NEW SYLLABUS
CBCS PATTERN

S.Y. B.B.A. (C.A.)
SEMESTER - IV

OPERATING SYSTEM

GAJANAN A. DESHMUKH
VINAYAK S. MORE

NIRALI PRAKASHAN
ADVANCEMENT OF KNOWLEDGE

SPPU New Syllabus

A Book Of

Operating System

For B.B.A.(C.A) : Semester - IV

[Course Code CA-403]

CBCS Pattern

As Per New Syllabus, Effective from June 2020

Prof. Gajanan A. Deshmukh
HOD - BCA Department
Smt. Kashibai Navale College of Commerce
Pune

Vinayak Sambhaji More
M.Sc. (C.S.), Asst. Professor
ATSS's College of Business Studies and Computer Applications
Chinchwad, Pune

Price ₹ 225.00

NIRALI PRAKASHAN
ADVANCEMENT OF KNOWLEDGE

N4955

Operating System **ISBN 978-93-90596-15-7**

First Edition : January 2021

Published By :
NIRALI PRAKASHAN
Abhyudaya Pragati, 1312, Shivaji Nagar,
Off J.M. Road, Pune – 411005
Tel - (020) 25512336/37/39, Fax - (020) 25511379
Email : niralipune@pragationline.com

Polyplate

Printed By :
YOGIRAJ PRINTERS AND BINDERS
Survey No. 10/1A, Ghule Industrial Estate
Nanded Gaon Road
Nanded, Pune - 411041
Mobile No. 9404233041/9850046517

➢ DISTRIBUTION CENTRES

PUNE

Nirali Prakashan (For orders within Pune) : 119, Budhwar Peth, Jogeshwari Mandir Lane, Pune 411002, Maharashtra
Tel : (020) 2445 2044; Mobile : 9657703145
Email : niralilocal@pragationline.com

Nirali Prakashan (For orders outside Pune) : S. No. 28/27, Dhayari, Near Asian College Pune 411041
Tel : (020) 24690204; Mobile : 9657703143
Email : bookorder@pragationline.com

MUMBAI

Nirali Prakashan : 385, S.V.P. Road, Rasdhara Co-op. Hsg. Society Ltd.,
Girgaum, Mumbai 400004, Maharashtra; Mobile : 9320129587
Tel : (022) 2385 6339 / 2386 9976
Email : niralimumbai@pragationline.com

➢ DISTRIBUTION BRANCHES

JALGAON

Nirali Prakashan : 34, V. V. Golani Market, Navi Peth, Jalgaon 425001, Maharashtra,
Tel : (0257) 222 0395, Mob : 94234 91860; Email : niralijalgaon@pragationline.com

KOLHAPUR

Nirali Prakashan : New Mahadvar Road, Kedar Plaza, 1st Floor Opp. IDBI Bank, Kolhapur 416 012
Maharashtra. Mob : 9850046155; Email : niralikolhapur@pragationline.com

NAGPUR

Nirali Prakashan : Above Maratha Mandir, Shop No. 3, First Floor,
Rani Jhanshi Square, Sitabuldi, Nagpur 440012, Maharashtra
Tel : (0712) 254 7129; Email : niralinagpur@pragationline.com

DELHI

Nirali Prakashan : 4593/15, Basement, Agarwal Lane, Ansari Road, Daryaganj
Near Times of India Building, New Delhi 110002 Mob : 08505972553
Email : niralidelhi@pragationline.com

BENGALURU

Nirali Prakashan : Maitri Ground Floor, Jaya Apartments, No. 99, 6th Cross, 6th Main,
Malleswaram, Bengaluru 560003, Karnataka; Mob : 9449043034
Email: niralibangalore@pragationline.com

Other Branches : Hyderabad, Chennai

niralipune@pragationline.com | www.pragationline.com

Also find us on www.facebook.com/niralibooks

Preface ...

We take this opportunity to present this book entitled as **"Operating System"** to the students of Fourth Semester - BBA (CA). The object of this book is to present the subject matter in a most concise and simple manner. The book is written strictly according to the New Syllabus (CBCS Pattern).

The book has its own unique features. It brings out the subject in a very simple and lucid manner for easy and comprehensive understanding of the basic concepts, its intricacies, procedures and practices. This book will help the readers to have a broader view on Operating System. The language used in this book is easy and will help students to improve their vocabulary of Technical terms and understand the matter in a better and happier way.

We sincerely thank Shri. Dineshbhai Furia and Shri. Jignesh Furia of Nirali Prakashan, for the confidence reposed in us and giving us this opportunity to reach out to the students of BBA (CA) studies.

We have given our best inputs for this book. Any suggestions towards the improvement of this book and sincere comments are most welcome on niralipune@pragationline.com.

Authors

■■■

Syllabus ...

1. Introduction to Operating System **[Lectures 3]**

1.1 What is Operating System
1.2 Computer System Architecture
1.3 Services provided by OS
1.4 Types of OS
1.5 Operating System Structure:
- Simple Structure
- Layered Approach
- Micro Kernels
- Modules

1.6 Virtual Machines – Introduction, Benefits

2. System Structure **[Lectures 3]**

2.1 User Operating System Interface
2.2 System Calls:
- Process or Job Control
- Device Management
- File Management

2.3 System Program
2.4 Operating System Structure

3. Process Management **[Lectures 4]**

3.1 Process Concept:
- The Process
- Process States
- Process Control Block

3.2 Process Scheduling:
- Scheduling Queues
- Schedulers
- Context Switch

3.3 Operation on Process:
- Process Creation
- Process Termination

3.4 Interprocess Communication:
- Shared Memory System
- Message Passing Systems

4. CPU Scheduling **[Lectures 6]**

4.1 What is Scheduling
4.2 Scheduling Concepts:
- CPU - I/O Burst Cycle
- CPU Scheduler
- Preemptive and Non-preemptive Scheduling
- Dispatcher

4.3 Scheduling Criteria
4.4 Scheduling Algorithms:
- FCFS
- SJF (Preemptive and Non-Preemptive)
- Priority Scheduling (Preemptive and Non-Preemptive)
- Round Robin Scheduling
- Multilevel Queues
- Multilevel Feedback Queues

5. Process Synchronization [Lectures 6]

5.1 Introduction
5.2 Critical Section Problem
5.3 Semaphores:
- Concept
- Implementation
- Deadlock & Starvation
- Types of Semaphores

5.4 Classical Problems of Synchronization:
- Bounded Buffer Problem
- Readers & Writers Problem
- Dining Philosophers Problem

6. Deadlock [Lectures 7]

6.1 Introduction
6.2 Deadlock Characterization
6.3 Necessary Condition
6.4 Deadlock Handling Technique:
- Deadlock Prevention
- Deadlock Avoidance:
  - Safe State
  - Resource Allocation Graph Algorithm
  - Bankers Algorithm
- Deadlock Detection
- Recovery from Deadlock:
  - Process Termination
  - Resource Preemption

7. Memory Management [Lectures 8]

7.1 Background:
- Basic Hardware
- Address Binding
- Logical versus Physical Address Space
- Dynamic Loading
- Dynamic Linking and Shared Libraries

7.2 Swapping
7.3 Contiguous Memory Allocation:
- Memory Mapping and Protection
- Memory Allocation
- Fragmentation

7.4 Paging:
- Basic Method
- Hardware Support
- Protection
- Shared Pages

7.5 Segmentation:
- Basic Concept
- Hardware

7.6 Virtual Memory Management:
- Background
- Demand Paging
- Performance of Demand Paging
- Page Replacement:
  - FIFO
  - OPT
  - LRU
  - Second Chance Page Replacement
  - MFU
  - LFU

8. File System [Lectures 7]

8.1 Introduction and File concepts (File Attributes, Operations on Files)

8.2 Access Methods:
- Sequential Access
- Direct Access

8.3 File Structure:
- Allocation Methods
- Contiguous Allocation
- Linked Allocation
- Indexed Allocation

8.4 Free Space Management:
- Bit Vector
- Linked List
- Grouping
- Counting

9. I/O System [Lectures 4]

9.1 Introduction

9.2 I/O Hardware

9.3 Application of I/O Interface

9.4 Kernel I/O Subsystem

9.5 Disk Scheduling:
- FCFS
- Shortest Seek Time First
- SCAN
- C- SCAN
- C- Look

■■■

Contents ...

■■■

1... Introduction to Operating System

Objectives...

- To know basic concepts of Operating System and its Layered structure.
- To get information about Computer System Architecture.
- To study about services provided by Operating System.
- To learn various types of Operating Systems.
- To know basic concept of Virtual Machine.

1.1 INTRODUCTION

- An operating system (sometimes abbreviated as "OS") is a program that, after being initially loaded into the computer with boot program, manages all the other programs in a computer. The other programs are called application programs. The application programs make use of the operating system by making requests for services through a defined Application Program Interface usually called as API.
- An Operating System is a software program or set of programs that mediate access between physical devices such as, a keyboard, mouse, monitor, disk drive or network connection and application programs such as, a word processor, World Wide Web browser or Electronic mail client.
- It is a program that controls the execution of application programs and acts as an interface between applications and the computer hardware. It is a program designed to run other programs on system.
- Operating systems are responsible for everything from the control and allocation of memory to recognizing input from external devices and transmitting output to computer displays. They also manage files on computer hard drives and control peripherals, like printers and scanners.
- Its job includes preventing unauthorized users from accessing the computer system.
- As a part of this process, the operating system manages the resources of the computer in an attempt to meet overall system goals such as efficiency and throughput. The details of this resource management can be quite complicated; however, the operating system usually hides such complexities from the user.

1.1.1 What is Operating System? [W-18]

- An operating system may be viewed as an organized collection of software extensions of hardware, consisting of control routines for operating a computer and for providing an environment for execution of programs.
- Other programs rely on facilities provided by the operating system to gain access to computer system resources, such as files and input/output devices.
- Programs usually invoke services of operating system by means of operating system calls. In addition users may interact with operating system directly by means of operating system commands. In either case, the operating system acts as interface between users and hardware of a computer system.

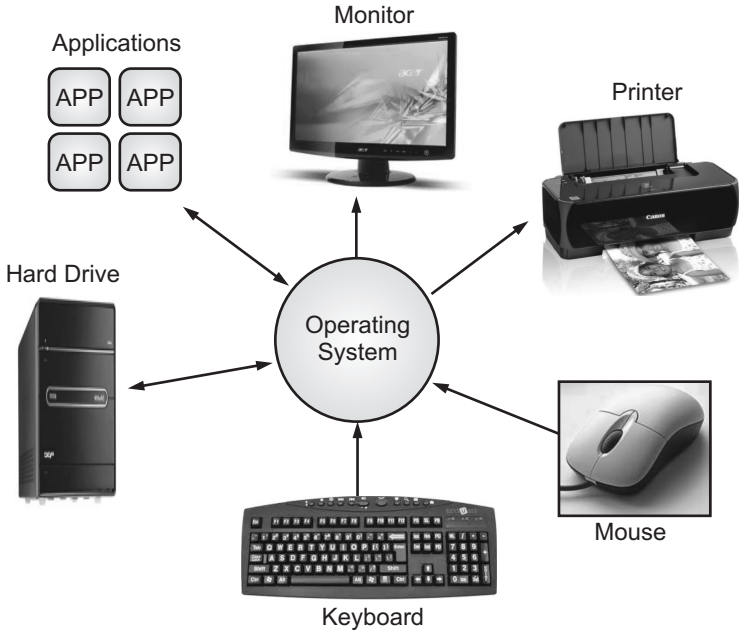


Fig. 1.1: Concept of Operating System

1.1.2 Definition of Operating System

- An Operating System is a computer program that manages the resources of a computer. It accepts keyboard or mouse inputs from users and displays the results of the actions and allows the user to run applications or communicate with other computers via networked connections.

OR

- Operating system is a set of computer programs that manage the hardware and software resources of a computer system. We can add to that definition to say that an operating system rationally processes electronic devices in response to approved commands.

OR

- An Operating system is a collection of programs that control the application software that users run and provides a link between the hardware and software currently

running on the computer. The operating system is also responsible for the management and control of all resources (memory, hard drives, monitor, etc.) that are shared amongst the different application programs that may be running simultaneously.

1.1.3 Characteristics of Operating System

- Some characteristics of an Operating System are:
  - **Multi-tasking:** Multiple programs can run on it simultaneously.
  - **Multi-processing:** It can take advantage of multiple processors.
  - **Multi-user:** Multiple users can run programs on it simultaneously.
  - **Security:** It can reliably prevent application programs from directly accessing hardware devices i.e. protected.
  - It has built-in support for graphics and Networks.
  - It manages the internal memory exchanges within the different installed applications.

1.1.4 Basic Functions of Operating System

- Various operating system functions are listed below:
  1. **A Task Scheduler:** The task scheduler is able to allocate the execution of the CPU to a number of different tasks. Some of those tasks are the different applications that the user is running and some of them are operating system tasks. The task scheduler is the part of the operating system that lets you print a document from your word processor in one window while you are downloading a file in another window and recalculating a spreadsheet is a third window.
  2. **A Memory Manager:** The memory manager controls the system's RAM and normally creates a larger virtual memory space using a file on the hard disk.
  3. **A Disk Manager:** The disk manager creates and maintains the directories and files on the disk. When you request a file, the disk manager brings it in from the disk.
  4. **A Network Manager:** The network manager controls all the data moving between the computer and the network.
  5. **Other I/O Services Manager:** The operating system manages the keyboard, mouse, video, display, printers etc.
  6. **Security Manager:** The operating system maintains the security of the information in the computer's files and controls that can access the computer.
  7. **Deadlocks:** If a process wants resources of system and if it is occupied by other process then it has to wait, if circular wait forms then deadlock can happen to avoid deadlocks and to resolve it there are some techniques used for it.

1.1.5 Examples of Operating System [S-18]

- Before 1950 the programmers directly interact with hardware; there is no operating system at that time.

- If the programmer wish to execute on those days, the following serial steps are necessary:
  - Type the program on the punched card.
  - Convert the punched card to card reader.
  - Submit to the computing machine, if there are any errors, the error condition was indicated by lights.
  - The programmer examines the registers and main memory to identify the cause of the error.
  - Take the output on the printers.
  - Then the programmer is ready for the next program.
- This mode of operating could be termed as processing. This type of processing is difficult for users, it takes much time and next program should wait for the completion of previous one.
- The programs are submitted to the machine one after another, therefore this method is said to be serial processing.

1. MS-DOS: **[S-18]**

- MS-DOS is one of the oldest and widely used operating system.
- Microsoft Disk Operating System(MS-DOS) is a non-graphical command line operating system derived from 86-DOS that was created for IBM compatible computers.
- MS-DOS originally written by Tim Paterson and introduced by Microsoft in August 1981 and was last updated in 1994 when MS-DOS 6.22 was released.
- DOS is a single tasking, single user and command-driven operating system.
- A DOS must provide a file system for organizing, reading, and writing files on the storage disk.
- The DOS commands are separated into two types:
  - **(i) Internal Commands:** Directly executable, are part of the core of the operating system.
  - **(ii) External Commands:** Separated from the original program, are additional programs. To run an external command, will also report its location on the hard disk or floppy disk (possibly via the PATH command).
- **Command Interpreter:** This is system program that reads textual commands from the user or from files and executes them.

  For example: The Command.com (CMD is in later versions of Windows) is command interpreter for DOS.

2. UNIX:

- The first version of UNIX was developed in 1969 by Ken Thompson of the research group of Bell laboratories to use for idle PDP-7.
- It serves as the operating system for all types of computers, including single-user personal computers, workstations, microcomputers, minicomputers and super computers as well as special purpose devices.

- The main features of UNIX are:
  - It is a Multi-user and Multi-tasking Operating System.
  - Portability to a wide range of machines.
  - Excellent network environment.
  - Adaptability and simplicity.
  - It provides better security.
  - Flexible file system.

Architecture:

- The architecture of UNIX can be divided into 4 layers: Hardware, Kernel, System call interface (shell), and application programs/libraries as shown in Fig. 1.2.

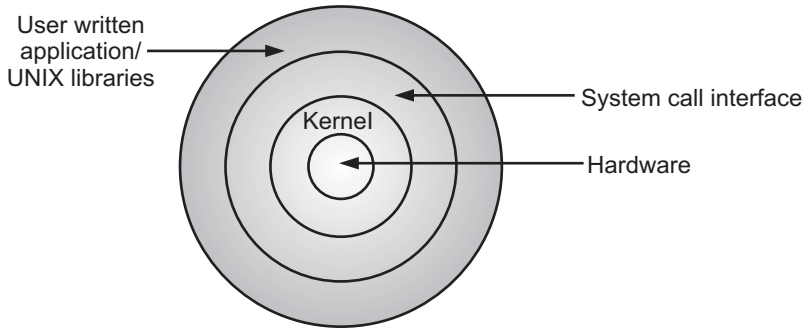


Fig. 1.2: Architecture of UNIX

(i) The Kernel: The Kernel is the part of the operating system; it interacts directly with the hardware of a computer, through device that is built into the Kernel. The main functions of Kernel are to manage computer memory to control access to the computer, to maintain the file system, to handle interrupts, to handle errors, to perform input and output services and to allocate the resources of the computers among users.

Programs interact with the Kernel through nearly 100 system calls. System calls tell the Kernel to carry out various tasks for the program, such as opening a file, writing to a file, obtaining information about a file, executing a program, terminating a process, changing the priority of a process and getting the time and data.

(ii) The Shell: Shell, it is the part of the operating system, it is a software program, and it acts as a mediator between Kernel and user. The shell reads the commands, what you typed at command lines, and interprets them and sends requests to execute a program. That is why Shell is also called Command Interpreter.

(iii) Hardware: The hardware contains all the parts of a computer, including clocks, timers, devices, ports, etc.

(iv) Unix Commands and Libraries: This layer includes user written applications, using Shell programming language and libraries of UNIX.

Modern UNIX Systems:

- A UNIX System that follows the designing principles and features of modern operating system is called Modern UNIX System.
- Examples are:

1. **BSD: (Berkeley Software Distribution):**
   - UNIX released by BSD has played a key role in the development of OS design theory.
   - The Kernel is below the system call interface and above the physical hardware in the architecture. The Kernel provides the file system. CPU scheduling memory management and other operating system functions through system calls.
2. **System V Release 4 (SVR4):**
   - SVR4 developed jointly by AT&T and SUN MICRO Systems. It is the combination of features from SVR3, 4.3 BSD, and Microsoft XENIX system V.
   - The new features of SVR4 are: Real time processing support, Process scheduling classes, dynamically allocated data structures, Virtual memory management, Virtual file system, and Preemptive Kernel.
   - SVR4 can run on any 32-bit Microprocessors up to super computers and is one of the most important operating systems ever developed.
3. **LINUX:**
   - Linux is UNIX based operating system originally developed as for Intel-Compatible PC's. It is now available for most types of hardware platforms, ranging from PDA's to main frames.
   - Linux is "Modern Operating System", meaning it has such features as virtual memory, memory protection and preemptive multitasking.
   - Linux is built and supported by a large international community of developers and users dedicated to free, open source software.
   - This community sees Linux as an alternative to such proprietary systems are Windows and Solaris, and a platform for alternatives to such proprietary applications as MS-Office, Internet Explorer and Outlook.
   - There is a very large collection of free software available for Linux. There are Graphical Environments (GUI's), office applications, developer's tools, system utilities, business applications, document publishing tools, network client and server applications.
   - Linux specifically refers to the Linux Kernel. Linux is most commonly distributed with the tool set and a collection of applications is called a "distribution".
   - The most common are Redhat, Mandrake, Suse and Debian, Ubuntu. Distributions differs in three basic ways the process for installing the distribution, the application available and the process for installing and managing these applications.

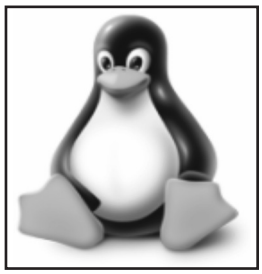


Fig. 1.3: Logo for Linux OS

- **Features of Linux**:
  - **(i) Configuration:** Linux distributions give the user full access to configure just about any aspect of their system. Linux also allows automating just about any task. Advanced scripting and high-level programming are standard features. Most operations are accessible via these scripting options.
  - **(ii) Convenience:** Once LINUX OS is set up, it requires low maintenance. Linux also offers complete remote access.
  - **(iii) Stability:** Linux is based on the UNIX Kernel. It provides preemptive multi-tasking that prevents any application from permanently stealing the CPU and locking of the machine. Protected memory prevents applications from interfering with the crashing one another.
  - **(iv) Community:** Linux is part of the greater open source community. This consists of thousands of developers and may more users world-wide who support open software. This user and developer base is also a support base.
  - **(v) Freedom:** Linux is free. This means that you are allowed to do whatever you want to with the software.

4. Windows Operating System:

- Windows is an operating system developed by Microsoft. It is a group of several proprietary graphical operating system families. Microsoft introduced an operating system on November 20, 1985, as a graphical operating system shell for MS-DOS.
- The main advantage of Windows OS is that it is user-friendly. It created a certain type of revolution in desktop computers allowing an average user to access the PC without any complexities. Windows OS provided handful of productive tools such as Microsoft Office that have made computers very useful.
- The first Windows launched in 1985 was Windows 1.0. Later many versions were released such as Windows XP, Windows Vista, Windows 7, Windows 8, Windows 10.
- Let us see the various versions of Windows Operating System:
  - **(i) Windows XP:**
    - Windows XP is an operating system produced by Microsoft as part of the Windows NT family of operating systems.
    - It was released to manufacturing on August 24, 2001, and broadly released for retail sale on October 25, 2001.

- Windows XP is more advance and high functionality window that is used in homes, offices and business places.
- Windows XP is a graphical user interface (GUI). It has pictures (graphical) that helps user to communicate (interface) with the computer.
- This operating system has multi-tasking capabilities.

(ii) Windows Vista:

- Latest windows operating system introduced in 2006. Windows Vista required high level of hardware compatibility.
- Windows Vista is an operating system developed by Microsoft. The business version was released at the end of 2006, while the consumer version shipped on January 30, 2007.
- The Vista operating system includes an updated look from Windows XP, called the "Aero" interface.

(iii) Windows 7:

- Windows 7 is a series of OS produced by Microsoft for use on personal computers for home and business purpose.

(iv) Microsoft Windows 8:

- This is the first touch-focused Windows system line and features major user interface changes over its predecessors.

(v) Microsoft Windows 10:

- Windows 10 is the newest member of the Microsoft Windows OS. Windows 10 introduces features such as an updated Start Menu, new login methods, a better taskbar, notification center, supports for virtual desktops, the Edge browser and a host of other usability updates.

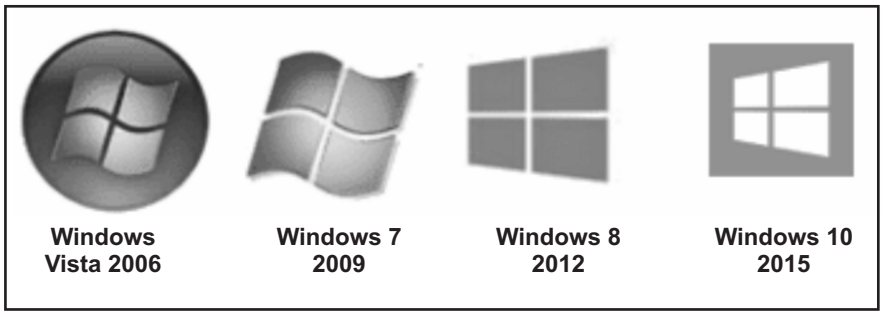


Fig. 1.4: Evolution of Windows OS

1.1.6 Operating System Layers

- Operating system structure can be viewed in a layered fashion as shown in Fig. 1.5. these layers are:
  - The lowest layer consists of Physical devices (hardware) such as integrated circuits, lines, cables, buses, the CPU, caches, hard-disks and so on.
  - The layer above is primitive software that directly controls the physical devices and provides a cleaner interface to the next layer. This layer of software is called Microprogramming, which is used to fetch machine language instructions and is usually stored in the physical device ROM.

- Above the Microprogramming layer is the Machine language layer. Machine language is mainly used to move data around the computer machine itself. The machine language layer controls I/O devices.
- The operating system resides above the Machine language layer. A major function of an operating system is to hide the above complexities and to provide the user with a convenient set of instructions to work with.

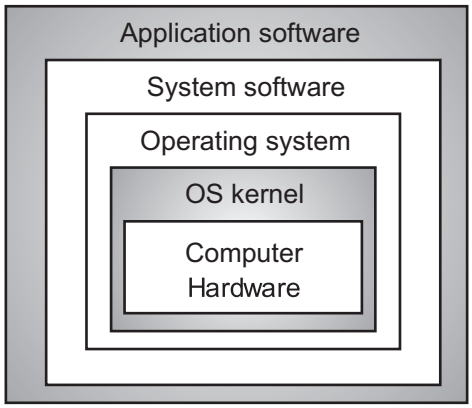


Fig. 1.5: Layers of Operating System

- Above the operating system are the rest of the System Programs, such as the command interpreter (shell), compilers, and editors and so on. These programs are distinct from the operating system, even though a systems manufacturer usually provides them.
- The operating system is the portion of software that runs in kernel mode or supervisor mode.
- An operating system is protected from users tampering with it by the hardware. Compilers and interpreters run in user mode.
- For example, you can write your own Object Oriented Compiler for Eiffel code if you want to.
- The top layer of this hierarchy is the Application Program. These programs are written by users to solve problems, such as data processing, games and so on.

