data even after the power is turned off or interrupted, it is termed as non-volatile storage. And, if the data is lost once the power is turned off or interrupted, it is called volatile storage. It is obvious from these properties that the primary storage units of the computer systems are volatile, while the secondary storage units are non-volatile. A non-volatile storage is definitely more desirable and feasible for storage of large volumes of data.

14.2.1 Primary Storage Devices

1. Static and Dynamic RAM

The main memory is the central storage unit in a computer system. It is a relatively large and fast memory. It is used to store programs and data during computer operations. The principal technology used for the main memory is based on semiconductor-integrated circuits. There are two possible modes in which the integrated circuit RAM chips are available. These modes are: *static* and *dynamic*.

The Static RAM (SRAM) stores binary information using clocked sequential circuits. The stored information remains valid only as long as power is applied to the unit. On the other hand, Dynamic RAM (DRAM) stores binary information in the form of electric charges that are applied to capacitors inside the chip. The stored charge on the capacitors tends to discharge with time and so must be periodically recharged by refreshing the dynamic memory. The dynamic RAM offers larger storage capacity and reduced power consumption. Therefore, large memories use dynamic RAM, while static RAM is mainly used for specialized applications.

The different types of memory discussed above are both of the read/write type. What about a memory where only one of the operations is possible, e.g., if we allow only reading from the memory (cannot change the information in the memory)? The memory might have some major importance; like an important bit of the computer's operating system which normally does not change can be stored in this kind of memory. Such a memory is called ROM (Read Only Memory).

2. Read-Only Memory (ROM)

Most of the memory in a general-purpose computer is made of RAM integrated circuit chips, but a portion of the memory may be constructed using ROM chips. Originally, RAM was used to refer to random-access memory, but now we use the term read/write memory to distinguish it from read-only memory (since ROM is also random access). RAM is used for storing the bulk of the programs and data that are subject to change, while ROM is used to store programs that are permanently resident in the computer and do not change once the production of the computer is completed.

Among other things, the ROM portion of the main memory is used for storing an initial program called the bootstrap loader, whose function is to get the computer software operating when power is turned on. Since RAM is volatile, its contents are destroyed when power is turned off. The contents of ROM remain unchanged even after the power is turned off and on again.

Read-only memories can be manufacturer-programmed or user-programmed. When the data is burnt into the circuitry of the computer by the manufacturer, it is called manufacturer-programmed ROM; for example, a personal computer manufacturer may store the boot program permanently in the ROM chip of the computers manufactured by it. Such chips are supplied by the manufacturer and are not modifiable by users. This is an inflexible process and requires mass production. Thus, a new kind of ROM, known as Programmable Read-only Memory (PROM), was designed. This is also non-volatile in nature. It can be written only once using some special equipment. The supplier or the customer can electrically perform the writing process in PROM.

In both ROM and PROM, you can perform write operations only once and you cannot change whatever you have written. But what about the cases where you mostly read but also write a few times? Another type of memory chip called EPROM (Erasable Programmable Read-only Memory) was developed to take care of such situations. EPROMs are typically used by R&D personnel who experiment by changing micro-programs on the computer system to test their efficiency.

Further, EPROM chips are of two types: EEPROMs (Electrically EPROM) in which high voltage electric pulses are used to erase stored information, and UVEPROM (Ultra Violet EPROM) in which stored information is erased by exposing the chip for a while to ultraviolet light.

Figure 14.1 summarizes the various types of Random Access Memories.

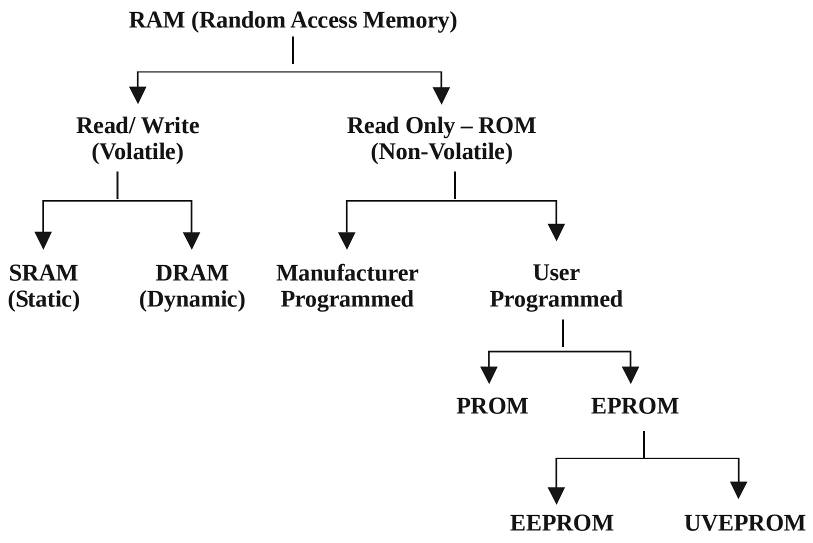


Fig. 14.1 *Types of Random Access Memories*

NOTES

14.2.2 Secondary Storage Devices

As discussed earlier, RAM is a volatile memory with limited storage capacity. The cost of RAM is also relatively higher as compared to secondary memory. Logic dictated that a relatively cheaper media, showing some sort of permanence of storage, be used. As a result, additional memory called *external* or *auxiliary memory* or *secondary storage* is used in most computers.

The magnetic medium was found to be long lasting and fairly inexpensive, and therefore, became an ideal choice for large storage requirements. The use of magnetic tapes and disks as storage media is very common. As optical technology is advancing, optical disks are turning out to be one of the major secondary storage devices.

1. Cache Memory

Cache memories are small, fast memories placed between the CPU and the main memory. They are faster than the main memory with access times closer to the speed of the CPU. Caches are fast, but very expensive. So, they are used only in small quantities. As an example, caches of size 64K, 128K are normally used in PC-386 and PC-486, which can have 1 to 8 MB of RAM or even more. Cache memories provide fast speed memory retrieval without compromising the size of the memory.

If the memory is so small, would it be advantageous to increase the overall speed of memory? This can be answered with the help of a phenomenon known as locality of reference. Let us examine what this means.

If we analyse a large number of typical programs,we would find that memory references at any given interval of time tend to be confined to a few localized areas in the memory. This phenomenon is called the property of locality of reference. This is true because most of the programs typically contain iterative loops (like 'for' or 'while' loops). During the execution of such programs, the same set of instructions (within the loop) are executed many times. The CPU repeatedly refers to the set of instructions in the memory that constitute the loop. Every time a specific subroutine is called, its set of instructions is fetched from the memory. Thus loops and subroutines tend to localize the references to memory for fetching instructions.

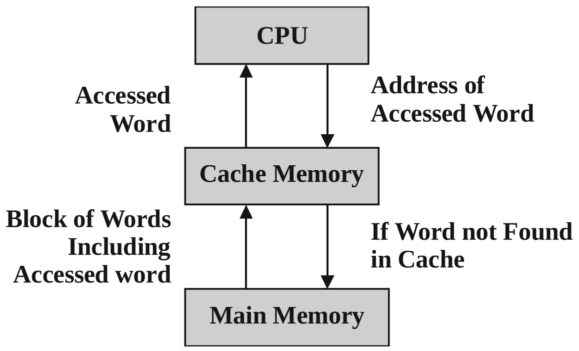

CPU
Accessed Word
Address of Accessed Word
Cache Memory
Block of Words Including Accessed word
If Word not Found in Cache
Main Memory

Fig. 14.2 *Functioning of the Cache Memory*

Based on the locality of reference, we understand that the cache has a copy of certain portions of main memory. First, the memory read or write operation is checked with the cache, and in case of availability of desired data in the cache, it is used directly by the CPU. Otherwise, a block of words is read from main memory to cache and the word is then used by the CPU from cache. Figure 14.2 explains the functioning of the cache memory.

3. Magnetic Tapes

Magnetic tapes are used for storing files of data that are sequentially accessed or not used very often and are stored off-line. They are typically used as backup storage for archiving of data.