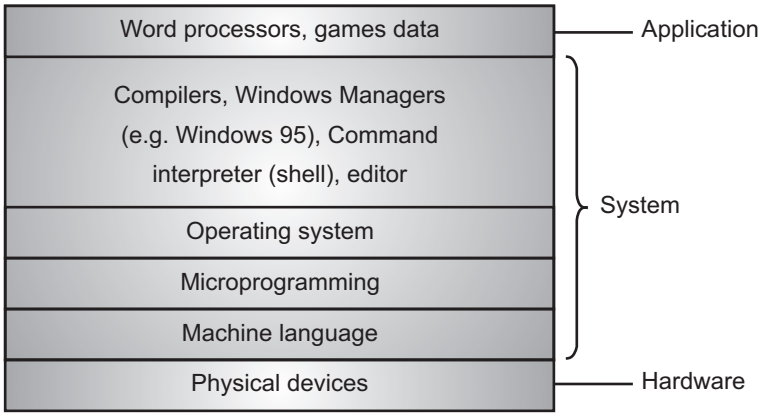


Fig. 1.6: Operating System Structure

1.2 COMPUTER SYSTEM ARCHITECTURE

- In this section, we will discuss about Computer Architecture. The computer system are categorized into single processor, dual processor, multiprocessor, parallel processor according to number of CPU used.

1.2.1 Single-Processor System

- Single-processor system contains only one CPU.
- The one CPU is capable of executing instruction set.
- Some device specific processors are used to reduce overhead of CPU.
- **Examples:** Disk-controller microprocessor receive a sequence of request from main CPU, creates its own disk queue and scheduling algorithm is implemented.

1.2.2 Multiprocessor System [S-19]

- Multiprocessor system is also called parallel system or tightly coupled system.
- The computer bus is shared by two or more processors. Also memory and peripheral devices can be shared.
- The multiprocessing systems are of two types:
  1. **Asymmetric Multiprocessing:** Each processor is assigned a specific task. This is master-slave relationship. The master processor control the system and other processor follows the master processor instruction e.g. SunOS version 4.
  2. **Symmetric multiprocessing (SMP):** Each processor performs all tasks within the operating system. All processors are peer, no master-slave relationship. All processors share the physical memory.
- **Examples:** Solaris, UNIX, Windows XP, Mac OS X, LINUX are SMP systems.

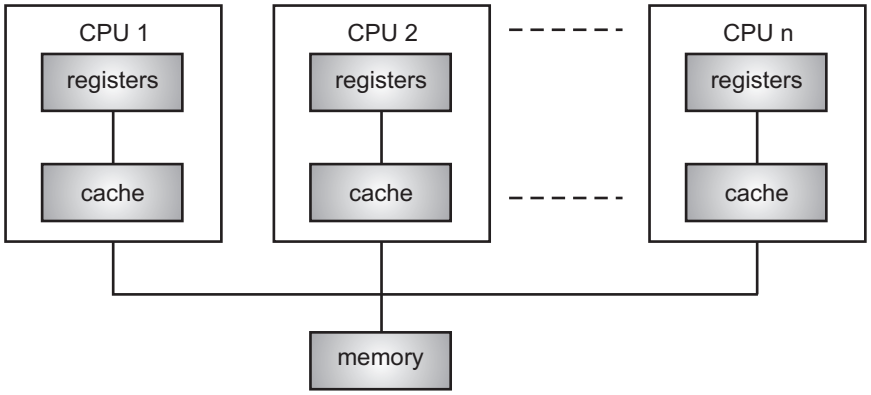


Fig. 1.7: SMP Architecture

- The recent CPU design includes multiple computing cores on a single chip. This is suitable for server systems such as database server and web server.
- In **Dual-core system**, two cores are present on the single chip, each core has its own register set and own local cache.
- In **Parallel processing**, two or more processors handle separate parts of overall task. It breaks up different parts of a task among multiple processors. This helps to reduce the amount of time to run a program.

Advantages of Multiprocessor System:

1. **Increased throughput:** As the number of processors increases, the result or output is faster. The time required is less to compute the task.
2. **Economy of scale:** The cost of multiprocessor system is less than that of multiple single-processor system, because peripheral devices, memory, bus is shared. If the multiple users wants to share the same data, then instead of storing same data on each processor, it is stored only on one disk and that can be shared by all processors.
3. **Increased reliability:** In multiprocessor system, if any of the processor fails, the work of that processor is shared by remaining processor and system will not halt.

   e.g. If we have 5 processors and one fails, then remaining 4 processors can pick up a share of the work of the failed processor.

Disadvantages of Multiprocessor System:

1. Failure of even one of the processor negatively affects the overall speed of the operating system.
2. More sophisticated operating systems are required to manage programs and data.
3. The computer system requires a large main memory.

1.2.3 Clustered Systems

- Clustered system keeps multiple CPU's together like parallel systems. Clustered system uses multiple CPU's like multiprocessor system, but they are composed of two or more individual systems or nodes joined together. The Fig. 1.8 shows the clustered system structure. This system is highly reliable to provide service to the user. The service will continue even if one or more systems in the cluster fail.
- Clustering are of two types:
  1. **Asymmetric Clustering:** In this, one machine is in standby mode and other is running the applications. The standby machine is monitoring the task of others.
  2. **Symmetric Clustering:** In this, two or more hosts are running the applications and are monitoring each other. This clustering is more efficient.

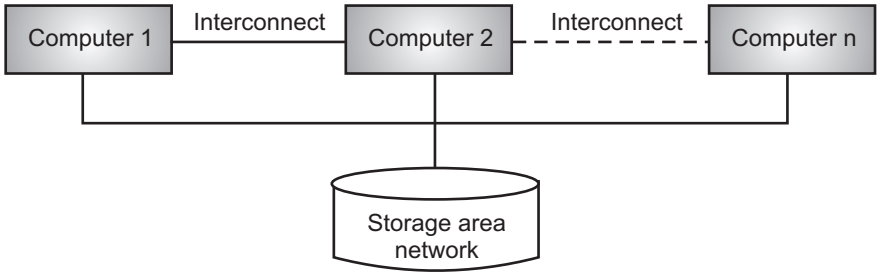


Fig. 1.8: Clustered System Structure

- A cluster consisting of many computers are connected via a network (LAN or WAN).
- Such a system is used to run application concurrently on all computers in the cluster.
- Many computers can share the same database.
- Increasing performance and throughput.

- More reliable.
- Designed for solving high-performance computing tasks.
- Example, oracle parallel server is version of oracle's database which runs on parallel clusters.

1.3 SERVICES PROVIDED BY OS [W-18, S-18]

- The services provided by the operating system differ from one another. The quality of the operating system depends upon the amount at services that the user can exploit with the system.
- Fig. 1.9 shows one view of the various operating system services and how they interrelate.

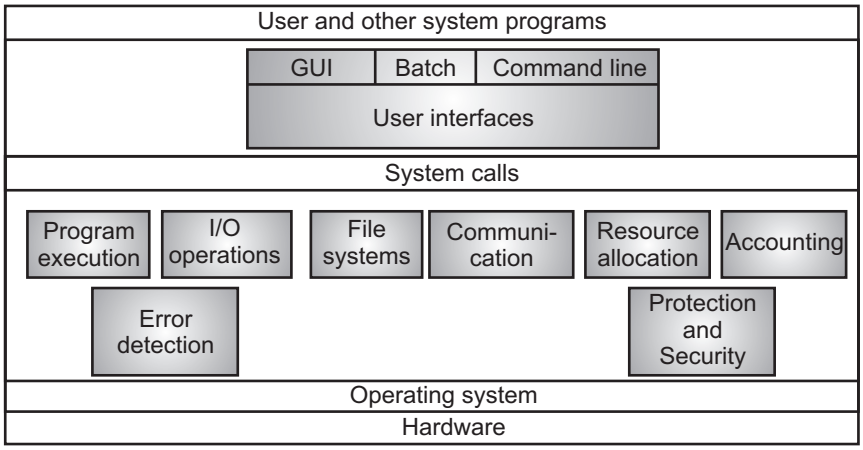


Fig. 1.9: A View of Operating System Services

- Following are the basic services are provided by the Operating System:
  1. **User Interface:** Almost all operating systems have a User Interface (UI). This user interface can take several forms. One is a Command-Line Interface (CLI), which uses text commands and a method for entering them. Another is a batch interface, in which commands and directives to control those commands are entered into files, and those files are executed. Most commonly, a Graphical User Interface (GUI) is used. The GUI interface is a window system with a pointing device to direct I/O, choose from menus and make selections and a keyboard to enter text.
  2. **Program Execution:** The system must be able to load a program into memory and run the program. If due to some reason the program halts abruptly then an error is indicated by the operating system.
  3. **I/O Operations:** All the programs dealing with I/O operations relating to specific devices are to be dealt by the operating system. For example, if a user wants to print a page then the operating system gives the command to I/O device for printing the particular pages.
  4. **File System Implementation:** The file system is of particular interest. Programs need to read and write files. Files can also be detected by their unique names.

5. **Communications:** The communication which takes place between the concurrent processes can be divided into two parts. The first one takes place between the processes that are running on the same computer and the other type of processes are those that are being executed on different computer systems through a computer network.
6. **Protection:** In a multi-user environment, protection of valuable resources plays an important role. It ensures that all the access to system resources should be in a controlled manner. This is implemented by the help of security assigned at various levels.
7. **Error Detection:** There are various types of errors that occur when the process is running. This error may be caused by CPU, memory hardware, I/O devices etc. It is the job of the operating system to keep track of these errors, raise appropriate errors at the user's screen.
8. **Accounting:** The record keeping work, as to which resource has been utilized by which process is being taken care at by the operating system. This record keeping also keeps track of the user who has used the resources and for how long so that he can be billed for that.
9. **Resource Allocation:** The operating system collects all the resources in the network environment and grants these resources to the requested process. Many different types of resources are managed by the operating system; these are CPU cycles, Main memory, I/O devices, and File storage and so on.

1.4 TYPES OF OS

- Following are major types of operating system:

1.4.1 Batch Operating System

- In olden days the computers were large systems run from a console. The common input devices were card readers, tape drives and output devices were line printers, tape drives and card punches.
- The computer system did not directly interact with the users instead the computer users used to prepare a format that consisted of the programs, the data and some control information about the nature of the job and submitted it to the computer operator.
- The job was usually in the form of punch cards. The process as a whole took a lot of time and was slow. To speed up the processing jobs with similar needs were batched together and run through the computer as a group.
- Fig. 1.10 shows the memory layout for a simple batch system.

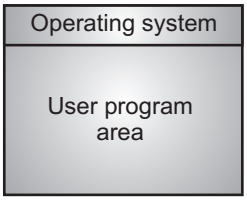


Fig. 1.10: Memory Layout for a Simple Batch System

- A batch operating system, thus normally Read a stream of separate jobs, each with its own control cards that predefine what the job does. When the job is complete, its output is usually printed. A batch is a sequence of user jobs.
- The important feature of a batch system is lack of interaction between the user and the job while that job is being executed. The job is prepared and submitted and at some later time, the output appears.
- In a batch processing system, a job is described by a sequence of control statements stored in a machine-readable form.
- The operating system can read and execute a series of such jobs without human intervention except for such functions as tape and disk mounting. The order in which the jobs are selected and executed can be scheduled using appropriate algorithms.
- Processing of a job involved physical actions by the system operator, e.g. loading a deck of cards into the card reader, pressing switches on the computer console to initiate a job, etc., all of which wasted a lot of computer time. This wastage could be reduced by automating the processing of a batch of jobs.

Working:

- A computer operator forms a batch by arranging user jobs in a sequence and inserting special markers to indicate the start and end of the batch.
- After forming a batch, the operator submits it for processing. The primary function of the batch processing system is to implement the processing of a batch of jobs without requiring any intervention of the operator.
- The operating system achieves this by making an automatic transition from the execution of one job to that of the next job in the batch.
- Batch processing is implemented by locating a component of the batch processing operating system, called the **batch monitor** (or batch supervisor), permanently in one part of the computer's main memory. The remaining part of the memory is used to process a user job i.e. the current job in the batch.
- Fig. 1.11 depicts the schematic of a batch processing system.

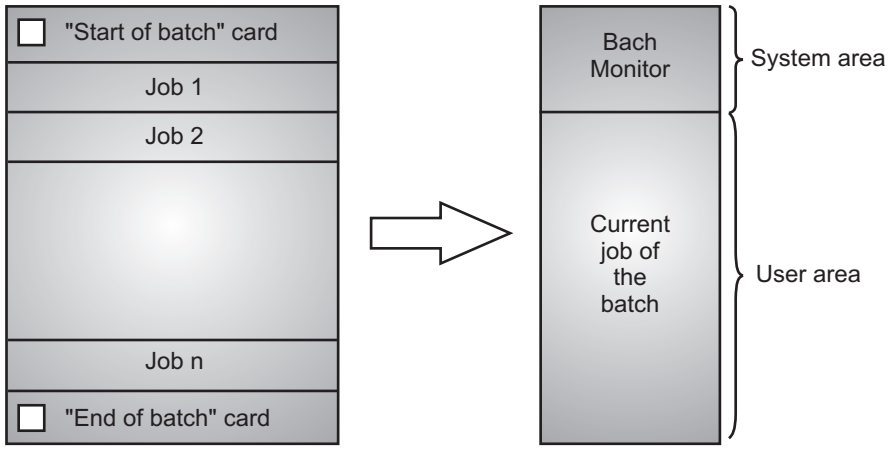


Fig. 1.11: Schematic of a batch processing system

- The batch monitor is responsible for:
  1. Accepting command from the system operator.
  2. Initiate the processing of a batch.
  3. Set up the processing of the first job.
  4. At the end of the job, terminate process and initiate execution of the next job.
  5. At the end of the batch, terminate batch and awaits initiation of the next batch by the operator.

Advantages of Batch Operating System:

1. Move much of the work of the operator to the computer.
2. Increased performance since it was possible for job to start as soon as the previous job finished.

Disadvantages of batch Operating System:

1. Turn around time can be large from user standpoint.
2. Difficult to debug program.
3. Due to lack of protection scheme, one batch job can affect pending jobs.
4. A job could corrupt the monitor, thus affecting pending jobs.
5. A job could enter an infinite loop.

Spooling:

- Spooling stands for "Simultaneous Peripheral Operations Online". So, in a Spooling, more than one I/O operations can be performed simultaneously i.e. at the time when the CPU is executing some process then more than one I/O operations can also be done at the same time.
- In the disk technology rather than the cards being read from the card reader directly into memory, and then the job being processed, cards are read directly from the card reader onto the disk.
- The location of the card images is recorded in a table kept by the operating system. When a job is executed, the operating system satisfied its request for card reader input by reading from the disk.
- Similarly, when the job requests the printer to output a line, that line is copied into a system buffer and is written to the disk. When the job is completed, the output is actually printed. This form of processing is called spooling as shown in Fig. 1.12.
- Spooling is used for data processing of remote sites. The CPU sends the data via communication paths to a remote printer. The remote processing is done at its own speed, with no CPU intervention.

Advantages of Spooling:

1. Spooling overlaps the I/O of one job with the computation of other jobs.
2. It has a direct beneficial effect on the performance of the system.
3. It can keep both the CPU and the I/O devices working at much higher rates.
4. Spooling operating uses a disk as a very large buffer.

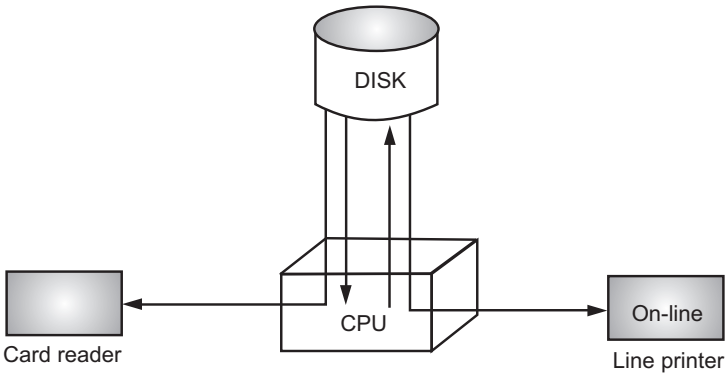


Fig. 1.12: Spooling

1.4.2 Multiprogramming [S-19]

- Multiprogramming is a technique to execute number of programs simultaneously by a single processor.
- In multiprogramming number of processes reside in main memory at a time and the operating system picks and begins executing one of the jobs in the main memory.
- The Figs. 1.13 and 1.14 shows the layout of the multiprogramming system, which consists of 5 jobs.

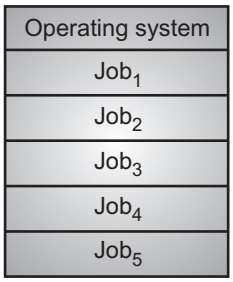


Fig. 1.13: Layout of Multiprogramming System

- In non-multiprogramming system, the CPU can execute only one program at a time, if the running program waiting for any I/O device, the CPU sits idle, this will affect the performance of the CPU.
- But in case of multiprogramming environment any I/O wait by a process, will switch the CPU from that job to another job in the job pool eliminating the CPU idle time.
- This type of system permits several user jobs to be executed concurrently. The operating system takes care of switching the CPU among the various user jobs. It also provides a suitable run-time environment and other support functions, so the jobs do not interfere with each other.

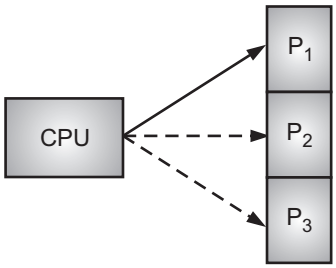


Fig. 1.14: CPU switches from one job to another

- The basic principle of multiprogramming is that while the I/O subsystem is busy with an I/O operation for a user job, the CPU can execute another user job.
- This requires the presence of multiple user jobs simultaneously in the computer's memory.

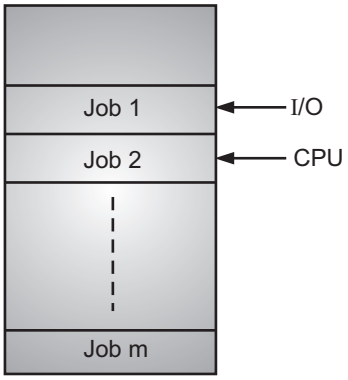


Fig. 1.15: Schematic of Multiprogramming

- Fig. 1.15 illustrates the schematic of a multiprogramming OS. The processor and I/O channel are busy with different user jobs residing in the memory. Thus, they access different areas of memory.
- This ensures that their activities would not interfere with one another.
- One part of the main memory is occupied by the supervisor and it consists of a permanently resident part and a transient part which is loaded whenever required.
- The architectural (or hardware) features essential for multiprogramming are:
  - I/O channels and the interrupted hardware,
  - Memory protection, and
  - Privileged mode of CPU operation.

Advantages:

1. Efficient memory utilization
2. CPU never sits idle, so it increases the CPU performance
3. Throughput of the CPU increases.
4. In non-multiprogramming environment (mono programming) the user/ program has to wait for CPU much time. But waiting time is limited in multiprogramming.

Disadvantages:

1. It is difficult to program a system because of complicated schedule handling.
2. Tracking all tasks/processes is sometimes difficult to handle.
3. Due to high load of tasks, long time jobs have to wait long.

1.4.3 Multiprocessing System

- This system is similar to multiprogramming system, except that there is more than one CPU available. In most multiprocessor systems, the processors share a common memory. Thus, the user can view the system as if it were a powerful single processor.

- Fig. 1.16 depicts the manner in which multiple processors may be used for multiprogramming. Usually, we visualize several separate processes as being in memory.
- Actually a process is often paged so that only part of it is in memory at one time; this allows the number of processes active in the system to be very large.

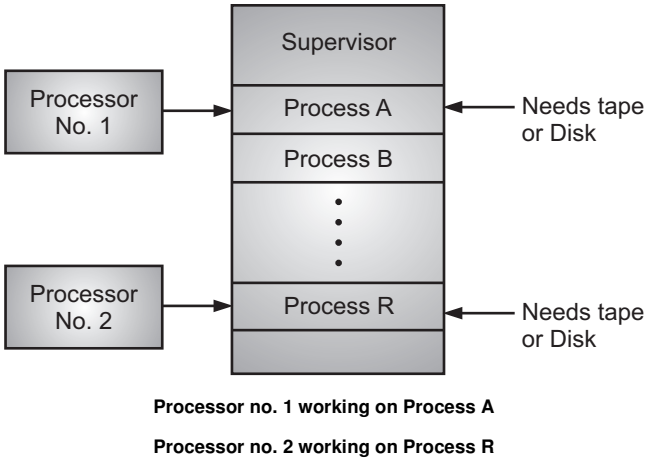


Fig. 1.16: Multiprogramming with Multiprocessors

- A processor is assigned to a task and operates on it until it is blocked. When a task is blocked, the processor selects another task and continues processing.
- After the blocking condition has been satisfied, a processor will eventually be assigned to the process; it need not be the same physical processor as before.

1.4.4 Multitasking or Time Sharing Systems

- Time sharing or multitasking, is a logical extension of multiprogramming. Multiple jobs are executed by the CPU switching between them, but the switches occur so frequently that the users may interact with each program while running.
- Time sharing systems were developed to provide interactive use of a computer at reasonable cost.
- A time shared operating system uses CPU scheduling and multiprogramming to provide each user with a small portion of a time-shared computer. Each user has at least one separate program in memory.
- A program that is loaded into memory and is being executed is commonly referred to as a process.
- A time shared operating system allows many users to share the computer simultaneously. Since each action or command in, in a time-shared system tends to be short, only a little CPU time is needed for each user.
- Time sharing operating systems are even more complex than multi-programmed operating systems. As in multi programming several jobs must be kept simultaneously in memory which requires some form of memory management and protection.

- If a reasonable time can be obtained, jobs may have to be snapped in and out of main memory to the disk that now serves as a backing store for main memory.
- A common method for achieving this goal is virtual memory, which is a technique that allows the execution of a job that may not be completely in memory.
- A time sharing system provides interactive or conversational access to a number of users. The operating system executes commands as they are entered, attempting to provide each user with a reasonably short response time to each command.
- Development of time sharing systems was motivated by the desire to provide fast response times to interactive users of a computer system.
- The response time is the time since the submission of a computational request by a user till its results are reported to the user.
- Emphasis on good response times, rather than good utilization efficiency or throughput. It requires certain basic changes in the design of the operating system. These changes mainly concern the scheduling and memory management components of the time sharing supervisor.

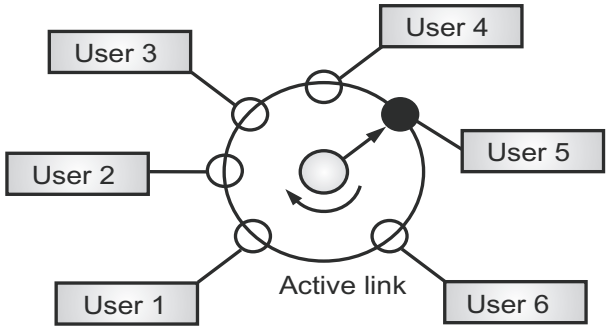


Fig. 1.17: Time Sharing System

- In above Fig. 1.17, the User 5 is active but User 1, User 2, User 3, and User 4 are in waiting state whereas user 6 is in ready status.
- As soon as the time slice of user 5 is completed, the control moves on to the next ready User i.e. User 6. In this state User 2, User 3, User 4, and User 5 are in waiting state and user 1 is in ready state. The process continues in the same way and so on.

Advantages:

1. Provide advantage of quick response.
2. Avoids duplication of software. Reduces CPU idle time.

Disadvantages:

1. Problem of reliability.
2. Question of security and integrity of user programs and data.
3. Problem of data communication.

1.4.5 Distributed Systems

- A distributed system is a collection of processors that do not share memory or a clock. Instead, each processor has its own local memory, and the processors communicate with each other through various communication lines.
- The processors in a distributed system vary in size and function.
- They may include small microprocessors, workstations, minicomputers and large general purpose computer systems.
- From the point of view of a specific processor in a distributed system, the rest of the processors and their respective resources are remote, whereas its own resources are local.
- The purpose of distributed system is to provide an efficient and convenient environment for this type of sharing of resources.

Distributed Operating System:

- In a distributed operating system, the users access remote resources in the same manner as they do local resources. Data and process migration from one site to another are under the control of the distributed operating system.
- A distributed operating system allows a more complex type of network organization. This kind of operating system manages hardware and software resources, so that a user views the entire network as a simple system.
- The user is unaware of which machine on the network is actually running a program or storing data.

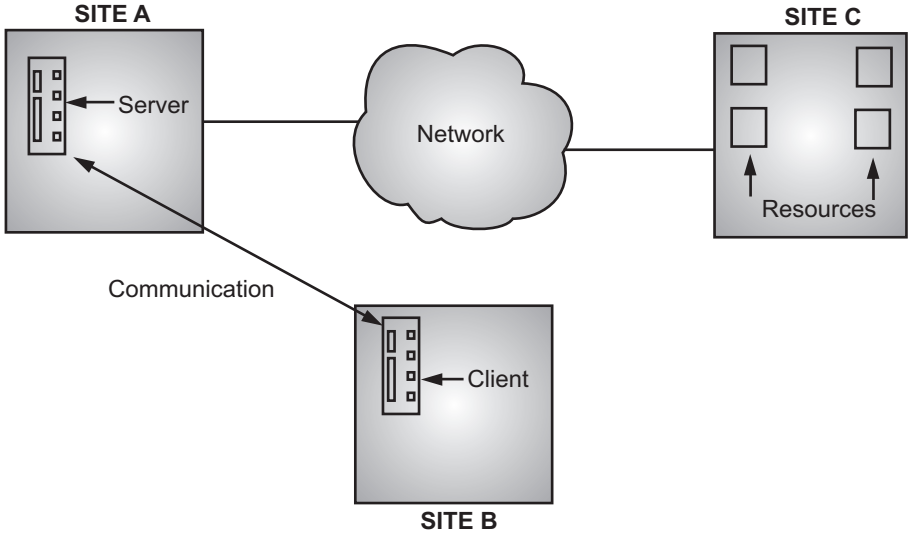


Fig. 1.18: A Distributed System

Types of Distributed Systems:

- There are four main types of distributed systems:
  1. **Client-server:** This type of system requires the client to request a resource, and then the server provides that resource which was requested. When a client is in contact with one server, the server itself may serve multiple clients at the same time.

     Both the server and the client will communicate over a computer network, which is part of a distributed system.
  2. **Three-tier:** The information about the client is stored in the middle tier, instead of storing it in the client. This is done to simplify development. This architecture is most common in web applications.
  3. **n-tier:** n-tier systems are used when the server or application needs to forward requests to additional enterprise services on the network.
  4. **Peer-to-peer:** This type of system contains nodes that are equal participants in data sharing. Furthermore, all the tasks are equally divided between all the nodes. These nodes will interact with each other as required as "share resources". To accomplish this, a network is needed.

Advantages:

1. **Economical:** Distributed computing reduces overall cost.
2. **Speed:** As computational load is spread across various nodes speed of execution increases.
3. **Reliability:** Distributed systems can function even if one of its node fails.
4. **Scalability:** The number of nodes can be increased or decreased as and when the need arises.

Disadvantages:

1. They present a potential compromise on the security of the system as multiple components could be more vulnerable to security breaches.
2. The system becomes more complex since it needs protocols for communication between various components in the system.

1.4.6 Network Operating Systems

- A Network Operating System provides an environment in which users, who are aware of the multiplicity of machines, can access remote resources by either logging into the appropriate remote machine or transferring data from the remote machine to their own machines.
- A Network Operating System is another specialized OS intended to facilitate communication between devices operating on LAN.

- Network Operating System provides the communication stack needed to understand network protocols in order to create, exchange and decompose network packets.
- Today, the concept of specialized NOS is largely obsolete because other OS types largely handle network communication.

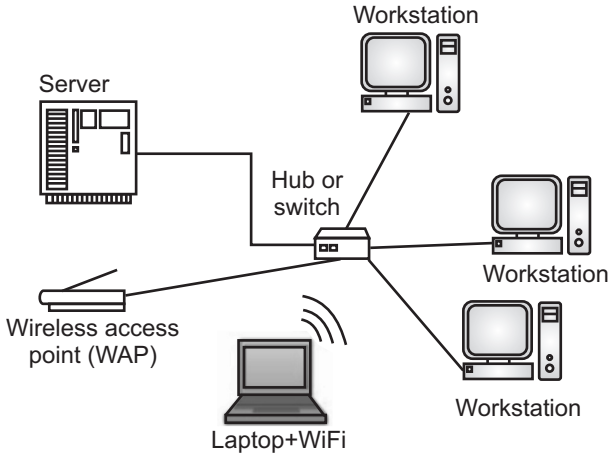


Fig. 1.19: Network Operating System

Advantages of Network Operating System:

1. Highly stable centralized servers.
2. Security concerns are handled through servers.
3. New technologies and hardware upgradation are easily integrated into the system.
4. Server access is possible remotely from different locations and types of systems.

Disadvantages of Network Operating System:

1. Servers are costly.
2. User has to depend on a central location for most operations.
3. Maintenance and updates are required regularly.

1.4.7 Real Time Systems

- Real time systems are the special type of operating system.
- A real time system is used when there are rigid time requirements on the operation of a processor or flow of data, and thus if often used as a control device in a dedicated application.
- A real-time system is designed to respond quickly to external signals such as those generated by data sensors.
- Sensors bring data to computer. The computer analyses the data and possibly adjust controls to modify the sensor inputs.
- Systems that control scientific experiments, medical imaging systems, industrial control systems and some display systems are real time systems.
- Also included are some automobile – engine fuel injection systems, home appliances controllers and weapon systems.

- Processing must be done within the defined constraints, or the system will fail. For instance, it would not do for a robot arm to be instructed to half after it had smashed into the car it was building.
- A real time system is considered to function correctly only if it returns the correct result within any time constraint.
- Real time systems are used on computers that monitor and control time-critical processes such as nuclear reactor operation or spacecraft flight.
- Hence, real-time application is an application which requires "timely" response from the computer system for the correctness of its functioning.
- A real-time operating system is one which helps to fulfill the worst-case response time requirements of an application.
- The real-time operating system provides the following facilities for this purpose:
  - Multitasking within an application.
  - Priority driven or deadline oriented scheduling.
  - Programmer defined interrupts.

Types:

- Real Time Systems are of two types:
  1. **Hard Real Time System:** Hard real-time systems guarantee that critical tasks are complete on time. This goal requires that all the delays in the system be bounded, from the retrieval of stored data to the time that it takes the operating system to finish any request made for it.
  2. **Soft Real Time System:** These are less restrictive type. A critical real time tasks get priority over other tasks and retain that priority until it completes.

1.4.8 Mobile Operating System

- These are designed to accommodate unique needs of mobile computing. And communication devices such as smart phones and tablets.
- Mobile devices typically offer limited computing resources compared to traditional PCs, and the OS must be scaled back in size and complexity in order to minimize its own resource use, while ensuring adequate resources for one or more applications running on the device.
- Mobile operating systems tend to emphasize efficient performance, user responsiveness and close attention to data handling tasks, such as supporting media streaming.
- Apple iOS and Google Android are examples of Mobile operating systems.

1.4.9 Embedded Operating System

- An embedded operating system is a type of operating system that is embedded and specifically configured for a certain hardware configuration.
- A huge collection of dedicated devices including home digital assistants, Automated Teller Machines (ATMs), Airplane systems, retail Point Of Sale (POS) terminals and Internet of Things (IoT) devices includes computers that require an embedded operating system.

- Embedded operating system helps to improve entire efficiency for controlling all hardware resources as well as decreasing the response times for specific task devices were made for.
- In the case of an embedded OS on a personal computer, this is an additional flash memory chip installed on a motherboard that is accessible on boot from the PC.
- A medical device used in a patient's life support equipment, for example, will employ an embedded OS that must run reliably in order to keep the patient alive.
- Embedded Linux is one example of an Embedded OS.

1.5 OPERATING SYSTEM STRUCTURE

- The structure of operating system consists of four layers, those are **Hardware, Operating system, System** and **Application programs** and **Users**.
- Fig. 1.20 shows the Structure of operating system.

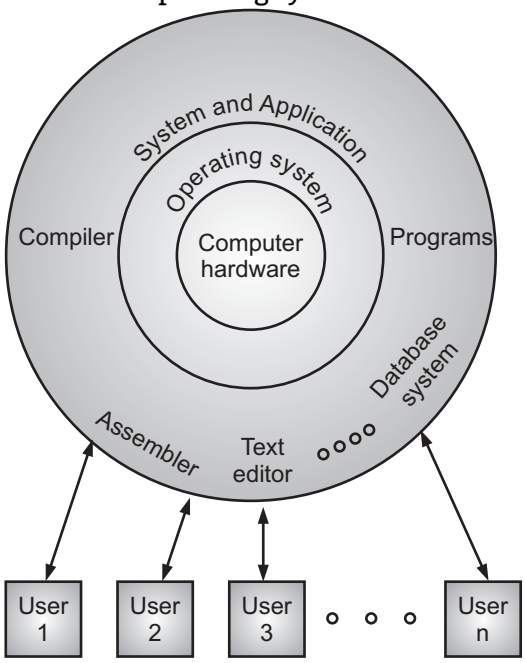


Fig. 1.20: Structure of operating system

- The hardware consists of Central Processing Unit (CPU), the main memory, I/O devices, secondary storage, etc.
- The operating system controls and co-ordinates the use of the hardware among the various application programs for the various users.
- The application programs such as word processors, spreadsheets, compilers and web browsers define the ways in which these resources are used to solve the Computing Problems of the users.
- There may be many different users as people, machines, and other computers trying to solve different problems. Accordingly there may be many different application programs.
- There are various types of operating system structures, some of them are given in this section.

1.5.1 Simple Structure

- Many commercial systems are not well defined structures. Such operating systems started as small, simple and limited systems. For example, MS-DOS: it was not divided into modules carefully.

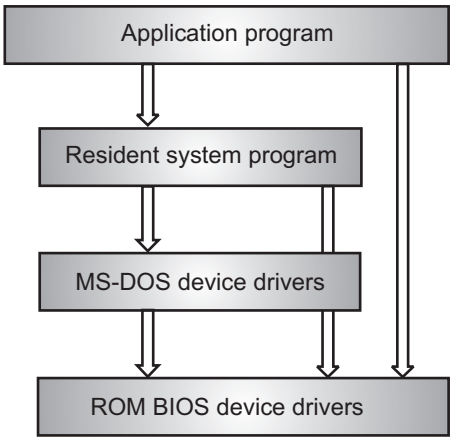


Fig. 1.21: MS-DOS Layer Structure

- UNIX is another system that was initially limited by hardware functionality. It consists of two parts one is kernel and second is system programs.
- The kernel provides the file systems, CPU scheduling, memory management and other operating system functions through System calls.
- System calls define the API to UNIX; the set of system programs commonly available defines the user interface. The programmer and user interfaces define the context that the kernel must support.
- The operating system has a much greater control over the computer and over the applications that make use of that computer.

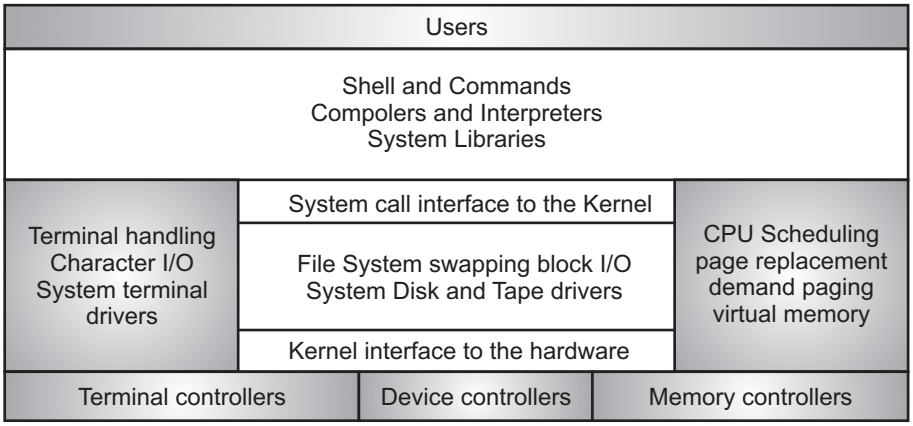


Fig. 1.22: UNIX System Structure

1.5.2 Monolithic System

- The operating systems are written as a collection of procedures, each of which can call any of the other ones whenever it needs to. When this technique is used, each procedure in the system has a well-defined interface in terms of parameters and results, and one is free to call any other one, if the latter provides some useful computation that the former needs.
- To construct the actual object program of the operating system when this approach is used, one compiles all individual procedures, or files containing the procedures and the binds them all together into a single object file using the system linker.
- In terms of information hiding, there is essentially none- every procedure is visible to every other one, (as opposed to a structure containing modules or packages, in which much of the information is local to a module and only officially designated entry points can be called from outside the module).
- The monolithic operating system is also known as the monolithic kernel. This is an old type of operating system. They were used to perform small tasks like batch processing, time sharing tasks in banks. Monolithic kernel acts as a virtual machine which controls all hardware parts

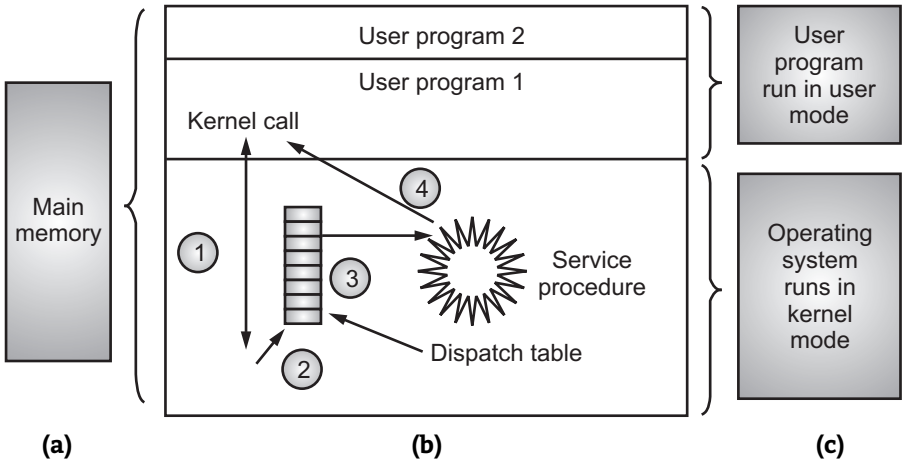


Fig. 1.23: Monolithic System

- Even in monolithic systems, however it is possible to have at least a little structure. The services (System Calls) provided by the operating system are requested by putting the parameters in well defined places, such as in registers or on the stack and then executing a special trap instruction known as a **Kernel call** or **Supervisor call**.
- This instruction switches the machine from user mode to kernel mode, and transfers control to the operating system as shown in Fig. 1.23.
- Most CPU's have two modes: **Kernel Mode** for operating system in which all instructions are allowed, and **User Mode** for user programs, in which I/O and certain other instructions are not allowed.
- The operating system then examines the parameters of the call to determine which system call is to be carried out.

- Next the operating system then examines indexes into a table that contains in slot k a pointer to the procedure that carries out system call k. This operation shown in Fig. 1.23, identifies the service procedure, which is then called. Finally, the system call is executed and control is transferred back to the user program.

Basic structure:

1. A main program that invokes the requested service procedure.
2. A set of service procedures that carry out the system calls.
3. A set of utility procedures that help the service procedures.

- In this mode, for each System Call there is one service procedure that takes care of it. The utility procedures do things that are needed by several service procedures, such as fetching data from user programs.
- This division of the procedure into three layers is shown in Fig. 1.24.

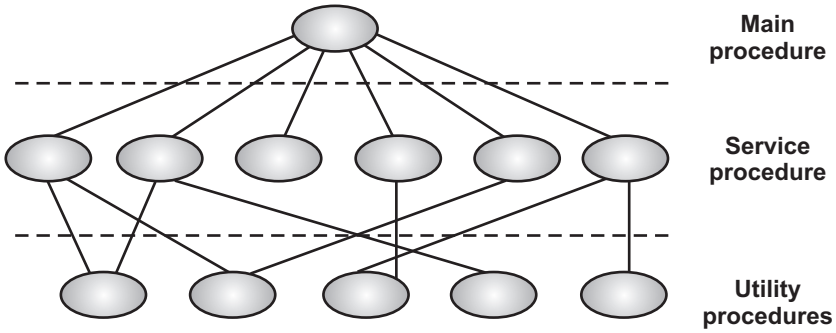


Fig. 1.24: Simple structuring model for a monolithic system

1.5.3 Layered Approach

- With proper hardware support, operating system can be broken into pieces that are smaller and more appropriate than allowed by MS DOS and UNIX OS.
- A system can be made modular in different ways one method is layered approach, here operating system is broken into number of layers or levels.

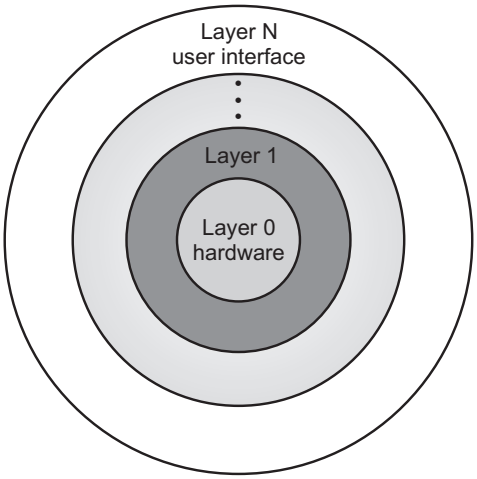


Fig. 1.25: Layered structure of OS

Layered structure of OS:

- The bottom layer is (layer 0) called hardware and the highest layers is user interface it is user interface.
- A typical operating system say Layer M consists of data structures and a set of routines that can be invoked by higher level layers. Layer M can invokes operations on lower level layers.
- The advantage of layered approach is simplicity of construction and debugging.
- The first layer can be debugged without any concern for the rest of the system, because it uses basic hardware. Once first layer is debugged its correct functioning can be assumed then second layer can be debugged and so on. If any error occur during debugging it becomes easy to identify so design and implementation of system are simplified.
- Each layer is implemented with only those operations provided by lower level layers. A Layers does not need to know how these operations are implemented; it needs to know only what these operations do.
- A generalized approach is to organize the operating system as a hierarchy of layers, each one constructed upon the one below it.
- The system had 6 layers as below.

Table 1.1: Functions of Layers

Layer	Function
5	The operator
4	User program
3	Input / Output management
2	Operator process communication
1	Memory and Drum management
0	Processor allocation and Multiprogramming

- **Layer 0** deals with allocation of the processor, switching between processes when interrupts occurred or timers expired.
- Above **Layer 0**, the system consists of sequential processes, each of which could be programmed without having to worry about the fact that multiple processes are running on a single processor. Layer 0 provides the basic multi programming of the CPU.
- **Layer 1** does the memory management. It allocates space for processes in main memory and on 512 K word drum used for holding parts of processes (pages) for which there was no room in main memory.

- Above **layer 1**, processes did not have to worry about whether they are in memory or on the drum; the layer 1 software takes care of making sure that pages are brought into memory whenever they are needed.
- **Layer 2** handles communication between each process and the operator console.
- Above this layer each process effectively had its own operator console. Layers 3 take care of managing the I/O devices and buffering the information streams to and from them.
- Above **Layer 3** each process can deal with abstract I/O devices with nice properties instead or real devices with many peculiarities.
- **Layer 4** is where the user programs are found. They do not have to worry about process memory, console or I/O management.
- The system operator process is located in **Layer 5**.
- The layering scheme is really only a design aid, because all the parts of the system are ultimately linked together into a single object program.

Advantages:

1. The main advantages of the layered approach is Modularity.
2. Easy for debugging and system verification.
3. Each layer hides the existence of certain data structures, operations and hardware from higher level layers.

Disadvantages:

1. Careful definition of the layers.
2. Less efficient than other types.

1.5.4 Micro-Kernels

- Kernel is core part of operating system which manages system resources. It is like bridge between hardware and application.
- **Micro-kernel** is a software or code which contains the required minimum amount of functions, data, and features to implement an operating system. It provides a minimal number of mechanisms, which is good enough to run the most basic functions of an operating system.
- It allows other parts of the operating system to be implemented as it does not impose a lot of policies.

Architecture:

- In micro-kernel architecture, only the most important services are put inside the kernel and rest of the OS service are present in the system application program. Now the user can easily interact with those not-so important services within the system applications and kernel i.e., microkernel is solely responsible for the three most important services of operating system namely Inter-process communication, Memory Management and CPU Scheduling.

- Fig. 1.26 shows Architecture of Micro-kernel.

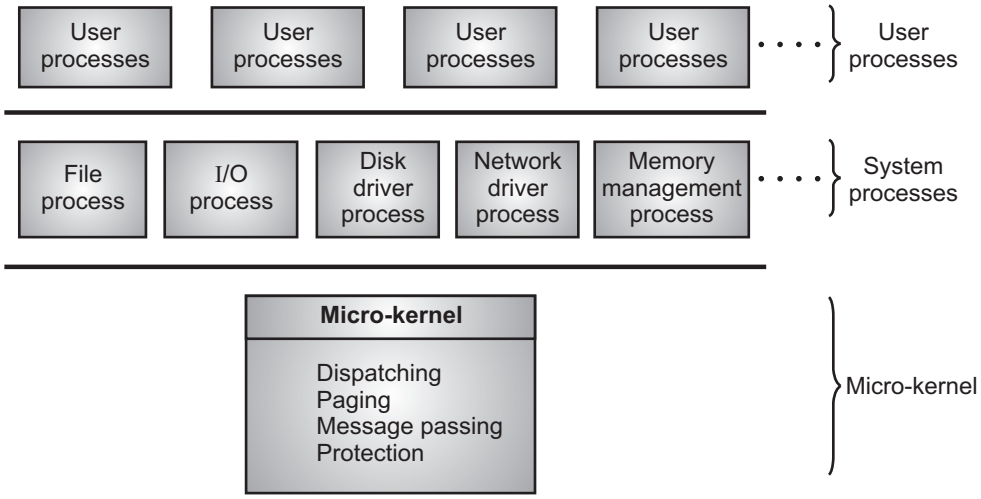


Fig. 1.26: Micro-kernel Architecture

Advantages:

1. **Communication:** The main advantage of the micro-kernel is to provide a communication facility between the client program and the various services that are also running in user mode. Communication is provided by message passing.
2. **Ease of extending:** The benefits of the micro kernel approach include the ease of extending the operating system. Extending of operating system is easier, since all new services are added to user space and consequently do not require modification of the kernel.
3. **Portability:** Because of small size of kernel, it is easy to port operating system from one hardware to other.
4. **Security and Reliability:** Micro-kernel also provides more security and reliability. Several contemporary operating systems have used the micro-kernel approach.

Disadvantages:

1. Suffer from performance decreases due to increased system function overhead.

1.5.5 Modules

- Modular Operating System is built with its various functions divided into unique processes, with its own interface.
- The primary benefit of modular approach is that each process operates independently, if any one of them fails or need updates; it does not affect any other functions.
- The main elements of modular operating system are kernel and set of dynamically loaded applications with their discrete memory spaces.
- The Linux kernel is modular which means it can extend its capabilities through dynamically loaded kernel modules.
- Modern OS development is object-oriented, with a relatively small core kernel and a set of modules which can be linked dynamically. See for example the Solaris structure, as shown in Figure below.

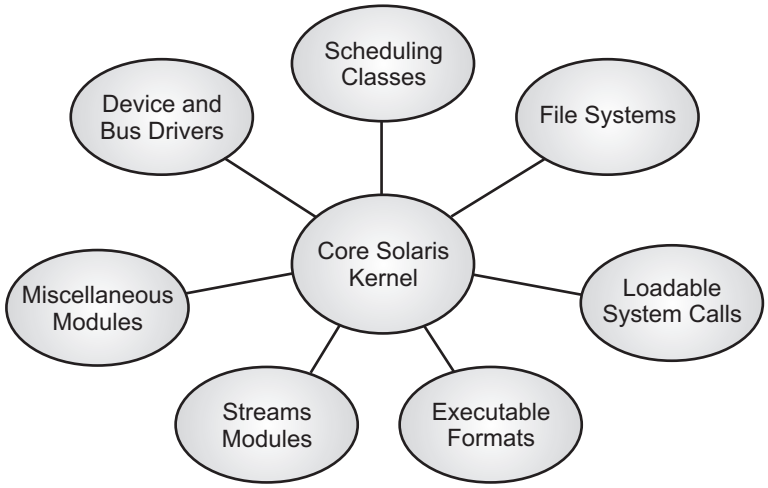


Fig. 1.27: Solaris OS Module

- Solaris operating system structure is organized around core kernel with different loadable kernel modules:
  1. Scheduling classes
  2. File systems
  3. Loadable System calls
  4. Executable formats
  5. STREAMS Modules
  6. Miscellaneous
  7. Device and bus drivers
- Such kind of design allows kernel to provide core services yet allows certain features to be implemented dynamically.
- The modular approach is similar to the layered approach in the sense that each kernel module has well-defined interfaces. However, it is more flexible than the layered approach as each module is free to call any other module.