In case of magnetic tapes, a tape (plastic ribbon usually 1/2 inch or 1/4 inch wide and 50 to 2400 feet long) is wound on a spool and its other end is threaded manually on a take-up spool. The Beginning Of the Tape (BOT) is indicated by a metal foil called a *marker*. When a write command is given, a block of data (records are usually grouped in blocks of two or more) is written on the tape. The next block is then written after a gap (called Inter Block Gap or IBG). A series of blocks are written in this manner. The End Of Tape (EOT) is indicated by an end-of-tape marker which is a metal foil stuck in the tape. After the data is written, the tape is rewound and kept ready for reading. Figure 14.3(a) shows the data organization on a magnetic tape. Figure 14.3(b) and 14.3(c) show magnetic tape reels and magnetic tape cartridges.

NOTES

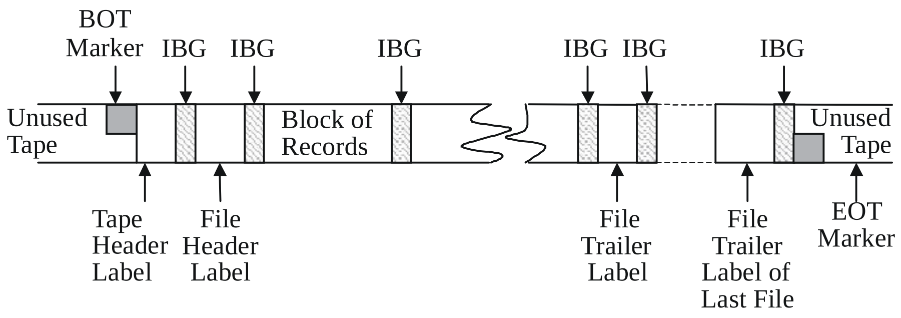


Fig. 14.3(a) *Data Organization on a Magnetic Tape*

Fig. 14.3(b) *Magnetic Tape Reel*

Fig. 14.3(c) *Magnetic Tape Cartridge*

The tape is read sequentially, i.e., data can be read in the order in which the data has been written. This implies that if the desired record is at the end of the tape, all the earlier records have to be read before it is reached. A typical example of a tape can be seen in a music cassette, where, to listen to the fifth song one must listen to, or traverse, the earlier four songs. The access time of information stored on tape is therefore very high as compared to that stored on a disk.

The storage capacity of the tape depends on its data recording density and the length of the tape. The data recording density is the amount of data that can be stored or the number of bytes that can be stored per linear inch of tape. The data recording density is measured in BPI (Bytes Per Inch).

Thus,

Storage Capacity of a Tape = Data Recording Density × Length of Tape

It is worth noting that the actual storage capacity for storing user data is much less owing to the file header labels, file trailer labels, BOT and EOT markers, and the use of IBGs.

Some commonly used magnetic tapes are:

- 1/2 inch tape reel
- 1/2 inch tape cartridge
- 1/4 inch streamer tape
- 4 mm DAT (Digital Audio Tape) – typical capacity of 4GB to 14 GB

4. Magnetic Disks

Magnetic disks are direct-access medium, and so are the most popular online secondary storage devices. Direct-access devices are also called random-access devices because information is literally available at random or in any order. There is direct access to any location on the device. Thus, approximately equal access time is required for each location. An example of this is a music CD where if you wish to listen to the fifth song, you can directly select the fifth track without having to fast forward the previous four.

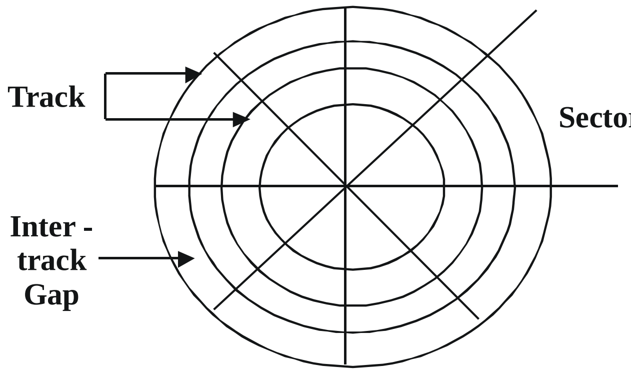


Fig. 14.4 *Logical Layout of a Magnetic Disk*

A magnetic disk refers to a circular plate made of metal or plastic and coated with magnetized material (ss shown in Figure 14.4). Often both sides of the disk are used. Data is recorded on the disk in the form of magnetized and non-magnetized spots (not visible to the naked eye) representing 1s and 0s.

(i) Disk Devices

A disk drive is a peripheral device used to store and collect information. It can be removable or fixed, high capacity or low capacity, fast or slow speed, and magnetic or optical.

Structurally, a drive is the object inside which a disk(s) is either permanently or temporarily stored. While a disk contains the media on which the data is stored, a drive contains the machinery and circuitry required for implementing the read / write operations on the disk.

The disk looks literally like a flat circular disk. The computer writes information onto the disk, where it is stored in the same form as it is stored on a cassette tape. Disks, as such, are just magnetically coated rolls or circular disks which are divided into sectors and tracks. The data is accordingly stored and numbered with respect to the track and sector number on the disk. Only the structure of the media is different from one to another. Examples of removable disk drives are DVD, CD-ROM, floppy disk drive, etc. The examples of non-removable disk drives include hard disk.

The method of accessing data could be sequential access (Magnetic Tape Drives) or random access (HDD, DVD), where the read/write head can directly go to any location on the disk and perform action.

(ii) Floppy Disks

The disks used with a floppy disk drive are small, removable disks made of plastic, and coated with magnetic recording material. There are two sizes commonly used, with diameters of 5.25 and 3.5 inches.

- The 5.25 inch disk is a 5.25 inch diameter floppy disk. Earlier, such disks recorded data only on one side and were called Single-Sided (SS) disks. Today both the surfaces are used for recording and are called Double-Sided (DS) disks. These are available in two capacities — Double Density (DD), and High Density (HD), where density refers to the number of bits that can be stored per square inch area.

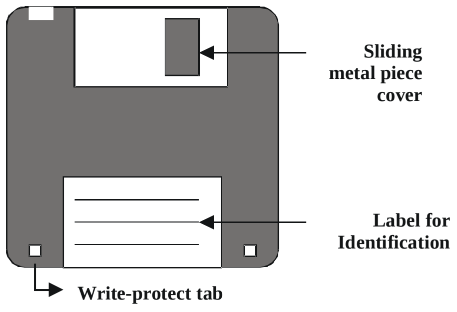


Fig. 14.5 *A 3.5 Inch Floppy Disk*

NOTES

- The 3.5 inch disk is a disk of 3½ inch diameter. These record data on both sides and are therefore double sided disks. Figure 14.5 displays a 3.5 inch floppy disk. These disks come in three different capacities — double density, high density and very high density. These are smaller and can store more data than can the 5.25 inch disks.

  The storage capacity for any disk can be calculated as:

  Storage Capacity = Number of Recording Surfaces × Number of Tracks Per Surface × Number of Sectors Per Track × Number of Bytes Per Sector

Thus, for a 3.5 inch high density disk which has 80 tracks, 18 sectors/track, and 512 bytes/sector, the disk storage capacity can be calculated as follows:

$$2 \times 80 \times 18 \times 512 = 14{,}74{,}560 \text{ bytes or } 1.4 \text{ MB (approximately)}$$

Table 14.1 provides the necessary details and associated storage capacities of various kinds of floppy disks.

Floppy disks were extensively used in personal computers as a medium for distributing software to computer users. Nowadays, CDs or DVDs are used for that purpose.

Table 14.1 *Storage Capacities of Floppy Disks*

Size (diameter in inches)	No. of Recording Surfaces	No. of Tracks	No. of Sectors/ Tracks	No. of Bytes/ Sector	Storage Capacity (approx)
5.25	2	40	9	512	3,68,640 bytes or 360kB
5.25	2	80	15	512	12,28,800 bytes or 1.2 MB
3.5	2	40	18	512	7,37,280 bytes or 720 kB
3.5	2	80	18	512	14,74,560 bytes or 1.4 MB
3.5	2	80	36	512	29,49,120 or 2.8 MB

(iii) Hard Disks

Unlike floppy disks, hard disks are made up of rigid metal. The sizes for the disk platters range between 1 to 14 inches in diameter. Depending on the way they are packaged, hard disks can be categorized as disk packs or Winchester disks.