1.6 VIRTUAL MACHINES – INTRODUCTION, BENEFITS

Introduction:

- Virtual machine is a virtual environment which simulates physical machine. It has its own CPU, memory, Network Interface and storage but they are independent from physical hardware.
- It is defined as software of a computer which provides functionality similar to physical computer.
- Software called hypervisor separates the machine resources from the hardware and provisions them appropriately so they can be used by the Virtual Machine.
- The physical machine equipped with hypervisor such as kernel based virtual machine is called Host machine or Host. The many VM's that use its resources are guest machines, the hypervisor treats computer resources like CPU, memory and storage as a pool of resources that can easily be relocated between existing guests or to new virtual machines.

- Virtual machines can exist on single piece of hardware like server. They can be moved between host servers depending on demand.
- When we run different processes on an operating system, it creates illusion that each process is running on different computer having its own virtual memory.

Benefits of Virtual Machine:

1. It provides good security.
2. Virtual machine supports the research and development of operating system.
3. It solves system compatibility problems.
4. It solves the problem of system development time.
5. They allow multiple operating system environments to exist simultaneously on the same machine.
6. They empower users to go beyond limitations of hardware to achieve their goals. Using VM's ensure application provisioning, better availability, easy maintenance and recovery.

Summary

- Operating System(OS) is a program that controls the execution of application programs and acts as an interface between applications and the computer hardware.
- An operating system is a program designed to run other programs on system.
- Operating systems are classified into different types such as single user single tasking OS (DOS), Multi-User Multi-tasking OS (Windows, Linux etc).
- System Architecture consists of Single processor system, Multi-processor system and Cluster system.
- Services provided by OS: User interface, Program Execution, I/O Operations, File System Implementation, Communications, Protection, Error detection, Accounting Resource allocation.
- Types of OS: Batch Operating System, Multi programming, multiprocessing systems, Multi-tasking or time sharing system, Distributed Systems, Real Time Systems etc.
- The structure of operating system consists of four layers. Those are hardware, operating system, application programs and users.
- Virtual Machine ~~It~~ is defined as software of a computer which provides functionality similar to physical computer. Benefits of virtual machines are flexibility, portability, isolation, security.

Check Your Understanding

1. Which of the following is not an operating system?
   (a) Windows (b) Linux
   (c) DB2 (d) Mac
2. What is meant by boot?
   (a) installing a software (b) restarting computer
   (c) scanning program (d) turn off computer

3. Which is the Linux operating system?
   (a) Private operating system (b) Windows operating system
   (c) Open-source operating system (d) None of these
4. Which is single user single tasking operating system?
   (a) Windows (b) Linux
   (c) DOS (d) Mac OS
5. To access services of operating system, the interface provided is _____.
   (a) API (b) GUI
   (c) System calls (d) High level instructions
6. Spooling stands for _____.
   (a) Simultaneous Peripheral Operations Online
   (b) Spontaneous Peripheral Operations Online
   (c) Serial Peripheral Operations Online
   (d) None of the above

Answers

1. (c)	2. (b)	3. (c)	4. (c)	5. (c)	6. (a)

Practice Questions

Q.I Answer the following questions in short:

1. Define Operating System.
2. List types of operating system?
3. List characteristics of operating system?
4. What is virtual system?
5. What is layered approach of operating system?
6. Which is bottom layer of operating system structure?
7. What is multiprogramming?

Q.II Answer the following questions:

1. What are the basic functions of operating systems?
2. With suitable diagram describe structure of operating system.
3. Define operating system? Enlist various characteristics of operating system.
4. List the different types of operating systems. Explain any one.
5. With suitable diagram describe batch operating system.
6. What are the objectives of operating system?
7. With the help of diagram describe distributed system.
8. Differentiate between multi programming and multitasking operating systems.
9. Explain computer system architecture.
10. What is virtual machine? Explain its benefits.
11. List and explain two types of multiprocessor system.

Q.III Define terms:

1. Real-time systems
2. Modules
3. Spooling
4. Time-sharing system
5. Batch operating system
6. Microkernel
7. Linux OS

Previous Exam Questions

Summer 2018

1. What is the purpose of command interpreter? **[2 M]**

Ans. Refer to Section 1.1.8.

2. List and explain services provided by the operating system. **[4 M]**

Ans. Refer to Section 1.3.

Winter 2018

1. Define the term Operating System. **[2 M]**

Ans. Refer to Section 1.1.

2. What is meant by multiprocessing system? **[2 M]**

Ans. Refer to Section 1.4.

3. List and explain services provided by operating system. **[4 M]**

Ans. Refer to Section 1.3.

Summer 2019

1. List two types of multiprocessor. Explain both in detail. **[4 M]**

Ans. Refer to Section 1.2.2.

■■■

2... System Structure

Objectives...

- To learn about different user Operating System Interfaces.
- To know about Concept of System Calls.
- To study about the System Program.

2.1 USER OPERATING SYSTEM INTERFACE

- Almost all operating systems have User Interface (UI). This interface can take several Forms one is Command-Line User Interface, Batch Interface, Menu Driven User Interface and Graphical User Interface (GUI).
- The importance of user interfaces are:
  - To assist users interacting with software.
  - To control how a user enters data and instructions.
  - To control how information is displayed.

Types of User Interface:

1. **Command-Line User Interface(CLI)**

Fig. 2.1 (a): Example of CLI

- The Command-line user interface requires a user to type commands or press special keys on the keyboard to enter data and instructions that instruct the Operating system what to do. It has to be typed one line at a time.
- The Command-line user interface also requires memorization. The advantage of Command-line interface is that it helps the user to operate the computer quickly after memorizing the keywords and syntax.
- Command line interfaces are also called command-line user interfaces, console user interfaces and character user interfaces.

2. Menu Driven User Interface:

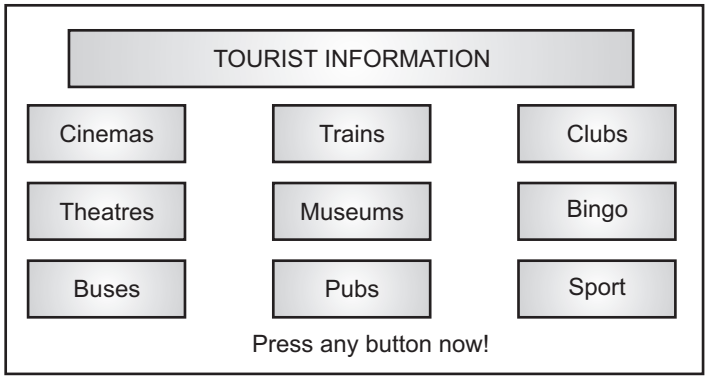


Fig. 2.1 (b): Example of Menu driven user interface

- It avoids memorizing the commands. Also it is much easier than Command Line Interface. Here user is displayed with menus and submenus by clicking the appropriate option specific tasks carried out.

3. Graphical User Interface:

- It is a software interface that user interact with a pointing device, such as mouse. These are very user friendly because they are easy to interact with system and to execute instructions. For example, when you are using Internet you are looking at the GUI of web browser.

Fig. 2.1 (c): Example of GUI

Basic Components of a GUI:

- Graphical user interfaces, such as Microsoft Windows and the one used by the Apple Macintosh, feature the following basic components:
  - **Pointer:** A symbol that appears on the display screen and that you move to select objects and commands. Usually, the pointer appears as a small angled arrow. Text processing applications, however, use an I-beam pointer that is shaped like a Capital I.

- **Pointing device:** A device, such as a mouse or trackball that enables you to select objects on the display screen.
- **Icons:** Small pictures that represent commands, files or windows. By moving the pointer to the icon and pressing a mouse button, you can execute a command or convert the icon into a window. You can also move the icons around the display screen as if they were real objects on your desk.
- **Desktop:** The area on the display screen where icons are grouped is often referred to as the desktop because the icons are intended to represent real objects on a real desktop.
- **Windows:** You can device the screen into different areas. In each window, you can run a different program or display a different file. You can move windows around the display screen and change their shape and size at will.
- **Menus:** Most graphical user interfaces let you execute commands by selecting a choice from a menu.

4. Form-based interfaces:

- Uses text-boxes, text-areas, check-boxes etc. to create electronic form. User completes in order to enter data into the system.

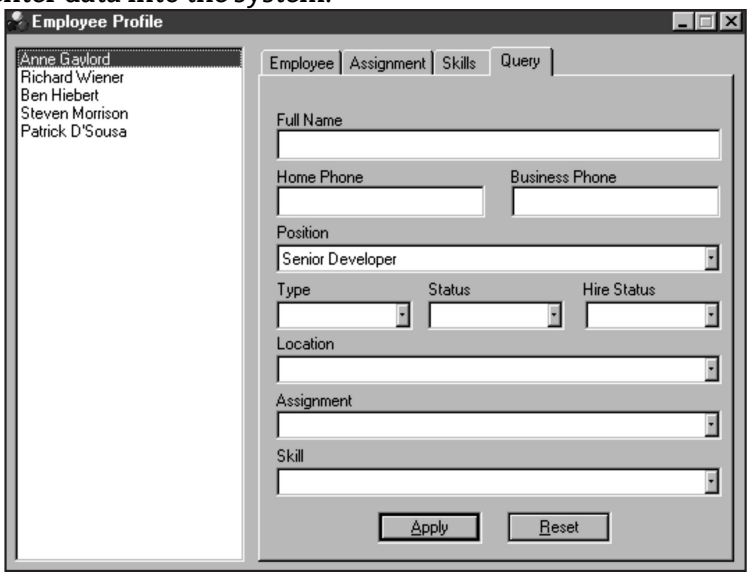


Fig. 2.1 (d): Example of a Form-based Interface

2.2 SYSTEM CALLS

- System calls are programming interface to the services provided by the operating system.
- A system call is a request by the user to the operating system to do something on user's behalf. System call provides an interface between a running program and an operating system.
- System calls are instructions that generate an interrupt that causes the operating system to gain control of the processor.
- When a program makes a system call, the mode is switched from user mode to kernel mode. This is called a **Context Switch**.

- Typically, system calls written in a high level languages like C or C++.
- The operating system then determines what kind of system call it is and performs the appropriate services for the system caller.
- For example, for I/O 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.

2.2.1 How to make a System Call?

- A system call is made using the system call machine language instruction. These calls are generally available as assembly language instructions and are usually listed in the manuals used by assembly - language programmers.
- Certain systems allow system calls to be made directly from a higher language program, in which case the calls normally resemble predefined function or subroutine calls. They may generate a call to a special run-time routine that makes the system call.
- Several languages such as C, C++ have been defined to replace assembly language for systems programming. These languages allow system calls to be made directly.
- For example, UNIX system calls may be invoked directly from a C or C++ program. System calls for Microsoft Windows Platforms are part of the Win32 API, which is available for use by all the compilers written for Microsoft Windows.
- Steps involved in making a system call:
  - We will take the example of a simple 'read system call.' It has three parameters along with it, the first one specifying the file, the second specifying the pointing to the buffer, and the third one giving the number of bytes to read.

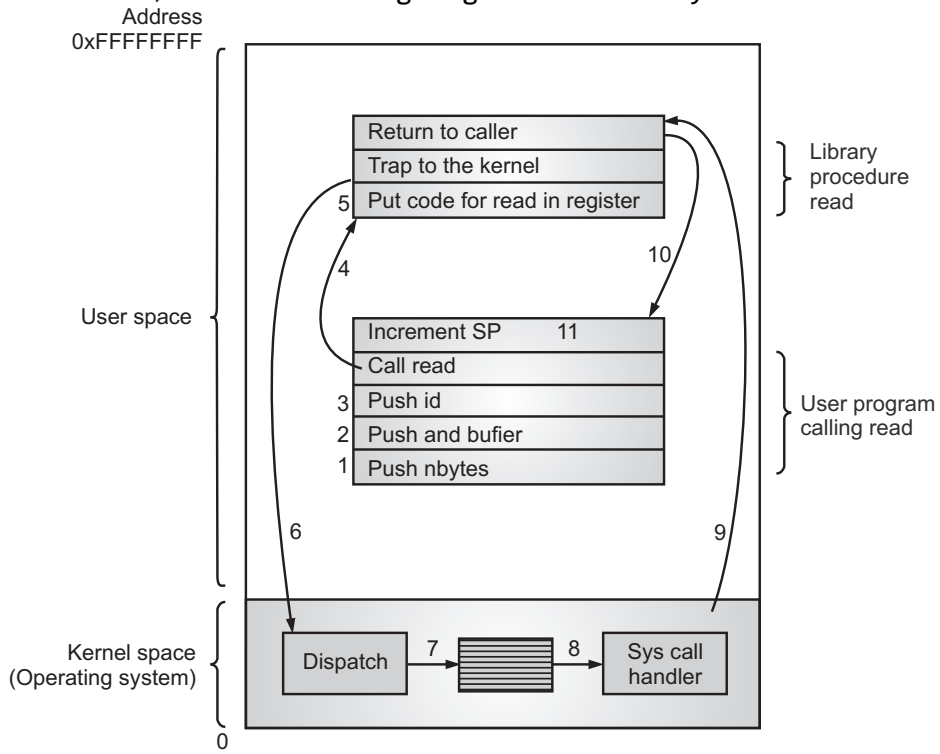


Fig. 2.2: Steps to make System Call

```
count= read(fd, buffer, nbytes)
```

- The count consists of total number of bytes to read and if some error occurs it is set to –1 which is also indicated by a global variable *errno*. The program should check this error variable timely to check if any error has occurred. The 1st and the 3rd parameters are passed by value and the 2nd parameter is passed by reference i.e. the buffer address is passed.

Steps 1-3 : The calling program pushes the parameters onto the stack.

Step 4 : The actual call to the library procedure is made. This instruction is the normal procedure call instruction used to call all procedures.

Step 5 : The library procedure places the system call number into the register.

Step 6 : The library procedure executes a TRAP instruction to switch from user mode to kernel mode and start execution at a fixed address within the kernel.

Step 7 : The kernel examines the system call number and then dispatches it to the correct system call handler. This correct number is given in the table of system call handlers by pointers referenced at the system call number.

Step 8 : The system call handler works.

Step 9 : The operation is completed, and the user is given back control once the TRAP instruction is set.

Step 10 : Then the control is transferred to the user program from the read procedure

Step 11 : Finally, SP is incremented to clean up the stack. In this way the job of read system call is completed.

Types of System Calls:

- System calls can be roughly grouped into following major categories:
  1. Process or Job Control.
  2. File Management.
  3. Device Management.
  4. Information Maintenance.
- In the following sections, we will see details of each category.

(i) File and I/O System Calls:

open	Get reading to read or write a file.
create	Create a new file and open it.
read	Read bytes from an open file.
write	Write bytes to an open file.
close	Indicate that you are done reading or writing a file.

(ii) Process Management System Calls:

create process	Create a new process
exit	Terminate the process making the system call
wait	Wait for another process to exit
fork	Create a duplicate of the process working the system call
execv	Run a new program in the process making the system call

(iii) Interprocess Communication System calls:

createMessageQueue	Create a queue to hold messages
SendMessage	Send a message to a message queue
ReceiveMessage	Receive a message from a message queue

2.3 PROCESS OR JOB CONTROL [S-18]

- Process is nothing but program in execution. The execution process must be in sequential manner.
- A running program needs to be able to halt its execution either normally (end) or abnormally (abort).
- If the program discovers an error in its input and wants to terminate abnormally, it may also want to define an error level.
- A process or job executing one program may want to load and execute another program.
- This allows the control card interpreter to execute program as directly by the control cards of the user job.
- If we create a new job or process, we should be able to control its execution. We may also want to terminate a job or process that we created (terminate process).
- If we find that it is incorrect or no longer needed we may require waiting time to finish execution (wait time). Another set of system calls are helpful in debugging a program.

System calls related to Process Control are:

(i) End, Abort.
(ii) Load, Execute.
(iii) Create process, Terminate process.
(iv) Ready process, Dispatch process.
(v) Suspend process, Resume process.
(vi) Get process attributes, Set process attributes.
(vii) Wait for Time.
(viii) Wait event, Signal event.
(ix) Change the priority of a process.

2.4 DEVICE MANAGEMENT

- Files can be thought of as abstract or virtual devices. Thus, many of the system calls for files are also needed for devices.
- If there are multiple users of the system however, the users must first request the device to ensure that they have an exclusive use of it.
- After the users are finished with the device, they must release it. These functions are similar to the open/close system calls for files.
- Once, the device has been requested the users can read, write and reposition the device just as with files.

- In fact, the similarity between input/output devices and files is so great that many operating systems merge the two into a combined file/device structure. In this case input/output devices are identified by special file names.

System calls related to Device Management are:

(i) Request a device, Release a device.

(ii) Read, Write, Reposition.

(iii) Get device attributes, Set device attributes.

(iv) Logically attach or detach devices.

2.5 FILE MANAGEMENT

- Operating System provides several common system calls dealing with files.
- Users need to be able to create and delete files. Such system calls require the name of the file and perhaps some of its attributes. Once the file is created, the users need to open it and use it.
- They may also need to read, write and reposition a file. Finally, they need to close the file, indicating that they are no longer using it.
- The users may need the same sets of operations for directories if there is a directory structure in the file system.
- In addition, for either files or directories, users need to be able to determine the values of various attributes and perhaps reset them if necessary.
- File attributes include the file name, file type, protection codes, accounting information and so on. Two system calls get file attributes and set file attributes are required for this function.

System calls for File Manipulation are:

(i) Create a file, Delete a file.

(ii) Open a file, Close a file.

(iii) Create a directory.

(iv) Read, Write, Reposition.

(v) Get file attributes, Set file attributes.

(vi) Create a link.

(vii) Change the working directory.

2.6 INFORMATION MAINTENANCE

- Many system calls exist simply for the purpose of transferring information between the user program and the operating system.
- For example, most systems have a system call to return the current time and date. Other system calls may return information about the system such as the number of current users, the version number of the operating system, the amount of free memory or disk space and so on.
- In addition the operating system keeps information about all of its jobs and processes and there are system calls to access this information.
- Generally, there are also calls to reset it, (get process attributes and set process attributes).

System calls related to Information Maintenance are:

(i) Get Time or Date, Set Time or Date.

(ii) Get system Data, Set system Data.

(iii) Get process, File or Device attributes.

(iv) Set process, File or Device attributes.

2.7 COMMUNICATION

- These system calls are useful for InterProcess Communication(IPC). They also deal with creating and deleting a communication connection.
- There are two common models of IPC, one is the Message-passing Model and another is the Shared Memory Model.
- Message-passing Model uses a common mailbox to pass messages between processes.
- Shared memory Model use certain system calls to create and gain access to create and gain access to regions of memory owned by other processes. The two processes exchange information by reading and writing in the shared data.
- The operating system and the various active application programs are only guaranteed to interact easily if the individual processes are well-coordinated with one another. For this reason, communication via relevant system calls is essential.

System calls related to Communication are:

(i) Create, delete communication connection.

(ii) Send, receive messages.

(iii) Transfer status information.

(iv) Attach and detach remote services.

2.8 SYSTEM CALL IMPLEMENTATION

- Basically, a number is associated with each system call. It is used to number the system calls.
- System call interface maintains a table indexed according to these numbers. The system call interface invokes intended system call is operating system kernel and return status of the system call and any return values.
- The caller needs to know nothing about how the system call is implemented. Just need to obey API and understand what operating system will do as a result call.
- Most details of operating system interface are hidden from programmer by API. It is managed by run-time support library.
- Most of the systems calls are available in assembly language. Some systems may allow system calls to be written in higher-level language program, in which the calls normally like predefined function or subroutine calls.
- They may be generated call to a special run-time routine that makes the system calls, or the system call may be generated directly in line.

- **Example:** Consider two files: Input file and Output file. The program is to read data from input file and copy into output file. To execute this program, the sequence of system calls occurs as follows:
  1. To obtain two file names from keyboard, system calls are required.
  2. Then system call is open input file and creates an output file.
  3. While opening, if file is not exist then program will print a message or terminate abnormally. This requires again system calls.
  4. If file exists, then for reading from the input file (a system call) and writing to output file (another system call) calls are required.
  5. For closing both files, we required two system calls.
- Some operating system provides API which invokes the actual system calls on behalf of the application programmer. Most of the programming languages provides system call interface, which intercepts function calls in the API and invokes system calls within operating system.
- The relationship between an API, the system call interface and the operating system is shown in Fig. 2.3.

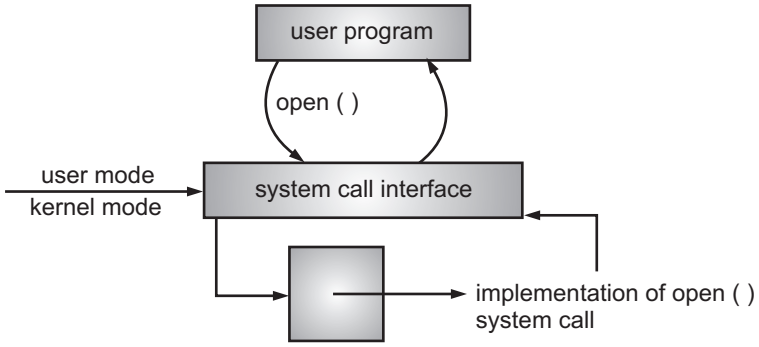


Fig. 2.3: Procedure to Invoke the system call

- The parameter can be passed by using registers to implement system call. Sometimes parameters are stored in a block in memory and the address of block or table is passed as a parameter in register. UNIX uses such a mechanism.
- Implementation of system call varies on different systems. Generally, a unique number identifies the type of system call and a system call is made by calling the particular number.
- Sometimes there is a need of sending some additional information along with, when the call is made. That is called as, parameter or parameter list of the system call [e.g. to open file, we need to specify file name and mode in which to open the file along with system call fopen( )].
- Special register is kept aside for the purpose of sending parameters. If parameters are too many in number and are beyond the capacity of the register to hold, parameters are stored in a block or table in the memory and the base address of it is passed as parameter through the register.
- Some programming languages like C permit system calls through program. To really understand what operating system does, we must examine these calls closely.

2.9 IMPORTANT SYSTEM CALLS

- **wait():**
  - In some systems, a process needs to wait for another process to complete its execution. This type of situation occurs when a parent process creates a child process, and the execution of the parent process remains suspended until its child process executes.
  - The suspension of the parent process automatically occurs with a wait() system call. When the child process ends execution, the control moves back to the parent process.
- **fork():**
  - The fork() system call is used to create processes. When a process (a program in execution) makes a fork() call, an exact copy of the process is created. Now there are two processes, one being the parent process and the other being the child process.
  - With the help of this system call, the parent process creates a child process and the execution of the parent process will be suspended till the child process executes.
- **exec():**
  - The exec() system call is also used to create processes. But there is one big difference between fork() and exec() calls. The fork() call creates a new process while preserving the parent process. But, an exec() call replaces the address space, text segment, data segment etc. of the current process with the new process.
- **exit():**
  - The exit() system call is used to terminate program execution. Specially in the multi-threaded environment, this call defines that the thread execution is complete. The OS reclaims resources that were used by the process after the use of exit() system call.

2.10 SUMMARY OF SYSTEM CALLS IN UNIX AND WINDOW

Table 2.1: Types of System calls in Unix and Windows

Types of System Calls	UNIX	Windows
Process control	fork() wait() exit()	CreateProcess() WaitForSingleObject() ExitProcess()
File manipulation	open() read() write() close()	CreateFile() ReadFile() WriteFile() CloseHandle()

contd. ...

Device manipulation	ioctl() read() write()	SetConsoleMode() ReadConsole() WriteConsole()
Information maintenance	getpid() alarm() sleep()	GetCurrentProcessId() SetTimer() Sleep()
Communication	pipe() shm_open() mmap()	CreatePipe() CreateFileMapping() MapViewOfFile()
Protection and Security	chmod() chown() umask()	SetFileSecurity() SetSecurityDescriptorGroup() IntializeSecurityDescriptor()

- **Example:** Consider 'C' fragment running on UNIX and invokes system call open() for a file. Initially the program is loaded into memory and starts the execution. The system uses fork() and exec() system calls. The configuration is shown in Fig. 2.4 (a).

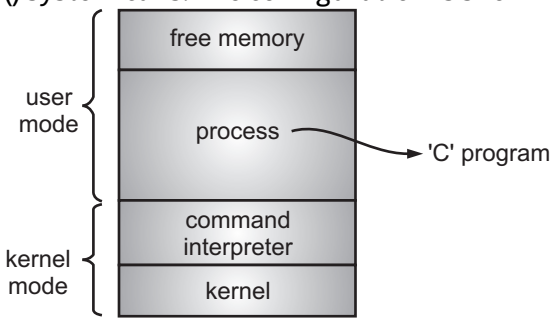


Fig. 2.4 (a): Running a program

- Let us assume 'C' program invoke a file opening statement. The Fig. 2.4 (b) shows how C library handles a open() calls.

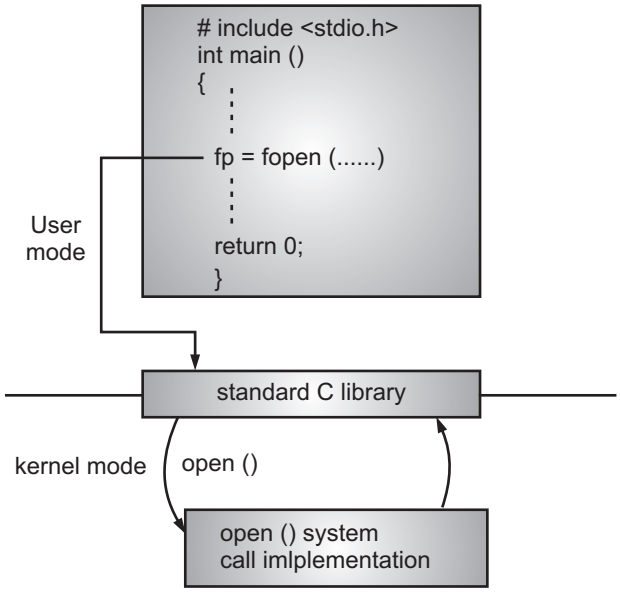


Fig. 2.4 (b): Handling a system call in 'C'

2.11 SYSTEM PROGRAM [S-18, S-19]

- System programs provide basic functioning to users so that they do not need to write their own environment for program development (editors, compilers) and program execution (shells).
- These programs act as an interface between the operating system and the application programs. They provide an environment in which application programs can be developed and executed in a convenient manner.
- These system programs can be divided into several categories:
  1. **File Manipulation:** These programs create, delete, copy, rename, print, dump, list and generally manipulate files and directories.
  2. **Status Information:** Some programs simply ask the operating system for the date, time, and amount of available memory or disk space, number of users or similar status information.
  3. **File Modification:** Several text editors may be available to create and modify the contents of files stored on a disk or a tape.
  4. **Programming Language Support:** Compilers, assemblers and interpreters for common programming languages. (Such as Fortran, Cobol, Pascal, Basic, C and so on) are often provided with the operating system.
  5. **Program Loading and Execution:** Once a program is assembled or compiled, it must be loaded into the memory to be executed. The operating system may provide absolute loaders, linkage editors and Debugging systems in order to achieve this.
  6. **Applications Programs:** In addition, most operating systems come with programs which are useful to solve some particularly common problems, such as Compilers, Text Formatters, Plotting Packages, Database Systems and so on.

2.12 CONCEPT OF OPERATING SYSTEM STRUCTURE

- This point we have already covered in first chapter. Please refer to Unit 1 point 1.5.

Summary

- Operating system provides interface to users to interact with applications through command line interface, menu driven interface, Graphical user Interface.
- System calls provide interface between the process and operating system. System calls are programming interface to the service provided by operating system. A system call is made using the system call machine language instruction. There are file and I/O system calls, Process management system calls, Inter-process communi-cation system calls.
- Process or job control is program in execution it must be in sequential manner. Operating system provides several common system calls with files.
- System calls related to device management are request a device, release a device, read, write, reposition, get device attributes, set device attributes, logically attach or detach devices.

- User need to do basic operations on files, system calls for file manipulations are : create file, open file, delete file, close a file, Read and write file, create a directory.
- System programs provide basic functioning to users so that they do not need to write their own environment for program development (editors, compilers) and program execution (shells). These are categorized into file manipulation, status information, file modification, programming language support, program loading and execution, application programs.

Check Your Understanding

1. In Layered operating system which is the highest level?
   (a) Hardware (b) User Interface
   (c) Kernel (d) None of Above
2. What is the another name used for command line interpreter?
   (a) Shell (b) Kernel
   (c) Command (d) Prompt
3. Which system call is used for creating a file?
   (a) Read (b) Write
   (c) Open (d) Close
4. Which system call does not return control to calling point, on termination?
   (a) fork (b) exec
   (c) wait (d) exit
5. A fork system call will fail if _____.
   (a) The previously executed statement is also fork call
   (b) The limit on maximum no of processes in system would be executed
   (c) The limit on minimum no of processes in system that can under execution
   (d) All of above

Answers

1. (b)	2. (c)	3. (c)	4. (b)	5. (b)

Practice Questions

Q.I Answer the following questions in short:

1. Which are major types of system call?
2. What is interface? State its types?
3. Which are Information Maintenance system calls?
4. What is Device Management?
5. What is File Management?
6. Define operating system structure?

Q.II Answer the following questions:

1. Explain controls of GUI?
2. Explain the following system calls:
   (i) wait
   (ii) exit

3. What is meant by system calls? How to make it?
4. With neat diagram describe structure of operating system.
5. What is system program? What are its types? Explain in detail.
6. Write any two system calls of device manipulation.
7. Write system calls related to file management.

Q.III Define terms:

1. Shared-memory model
2. Message passing model
3. Process or job control
4. System call
5. System program

Previous Exam Questions

Summer 2018

1. What is Process? **[2 M]**

Ans. Refer to Section 2.3.

2. Define System Program. **[2 M]**

Ans. Refer to Section 2.11.

3. List and explain system calls related to Process and Job control. **[4 M]**

Ans. Refer to Section 2.3.

Summer 2019

1. Explain various types of system program. **[4 M]**

Ans. Refer to Section 2.11.

■■■

3... Process Management

Objectives...

- To learn about concept of the Process.
- To get knowledge about Process Scheduling.
- To study about operations on Process such as Process Creation and Termination.
- To know about Interprocess Communication.

3.1 WHAT IS PROCESS?

- A process is a program in execution. As the program executes the process changes state.
- The state of a process is defined by its current activity.
- Process execution is an alternating sequence of CPU and I/O bursts, beginning and ending with a CPU burst. Thus, each process may be in one of the following states: **New, Active, Waiting** or **Halted**.
- In Process model, the operating system is organized into a number of sequential processes, or just processes for short.
- A process is just an executing program, including the current values of the program counter, register and variables.
- Conceptually, each process had its own virtual CPU.
- In reality, of course, the real CPU switches back and forth from process to process; thus it is much a collection of processes running in parallel. This rapid switching back and forth is called Multiprogramming.
- In Fig. 3.1 (a), we see a computer multiprogramming four programs in the memory. In Fig. 3.1 (b), we can see how this has been abstracted into four processes, each with its own flow of control (i.e. its own program counter), and each one running independent of the other ones. In third Fig. 3.1 (c), we can see that viewed over a long enough time interval, all the processes have made progress, but at any given instant only one process is actually running.

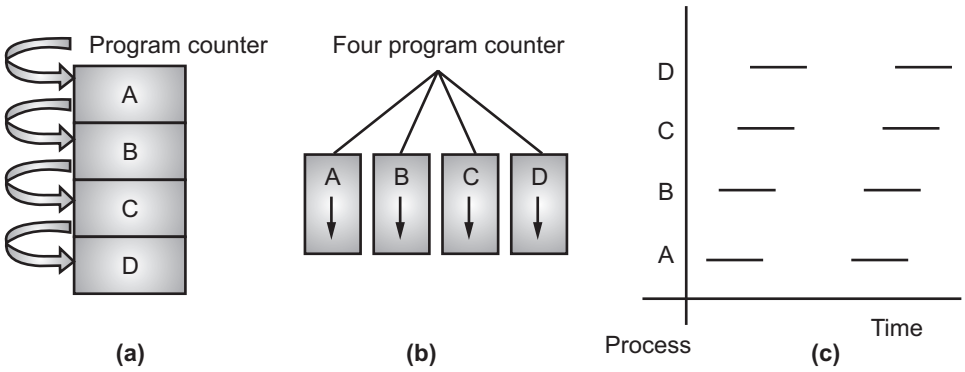


Fig. 3.1: The Process Model

- With the CPU switching back and forth among the processes, the rate at which a process performs its computation will not be uniform, and probably not even reproducible if the same processes are run again.
- Thus, processes are an activity of some kind. It has a program, input, output and a state a single processor may be shared among several processes, with some scheduling algorithm being used to determine when to stop work on one process and service a different one.

3.1.1 Process States [S-18]

- A process is a program in execution which includes the current activity and this state is represented by the program counter and the contents of the processor's register.
- There is a process stack for storage of temporary data.
- There may be a situation when two processes may be associated with the same program but they are considered for two separate execution processes.
- A user can have several programs running and all these programs may be of a similar nature but they must have different processes.
- As a process executes, it changes state. The state of a process is defined in part by the current activity of that process.
- We have two types of process state models which are :
  1. **Two State Model:** This represents a simple model by observing that a process is either being executed by a processor or not. It can be two states Running or Not running.

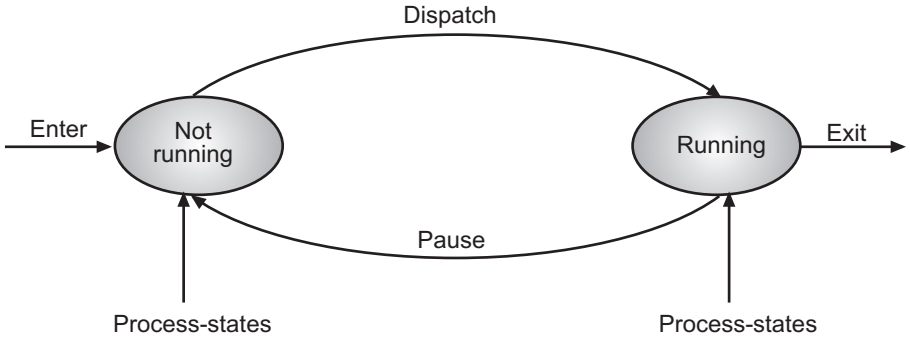


Fig. 3.2: State Transition diagram of Two-State Process Model

(a) Running state:

- When a new process is created, it enters into the system as in the running state.
- The process is waiting for an opportunity to operate, at times the currently running process will be interrupted and the dispatcher of the operating system will select a new process to run, one of the processes is now in the **running** state.

(b) Not running State: Processes that are not running must be kept in some queue, waiting their turn to be executed. Each entry in the queue is a pointer to a particular process.

2. Five State Model : In this, there are five stages and each process may be in one of the following states :

(a) New : The new process is created when a specific program calls from secondary memory/ hard disk to primary memory/ RAM

(b) Running : The process is being executed.

(c) Waiting : The process is waiting for some event to occur such as an I/O completion.

(d) Ready : The process should be loaded into the primary memory, which is ready for execution.

(e) Terminated : The process has finished execution.

- These names are arbitrary and they vary from operating systems to operating systems. The important thing is only one process can be running in any processor at any time. Many processes may be ready and waiting state.
- The state diagram corresponding to these states is shown in Fig. 3.3.

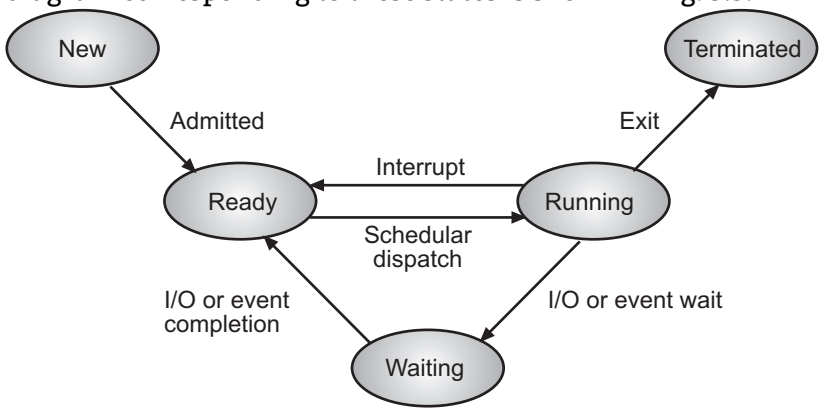


Fig. 3.3: Process States

- **New → Ready:** The operating system creates a process and prepares the process to be executed, and then the operating system moves the process into "Ready" queue.
- **Ready → Running:** When it is time to select a process to run, the operating system selects one of the jobs from the ready queue and moves the process from ready state to running state.
- **Running → Terminated:** When the execution of a process has completed, then the operating system terminates that process from running state. Sometimes, operating

system terminates the process some other reasons also include time limit exceeded, memory unavailable violation, protection error, I/O failure, data misuse and so on.

- **Running → Ready:** When the time slot of the processor expired or if processor received any interrupt signal, then the operating system shifted running process to ready state. For example, process P1 is executed by processor; in the meantime process P2 generates an interrupt signal to the processor. Then, the processor compares the priorities of process P1 and P2, if P1 > P2 then the processor continues the process P1 otherwise, the processor switched to process P2, and the process P1 moved to ready state.
- **Running → Waiting:** A process is put into waiting state, if the process need an event to occur or an I/O device requires. The operating system does not provide the I/O or event immediately then the process moved to waiting state by the operating system.
- **Waiting → Ready:** A process in the blocked state is moved to ready state when the event for which it has been waiting to occur. For example, a process is in running state need an I/O device, then the process moved to block or waiting state. When the I/O device provided by the operating system, the process moved to ready state from waiting or blocked state.

3.1.2 Process Control Block (PCB) [W-18, S-19]

- Each process is represented in the operating system by a **Process Control Block** (PCB) also called as **Task Control Block**.
- The operating system groups all information that it needs about a particular process into a data structure called a **PCB** or **Process Descriptor**.
- When a process is created, the operating system creates a corresponding PCB and releases whenever, the process terminates.
- The information stored in a PCB includes:
  - Process name (ID): Each process is given an integer identifier, termed as Process identifier, or PID.
  - Priority.

Structure of the Process Control Block:

- A PCB is shown in Fig. 3.4. It contains many data items associated with a specific process.

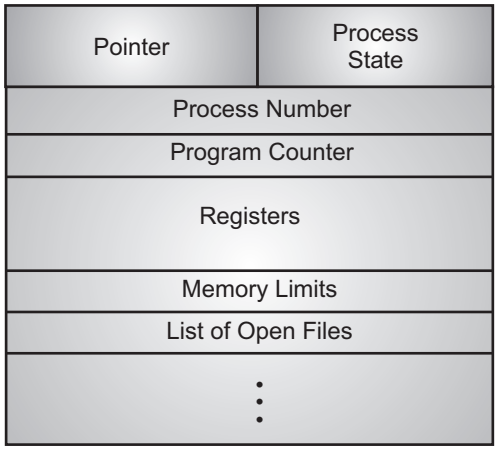


Fig. 3.4: Process Control Block (PCB)

- The following are the data items in PCB:
  - **Process State:** The state may be New, Ready, Running, and Waiting, Halted and so on.
  - **Program Counter:** The counter indicates the address of the next instruction to be executed for this process.
  - **CPU Registers:** The registers vary in number and type, depending on the computer architecture. They include accumulators, index registers, stack pointers and general purpose registers, plus any condition - code information.
  - **CPU Scheduling Information:** The information includes a process priority, pointers to scheduling queues, and any other scheduling parameters.
  - **Memory Management Information:** This information may include such information as the value of the base and limit registers, the page tables or the segment tables, depending on the memory system used by the operating system.
  - **Accounting Information :** This information includes the amount of CPU and real time used, time limits, account number, job or process numbers and so on.
  - **I/O Status Information:** The information includes the list of I/O devices allocated to this process, a list of open files and so on.

Example: The PCB in the Linux operating system contains C structure `task_struct`. It contains all necessary information about process like state of the process, its Scheduling and memory management related information, how many lists of open files, and pointers to the process's parent and children.

The structure has some of the following field members:

```
pid_t id;                    //process identifier
long state;                  //state of the process
long int time_slice          //scheduling information
struct task_struct *parent   //pointer to parent process
struct task_struct *parent   //pointer to child process
struct file_struct *files    // list of open files
struct mm_struct *mm;        //address space for the processed
```

Threads:

- Thread is the segment of a process. A process can have multiple threads.
- The process model discussed so far has implied that a process is a program that performs a single thread of execution of execution.
- For example, when a process is running a word-processor program, a single thread of instructions is being executed. This single thread of control allows the process to perform only one task at one time. For this reason, the user cannot simultaneously type in characters and run the spell checker within the same process.
- Number of modern operating systems have extended the process concept to allow a process to have multiple threads of execution and thus to perform more than one task at a time.
- On a system that supports threads, the PCB is expanded to include information for each thread. Other changes throughout the system are also needed to support threads.

3.2 PROCESS SCHEDULING [W-18, S-19]

- In operating system always a process should be executing to maximize CPU utilization. The main objective of time sharing system is to switch CPU between different processes such that user can interact with each program while running.
- The process scheduler selects the available process (from the set of available processes) for execution purpose. If there are other processes then they have to wait until the CPU is free and it can be rescheduled.

3.2.1 Scheduling Queues

- The OS maintains all PCBs in Process Scheduling Queues. The OS maintains a separate queue for each of the process states and PCBs of all processes in the same execution state are placed in the same queue.
- The Operating System maintains the following important process scheduling queues:
  1. **Job queue:** All processes when enters into the system are stored in the Job Queue.
  2. **Ready queue:** Processes in the Ready state are placed in the Ready Queue.
  3. **Device queues:** Processes waiting for a device to become available are places in device queues. There are unique device queues available for each I/O device.
- The most common representation of process scheduling is queueing diagram, in the following diagram the rectangular boxes represents queue, there are two types of queue ready queue, and set of device queue. The circle denotes resources that serves queue, arrow indicates flow of process in the system.
- When a new process is arrives it is kept in ready queue, it has to wait until it gets selected for execution, or is dispatched.
- When the process is selected for execution, one of following states can occur:
  - The process could issue an I/O request and then it is placed in I/O queue.
  - The process can create new sub process and waits for its subprocess termination.
  - The process can be removed forcefully, from the CPU, when the interrupt occurs and can be taken back in ready queue.

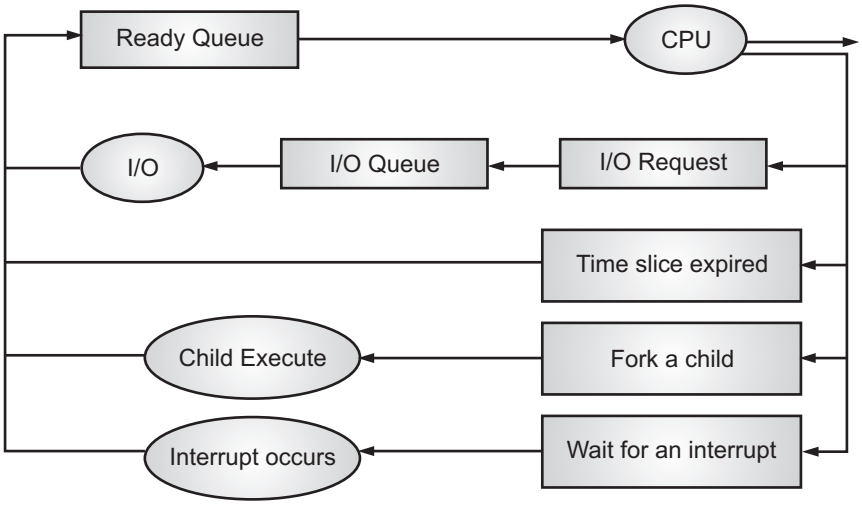


Fig. 3.5: Queuing diagram for Process Scheduling

3.2.2 Schedulers [S-18, 19]

- Schedulers are special system software which handles the process scheduling in various ways.
- Their main task is to select the jobs to be submitted into the system and to decide which process to run.
- Schedulers are of three types:
  1. Long-Term Scheduler
  2. Short-Term Scheduler
  3. Medium-Term Scheduler

1. Long-Term Scheduler

- The Long term scheduler or job scheduler selects the process from the pool and loads it into the main memory.
- Long term scheduler runs less frequently. It decides which program must get into the job queue. From the job queue, job processor selects process and loads into memory for execution. The main objective of scheduler is to maintain the degree of multiprogramming.

2. Short-Term Scheduler

- The Short Term Scheduler or CPU Scheduler selects among the processes that are ready to execute and allocate CPU to one of them. Short term scheduler runs very frequently.
- Primary aim of this is to enhance CPU performance and increase process execution rate.

3. Medium-Term Scheduler

- Medium term scheduler takes care of swapped out processes. If the running process needs some I/O time for completion then it needs to change its state from running to waiting.
- Medium term scheduler is used for this purpose. It removes the running process and makes room for new process. Such processes are called swapped out processes and this procedure is called as Swapping.
- The medium term scheduler is used for suspending and resuming the process.

3.2.3 Context Switch [W-18]

- It is a process that involves switching of the CPU from one process to another. In this, execution of the process that is present in the running state is suspended by the kernel and another process that is present in ready state is executed by the CPU.
- Switching the CPU to another process requires saving the state of the old process and loading the saved state for the new process. This task is known as a Context Switch.
- Context switch can happen due to the following reasons:
  1. When high priority process comes in the ready state, in this case running process is stopped and higher priority process should be given to CPU for execution.
  2. When an interrupt occurs then running process should be stopped and CPU should handle the interrupt which is occurred.
  3. When transaction between user mode and kernel mode is required then context switch is performed.

- While performing context switch following steps occurred:
  1. Save the context of the process that is currently running by the CPU, update the PCB and other important details.
  2. Move the PCB of above process into ready queue, I/O queue etc.
  3. Select new process for execution.
  4. Update the PCB of the selected process. It includes updating the process state to running.
  5. Update memory management data structure as required.
  6. Restore the context of the process that was previously running when it is loaded again on the processor.
- Context switch is used for multitasking that is multi programming with time sharing, here the context switch occurs so fast that user feels that the CPU is executing more than one task at same time.
- Context-switch time is pure overhead, because the system does no useful work while switching. Its speed varies from machine to machine, depending on the memory speed, the number of registers that must be copied, and the existence of special instructions.
- Context-switch times are highly dependent on hardware support. For instance, some processors provide multiple sets of registers.
- A context switch simply includes changing the pointers to the current register set. Of course, if active processes exceed register sets, the system resorts to copying the register data to and from memory.
- Also, the more complex the operating system, the more work must be done during a context switch.

3.3 OPERATIONS ON PROCESSES

- The processes in the system can execute concurrently and they must be created and deleted dynamically.
- Operations on processes are Process Creation and Termination.

3.3.1 Process Creation

- When a new process is to be added to those currently being managed, the operating system builds the data structures that are used to manage the process, and allocates address space in main memory to the process. This is the creation of a new process.

Common events lead to a Process Creation:

- In a batch environment, a process is created in response to the submission of a job. In an interactive environment, a process is created when a new user attempts to log on. In both cases, the operating system is responsible for the creation of the new process.
- When operating system decides to create a new process, it can proceed as follows:
  1. Assign a unique identifier to the new process.
  2. Allocate space for the process.
  3. Initialize the process control block.
  4. Set the appropriate linkages.
  5. Create or expand other data structures.

- Operating system created all processes in a way that was transparent to the user or application program, and this is still commonly found with many contemporary operating systems. When the operating system creates a process at the explicit request of another process, the action is referred to as process spawning.
- When one process spawns another, the former is referred to as the parent process, and the spawned process is referred to as the child process. The parent may have to partition its resources among its children, or it may be able to share some resources among several of its children.
- A sub-process may be able to obtain its resources directly from the operating system. When a process creates a new process, two possibilities exist in terms of execution :
  - The parent continues to execute concurrently with its children.
  - The parent waits until some or all of its children have terminated.

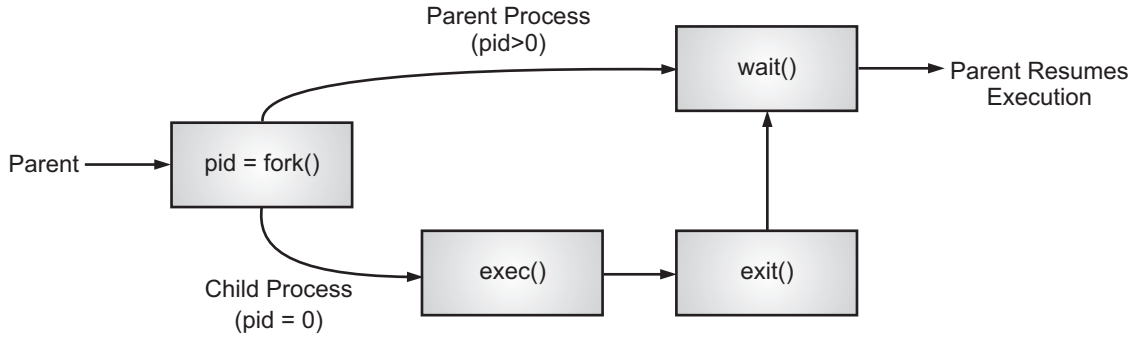


Fig. 3.6: Process Creation using fork()

Example of Process Creation on UNIX system:

- A process creates a child process in UNIX through a system call *fork*.
- fork system call creates a child process and sets up its execution environment, then it allocates an entry in the process table i.e. a PCB for the newly created process and marks its state as ready.
- Fork also returns the id of the child process to its creator also called the parent process.
- The child process shares the address space and file pointers of the parent process, hence data and files can be directly shared.
- A child process can in turn create its own child processes, thus leading to the creation of a process tree.
- The UNIX operating system keeps track of the parent-child relationships throughout the lives of the parent and child processes.
- The child process in UNIX environment called its context is a copy of the parent's environment. Hence, the child executes the same code as the parent. At creation, the program counter of the child process is set at the instruction at which the fork call returns.
- The only difference between the parent and the child processes is that in the parent process fork returns with the process id of the child process, while in the child process it returns with a '0'.