- **Disk Packs** consist of two or more hard disks mounted on a single central shaft. Because of this, all disks in a disk pack rotate at the same speed. It consists of separate read/write heads for each surface (excluding the upper surface of the topmost disk platter and the lower surface of the bottommost disk platter). Disk packs are removable in the sense that they can be removed and kept offline when not in use (typically stored away in plastic cases) as displayed in Figure 14.6. They have to be mounted on the disk drive before they can be used. Thus different disk packs can be mounted on the same disk drive at different instances, thereby providing virtually unlimited (modular) storage capacity.

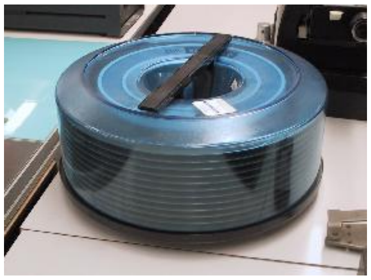


Fig. 14.6 *A Disk Pack*

- ***Winchester Disks*** also consist of two or more hard disk platters mounted on a single central shaft but are of the fixed type. The disk platters are sealed in a contamination-free container. Due to this fact, all the disk platters, including the upper surface of the topmost disk platter and the lower surface of the bottommost platter, are used for storing data. So, even though Winchester disks have limited storage capacity as opposed to disk packs, they can store larger amounts of data as compared to the same number of disk platters. Figure 14.7 displays a Winchester disk.

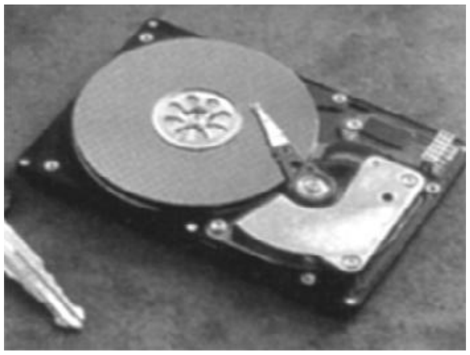


Fig. 14.7 *A Winchester Disk*

Another kind of disk called the zip disk is very common today. This consists of a single hard disk platter encased in a plastic cartridge. Such a disk typically has a capacity of about 100 MB. The zip drive can further be fixed or portable. The fixed zip drive is permanently connected to the computer system while the portable ones can be carried around and connected to any computer system for the duration of its use. In both cases, however, the zip cartridge (the actual storage medium) is portable just like a floppy, albeit with a nearly 100 times larger storage capacity. Figure 14.8 displays zip disks and zip drive.

Fig. 14.8 *Zip Disks and Zip Drive*

5. Optical Storage Devices

(i) Optical Disks

Optical disks are storage devices with huge storage capacities. They are a relatively new storage medium and use laser beam technology for writing and reading of data.

Optical disks consist of one large track that starts from the outer edge and spirals inward towards the center (this is unlike the magnetic disk in which tracks are concentric circles on the disk platter). An optical disk is also split into sectors but these are of the same length regardless of their location on the track. Data is therefore packed at maximum density over the disk.

The storage capacity of an optical disk is determined as follows:

Storage Capacity = Number of Sectors × Number of Bytes Per Sector

(Note that we do not consider the number of tracks since there is only one track in this case)

Thus, a 5.25 inch optical disk having 3,30,000 sectors and storing 2,352 bytes per sector, will have a storage capacity

3,30,000 × 2352 = 77,61,60,000 bytes or 740 MB (approx.)

The technology used in optical disks uses laser beams to write and read data as opposed to the read /write head used in magnetic disks. Data is recorded by etching microscopic pits (burnt surface) on the disk surface. A high-intensity laser beam is used to etch the pits, while for retrieving data, a low-intensity laser beam is used. Figure 14.9 displays an optical disk formats.

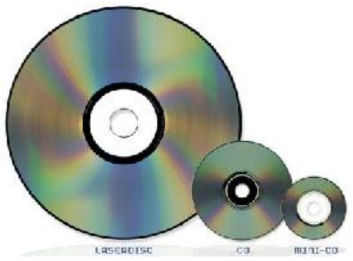


Fig. 14.9 *An Optical Disk and Disk Drive*

There are three optical memory devices that are becoming increasingly popular in various computer applications: CD-ROM, WORM, and Erasable Optical disks. We shall discuss these in the succeeding section.

CD-ROM: The Compact Disk Read-Only Memory(CD-ROM) is a direct extension of the audio CD. It is generally made from a resin called polycarbonate that is coated with aluminium to form a highly reflective surface. The information on a CD-ROM is stored as a series of microscopic pits on the reflective surface (using a high-intensity laser beam). The process of recording information on these disks is known as 'mastering'. This is so-called because this master disk is then used to make a 'die'that is used to make copies.

The information is retrieved from a CD-ROM using a low-powered laser that is generated in an optical disk drive unit. The disk is rotated and the laser beam is aimed at the disk. The intensity of the laser beam changes when it encounters a pit. A photo-sensor detects the change in intensity, hence recognizing the digital signals that are recorded on the surface of the CD-ROM and converts them into electronic signals of 1s and 0s.

As the name suggests, information stored in a CD-ROM can be read only. It cannot be modified in any way. It is therefore useful for applications in which there is a database of information that is useful as it is and does not need changing in any way, for example, a directory such as Yellow Pages. CD-ROMs are very handy in the distribution of large amounts of information to a large number of users. The strengths of CD-ROMs lie in the fact that they provide for:

- Large storage capacity for information/data.
- Fast and inexpensive mass replication.
- Suitable for archival storage since they are removable disks.

The weaknesses of CD-ROMs are:

- They are read-only and cannot be updated.
- Access time in them is greater than that of a magnetic disk.

WORM: The drawbacks of CD-ROM were partially resolved by the introduction of WORM ('Write-Once, Read Many').

In some applications, only a limited copies of compact disks are required to be made. This makes the CD-ROM production economically not viable. This is because manufacturers duplicate CD-ROMs by using expensive duplication equipment. To overcome such a problem, write-once read-many CDs have been developed.

WORM disks allow users to create their own CDs by using a CD-Recordable (CD-R) drive. This can be attached as a peripheral device to the computer system. WORM disks recorded like this can be read by any CD-ROM drive.

Erasable Optical Disk: The erasable optical disk is the latest development in optical disks. Like magnetic disks, the data in the erasable optical disk can be changed repeatedly. Erasable optical disks are also known as rewritable optical disks.

These disks integrate the magnetic and optical disk technologies to enable rewritable storage with the laser-beam technology and so are also called magneto-optical disks. In such systems, a laser beam is used along with a magnetic field to read or write information on a disk that is coated with a magnetic material.

To write, the laser beam is used to heat a specific spot on the magnetic coated material. At this increased temperature, a magnetic field is applied so as to change the polarization of that spot, thereby recording the data that is required. It may be noted here that this process does not cause any physical changes in the disk. Hence, it can be repeated many times. The degree of rotation of the polarized laser beam reflected from the surface is detected to perform reading function. This implies that as the disk spins, the polarized spots pass under the laser beam, and depending on their orientation or alignment, some of them reflect the light while others scatter it. This produces patterns of 'on' and 'off' that are converted into electronic signals of binary 1s and 0s.

The capacity of an erasable disk is very high in comparison to a magnetic disk; for example, a 5.25 inch optical disk can store around 650 MB of data while Winchester disks normally can store a maximum capacity of 320 MB. This is why magneto-optical disks are ideal for multimedia applications that require large storage capacities.

DVD: Known as the Digital Versatile Disk or the Digital Video Disk, it has the same physical dimensions as that of a CD-ROM, but it can hold up to 4.4GB data on a Single layer disk and up to 8.47GB on a dual layer disk with the maximum data transfer of 27MB/s at 20x speeds. The laser used to read/write data on a DVD is much more precise and has a wavelength of 650mm, which is one reason why a DVD can hold more data.

HD-DVD: This is a high density, mostly single-sided, double-layered optical disc that can hold up to 15GB on a single layer and 30GB on a dual layer disc. The read/write speed on an HD-DVD varies between 36 MBPS to 72 MBPS. These were primarily designed for the storage of high definition videos and large volumes of data. The basic look and feel of an HD-DVD drive and disk is the same as that of a CD-ROM and DVD except that it uses a laser of a different wavelength and the microscopic structure of storage on a disk is different.