3.3.2 Process Termination

- When processes terminate, it returns data to its parent process. Resources like memory, files and I/O are de-allocated by the operating system. A child process may be terminated if its parent process requests for its termination.
- Process will terminate usually following reasons:
  1. **Due to Normal exit:** When compiler has compiled the program, the compiler executes a system call to inform the operating system that task has finished. This call is exit in UNIX and Exit Process in windows.
  2. **Error Exit:** Another reason for termination is that process finds fatal error, suppose user types the command *sample.c* and no such files present, the compiler simply exits.
  3. **Fatal Error:** The third reason for termination is an error caused by the process, often due to a program bug. Examples include executing an illegal instruction, referencing nonexistent memory, or dividing by zero. In some systems (e.g., UNIX), a process can tell the operating system that it wishes to handle certain errors itself, in which case the process is signaled (interrupted) instead of terminated when one of the errors occurs.
  4. **Killed by another Process:** The fourth reason a process might terminate is that the process executes a system call telling the operating system to kill some other process. In UNIX this call is kill. The corresponding Win32 function is *Terminate Process*.

Example:

- Any process p_i can terminate itself through the exit system call,

  ```
  exit (status_code);
  ```

  Where, the value of `status_code` is saved in the kernel for access by the parent of p_i. If the parent is waiting for the termination of p_i, a signal is sent to it. The child processes of p_i are made the children of a kernel process.
- Waiting for process termination : A process p_i can wait for the completion of a child process through the system call,

  ```
  wait(add(abc));
  ```

  Where `abc` is a variable. When a child of p_i terminates the wait call returns after storing the termination status of the terminated child process into `abc`. The wait call returns with '–1' if p_i has no children.

```
main ()
{
int saved_status;
for (i = 0; i < 3; i++)
{
   if fork()== 0)
   {
      /* code for child processes */
          exit ();
   }
}
while(wait(&saved_status) != -1)
      /* All child processes terminated? */
}
```

3.4 INTERPROCESS COMMUNICATION

- There are independent processes and co-operating processes in the operating system.
  1. **Independent Process:** The processes which are independent that can not affect or be affected by other processes executed in system. Or we can say that the process which does not share data with other process is independent process.
  2. **Co-operating Process:** Co-operating process shares data with other process and its execution can get affected by other process.
- There are many reasons for providing environment that allows process cooperation.
  1. **Information Sharing:** Shared data can be required by several processes. Like shared file can be used by two or more processes, so we must allow concurrent access to such information.
  2. **Computation speedup:** To execute task faster we should break up into subtasks, so each subtask will execute parallel with other. This can be achieved only if computer has multiple processing elements.
  3. **Modularity:** Dividing the system functions into separate processes or threads.
  4. **Convenience:** Individual user may work on multiple tasks at a same time. Like, user may be editing, compiling, printing in parallel.

3.4.1 Shared Memory System

- In the shared memory system, the cooperating processes communicate with each other by establishing the shared memory region, in its address space.
- It is the fundamental model of interprocess communication. It allows fastest interprocess communication.

Working:

- In Shared Memory system, the cooperating processes communicate, to exchange the data or the information with each other. For this, the cooperating processes establish a shared region in their memory. The processes share data by reading and writing the data in the shared segment of the processes.
- Consider a situation that there are two cooperating processes P1 and P2. Both the processes have their different address space. Now P1 wants to share some data with P2. So both processes will have to perform following steps.

  Step 1 : As Process P1 has some data to share with P2. Process P1 has to take initiative and establish a shared memory region in its address space and store its data to be shared in its shared memory region.

  Step 2 : Now, P2 requires information which is in shared segment of P1. So P2 has to attach itself to shared address space of P1. Now, Process P2 can read out the data from there.

  Step 3 : Now, Processes P2 and P1 can exchange information by reading and writing data in shared segment of the process.

3.4.2 Message Passing Systems

- Operating system provides the means for cooperating processes to communicate with each other via message passing facility.
- In this method, processes communicate with each other without using any kind of shared memory.
- This communication could involve a process letting another process to know that some event has occurred or transferring of data from one process to another. This communication model is message passing model.
- This model allows multiple processes to read and write data to message queue without connected to each other. Messages are stored in the queue until their recipient retrieves them. Message queues are very useful for Interprocess Communication.

Working:

- Following diagram demonstrates the Message passing model of the process communication.

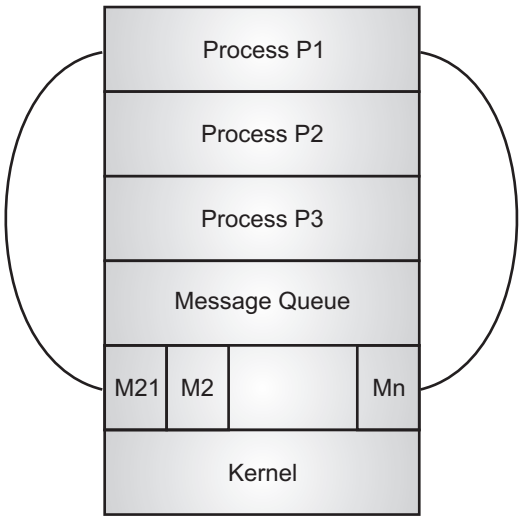


Fig. 3.7: Message Passing Model

- Message passing facility provides at least two operations: send (message) and receive (message).
- The messages send by the process can be fixed size or variable size. If fixed size messages are sent, the system level implementation is straight forward.
- Variable size messages require more complex system level implementation, but the programming task becomes simple.
- A standard message can have two parts: header and body.
- Header part is used for storing Message Type, Destination Id, Source Id, Length of Message, and Control Information.
- If the Processes P and Q want to communicate, they must send and receive messages from each other; communication link must exist between them.
- There are several methods for logically implementing a link and send() and receive() operations.

Methods of implementing communication link:

- A link has some capacity that determines the number of messages that can reside in it temporarily for which every link has a queue associated with it which can be of zero capacity, bounded capacity, or unbounded capacity.
- In zero capacity, the sender waits until the receiver informs the sender that it has received the message.
- In non-zero capacity cases, a process does not know whether a message has been received or not after the send operation. For this, the sender must communicate with the receiver explicitly. Implementation of the link depends on the situation; it can be either a direct communication link or an in-directed communication link.

1. Direct and Indirect Communication:

- Direct Communication links are implemented when the processes use a specific process identifier for the communication, but it is hard to identify the sender ahead of time. For example: the print server.
- In-direct Communication is done via a shared mailbox (port), which consists of a queue of messages. The sender keeps the message in mailbox and the receiver picks them up.

2. Synchronous and Asynchronous Communication:

- Communication between processes take place through calls to send() and receive() primitives. There are different design options for implementing each primitive message passing may be either Blocking or Non-blocking also called as Synchronous and Asynchronous.
    - **Synchronous:** A synchronous operation blocks process till the operation completes.
    - **Asynchronous:** This operation is non-blocking and only initiates the operation. The caller could discover completion by some other mechanism notion of synchronous operations requires an understanding of what it means for an operation to complete.

Summary

- Process is instance of computer program that is being executed by one or more threads. Process has its different states like, New, Ready, Running, Waiting and Terminated.
- **Process Models:** Two state model and five state model.
- Process Control Block is a data structure that contains information of the process related to it. The process control block is also known as a task control block, entry of the process table.
- Entries in the PCB are Process state, program counter, CPU Registers, CPU – Scheduling Information, Memory Management Information, Accounting information, I/O Status information.
- We have to schedule processes to CPU for multiprogramming purpose. The main objective is to increase degree of multiprogramming. Schedulers play this role to schedule processes. Short Term Scheduler, Medium Term Scheduler and Long Term Scheduler are used for this purpose.

- Switching the CPU to another process requires saving the state of the old process and loading the saved state for the new process. This task is known as a Context Switch.
- There are different operations performed on processes like Creation of process and Termination of process.
- Interprocess communication is a set of programming interface which allow a programmer to coordinate activities among various program processes which can run concurrently in an operating system. This allows a specific program to handle many user requests at the same time.
- Interprocess communication (IPC) is performed by Shared Memory Systems and Message Passing Systems.
- Message passing is a mechanism for a process to communicate and synchronize.
- In Shared Memory System, a particular region of memory is shared between cooperating processes.

Check Your Understanding

1. A process is more than program code, which is sometimes known as _____.
   (a) text section (b) contents of processor registers
   (c) stack (d) data section
2. When process gets executes it changes its state, the process may be in one of the following states.
   1. New 2. running
   3. waiting 4. ready
   5. terminated
   (a) 1, 2 (b) 1,2,4
   (c) 1,2,3 (d) 1,2,3,4,5
3. Which of the following stage defines "the process is waiting for some other event to occur"?
   (a) ready (b) running
   (c) waiting (d) terminated
4. In operating systems each process is represented by_____.
   (a) print dialog box (b) process control block
   (c) program control box (d) process command box
5. The address of next instruction to be executed for the current process is stored in _____.
   (a) CPU registers (b) program registers
   (c) stack pointers (d) program counter
6. The objective of multiprogramming is _____.
   1. to have running some processes all the time
   2. to execute single process at a time
   3. to minimize the CPU utilization
   4. to maximize the CPU utilization
   (a) 1,2 (b) 1,3
   (c) 1,4 (d) none of above

7. The process which is ready to execute and residing in main memory is kept in _____.
   (a) device queue (b) processor queue
   (c) ready queue (d) job queue
8. Which among following is true for short term scheduler?
   1. Select from among the processes that are ready to execute and allocate CPU to them.
   2. Select process from mass storage device and loads them into memory for execution.
   3. Short term scheduler must select new process for CPU.
   4. It executes less frequently.
   (a) 1 only (b) 2 and 3 only
   (c) 1, 2, 4 only (d) 1, 3
9. Copying process from memory to disk to allow space for other process is called _____.
   (a) deadlock (b) swapping
   (c) shifting (d) copying
10. PCB does not contain following _____.
   (a) code (b) stack
   (c) data (d) bootstrap program
11. Which of the following is used as synchronization tool?
   (a) thread (b) pipe
   (c) semaphore (d) socket
12. Which of the following is not a scheduler type?
   (a) short term scheduler (b) medium term scheduler
   (c) block scheduler (d) long term scheduler

Answers

1. (a)	2. (d)	3. (c)	4. (b)	5. (d)	6. (c)	7. (c)	8. (d)	9. (b)	10. (d)
				11. (c)	12. (c)				

Practice Questions

Q.I Answer the following questions in short:

1. Define the process?
2. What is interprocess communication?
3. What is swap in and swap out?
4. Define scheduler.
5. List types of interprocess communication.
6. What is thread in Process Management?

Q.II Answer the following questions:

1. What is Process? Explain with its state?
2. What is process control block?
3. Explain the term process concept in detail.
4. Explain context switch between the processes.
5. Explain the process creation and process termination
6. What is scheduler? Explain scheduling queue?
7. Explain shared memory system concept?
8. Explain message passing systems?

Q.III Define terms:

1. Shared memory
2. Message passing
3. Scheduling queues
4. Process control block
5. Context switch

Previous Exam Questions

Summer 2018

1. Explain process states in detail. **[4 M]**

Ans. Refer to Section 3.1.2.

2. Explain medium term scheduler. **[4 M]**

Ans. Refer to Section 3.2.2

Winter 2018

1. What is Context switch? **[2 M]**

Ans. Refer to Section 3.2.3.

2. Explain Process Control Block (PCB) in detail with the help of diagram. **[4 M]**

Ans. Refer to Section 3.1.3.

3. Describe process state with suitable diagram **[4 M]**

Ans. Refer to Section 3.1.2.

4. Explain medium term scheduler in detail. **[4 M]**

Ans. Refer to Section 3.2.

Summer 2019

1. Which scheduler controls the degree of multiprogramming? How? **[2 M]**

Ans. Refer to Section 3.2.

2. Explain the process Control Block with a diagram. **[4 M]**

Ans. Refer to Section 3.1.3.

3. Write a short note on medium-term scheduler. **[4 M]**

Ans. Refer to Section 3.2.2.

■■■

4...

CPU Scheduling

Objectives...

- To study about CPU Scheduling Concepts and Criteria.
- To get knowledge of CPU – I/O Burst Cycle.
- To learn about various Scheduling Algorithms such as First-Come First-Served Scheduling (FCFS), Shortest-Job-First Scheduling (SJF), Priority Scheduling, Round Robin (RR) Scheduling, Multilevel Queue Scheduling, Multilevel Feedback Queue Scheduling.

4.1 WHAT IS SHEDULING? [W-18]

- CPU scheduling is the basis of multiprogrammed operating systems.
- The idea of multiprogramming is relatively simply, if a process (job) is waiting for an I/O request, then the CPU switches from that job to another job, so the CPU is always busy in multiprogramming.
- But in a simple computer system, the CPU sit idle until the I/O request is granted.
- By switching the CPU among processes, the operating system can make the computer more productive.
- Scheduling is a fundamental operating system function, almost all the computer resources are scheduled before use.

CPU Scheduling Algorithm:

- The CPU scheduling algorithm determines how the CPU will be allocated to the process.
- CPU scheduling algorithms are two types one is non-preemptive and second one is preemptive scheduling algorithms.
- In the non-preemptive scheduling once the CPU is assigned to a process, the processor do not **release Switch** until the completion of that process.
- The CPU is assigned to some other job only after the previous job has been finished. But in the preemptive scheduling the CPU can release the processes even in the middle of the execution.
- For example, when the CPU executing the process P1 receives a request signal from process P2 in the middle of the execution, then the operating system compares the priorities of P1 and P2.

- If the priority of P1 is higher than P2, then the CPU continues the execution of process P1. Otherwise the CPU preempts the process P1 and assigns to process P2 [priority (P1 < P2)].
- CPU scheduling is the basis of multiprogrammed operating system. By switching the CPU among different processes, operating system can improve your degree of resource utilization.
- In this chapter, we can study different scheduling policies and how to select particular algorithm, which is best suited for our system.

4.2 SCHEDULING CONCEPTS

- The main goal of multiprogrammed system is to maximize the CPU utilization. For uniprocessor system there will never be more then one running process, all other processes are in state of waiting.
- In multiprogramming environment several processes are kept in memory at one time.
- When one process is in the state of wait, Operating system switches the CPU from this process to another one. This pattern continues among all.
- Scheduling is the one of the most important function of operating system so almost all computers resources are scheduled before use.
- So CPU is one of the primary computer resources for this reason its scheduling is central to operating system design.

4.2.1 CPU-I/O Burst Cycle [S-18]

- CPU-I/O burst cycle contains:
  - **CPU Burst:** A period of uninterrupted CPU activity.
  - **I/O Burst:** A period of uninterrupted input/output activity.

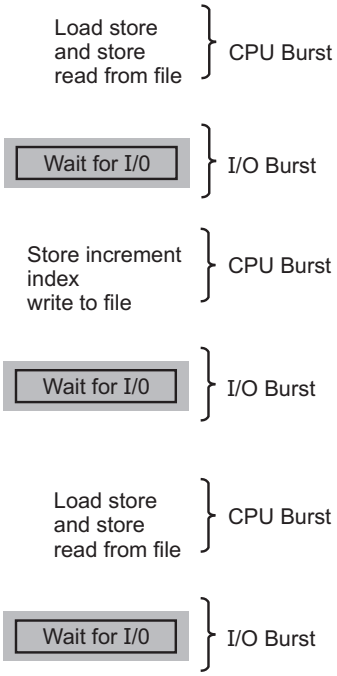


Fig. 4.1: Alternating Sequence of CPU and I/O Bursts

- Process execution consists of a cycle of CPU execution and I/O wait. Processes alternate back and forth between these two states. Process starts with CPU burst followed by I/O burst and so on.
- An I/O bound program would typically have many short CPU bursts where as CPU bound program consist of few long CPU bursts.

4.2.2 CPU Scheduler

- The operating system must select one of the processes in the ready queue to be executed. This is carried out by short term scheduler.
- A ready queue may be FIFO queue, Priority queue, Tree, Linked list.
- Operating system selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.

4.2.3 Preemptive Scheduling [W-18]

- In this section, we will look at several methods of preemptive scheduling of the processor.
- We will assume a system where processes enter the processor scheduling system and remain there sometimes executing and sometimes waiting to execute until they have finished execution.
- By "finished execution", we do not mean that the process terminates rather we mean that the process becomes blocked waiting for an event waiting for a message or an I/O operation to complete.
- At that point, the process can no longer use the processor and so it is considered to be finished as shown in Fig. 4.2.

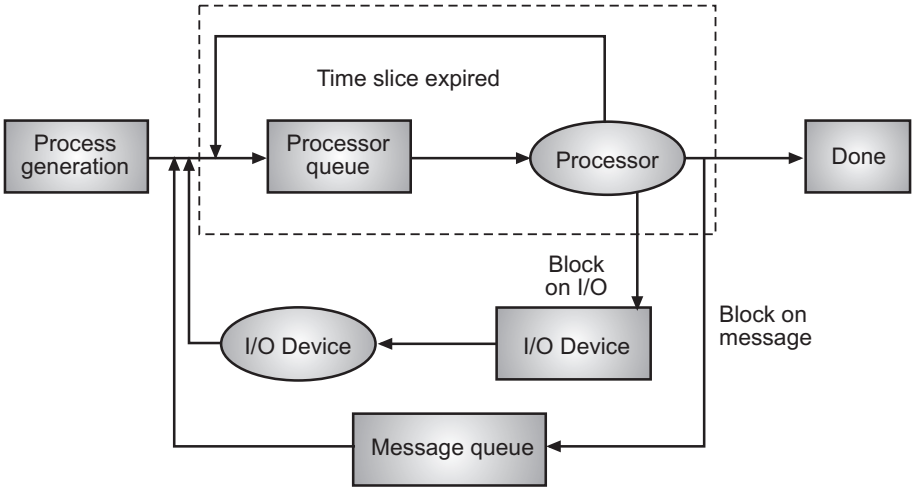


Fig. 4.2: The Flow of Processes in an Operating System

- The currently running process may be interrupted and moved to the ready state by the operating system. This is known as Preempting.
- The decision to preempt may be performed:
  - When a new process arrives.
  - When an interrupt occurs that places a blocked process in the ready state or
  - Periodically based on a clock interrupt.

- A process will enter and leave the processor scheduling system many times during its entire execution, may be hundreds of thousands of times. Still, each one is a separate transaction as far as the processor scheduling system is concerned.
- The processor scheduling algorithm might look at the history of the process, that is, what happened on previous trips through the processor scheduling system. For example, the scheduler might want to estimate how much processor time the process will consume on this trip though the scheduling system.
- One rough estimate would be the same amount of time it used last trip. A better estimate might be weighted average of the last 10 trips.

4.2.4 Non-preemptive Scheduling [W-18]

- In non-preemptive scheduling, once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or by switching to the working state.
- This method uses some hardware platforms. Microsoft Windows and Apple Macintosh Operating System use this type of scheduling method. Non-preemptive scheduling is attractive due to its simplicity.
- In non-preemptive scheduling, once a process is in the running state, it continues to execute until it terminates or blocks itself to wait for I/O or by requesting some operating system services.
- In short, we can define the preemptive and non-preemptive scheduling as follows:
  1. **Preemptive scheduling:** CPU allocated to a process may be switched if another process is scheduled which is of higher priority.
  2. **Non-preemptive scheduling:** Once the CPU has been allocated to a process, it keeps the CPU until process terminates or by switching to the wait state.

4.2.5 Dispatcher [S-19]

- A Dispatcher is a module; it connects the CPU to the process selected by the short-term scheduler.
- The main function of the dispatcher is switching, it means switching the CPU from one process to another process.
- The function of the dispatcher is 'jumping to the proper location in the user program and ready to start execution'.
- The dispatcher should be fast, because it is invoked during each and every process switch.
- **Dispatch Latency:** The time it takes by the dispatcher to stop one process and start another running is known as the 'dispatch latency'.
- The degree of multiprogramming is depending on the dispatch latency. If the dispatch latency is increasing then the degree of multiprogramming decreases.
- **Worst Case Latency:** This term is used for the maximum time taken for execution of all the tasks.

4.3 SCHEDULING CRITERIA [W-18, 19]

- There are many different CPU - Scheduling algorithms are available and different CPU scheduling algorithms have different properties and may favour one class of processes over another.
- To choose a particular algorithm for a particular situation, we must consider the properties of the various algorithms.
- For comparing CPU - scheduling algorithms many criteria have been suggested. The characteristics used for comparison can make a substantial difference in the determination of the best algorithm.
- The following scheduling criteria are listed below:
  1. **CPU utilization:** Our main aim is to keep the CPU as busy as possible. The utilization of CPU may range from 0 to 100%. In real time, the CPU range is from 40% (for lightly loaded systems) to 90% (for heavily loaded systems).
  2. **Throughout:** If the CPU is busy executing processes, then work is being done. One measure of work is the number of processes completed per time unit, called *throughput*. For long processes, this rate may be one process per hour, for short transactions, throughput might be 10 processes per second.
  3. **Turnaround time:** For a process, the important criterion is how long it takes to execute that process. The interval from the time of submission of a process to the time of completion is the *turnaround time.* It is the sum of the periods spends waiting to get into memory, waiting in the ready queue, executing on the CPU and doing I/O.
  4. **Waiting time:** The CPU scheduling algorithm does not affect the amount of time during which a process executes or does I/O. The CPU - scheduling algorithm affects only the amount of time during which a process spends waiting in the ready queue. Waiting time is the addition of the periods spends waiting in the ready queue.
  5. **Response time:** Response time is the time from the submission of a request until the first response is produced or we can say that it is the amount of time it takes to start responding, but not the time that it takes the output that response. To guarantee that all users get good service, we may want to minimize the maximum response time.
  6. **Load average:** It is the average number of processes residing in the ready queue waiting for their turn to get into the CPU.
- In general CPU utilization and throughput are maximized and other factors are reduced for proper optimization.

4.4 SCHEDULING ALGORITHMS [S-18, 19]

- CPU scheduling algorithms deals with the problem of deciding which of the processes in the ready queue is to allocate the resource that is the CPU.
- There are many different algorithms, here we will discuss the below mentioned algorithms only.

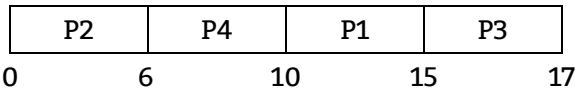
4.4.1 First-Come, First-Serve Scheduling (FCFS) [S-18]

- The simplest among all is the FCFS scheduling algorithm.
- The process that requests the CPU first, is allocated CPU first.
- It can be implemented with FIFO queue.
- When a process enters in a ready queue, get allocated with CPU.
- When CPU is free it is allocated at the head of the queue.
- The running process is then removed from the queue. The average waiting time under FCFS policy, however, is often quite long.

Example 1: Calculate Average Turnaround Time and Average Waiting Time for all set of processes using FCFS:

Process	Burst Time	Arrival Time	Waiting Time	Turnaround Time	Turnaround Time
P1	5	1	9	15	14
P2	6	0	0	6	6
P3	2	2	13	17	15
P4	4	0	6	10	10

The Gantt chart for the schedule is:



Solution:

(i) Waiting time:

Waiting Time = Start Time – Arrival Time

P1 Wait Time = 9

P2 Wait Time = 0

P3 Wait Time = 13

P4 Wait Time = 6

$\therefore$ Average waiting time = $\frac{9 + 0 + 13 + 6}{4} = \frac{28}{4} = 7$ milliseconds

(ii) Turnaround Time:

Turnaround Time = Finished Time – Arrival Time

P1 Turnaround Time = 15 – 1 = 14 milliseconds

P2 Turnaround Time = 6 – 0 = 6 milliseconds

P3 Turnaround Time = 17 – 2 = 15 milliseconds

P4 Turnaround Time = 10 – 0 = 10 milliseconds

$\therefore$ Average waiting time = $\frac{14 + 6 + 15 + 10}{4} = \frac{45}{4} = 11.25$ milliseconds

- Here, we have the waiting period for process P1 is 9 and for process P2 waiting period is 0 and for process P3 the waiting period is 13 and for process P4 the waiting period is 6.

- Then the average waiting period/time is 4, therefore, we can say that the waiting Time is much better than the previous case.
- Now consider the performance of FCFS scheduling in dynamic situation. Assume that we have one CPU bound process and many I/O bound processes.
- As processes flow around the system, we may have the following scenario, the CPU bound process will get the CPU and it will hold it.
- During that time all other processes will finish their I/O and now they are ready to move into the ready queue, and waiting for the CPU. As the processes are waiting in the ready queue mean while the I/O devices are sitting idle.
- The CPU bound process finishes its CPU burst and moves to an I/O device for I/O related work. Now all the I/O bound processes, which have very short CPU burst gets execute quickly and moves back to the I/O queues.
- At this point the CPU remains idle. The CPU-bound process will then move back to the ready queue and be allocated the CPU. Again, all the I/O processes end up waiting in the ready queue until the CPU bound process is done.

Example 2: Consider the following set of processes which all arriving at time 0 and the burst time of each process is given below:

Process	Burst Time (milliseconds)
P1	5
P2	24
P3	16
P4	10
P5	3

The processes arrive in the order given below as P1, P2, P3, P4 and P5.

The Gantt chart for the above scenario would be as shown below:

P1	P2	P3	P4	P5

0 5 29 45 55 58

Calculate:

(i) Average Waiting Time

(ii) Average Response Time

(iii) Average Turnaround time.

Solution:

(i) Average Waiting Time:

Waiting Time = Starting Time – Arrival Time

Process	Starting Time	Arrival Time	Waiting Time
P1	0	0	0
P2	5	0	5
P3	29	0	29
P4	45	0	45
P5	55	0	55

$$\therefore \text{ Average waiting time } = \frac{0 + 5 + 29 + 45 + 55}{5}$$

$$= 26.8 \text{ milliseconds}$$

(ii) Average Response Time:

Response Time = First Response – Arrival Time

Process	Starting Time	Arrival Time	Waiting Time
P1	0	0	0
P2	5	0	5
P3	29	0	29
P4	45	0	45
P5	55	0	55

$$\therefore \quad \text{Average Response Time} = \frac{0 + 5 + 29 + 45 + 55}{5}$$

$$= 26.8 \text{ milliseconds}$$

(iii) Average Turnaround Time:

Turnaround Time = Finished time – Arrival Time

Process	Starting Time	Arrival Time	Waiting Time
P1	5	0	5
P2	29	0	29
P3	45	0	45
P4	55	0	55
P5	58	0	58

$$\therefore \quad \text{Average Turnaround Time} = \frac{5 + 29 + 45 + 55 + 58}{5}$$

$$= 38.4 \text{ milliseconds}$$

Since, it is non-preemptive scheduling algorithms the average response time and average waiting time is same in both the cases.

Note: Non-preemptive means once the CPU is allocated to one process it keeps it as long as it needs and release only when it finishes its jobs either by terminating or by requesting an I/O.

Example 3: In Example 2, the arrival time for all the processes was 0. But if we change the arrival time of the processes then the whole calculation changes as shown below.

Process	CPU burst Time	Arrival Time
P1	3	0
P2	6	2
P3	4	4
P4	5	6
P5	2	8

Order of Arrival of processes is P1, P2, P3, P4 and P5.

P1 arrived at Time 0 milliseconds

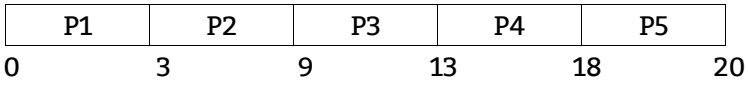
P2 arrived after 2 milliseconds

P3 arrived after 4 milliseconds

P4 arrived after 6 milliseconds

P5 arrived after 8 milliseconds.

The Gantt chart of the above problem is shown below:



Calculate:

(i) Average Relative Delay

(ii) Average Response Time

(iii) Average Turnaround Time.

Solution:

(i) Average Relative Delay:

$$\text{Relative delay} = \frac{\text{Turn around time}}{\text{Burst Time}}$$

Process	Turnaround Time	Burst Time	Relative delay
P1	3	3	1
P2	7	6	1.16
P3	9	4	2.25
P4	12	5	2.4
P5	12	2	6

$$\text{Average relative delay} = \frac{1 + 1.16 + 2.25 + 2.4 + 6}{5}$$

$$= \frac{12.81}{5} = 2.56 \text{ milliseconds}$$

(ii) Average Turnaround Time:

$$\text{Turnaround Time} = \text{Finish Time} - \text{Arrival Time}$$

Process	Finish Time	Arrival Time	Turnaround Time
P1	3	0	3
P2	9	2	7
P3	13	4	9
P4	18	6	12
P5	20	8	12

$$\therefore \quad \text{Average Turnaround Time} = \frac{3 + 7 + 9 + 12 + 12}{5}$$

$$= 8.6 \text{ milliseconds}$$

(iii) Average Response Time:

$$\text{Response Time} = \text{First Response Time} - \text{Arrival Time}$$

Process	Finish Time	Arrival Time	Response Time
P1	0	0	0
P2	3	2	1
P3	9	4	5
P4	13	6	7
P5	18	8	10

$$\therefore \quad \text{Average Response time} = \frac{0 + 1 + 5 + 7 + 10}{5}$$

$$= 4.6 \text{ milliseconds}$$

Advantages:

1. Easy to implement.
2. It is very simple.

Disadvantages:

1. Difficult with some time sharing systems.
2. Average waiting time is very high with respect to others.
3. Because of this performance is affected or degraded.

Note: To summarize FCFS algorithm

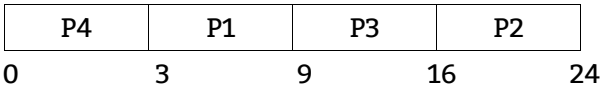
- FCFS scheduling algorithm is non-preemptive.
- For time-sharing system we cannot implement FCFS, because process will hold the CPU until it finishes or changes a state to wait state.
- Average waiting time for FCFS algorithm is not minimal, and it also varies substantially if the process CPU burst time vary greatly.

4.4.2 Shortest-Job-First Scheduling (SJF)

- The Shortest-Job-First (SJF) is the method of CPU scheduling.
- It allocates the CPU to a process having smallest next CPU burst. When the CPU is available, it is assigned to the process that has the smallest next CPU burst.
- If the two processes having same CPU burst then they will be scheduled according to FCFS algorithm. For example consider the following set of processes.

Process	Burst-time (milliseconds)
P1	6
P2	8
P3	7
P4	3

- The Gantt chart for the above problem is,



Waiting Time for Process P1 = 3 milliseconds

Waiting Time for Process P2 = 16 milliseconds

Waiting Time for Process P3 = 9 milliseconds

Waiting Time for Process P4 = 0 milliseconds

$\therefore$ Average Waiting Time = $\frac{(3 + 16 + 9 + 4)}{4}$ = 8 milliseconds

- The SJF scheduling algorithm is probably optimal; it gives minimal average waiting time for a given set of processes.
- By moving short process before a long one, the waiting time of short process decreases. Consequently average waiting time reduces.
- The real difficulty with SJF knows the length of next CPU request so it can not be implemented at the level of short term scheduling. Instead it can be used for long term scheduling where user estimates the process time limit.

- At short-term scheduling, there is no way to find out the length of next CPU burst. One approach is to approximate SJF scheduling. We can predict the next CPU burst from previous value.
- The next CPU burst is generally predicted as an exponential average of the measured lengths of previous CPU burst.
- Let t_n be the length of the n^{th} CPU burst, and let T_{n+1} be our predicted value for the next CPU burst. Then for $\propto$, $0 <= \propto <= 1$,

$$\boxed{T_{n+1} = \propto t_n + (1 - \propto) T_n}$$

- This gives exponential average. Here,
  - t_n : Most recent information
  - T_n : Stores the past history
  - $\propto$: Relative weight of recent and past history

  If $\propto = 0$, Then $T_{n+1} = T_n$ (Current T_n and recent having same values)

  If $\propto = 1$, Then $T_{n+1} = t_n$ (Most recent CPU burst matters)

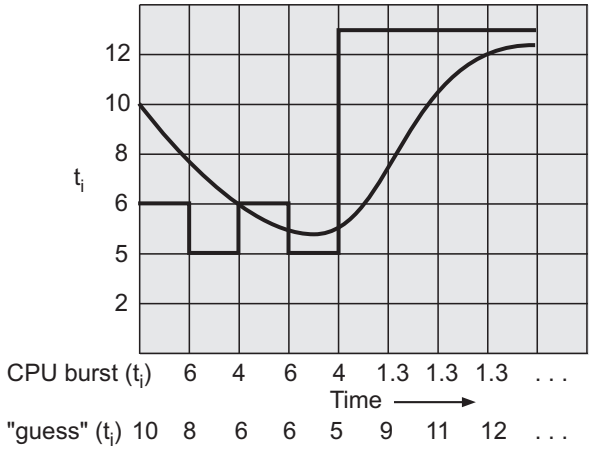


Fig. 4.3: Prediction of the length of the next CPU Burst

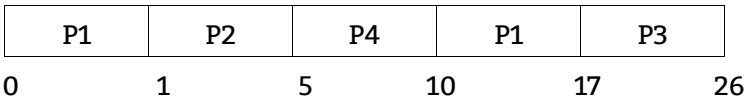
Types of SJF Algorithm:

- Two types of SJF algorithm are:
  1. Preemptive
  2. Non-preemptive
- The choice arises when a new process arrives at the ready queue while a previous process is executing.
- The new process may have a shorter next CPU burst than what is left at the currently executing process.
- A **Preemptive SJF algorithm** will preempt the currently executing process, whereas as a **Non-preemptive SJF** algorithm will allow the currently running process to finish its CPU burst.
- Preemptive SJF scheduling is sometimes called shortest-remaining time first scheduling.

- For example consider the processes below:

Process	Arrival Time	Burst-time (milliseconds)
P1	0	8
P2	1	4
P3	2	9
P4	3	5

- The Gantt chart for the above problem is shown below:



- Process P1 arrives at Time 0.
- Process P2 arrives at Time 1. The remaining time for process P1 is 7 milliseconds is larger than the time required by process P2 i.e. 4 milliseconds, so process P1 is preempted and process P2 is scheduled.
- The average waiting time for this is:

  Formula for waiting time in Preemptive SJF Scheduling:

  Waiting time = Start time – Arrival time + New start time – Old finish time.

  $\therefore$ Waiting time $= \frac{((10-1)+(1-1)+(17-2)+(5-3))}{4} = \frac{(9+0+15+2)}{4}$

  $\therefore \quad = \frac{26}{4}$

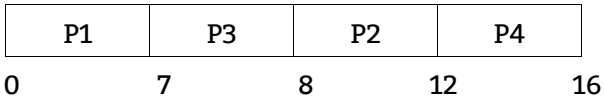
  $\therefore \quad = 6.5$ milliseconds
- A **Non-preemptive SJF scheduling** would have average waiting time at 7.75 milliseconds. The calculation is left for the students to solve.

Example 4: Non-Preemptive SJF.

Process	Arrival Time	Burst Time
P1	0.0	7
P2	2.0	4
P3	4.0	1
P4	5.0	4

Solution:

SJF (Non-Preemptive)



$\therefore$ Average waiting time $= \frac{0+6+3+7}{4} = \frac{16}{4} = 4$

Example 5: Preemptive SJF.

Process	Arrival Time	Burst Time
P1	0.0	7
P2	2.0	4
P3	4.0	1
P4	5.0	4

Solution:

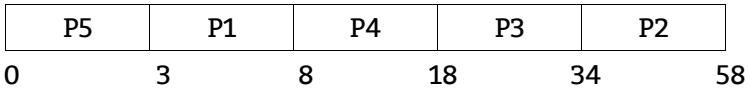
SJF (Preemptive)

$\therefore$ Average waiting time $= \frac{9+1+0+2}{4} = \frac{12}{4} = 3$

Example 6: Consider the process and CPU burst time in milliseconds.

Process	CPU Burst-time (milliseconds)
P1	5
P2	24
P3	16
P4	10
P5	3

The Gantt chart for the above problem is shown below:



By observing the above example, we see that P5 has the shortest CPU burst time. So we allocate P5 the CPU first. After finishing P5 the next shortest job is searched which is P1 and CPU is given to P1 the same process continuous till all the processes are finished. Now we calculate the average waiting time, Average turnaround time and Average response time.

(i) Average Waiting Time:

Waiting time = Starting time – Arrival time

Process	Starting Time	Arrival Time	Waiting Time
P1	3	0	3
P2	34	0	34
P3	18	0	18
P4	8	0	8
P5	0	0	0

$\therefore$ Average waiting time $= 3 + 34 + 18 + 8 + 0$

$= \frac{63}{5} = 12.6$ milliseconds

(ii) Average Turnaround Time:

Turnaround time = Finish time – Arrival time

Process	Finish Time	Arrival Time	TurnAround Time
P1	8	0	8
P2	58	0	58
P3	34	0	34
P4	18	0	18
P5	3	0	3

$$\therefore \text{ Average turnaround time } = \frac{8 + 58 + 34 + 18 + 3}{5}$$

$$= \frac{121}{5} = 24.2 \text{ milliseconds}$$

(iii) Average Response Time:

Response time = First Response Time – Arrival time

Process	First Response Time	Arrival Time	Response Time
P1	3	0	3
P2	34	0	34
P3	18	0	18
P4	8	0	8
P5	0	0	0

$$\therefore \text{ Average Response time } = \frac{3 + 34 + 18 + 8 + 0}{5}$$

$$= \frac{63}{5} = 12.6 \text{ milliseconds}$$

Advantage:

1. It has the least Average waiting time, Average turnaround time and Average Response time.

Disadvantages:

1. It is difficult to know the length of the next CPU burst time.
2. It is optimal algorithm it cannot be implemented in short-term CPU scheduling.
3. Aging is another problem where big jobs are waiting for long-time in the CPU.
4. Starvation of process having long burst time may cause because processor is selecting the process having smallest burst time.

4.4.3 Priority Scheduling

- SJF algorithm is a special case of Priority scheduling algorithm.
- Priority is assigned to each process and CPU is allocated to the process with highest priority. Equal priority process is scheduled with FCFS algorithm.

- Priorities are the numbers ranging from 0 to 7 or 0 to 4095 have given to each processes. Some systems use low numbers to represent low priority and other use low numbers for high priority. Here we use low numbers to represent high priority.
- Protocol is set whether 0 is the highest or lowest priority.
  1. **Starvation:** A major problem with priority scheduling is indefinite blocking or starvation. The high priority process indefinitely blocks a low priority process in a heavily loaded system. As per algorithm high priority processes can prevent the low priority process for allocating CPU and thus such a process will never get a chance to allocate a CPU.
  2. **Aging:** A solution to this problem is to gradually increase the priority of a process that wait in the system for a long time called Aging.

Note: To summarize Priority Scheduling

- A priority number (integer) is associated with each process.
- The CPU is allocated to the process with the highest priority (smallest integer = highest priority).
- Preemptive
- Non-preemptive
- SJF is a priority scheduling where priority is the predicted next CPU burst time.
- Problem = Starvation-low priority processes may never execute.
- Solution = Aging-as time progresses increase or decrease the priority of the process.

- Priorities are of two types:
  1. **Internal priorities:** Internally defined priorities use some measurable quantity to compute the priority of a process.
  2. **External priorities:** External priorities are set by criteria that are external to the operating system, such as the importance of the process.

Modes of Priority Scheduling:

- Priority scheduling are of two modes:
  1. Preemptive
  2. Non-preemptive.
- When a process arrives at the ready queue, its priority is compared with the priority of the currently running process.
- A **Preemptive priority**: This scheduling algorithm will preempt the CPU if the priority of the newly arrived process is higher than the priority of the currently running process.
- A **Non-preemptive priority**: This scheduling algorithm will simply put the new process at the head of the ready queue.

Advantages:

1. It considers the priority of the processes and allows the important processes to run first.
2. Priority scheduling in preemptive mode is best suited for real time operating system.
3. Priorities in the Priority scheduling are chosen on the basis of time requirements, memory requirements, and user preferences.

Disadvantages:

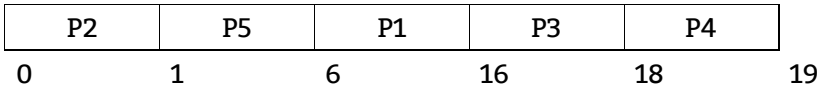
1. Processes with lesser priority may starve for CPU.
2. In priority scheduling, if the system is crashed, then all low-priority processes that are not yet completed will also get lost.
3. There is no idea of response time and waiting time.

Example 7: Consider the following set of processes. All processes arrived time 0 and in the order P1, P2, P3, P4, P5 having CPU burst time in milliseconds. Calculate average waiting time using Non-Preemptive Priority Scheduling Technique.

Solution:

Process	Burst Time (in milliseconds)	Priority
P1	10	3
P2	1	1
P3	2	3
P4	1	4
P5	5	2

The Gantt chart is:



Waiting time of process P1 = 6 milliseconds
Waiting time of process P2 = 0 milliseconds
Waiting time of process P3 = 16 milliseconds
Waiting time of process P4 = 18 milliseconds
Waiting time of process P5 = 1 milliseconds

Average waiting time = $\frac{6 + 0 + 16 + 18 + 1}{5}$ = 8.2 milliseconds

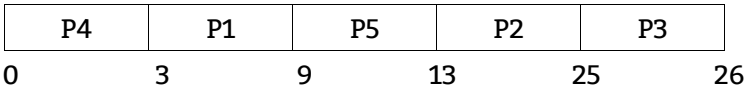
---

Example 8: Consider the following set of processes:

Process	CPU Burst Time (in milliseconds)	Priority
P1	6	2
P2	12	4
P3	1	5
P4	3	1
P5	4	3

By observing the above example we can say that process P4 has highest priority. Therefore, we allocate CPU to process P4 first. The next priority is given to process P1 and CPU is allocated to process P1, next priority is given to process P5, so the CPU is allocated to P5 and this process continuous until all the process are completed.

The Gantt chart for the above example is shown below:



(i) The Average Waiting Time:

Process	Waiting time
P1	3
P2	13
P3	25
P4	0
P5	9

$$\text{Average waiting time} = \frac{3 + 13 + 25 + 0 + 9}{5}$$

$$= \frac{50}{5}$$

$$= 10 \text{ milliseconds}$$

(ii) The Average Turnaround Time:

Process	Turnaround time
P1	9
P2	25
P3	26
P4	3
P5	13

$$\therefore \quad \text{Average turnaround time} = \frac{9 + 25 + 26 + 3 + 13}{5}$$

$$= \frac{76}{5}$$

$$= 15.2 \text{ milliseconds}$$

4.4.4 Round Robin Scheduling [W-18, S-19]

- Round Robin (RR) Scheduling is designed for time-sharing system.
- This Scheduling is also called as FCFS scheduling along with preemption to switch between processes.
- In this scheduling algorithm, we will define a time slice or time quantum generally from 10 to 100 ms. Ready queue is treated as circular queue and processes are entered in ready queue as they are coming in the system.
- CPU scheduler goes around this circular queue and CPU is allocated to each process for a fixed time slice.
- To implement RR algorithm, we keep the ready queue as a FIFO.
- New processes are added to the tail of the ready queue.

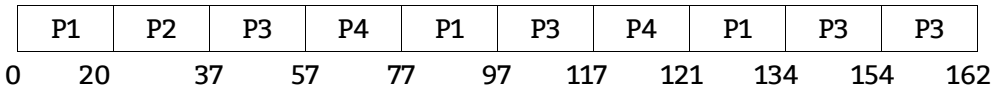
- The CPU scheduler picks from the ready queue, sets a timer to interrupt after 1 time quantum which will interrupt the operating system.
- A context switch will be executed; process will be put at a tail of ready queue.

Example 9: Example of RR with Time Quantum = 20

Process	Burst Time
P1	53
P2	17
P3	68
P4	24

Solution:

The Gantt chart is:



Average waiting time is (0 + 20 + 37 + 57 + 57 + 40 + 40) ms.

- Typically, higher average turnaround than SJF, but better response.
- In RR scheduling algorithm, no process is allocated to the CPU for more than one time quantum in a row. If a process CPU burst exceeds 1 time quantum, that process is preempted and is put back in ready queue. RR scheduling algorithm is preemptive.
- If there are n processes in ready queue and the time quantum is q, then each process gets 1/n of the CPU time in chunks of at most q time units.
- Each process must wait no longer than (n-1)*q time units until its next time quantum.
- For example, if there are 5 processes, with a time quantum of 20 ms, then each process will get up to 20 ms every 100 ms.
- The performance of RR algorithm depends heavily on the size of time quantum.
- At one extreme, if the time quantum is very large (Infinite), RR policy is the same as the FCFS policy.
- If the time quantum is very small (say 1 ms), RR approach is called processor sharing, and appears (in theory) to the users as though each n processes has its own processor running 1/n in the speed of the real processor. This approach was used in the Control Data Corporation (CDC).
- In software, however, we need also to consider the effect of context switching on the performance of RR scheduling.
- Let's assume that we have only one process of 10 time units. If the quantum 12 time units, the process finishes in less than 1 time quantum with no overhead.
- If the quantum is 6 time units, however the process required 2 quantum, resulting in a context switch. If the time quantum is 1 time unit, 9 context switches will occurred, slowing the execution of the process accordingly, (Fig. 4.4)

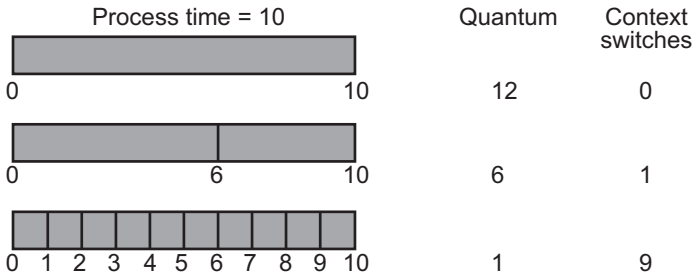
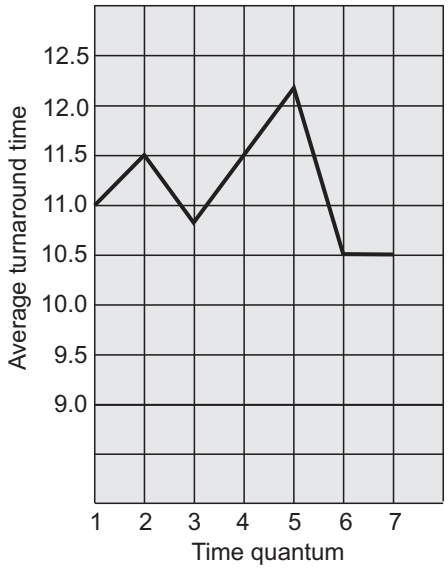


Fig. 4.4: Time Quantum and Context Switch Time

- Thus, we want the time quantum to be large with respect to the context switch time. If the context switch time is approximately 10 % of the time quantum, then about 10% of the CPU time will be spent in the context switch.
- Turnaround time also depends on the size of time quantum. As we can see from Fig. 4.5, the average turnaround time of a set of processes does not necessarily improve as the time quantum size increases.
- In general, the average turnaround time can be improved if most processes finish their next CPU burst in a single time quantum.
- For example, given three processes of 10 time units each and a quantum of 1 time unit, the average turnaround time is 29. If the time quantum is 10, however, the average turnaround time drops to 20.
- If context-switch time is added in, the average turnaround time increases for a smaller time quantum, since more context switches will be required.



Process	Time
P_1	6
P_2	3
P_3	1
P_4	7

Fig. 4.5: Turnaround time varies with the Time Quantum

- On the other hand, if the time quantum is too large, RR scheduling degenerates to FCFS policy. A rule of thumb is that 80% of CPU bursts should be shorter than the time quantum.

Advantages:

1. If you know the total number of processes on the run queue, then you can also assume the worst-case response time for the same process.
2. It deals with all process without any priority.
3. Allows OS to use the Context switching method to save states of preempted processes.
4. It gives the best performance in terms of average response time.

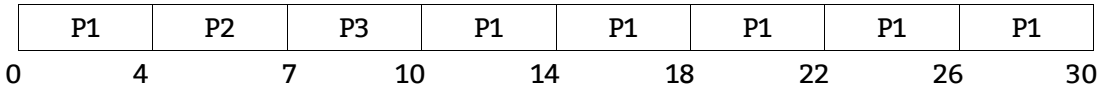
Disadvantages:

1. Its performance heavily depends on time quantum.
2. Deciding a perfect time quantum is really a very difficult task in the system.
3. Priorities can not be set for the processes.

Example 10: Consider the following set of process that arrives at time 0, with the length of the CPU burst time given in milliseconds.

Process	Burst Time
P1	24
P2	3
P3	3

Solution: If we use a time quantum of 4 milliseconds, then process P1 gets the first 4 milliseconds. Since, it requires another 20 milliseconds, it is preempted after the first time quantum and the CPU is given to the next process in the queue process P2. Since, process P2 does not need 4 milliseconds, it quits before its time quantum expires. The CPU is then given to the next process in the queue, process P3. Once, each process has received 1 time quantum, the CPU is returned to process P1 for an additional time quantum. The resulting RR schedule is:



Round robin scheduling is always Preemptive Scheduling.