Blu-Ray Disc: Another high density optical storage media format is gaining increasing popularity these days. Its main uses are high-definition video and data storage. A dual layer Blu-ray Disc can store 50 GB, almost six times the capacity of a double-dual layer DVD (or more than 10 times if single-layer). The data can be read from 36 MBPS to 432 MBPS which is higher than HD-DVD. When used for HD video playback, the video is encoded on it in MPEG-2, AVC, VC-1 (H.264) format, which is the same as HD-DVD.

With the wide variety of storage choices available and constant increase in their number due to changes in technology, it is impossible to say that a single medium of storage can suit the needs of all. Therefore, it is necessary to make the choice of right media based upon factors like capacity, speed, durability, life span and cost of media.

NOTES

6. Computer Output Microfilm (COM)

COM refers to a process characterized by copying/printing data from media located on personal computers, mini, or mainframe computers onto a microfilm. It comprises a high-speed recorder, which transfers digital data onto a microfilm applying laser technology, and a processor which develops the microfilm once exposed to the light source.

A computer output microfilm device translates information normally held on magnetic tape into miniature images on a microfilm (also called microfiche–'fiche' pronounced as 'fish'). The device displays information as characters on a CRT screen and then using photographic methods, records the display onto the film. Drawings and images can also be displayed along with narrative text.

A special reader/printer can be subsequently used to view the processed film. The reader operates on a 'back projection' principle displaying one frame at a time on a translucent screen, typically about A4 size. The printer can then be used selectively to produce a hard copy of what is presented on screen.

Figure 14.10 identifies the various steps in COM production.

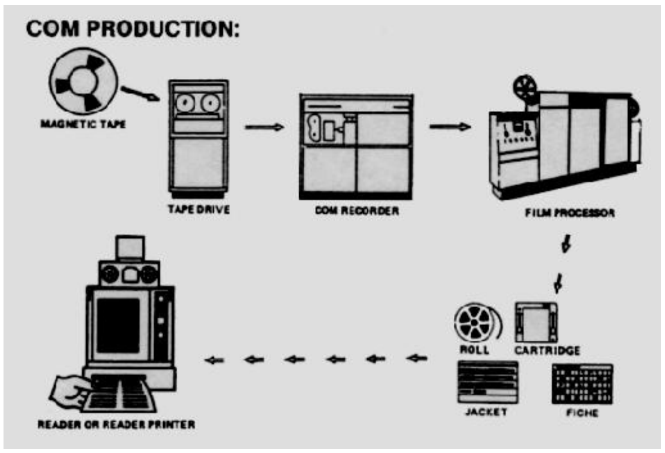


Fig. 14.10 *Steps in a COM Production*

A COM system provides an easy and compact way of recording and storing information, and subsequently retrieving desired pieces of information. It offers various advantages like reduction of paper, reduction in cost (since it is cheaper than most electronic media), improved quality (COM technology provides superior image quality), and electronic record retention/archiving.

COM is best suited for data requiring long-term storage. This is because microfilm is less volatile than magnetic media like disks and tapes. COM stores data in a very compact format. It is to be noted that up to 270 pages can be contained in a single 4 × 6 inch fiche.

Conversion of magnetic tapes to microfilms is cost-effective for closed files. However, in case of highly active data or data requiring regular updating, using microfilms may not be as efficient as retaining the information online. It is useful for data that must be archived for long periods of time and referenced only occasionally, e.g., information that must be archived to comply with legal regulations, information maintained by insurance companies, banks, government agencies, and various other organizations of this type.

Measuring Drive Performance

Disk performance can be categorized by the speed at which the data can be read or written. Over the years there have been changes in disk drive interface, rotation speeds, number of heads and cylinders and storage format, all of which have led to a decrease in data access time.

The various types of disks currently available in the market are:

- IDE – Integrated Drive Electronics
- EIDE – Enhanced Integrated Drive Electronics
- SCSI – Small Computer System Interface
- SATA I & II – Serial Advanced Technology Attachment

There are two standard methods for accessing and writing data on a disk — 'sequential' 'access' and 'random access'.

Sequential Access is when you read or write on the disk blocks in sequential or continuous order, that is, one block after another. Examples of where Sequential Access is used in computing or data storage would be the backing up of data onto tape drives or the process of writing data onto CDs and DVDs. Any storage medium based on magnetic tape, VHS, audio cassette etc, are read and written by Sequential Access.

Random Access, as the name suggests, is performed when the hard drive head needs to read/write data from/at various locations on the disk. In this case, the disk heads move rapidly from one place to another and the seek time to access data is increased because it involves mechanical operations. Most of the disk operations performed during routine computer work are random access. This is also the reason why random access time is more important while measuring disk performance than sequential access time. While data is written onto optical media sequentially, data on CDs and DVDs can be read randomly.

For Random Access, the *average seek time* and *average latency time* are added to come up with the total time it takes for the disk to read and write data on it. The average seek time is the time it takes to move the head arm from one position to another, and average latency time is the time it takes for the required data block to come under the head for the read/write operations. The average latency time depends on the Rotations Per Minute (RPM) of the disk, which is the speed at which the magnetic or optical disk rotates.

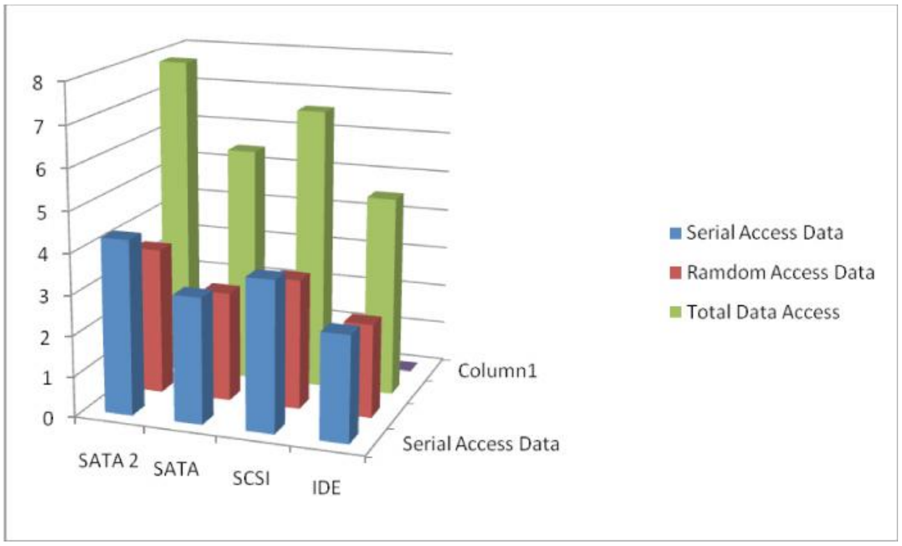


Fig. 14.11 *Data Accessed by Different Disk Drives*

The above bar chart shows the relative amount of data accessed by each type of disk drive.

14.3 DISK MANAGEMENT AND SCHEDULING

Disks are used to store bulk data as secondary storage for modern computers. A hard disk has a device controller which translates numerical addresses into head movements. The device controller gets the instructions for execution from the operating system.

A Small Computer System Interface (SCSI) drive is the most popular drive used for large personal computers and workstations. An SCSI is of four types—SCSI 1, SCSI 2, fast SCSI 2 and SCSI 3. An SCSI drive is coupled with a bus and identifies an SCSI address. Each SCSI controller can address up to seven units.

Each track is usually divided into sectors; a sector is a fixed size region which is present along the track. When writing to a disk, data is written in units of a whole number of sectors. On a disk, the number of sectors per track is not constant, nor is the number of tracks. If a sector is damaged, it must be patched up by some repair program. Usually, data in that sector is lost. Disks must be checked for damages. When a disk is formatted, the damaged sectors are not used for storing data. The SCSI drive keeps a defect list that consists of all bad sectors. A new disk contains a list of sectors and is updated as more defects occur. Formatting is a process by which the sectors of a disk are created along the tracks; each sector is labelled with an address, so that the disk controller can find the correct sector.

On simple disks, formatting is done manually. On complex, disks such as SCSI drives, a low level formatting on the disk is done by the manufacturer. High level formatting is not necessary when a file system is able to manage the hardware sectors.

Data consistency is checked by writing data to the disk and reading back the data. An error occurs if there is any disagreement. Some device controllers detect bad sectors and move data to a good sector if there is an error. Another way of checking data consistency is to calculate a number for each sector in which the number is calculated on the basis of what data is present in the sector. The calculated number is stored in the sector. When data is read back, the number is recalculated and if there is disagreement, then an error is generated. This is called Cyclic Redundancy Check (CRC) or error correcting code.

The disk drive rotates at the speed 60 to 200 rotations per second. Transfer rate is the rate at which data flows between a drive and a computer. Positioning time (random-access time) is the time for moving the disk arm to the desired cylinder and is also called seek time. The time for the desired sector to rotate under the disk head is called rotational latency. Head crash results from the disk head making contact with the disk surface. Host controller in computer uses bus to talk to disk controller built into the drive or storage array.

14.3.1 Disk Structure

A disk is the permanent storage media. It is divided into a number of cylinders in which each cylinder is divided into a number of tracks; each track is divided into a number of sectors and each sector has several blocks, as shown in Figure 5.26. Addressing of disk drive is done as large one-dimensional arrays of logical blocks. A logical block is the smallest unit of transfer. The size of a logical block is generally 512 bytes. The one-dimensional array of logical blocks is mapped into the sectors of the disk sequentially. Sector 0 is the first sector of the first track on the outermost cylinder. Mapping proceeds in order through that track, then the rest of the tracks in that cylinder, and then through the rest of the cylinders from the outermost to the innermost.

To access the disk faster, each track is usually divided into sectors; a sector is a fixed size region which is present along the track. When writing to a disk, data is written in units of a whole number of sectors. Because the heads of the disk move simultaneously on all the surfaces of the disks, we can increase read/write efficiency by allocating blocks related to one file in parallel across all surfaces.

The physical properties of a disk are number of tracks, sectors per track, speed of revolution and the number of heads (Refer Figure 14.12). An operating system must access all the different types of disks without any difficulty. On a disk, neither the number of sectors per track is constant nor is the number of tracks.

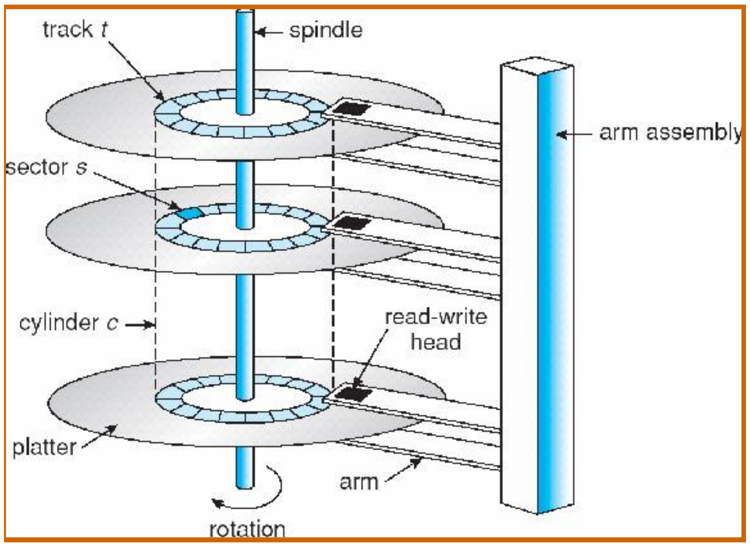


Fig. 14.12 *Physical Properties of a Disk*

Check Your Progress

1. What do you mean by access modes and its types?
2. Define the term magnetic tapes.
3. What is meant by magnetic disks?
4. What do you understand by the usage of disk drives?
5. State the formulation for storage capacity of any floppy disks and optical disk.
6. What are the two standard methods for accessing and writing data on a disk?
7. Define the significance of small computer system interface.
8. What are the physical properties of a disk?

14.4 DISK ATTACHMENT

Computers access disk storage in two ways. One way is via I/O ports (or host-attached storage); this is common on small systems. The other way is via a remote host in a distributed file system; this is referred to as network-attached storage.

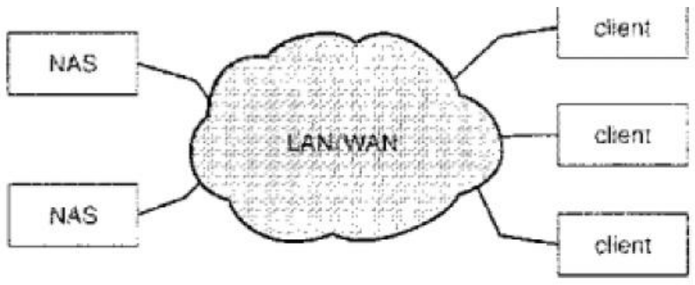
Host-Attached Storage: Host-attached storage is storage accessed through local I/O ports. These ports use several technologies. The typical desktop PC uses an I/O bus architecture called IDE or ATA. This architecture supports a maximum of two drives per I/O bus. A newer, similar protocol that has simplified cabling is SATA. High-end workstations and servers generally use more sophisticated I/O architectures, such as SCSI and Fibre Channel (FC). SCSI is a

NOTES

bus architecture. Its physical medium is usually a ribbon cable having a large number of conductors (typically 50 or 68). The SCSI protocol supports a maximum of 16 devices on the bus.

Generally, the devices include one controller card in the host (the SCSI initiator) and up to 15 storage devices (the SCSI targets). A SCSI disk is a common SCSI target, but the protocol provides the ability to address up to 8 logical units in each SCSI target. A wide variety of storage devices are suitable for use as host-attached storage. Among these are hard disk drives, RAID arrays, and CD, DVD, and tape drives. The I/O commands that initiate data transfers to a host-attached storage device are reads and writes of logical data blocks directed to specifically identified storage units (such as bus ID, SCSI ID, and target logical unit).

Network-Attached Storage: A Network-Attached Storage (NAS) device is a special-purpose storage system that is accessed remotely over a data network. Clients access network-attached storage via a remote-procedure-call interface such as NFS for UNIX systems or CIFS for Windows machines. The Remote Procedure Calls (RPCs) are carried via TCP or UDP over an IP network—-usually the same Local-Area Network (LAN) that carries all data traffic to the clients.



The network attached storage unit is usually implemented as a RAID array with software that implements the RPC interface. It is easiest to think of NAS as simply another storage-access protocol. For example, rather than using a SCSI device driver and SCSI protocols to access storage, a system using NAS would use RPC over TCP/IP.

Network-attached storage provides a convenient way for all the computers on a LAN to share a pool of storage with the same ease of naming and access enjoyed with local host-attached storage. However, it tends to be less efficient and have lower performance than some direct-attached storage options. ISCSI is the latest network-attached storage protocol.

Storage-Area Network: One drawback of network-attached storage systems is that the storage I/O operations consume bandwidth on the data network, thereby increasing the latency of network communication. This problem can be particularly acute in large client-server installations—the communication between servers and clients competes for bandwidth with the communication among servers and storage devices. A Storage-Area Network (SAN) is a private network (using storage protocols rather than networking protocols) connecting servers and storage units.

14.5 DISK SCHEDULING

NOTES

The operating system is responsible for the efficient use of the disk drives which means having fast access time and a high bandwidth for the disks. The disk header moves only in the forward or backward direction. The access time has the following three major components:

- Seek time is the time required for the disk to move the heads to the cylinder containing the desired sector.
- Rotational latency is the time required for the disk to rotate the desired sector.
- Transfer time is the time taken to transfer the data present in the blocks in a sector.

The transfer time is so fast that it can be negligible. The rotation latency is normally 7200 rpm, i.e., 120 rotations per second. Even this value is negligible. The efficiency of the operating system is directly dependent on the seek time. The objective of the operating system is to minimize seek time.

Disk bandwidth is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer.

Disk access time for any request must be as fast as possible. Scheduling is done to improve the average disk access time. The speed of access depends on seek time. Each disk drive has a queue of pending requests. A request can be the input request (write request) or output (read request). A disk address consists of disk number, cylinder number, surface number, sector number and the number of blocks to transfer.

Data in the blocks is transferred into the main memory address. Block numbers indicate the amount of information to be transferred, i.e., the byte count.

14.5.1 Disk Scheduling Algorithms

Servicing of disk I/O requests can be done by using disk scheduling algorithms.

We illustrate disk scheduling algorithms with an example.

Consider a disk with 200 tracks from 0 to 199. Let the request queue be as follows:

98, 183, 37, 122, 14, 124, 65, 67

And, let the head point to 53 track.

In the above example, the disk has 200 tracks. Request queue has 98 as its first request. Here, the disk must be moved from 53 track to 98 track as the disk header is positioned at 53 track initially.

NOTES

(a) FCFS Scheduling

First Come, First Serve (FCFS) scheduling is the simplest form of disk scheduling. It is a fair algorithm and may not provide the best possible service.

For the ordered disk queue with requests on tracks is shown in the screen below.

98; 183; 37; 122; 14; 124; 65; 67

Read/write head initially at track 53

From 53 track the header moves to 98 track

From 98 track the header moves to 183 track

From 183 track the header moves to 37 track

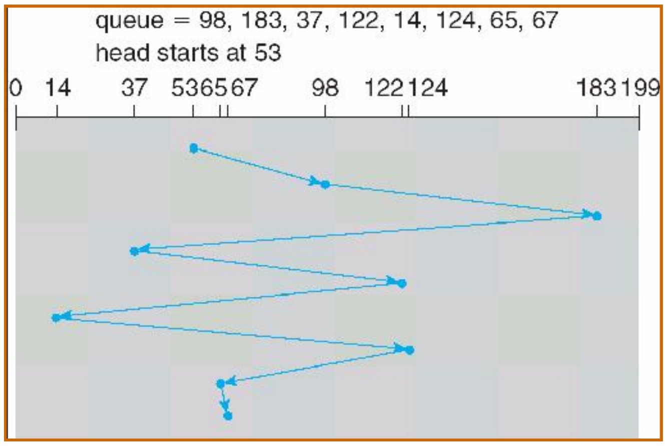
From 37 track the header moves to 122 track

From 122 track the header moves to 14 track

From 14 track the header moves to 124 track

From 124 track the header moves to 65 track

From 65 track the header moves to 67 track



Total head movement = 640 tracks

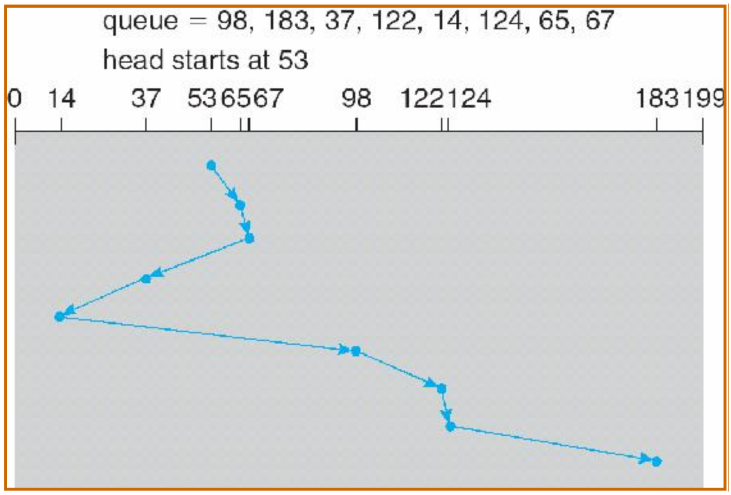
And there is a wild swing from 122 to 14 and back to 124. Wild swings occur because the requests do not always come from the same process and they are inter leaved with requests from other processes.

(b) SSTF Scheduling

Shortest Seek Time First (SSTF) scheduling selects the request with the minimum seek time from the current head position. SSTF scheduling is a form of Shortest-Job-First (SJF) scheduling and it may cause starvation of some requests. This algorithm services all requests close to the current head position before moving the head far away, i.e., moves the head to the closest track in the service queue.

A service queue can be serviced as shown in screen below.

53; 65; 67; 37; 14; 98; 122; 124; 183



53; 65; 67; 37; 14; 98; 122; 124; 183

From 53 track the header moves to 65 track

From 65 track the header moves to 67 track

From 67 track the header moves to 37 track

From 37 track the header moves to 14 track

From 14 track the header moves to 98 track

From 98 track the header moves to 122 track

From 122 track the header moves to 124 track

From 124 track the header moves to 183 track

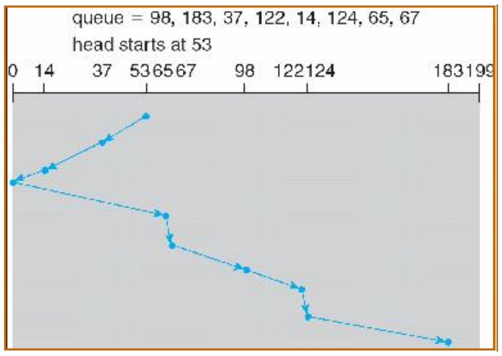
Total head movement = 263 tracks.

(c) SCAN Scheduling

The disk arm starts at the one end of the disk and moves toward the other end servicing requests until it gets to the other end of the disk where the head movement is reversed and servicing continues.

This is also termed as elevator algorithm due to its similarity with building elevators. The head constantly scans the disk from one end to the other. For example, the read/write head begins at one end of the disk and proceeds towards the other end servicing all requests as it reaches each track. At the other end, the direction of head movement is reversed and servicing continues. For example, assume a head moving towards 0 in a queue 53; 37; 14; 0; 65; 67; 98; 122; 124; 183 as shown in screen below.

NOTES

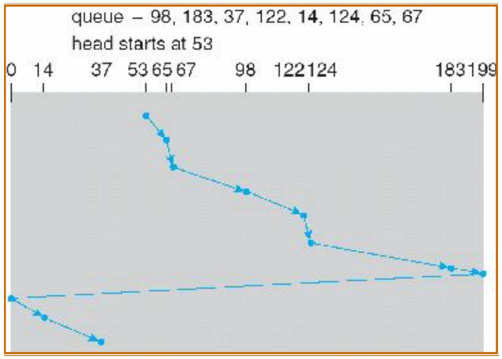


Total head movement = 236 track.

When head reverses, we have only a few requests. Heaviest density of requests is at the other end.

(d) C SCAN Scheduling

Circular SCAN (C SCAN) refers to a variation of SCAN scheduling. It moves the head from one end to the other. When it reaches the other end, it immediately come back to the first end without servicing any requests on the way.



(e) LOOK Scheduling

LOOK scheduling moves the head only till the last request in that direction. No more requests in the current direction reverses the head movement. Before you move in that direction, you need to look for a request.

Placing directories halfway between the inner and outer tracks of a disk reduce the head movement.

SSTF is common and has a natural appeal.

SCAN and C-SCAN perform better for systems that place a heavy load on the disk.

Performance depends on the numbers and types of requests.

Requests for disk service can be influenced by the file allocation method.

A disk scheduling algorithm should be written as a separate module of the operating system, allowing it to be replaced with a different algorithm, if necessary.

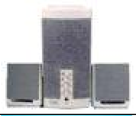
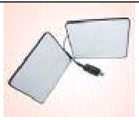
Either SSTF or LOOK is a reasonable choice for the default algorithm.

When a user wants to open a file, an operating system needs to locate the file in the directory structure. The operating system searches the directory structure frequently. If the directory structure is present in the first track and the file data is present on the last track then the head must be moved from the first track to the last track. This decreases the performance. To improve the performance, the directory structure is cached for faster access. The disk scheduling algorithms are complex to implement hence, they are written in separate module of the operating system.

NOTES

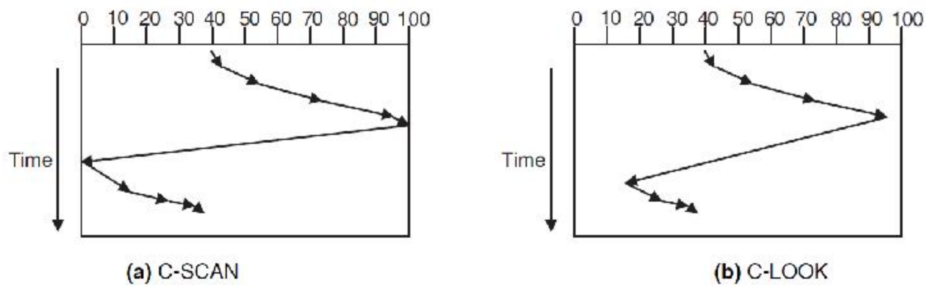
Table 14.2 summarizes the various types of tools and devices used in multimedia system.

Table 14.2 *Devices used in Intelligent Multimedia System*

Devices	Functions
	This device (speaker) is used in multimedia system and virtual reality applications. It accepts 80Hz-20KHz input sensitivity.
	This device (CD/CD R/CD RW) is also used in multimedia system and virtual reality applications.
	This audio CD Player PCD-901 device plays with CD R/CD/DC RW device.
	This headphones device keeps input power 50mw frequency response and its 20Hz-20KHz capacity uses 98dB/mw at 1 KHz.
	Video VCD player device is used to play the audio and video files.
	The wireless speaker is used to maintain the system frequency response for 35Hz-20kHz degree of distortion.
	The multimedia speaker W-400D consists of 4.1 channel audio super-woofer output with low noise power audio, for example 560W and is used in multimedia system and virtual reality applications.
	This device is a type of USB portable panel speakers and for this, no power is needed. Its size is 150mm (W) ×120mm (H). W represents width and H represents height of the device respectively.
	This PC camera CMOS sensor device is basically used for image resolution. This device uses 352×288 file format and 15 frames per second (fps) catch speed for Audio Video Interleaved (AVI) images. This device is used to process the multiple video files at a time.

C-SCAN and C-LOOK Algorithms

The C-SCAN (circular SCAN) and C-LOOK (circular LOOK) are variants of SCAN and LOOK algorithms respectively, and are designed to provide a more uniform wait time. In these algorithms, though the head scans through the disk in both directions, it services requests in one direction only. That is, when the head reaches the other hand, it promptly returns to the starting end without servicing any requests. Figure 5.25 illustrates the C-SCAN and C-LOOK algorithm for our example queue of requests.



(a) C-SCAN **(b) C-LOOK**

Fig. 5.25 *C-SCAN and C-LOOK Algorithms*

Check Your Progress

9. State the significance of network-attached storage device.
10. What is the major drawback of any network-attached storage system?
11. Define the term disk bandwidth.
12. What do you understand by shortest seek time first scheduling algorithm?
13. How do we improve the performance while disk scheduling?
14. Write the steps required while disk head scheduling.

14.6 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. Access Mode: Memory comprises various locations. Access mode is the mode in which information is accessed from the memory. The user can access memory devices in any of the following ways:
    (a) Random Access Memory (RAM): This refers to the mode in which any memory location can be accessed in any order in the same amount of time. Ferrite and semiconductor memories, which usually constitute the primary storage or main memory, are of this nature.
    (b) Sequential Access: Memories that can be accessed only in a pre-defined sequence are sequential access memories. Here, since

sequencing through other locations precedes the arrival at a desired location, the access time varies according to the location. Information on a sequential device can be retrieved in the same sequence in which it was stored. Songs stored on a cassette, that can be accessed only one by one, is an example of sequential access. Typically, magnetic tapes are sequential access memory.

2. Magnetic tapes are used for storing files of data that are sequentially accessed or not used very often and are stored off-line. They are typically used as backup storage for archiving of data.
3. Magnetic disks are direct-access medium, and so are the most popular online secondary storage devices. Direct-access devices are also called random-access devices because information is literally available at random or in any order. There is direct access to any location on the device. Thus, approximately equal access time is required for each location.
4. A disk drive is a peripheral device used to store and collect information. It can be removable or fixed, high capacity or low capacity, fast or slow speed, and magnetic or optical.
5. The storage capacity for any disk can be calculated as:

   Storage Capacity = Number of Recording Surfaces × Number of Tracks Per Surface × Number of Sectors Per Track × Number of Bytes Per Sector

   The storage capacity of an optical disk is determined as follows:

   Storage Capacity = Number of Sectors × Number of Bytes Per Sector
6. There are two standard methods for accessing and writing data on a disk — sequential access and random access.
7. A Small Computer System Interface (SCSI) drive is the most popular drive used for large personal computers and workstations. An SCSI is of four types—SCSI 1, SCSI 2, fast SCSI 2 and SCSI 3. An SCSI drive is coupled with a bus and identifies an SCSI address. Each SCSI controller can address up to seven units.
8. The physical properties of a disk are number of tracks, sectors per track, speed of revolution and the number of heads. An operating system must access all the different types of disks without any difficulty. On a disk.
9. A Network-Attached Storage (NAS) device is a special-purpose storage system that is accessed remotely over a data network. Clients access network-attached storage via a remote-procedure-call interface such as NFS for UNIX systems or CIFS for Windows machines.
10. One drawback of network-attached storage systems is that the storage I/O operations consume bandwidth on the data network, thereby increasing the latency of network communication. This problem can be particularly acute in large client-server installations—the communication between servers and

clients competes for bandwidth with the communication among servers and storage devices. A Storage-Area Network (SAN) is a private network (using storage protocols rather than networking protocols) connecting servers and storage units.

11. Disk bandwidth is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer.
12. Shortest Seek Time First (SSTF) scheduling selects the request with the minimum seek time from the current head position. SSTF scheduling is a form of Shortest-Job-First (SJF) scheduling and it may cause starvation of some requests. This algorithm services all requests close to the current head position before moving the head far away, i.e., moves the head to the closest track in the service queue.
13. To improve the performance, the directory structure is cached for faster access. The disk scheduling algorithms are complex to implement hence, they are written in separate module of the operating system.
14. The disk head scheduling works on various mechanism, such as FCFS, SSTF, etc. The SSTF solves the starvation problem by using the two techniques known as batching and aging. Elevator algorithm is required for disk head scheduling. The elevator algorithm starts at cylinder 0 with direction 'up'.

    The required steps are as follows:

    Step 1: Let current cylinder be N.

    Step 2: The second step is required to select the closest request for cylinder 'N' for up direction otherwise select the else option that is less higher for cylinder 'N' for low direction.

    Step 3: The switch direction can be used if no request direction is determined.

    Step 4: The Step 2 is repeated at this step.

14.7 SUMMARY

- Storage capacity represents the size of the memory. It is the amount of data that can be stored in the storage unit.
- Primary storage units have less storage capacity as compared to secondary storage units.
- The capacity of internal memory and main memory can be expressed in terms of the number of words or bytes. The capacity of external or secondary storage, on the other hand, is measured in terms of bytes.
- Access time is the time required to locate and retrieve the data from the storage unit. It is dependent on the physical characteristics and access mode

used for that device. Primary storage units have faster access time as compared to secondary storage units.

- Access mode is the mode in which information is accessed from the memory.
- Random Access Memory (RAM) refers to the mode in which any memory location can be accessed in any order in the same amount of time. Ferrite and semiconductor memories, which usually constitute the primary storage or main memory, are of this nature.
- Sequential access are memories that can be accessed only in a pre-defined sequence. Here, since sequencing through other locations precedes the arrival at a desired location, the access time varies according to the location. Information on a sequential device can be retrieved in the same sequence in which it was stored. Typically, magnetic tapes are sequential access memory.
- In the direct access, a separate read/write head exists for each track, and on a track, you can access the information serially. This semi-random mode of access exists in magnetic disks.
- If the storage unit can retain the data even after the power is turned off or interrupted, it is termed as non-volatile storage. And, if the data is lost once the power is turned off or interrupted, it is called volatile storage. A non-volatile storage is definitely more desirable and feasible for storage of large volumes of data.
- The Static RAM (SRAM) stores binary information using clocked sequential circuits. The stored information remains valid only as long as power is applied to the unit.
- The Dynamic RAM (DRAM) stores binary information in the form of electric charges that are applied to capacitors inside the chip. The stored charge on the capacitors tends to discharge with time and so must be periodically recharged by refreshing the dynamic memory. The dynamic RAM offers larger storage capacity and reduced power consumption.
- Cache memories are small, fast memories placed between the CPU and the main memory. They are faster than the main memory with access times closer to the speed of the CPU.
- Magnetic tapes are used for storing files of data that are sequentially accessed or not used very often and are stored off-line. They are typically used as backup storage for archiving of data.
- The tape is read sequentially, i.e., data can be read in the order in which the data has been written. This implies that if the desired record is at the end of the tape, all the earlier records have to be read before it is reached.
- The storage capacity of the tape depends on its data recording density and the length of the tape. The data recording density is the amount of data that can be stored or the number of bytes that can be stored per linear inch of tape. The data recording density is measured in BPI (Bytes Per Inch).

NOTES

NOTES

- Magnetic disks are direct-access medium, and so are the most popular online secondary storage devices. Direct-access devices are also called random-access devices because information is literally available at random or in any order. There is direct access to any location on the device. Thus, approximately equal access time is required for each location.
- A disk drive is a peripheral device used to store and collect information. It can be removable or fixed, high capacity or low capacity, fast or slow speed, and magnetic or optical.
- Structurally, a drive is the object inside which a disk(s) is either permanently or temporarily stored. While a disk contains the media on which the data is stored, a drive contains the machinery and circuitry required for implementing the read / write operations on the disk.
- The disks used with a floppy disk drive are small, removable disks made of plastic, and coated with magnetic recording material. There are two sizes commonly used, with diameters of 5.25 and 3.5 inches.
- Hard disks are made up of rigid metal. The sizes for the disk platters range between 1 to 14 inches in diameter. Depending on the way they are packaged, hard disks can be categorized as disk packs or Winchester disks.
- Disk Packs consist of two or more hard disks mounted on a single central shaft. Because of this, all disks in a disk pack rotate at the same speed. It consists of separate read/write heads for each surface (excluding the upper surface of the topmost disk platter and the lower surface of the bottommost disk platter).
- Winchester Disks also consist of two or more hard disk platters mounted on a single central shaft but are of the fixed type. The disk platters are sealed in a contamination-free container. Due to this fact, all the disk platters, including the upper surface of the topmost disk platter and the lower surface of the bottommost platter, are used for storing data.
- Optical disks are storage devices with huge storage capacities. They are a relatively new storage medium and use laser beam technology for writing and reading of data.
- Optical disks consist of one large track that starts from the outer edge and spirals inward towards the center (this is unlike the magnetic disk in which tracks are concentric circles on the disk platter).
- An optical disk is also split into sectors but these are of the same length regardless of their location on the track. Data is therefore packed at maximum density over the disk.
- The Compact Disk Read-Only Memory (CD-ROM) is a direct extension of the audio CD. It is generally made from a resin called polycarbonate that is coated with aluminium to form a highly reflective surface. The information

on a CD-ROM is stored as a series of microscopic pits on the reflective surface (using a high-intensity laser beam).

- WORM disks allow users to create their own CDs by using a CD-Recordable (CD-R) drive. This can be attached as a peripheral device to the computer system. WORM disks recorded like this can be read by any CD-ROM drive.
- The Digital Versatile Disk or the Digital Video Disk has the same physical dimensions as that of a CD-ROM, but it can hold up to 4.4GB data on a Single layer disk and up to 8.47GB on a dual layer disk with the maximum data transfer of 27MB/s at 20x speeds. The laser used to read/write data on a DVD is much more precise and has a wavelength of 650mm, which is one reason why a DVD can hold more data.
- HD-DVD is a high density, mostly single-sided, double-layered optical disc that can hold up to 15GB on a single layer and 30GB on a dual layer disc. The read/write speed on an HD-DVD varies between 36 MBPS to 72 MBPS. These were primarily designed for the storage of high definition videos and large volumes of data.
- Disk performance can be categorized by the speed at which the data can be read or written. Over the years there have been changes in disk drive interface, rotation speeds, number of heads and cylinders and storage format, all of which have led to a decrease in data access time.
- Each track is usually divided into sectors; a sector is a fixed size region which is present along the track. When writing to a disk, data is written in units of a whole number of sectors.
- Data consistency is checked by writing data to the disk and reading back the data. An error occurs if there is any disagreement. Some device controllers detect bad sectors and move data to a good sector if there is an error. Another way of checking data consistency is to calculate a number for each sector in which the number is calculated on the basis of what data is present in the sector.
- A disk is the permanent storage media. It is divided into a number of cylinders in which each cylinder is divided into a number of tracks; each track is divided into a number of sectors and each sector has several blocks.
- Host-attached storage is storage accessed through local I/O ports. These ports use several technologies. The typical desktop PC uses an I/O bus architecture called IDE or ATA. This architecture supports a maximum of two drives per I/O bus. A newer, similar protocol that has simplified cabling is SATA.
- High-end workstations and servers generally use more sophisticated I/O architectures, such as SCSI and Fibre Channel (FC). SCSI is a bus

architecture. Its physical medium is usually a ribbon cable having a large number of conductors (typically 50 or 68). The SCSI protocol supports a maximum of 16 devices on the bus.

- A Network-Attached Storage (NAS) device is a special-purpose storage system that is accessed remotely over a data network. Clients access network-attached storage via a remote-procedure-call interface such as NFS for UNIX systems or CIFS for Windows machines.
- The operating system is responsible for the efficient use of the disk drives which means having fast access time and a high bandwidth for the disks. The disk header moves only in the forward or backward direction.
- Seek time is the time required for the disk to move the heads to the cylinder containing the desired sector.
- Rotational latency is the time required for the disk to rotate the desired sector.
- Transfer time is the time taken to transfer the data present in the blocks in a sector.
- Disk bandwidth is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer.
- Disk access time for any request must be as fast as possible. Scheduling is done to improve the average disk access time.

14.8 KEY WORDS

- Magnetic tapes: These are used for storing files of data that are sequentially accessed or not used very often and are stored off line.
- Disk drive: A peripheral device used to store and collect information.
- Small Computer System Interface (SCSI): A SCSI drive is the most popular drive used for large personal computers and workstations.
- Network-Attached Storage (NAS): A NAS device is a special-purpose storage system that is accessed remotely over a data network.

14.9 SELF-ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. Define the term mass storage devices.
2. What is the importance of using magnetic disks and tapes?

3. What are the optical storage devices in any operating system?
4. What is the significance of measuring drive performance?
5. Define the term disks.
6. State functions of the disk structure.
7. In how many ways the disk attachment can be performed? List them.
8. What is the importance of studying disk scheduling algorithms?
9. Differentiate between FCFS, SSTF and SCAN scheduling algorithms.
10. Define the disk head scheduling.

Long-Answer Questions

1. What are the major advantages of primary and secondary storage devices? Explain each of them briefly.
2. What are magnetic disks and magnetic tapes? State the examples of some magnetic disk storage units.
3. Analyse the disk structure and its physical properties along with the suitable diagram.
4. Explain the terms, such as host-attached storage, network-attached storage and storage –area network.
5. Define disk management. Illustrate the techniques how the disks can be easily scheduled and manipulated.
6. What is disk scheduling? Explain its various types. What role does disk access time plays in disk scheduling?
7. Consider a disk with 200 tracks and the queue has random requests from different processes in the order: 55, 58, 39, 18, 90, 160, 150, 38, and 184.

14.10 FURTHER READINGS

Silberschatz, Abraham, Peter B. Galvin and Greg Gagne. 2008. *Operating System Concepts*, 8th Edition. New Jersey: John Wiley & Sons.

Tanenbaum, Andrew S. 2006. *Operating Systems Design and Implementation*, 3rd Edition. New Jersey: Prentice Hall.

Tanenbaum, Andrew S. 2001. *Modern Operating Systems*. New Jersey: Prentice Hall.

Deitel, Harvey M. 1984. *An Introduction to Operating Systems*. Boston (US): Addison-Wesley.

NOTES

Stallings, William. 1995. *Operating Systems,* 2nd Edition. New Jersey: Prentice Hall.

Milenkovic, Milan. 1992. *Operating Systems: Concepts and Design*. New York: McGraw Hill Higher Education.

Mano, M. Morris. 1993. *Computer System Architecture*. New Jersey: Prentice Hall Inc.