Waiting time = Start time – arrival time + new start time – old finish time.

Waiting time for process P1 = (0 – 0) + (10 – 4) = 6 milliseconds

Waiting time for process P2 = (4 – 0) = 4 milliseconds

Waiting time for process P3 = (7 – 0) = 7 milliseconds

$\therefore$ Average Waiting Time is $\frac{17}{3}$ = 5.66 milliseconds.

In RR scheduling algorithm, no process is allocated to the CPU for more than 1 time quantum in a row. If a process CPU burst time exceeds 1 time quantum, that process is preempted and is input back in the ready queue. The RR scheduling algorithm is preemptive.

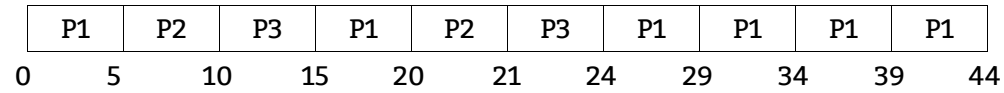
Example 11: Consider the following set of processes:

Process	CPU Burst Time (in milliseconds)
P1	30
P2	6
P3	8

Given the quantum is of 5 milliseconds.

Solution: Initially process P1 is given 5 milliseconds. When time quantum of P1 expired, the CPU switches from process P1 to P2. When the time quantum of P2 expired, the process switches to process P3. When time quantum of P3 expired, the CPU switch to P1 as P1 was in the ready queue.

The below Gantt chart shows this:



(i) Average Waiting Time:

Waiting time = Start time – Arrival time + New start time – Old finish time

Waiting time For P1 = 0 + (15 – 5) + (24 – 20) = 10 + 4=14 milliseconds

P2 = 5 + (20 – 10) = 15 milliseconds

P3 = 10 + (21 – 15) = 16 milliseconds

Average waiting time = 45/3 = 15 milliseconds

(ii) Average Turnaround Time:

Turnaround time for P1 = 44

Turnaround time for P2 = 21

Turnaround time for P3 = 24

$$\therefore \text{Average Turnaround Time} = \frac{44 + 21 + 24}{3}$$

$$= \frac{89}{3}$$

= 29.66 milliseconds

(iii) Average Response Time:

Response time for P1 = 0

Response time for P2 = 5

Response time for P3 = 10

$$\therefore \text{Average Response Time} = \frac{0 + 5 + 10}{3}$$

$$= \frac{15}{3}$$

= 5 milliseconds

The performance of round robin depends on the size of time quantum chosen.

4.4.4.1 Multilevel Queues

- These scheduling algorithms are created for areas in which we classify process into different groups.
- For example, a common division can be made between foreground (or interactive) processes and background (or batch) processes.
- Priority of foreground processes may be higher than background processes.
- The multilevel queue - scheduling algorithm partitions the ready queue into several separate queues as shown in Fig. 4.6.

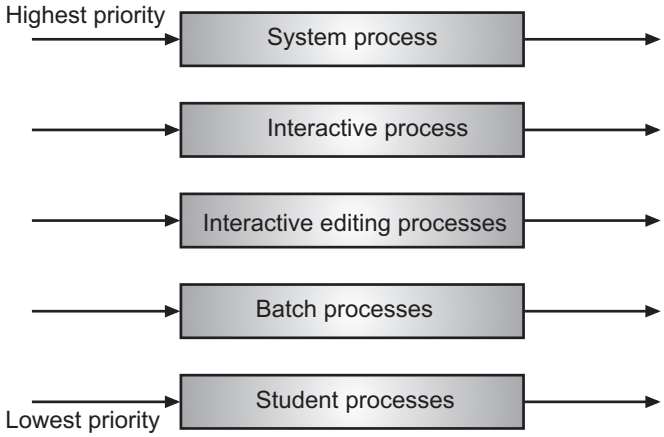


Fig. 4.6: Multilevel Queue Scheduling

- The processes are assigned to one queue depending on some property of the process.
- The property may be memory size, process priority or may be process type. Each queue is associated with its own scheduling algorithm.
- For example, separate queues might be used for foreground and background processes. The foreground queue might be scheduled by an RR algorithm and the background queue is scheduled by an FCFS algorithm.

4.4.4.2 Multilevel Feedback Queues [S-18]

- Multilevel feedback queue allows a process to move between queues. The idea is to separate process with different CPU - Burst characteristics.
- If a process uses too much CPU times, it will be moved to a lower priority queue. This leaves I/O bound and interactive processes in the higher priority queues.
- If a process waits for long time in a lower priority queue, then it is moved to a higher priority queue. This form of aging prevents starvation.
- For example, consider a multilevel feedback queue scheduler with three queues and number them from 0 to 2 as shown in Fig. 4.7.

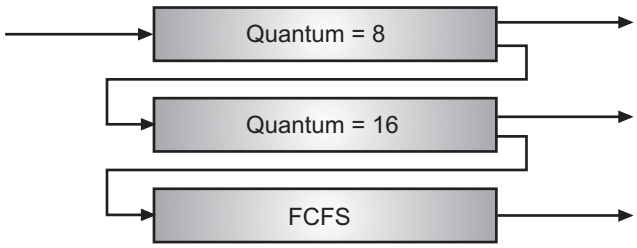


Fig. 4.7: Multilevel Feedback Queues

- The first task the scheduler does is that it executes all processes in queue 0. Only when queue 0 is empty then only it will go to queue 1 and then executes the process there in queue 1.
- Similarly, when queue 0 and queue 1 are empty then only it will process the processes in queue 2. But when a process is getting executed in queue 2 and at that time if a process arrives for queue 1 then it will preempt a process in queue 2. Similarly, when a process arrives for queue 0 will in turn preempt a process in queue 1.
- A process entering the ready queue is put in queue 0.
- A process is queue 0 is given a time quantum of 8 milliseconds.
- If it does not finish within this time, it is moved to the tail of queue 1.
- If queue 0 is empty, the process at the head of queue 1 is given a quantum of 16 milliseconds. If it does not complete, it is preempted and is put into queue 2.
- Processes in queue 2 are run on an FCFS basis only when queues 0 and 1 are empty.

Parameters:

- The below are the parameters for a multilevel feedback queue scheduler:
  - The number of queues.
  - The scheduling algorithm for each queue.
  - The method used to determine when to upgrade a process to a higher-priority queue.
  - The method used to determine when to demote a process to a lower priority queue.
  - The method used to determine which queue a process will enter when that process needs service.

SOLVED EXAMPLES OF PREVIOUS EXAMS

Example 1: Consider the following set of processes with the length of CPU burst time and arrival time.

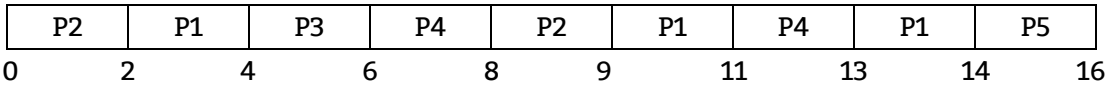
Process	Burst time	Arrival time
P1	5	1
P2	3	0
P3	2	2
P4	4	3
P5	2	13

Calculate turnaround time, waiting time, average waiting time and average turnaround time using Round Robin Algorithm with time quantum = 2.

Solution:

Process	Burst Time (ms)	Arrival Time (ms)	Start Time (ms)	Finish Time (ms)	Waiting Time (ms)	Turnaround Time (ms)
P1	5	1	2	14	8	13
P2	3	0	0	9	6	9
P3	2	2	4	6	2	4
P4	4	3	6	13	6	10
P5	2	13	14	16	1	3

The Gantt chart for the above scenario would be as shown below:



(i) Waiting Time: The Waiting time formula for Round Robin Scheduling.

Waiting Time = Starting Time – Arrival Time +new start time – old finish time

So waiting time of P1 = (2 – 1) + (9 – 4) + (13 – 11)
= 1 + 5 + 2
= 8 ms

Waiting time of P2 = (0 – 0) + (8 – 2)
= 6 ms

Waiting time of P3 = (4 – 2)
= 2 ms

Waiting time of P4 = (6 – 3) + (11 – 8) = (3 + 3)
= 6Ms

Waiting time of P5 = (14 – 13) = 01 ms

Formula for Average Waiting time = $\frac{\text{Sum of all processes waiting time}}{\text{Total number of processes}}$

$\therefore$ Average Waiting Time = $\frac{8 + 6 + 2 + 6 + 1}{5}$

= 4.6 milliseconds

Turnaround Time:

Turnaround Time = Finished time – Arrival Time

Process	Turnaround Time
P1	13
P2	9
P3	4
P4	10
P5	3

(ii) Average Turnaround time = $\frac{\text{Sum of turnaround time of processes}}{\text{Total number of processes}}$

$\therefore$ Average Turnaround Time = $\frac{13 + 9 + 4 + 10 + 3}{5}$

= 7.8 milliseconds

Example 2: Consider the following set of processes with the length of CPU Burst time and Arrival time given in milliseconds:

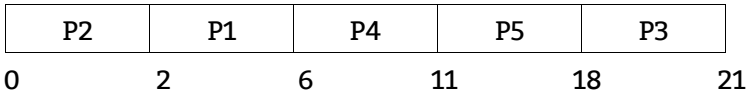
Process	Burst time	Arrival time
P1	4	1
P2	2	0
P3	3	3
P4	5	2
P5	7	2

Calculate turnaround time, waiting time, average waiting time and average turnaround time using FCFS algorithm.

Solution:

Process	Burst Time (ms)	Arrival Time (ms)	Start Time(ms)	Finish Time (ms)	Waiting Time (ms)	Turnaround Time (ms)
P1	4	1	2	6	1	5
P2	2	0	0	2	0	2
P3	3	3	18	21	15	18
P4	5	2	6	11	4	9
P5	7	2	11	18	9	16

The Gantt chart for the above scenario would be as shown below:



(i) Waiting Time:

The Waiting time formula for Non Preemptive scheduling using FCFS:

Waiting Time = Starting Time – Arrival Time

So, Waiting Time of P1 = 2 – 1 = 1 ms

Waiting Time of P2 = 0 – 0 = 0 ms

Waiting Time of P3 = 18 – 3 = 15 ms

Waiting Time of P4 = 6 – 2 = 4 ms

Waiting Time of P5 = 11 – 2 = 9 ms

(ii) Average Waiting time:

$$\text{Average Waiting time} = \frac{\text{Sum of all processes waiting time}}{\text{Total number of processes}}$$

$$\therefore \quad \text{Average Waiting Time} = \frac{1 + 0 + 15 + 4 + 9}{5}$$

$$= 5.8 \text{ milliseconds}$$

(iii) Turnaround Time:

Turnaround Time = Finished time – Arrival Time

Process	Turnaround Time
P1	5
P2	2
P3	18
P4	9
P5	16

(iv) **Average Turnaround time** = $\frac{\text{Average Turn Around time}}{\text{Total number of processes}}$

$\therefore$ Average Turnaround Time = $\frac{5 + 2 + 18 + 9 + 16}{5}$

= 10 milliseconds

Example 3: Consider the following set of processes with the length of CPU Burst time and arrival time in milliseconds:

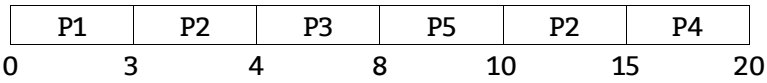
Process	Arrival	Time Burst Time
P1	0	3
P2	2	6
P3	4	4
P4	6	5
P5	8	2

Calculate turnaround time, waiting time, average waiting time and average turnaround time using preemptive SJF Scheduling algorithm.

Solution:

Process	Burst Time (ms)	Arrival Time (ms)	Start Time(ms)	Finish Time (ms)	Waiting Time (ms)	Turnaround Time (ms)
P1	0	3	0	3	0	3
P2	2	6	3	15	7	13
P3	4	4	4	8	0	4
P4	6	5	15	20	9	14
P5	8	2	8	10	0	2

The Gantt chart for the above scenario would be as shown below:



(i) Waiting Time:

The Waiting time formula for SJF Preemptive Scheduling Algorithm.

Waiting Time = Starting Time − Arrival Time + New start time − Old finish time

So waiting time of P1 = (0 − 0) = 0 ms

Waiting time of P2 = (3 − 2) + (10 − 4) = 7 ms

Waiting time of P3 = (4 − 4) = 0 ms

Waiting time of P4 = (15 − 6) = 9 ms

Waiting time of P5 = (8 − 8) = 0 ms

(ii) Average Waiting time:

$$\text{Average Waiting time} = \frac{\text{Sum of all processes waiting time}}{\text{Total number of processes}}$$

$$\therefore \quad \text{Average Waiting Time} = \frac{0+7+0+9+0}{5}$$

$$= 3.2 \text{ milliseconds}$$

(iii) Turnaround Time:

Turnaround Time = Finished time – Arrival Time

Process	Turnaround Time
P1	3
P2	13
P3	4
P4	14
P5	2

(iv) Average Turnaround time:

$$\text{Average Turnaround Time} = \frac{\text{Sum of turnaround time of processes}}{\text{Total number of processes}}$$

$$\therefore \quad \text{Average Turnaround Time} = \frac{3+13+4+14+2}{5}$$

$$= 7.2 \text{ milliseconds}$$

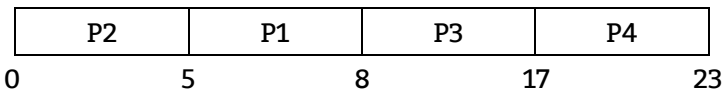
Example 4: Calculate average turnaround time and average waiting time for all set of processes using FCFS algorithm:

Process	Burst Time	Arrival Time
P1	3	1
P2	5	0
P3	9	4
P4	6	5

Solution:

Process	Burst Time (ms)	Arrival Time (ms)	Start Time(ms)	Finish Time (ms)	Waiting Time (ms)	Turnaround Time (ms)
P1	3	1	5	8	4	7
P2	5	0	0	5	0	5
P3	9	4	8	17	4	13
P4	6	5	17	23	12	18

The Gantt chart for the above scenario would be as shown below:



(i) Waiting Time:

The Waiting time formula for FCFS Scheduling:

Waiting Time = Starting Time – Arrival Time

So waiting time of P1 = (5 – 1) = 4 ms

Waiting time of P2 = (0 –0) = 0 ms

Waiting time of P3 = (8 – 4) = 4 ms

Waiting time of P4 = (17 – 5) = 12 ms

(ii) Average Waiting time:

$$\text{Average Waiting Time} = \frac{\text{Sum of all processes Waiting Time}}{\text{Total number of processes}}$$

$$\text{Average Waiting Time} = \frac{4 + 0 + 4 + 12}{4}$$

$$= 4.5 \text{ milliseconds}$$

(iii) Turnaround Time:

Turnaround Time = Finished time – Arrival Time

Process	Turnaround Time
P1	7
P2	5
P3	13
P4	18

(iv)

$$\text{Average Turnaround time} = \frac{\text{Sum of turnaround time of processes}}{\text{Total number of processes}}$$

$$\text{Average Turnaround Time} = \frac{7 + 5 + 13 + 18}{4}$$

$$= 10.75 \text{ milliseconds}$$

Summary

- CPU Scheduling algorithms are very important in scheduling the processor amongst different processes as the CPU has to allocate to multiple processes and idle time of CPU should be minimized, it is achieved with CPU Scheduling algorithms.
- A component involved In the CPU-scheduling function is the dispatcher, which is the module that gives control of the CPU to the process selected by the short-term scheduler. It receives control in kernel mode as the result of an interrupt or system call.
- Schedulers are special system software which handles process scheduling in various ways. Their main task is to select the jobs to be submitted into the system and to decide which process to run.

- Schedulers are of three types:Long Term Scheduler, Medium Term Scheduler, and Short Term Scheduler.
- In FCFS, the processor is allocated to the processes according to arrival time which process arrives first CPU will be allocated to it.
- In SJF Scheduling algorithm, the processor will be allocated to shortest burst time of processes that may lead to starvation and aging. SJF can be implemented with Non-preemptive and preemptive scheduling.
- Preemption stands for switching the processor amongst processes. No preemption stands for not switching the processor until the task is finished. In Priority Scheduling process has priority assigned to it the processor will be allocated to that process which has highest priority.
- Priority is the number which can ranges from 0 to 4999. In this case, there are chances that low process will have high waiting time.
- Round Robin(RR) scheduling technique uses time quantum or time slot where equal time slot is given to processes. It uses preemptive scheduling algorithm where equal amount of CPU time is given for all processes if the process doesn't complete in time quantum the processor will be allocated to another process. RR tries to give fare justice to all the processes.
- Multilevel Queue Scheduling algorithms are created for areas in which we classify process into different groups.
- Multilevel feedback queue allows a process to move between queues. The idea is to separate process with different CPU - Burst characteristics.

Check Your Understanding

1. Process in main memory which is ready and waiting to execute is kept in _____.
   (a) process queue (b) ready queue
   (c) schedule (d) dispatcher
2. The time interval from time of submission of a process to completion of time is called as _____.
   (a) completion time (b) turnaround time
   (c) waiting time (d) none of above
3. The unit of measurement of work of CPU is called as _____.
   (a) latency (d) delay
   (c) throughput (d) context switch
4. In priority scheduling the process is allocated with _____.
   (a) low priority (b) high priority
   (c) equal priority (d) extra priority

5. Which scheduling technique applies time quantum?
   (a) SJF (b) FCFS
   (c) Round Robin (d) Priority
6. In multilevel feedback queue algorithm _____.
   (a) a process can move to different classified ready queue
   (b) processes are not classified into different groups
   (c) classification of ready queue is not permanent
   (d) process is upgraded

Answers

1. (b)	2. (b)	3. (c)	4. (b)	5. (c)	6. (a)

Practice Questions

Q.I Answer the following questions in short:

1. What is meant by CPU scheduling?
2. What is CPU scheduler?
3. Describe the following scheduling schemes:
   (a) Preemptive scheduling,
   (b) Non-primitive scheduling.
4. What is priority?
5. Define dispatch latency.

Q.II Answer the following questions:

1. Explain scheduling criteria in detail.
2. With the help of example describe following scheduling algorithms:
   (i) FCFS (ii) RR (iii) SJF
3. With the help of diagram explain multilevel queue scheduling.
4. Describe the term priority scheduling with example.
5. What is the difference between preemptive and non-preemptive scheduling?
6. With the help of diagram describe CPU-I/O burst cycle.
7. Explain the term dispatcher in detail.
8. With suitable diagram describe multilevel feedback queue scheduling.
9. Compare RR and SJF scheduling algorithm.
10. Differentiate SJF and FCFS.
11. Calculate Average Turnaround Time and Average Waiting Time for all set of processes using SJF.

Process	Burst Time	Arrival Time
P1	4	1
P2	3	0
P3	2	2
P4	4	3
P5	1	2

12. Consider the following set of processes:

Process CPU	Burst Time (in milliseconds)
P1	30
P2	6
P3	8

Calculate the Average Waiting Time and Average Turnaround Time by using Round Robin CPU Scheduling Algorithm. (The time quantum is of 5 milliseconds).

Q.III Define terms:

1. Waiting Time
2. Turnaround Time
3. Throughput
4. Relative Delay.
5. Burst time

Previous Exam Questions

Summer 2018

1. Define Burst Time. **[2 M]**

Ans. Refer to Section 4.3.

2. What is Turn-Around Time? **[2 M]**

Ans. Refer to Section 4.4.

3. What is CPU I/O Burst Cycle? **[2 M]**

Ans. Refer to Section 4.3.

4. Explain multilevel feedback queue algorithm. **[4 M]**

Ans. Refer to Section 4.4.4.2.

5. Consider the following set of processes with the length of CPU Burst Time and Arrival Time. **[4 M]**

Process	Burst Time	Arrival Time
P1	5	1
P2	3	0
P3	2	2
P4	1	3

Calculate turnaround time, waiting time, average turnaround time, average waiting time using FCFS CPU scheduling algorithm.

Ans. Refer to Section 4.5.1.

Winter 2018

1. Consider the following set of processes: **[4 M]**

Processes	CPU Burst Time (in ms)	Arrival Time (in ms)
P1	28	3
P2	7	1
P3	9	2

Calculate the Average Waiting Time and Average Turn-around Time by using Round Robin CPU Scheduling Algorithm. (The time quantum is of 5 milliseconds)

Ans. Refer to Section 4.5.4.

2. What is CPU Scheduler? State the criteria of CPU scheduling. **[4 M]**

Ans. Refer to Section 4.1.

3. What are the differences between Preemptive and Non-preemptive Scheduling? **[4 M]**

Ans. Refer to Sections 4.2.3 and 4.2.5

Summer 2019

1. Define Dispatch Latency Time. **[2 M]**

Ans. Refer to Section 4.2.5.

2. Round Robin algorithm is non-preemptive. Comment. **[2 M]**

Ans. Refer to 4.5.4.

3. List and explain four criteria for computing various scheduling algorithms. **[4 M]**

Ans. Refer to Section 4.4.

■■■

5...

Process Synchronization

Objectives...

- To introduce concept of Process synchronization.
- To learn about Critical Section Problem.
- To get knowledge of Semaphores.
- To study about Classical Problems of Synchronization.

5.1 INTRODUCTION

- Process synchronization means sharing resources by process in such a way that no two processes can share data and resources at the same time.
- It is specially used in multi process system when multiple processes wants to acquire same shared resource or data at the same time.
- When two or more processes wants to acquire shared region data it can lead to inconsistency of shared data, processes need to be synchronized with each other.
- The basic technique used to implement synchronization is to block a process until an appropriate condition is fulfilled.
- There are two kinds of synchronization one is control synchronization and another is data access synchronization.
    - **Control synchronization:** In this synchronization the processes may wish to co-ordinate their activities with respect to one another such that a process performs an action only when some other processes reach specific points in their execution.
    - **Data access synchronization:** This synchronization ensures that shared data do not lose consistency when they are updated by several processes. It is implemented by ensuring that accesses to shared data are performed in a mutually exclusive manner.
- Process Synchronization was introduced to handle problems that arise while multiple process executions. Some of the problems are discussed below.

5.2 CRITICAL SECTION PROBLEM [S-18, 19]

- A race condition occurs when multiple processes or threads read and write data items so that the final result depends on the order of execution of instructions in the multiple processes.

- Consider a system consisting of n processes {P_0, P_1 ... P_{n-1}}. Each process has a segment of code, called a **Critical Section,** in which the process may be changing common variables, updating a table, writing a file and so on.
- In a sense, updating of a shared variable may be regarded as a critical section. The critical section is a sequence of instructions with a clearly marked beginning and end. It usually safeguards of updating of one or more shared variables.
- When one process is executing in critical section, no other process is allowed to enter in critical section. Each process must request to enter into the critical section.
- There are three sections which can be categorized as Entry Section, Critical Section, and Exit Section. The process has to make request in first section that is entry section.
- When a process enters a critical section, it must complete all instructions there in before any other process is allowed to enter the same critical section. Only the process executing the critical section is allowed to access the shared variable.
- Thus, is often referred to as **mutual exclusion,** in which a single process temporarily excludes all other from using a shared resource in order to ensure the system's integrity.
- To be acceptable as a general tool, a solution to mutual exclusion problem should:
  - Ensure mutual exclusion between processes accessing the protected shared resources.
  - Make no assumption about relative speeds and priorities of contending processes.
  - Guarantee that crashing of process outside of its critical section does not affect the ability of other contending processes to access the shared resource.
  - When more than one process wishes to enter critical section, grant entrance to one of them in finite time.

```
do
   {
      entry section
      critical section
      exit section
      remainder section
   }  while(1);
```

Fig. 5.1: General Structure of a Typical Process Pi

- A solution to critical section problem must satisfy the following conditions:
  1. **Mutual Exclusion:** If process P1 is executing in the critical section, no other process can be executing in the critical section.
  2. **Progress:** When no process is executing in its critical section and some processes wish to enter their critical sections, then only those processes that are not executing in their remainder sections can participate in deciding which will enter its critical section next, and this selection cannot be postponed indefinitely.

3. **Bounded waiting:** There exists a bound 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 that request is granted.

5.3 SEMAPHORES [S-18, 19]

5.3.1 Concept

- It is a method or tool which is used to prevent race condition. When co operating processes try to gain access to shared region data at that time race condition can occur so semaphore is one of the tools which are used to apply synchronization between processes and to avoid race conditions and deadlocks.
- A semaphore is a shared integer variable with non-negative values which can only be subjected to following two operations.
  1. Initialization,
  2. Invisible operations.
- The first major advance in dealing with the problems of concurrent processes came in 1965 with **Dijkstra's** solution. Dijkstra's proposal for mutual exclusion among an arbitrary number of processes is called **Semaphore.**
- A semaphore mechanism basically consists of two primitive operations SIGNAL and WAIT, (originally defined as P and V by Dijkstra), which operate on a special type of semaphore variables s.
- The semaphore variable can assume integer values, and except possibly for initialization may be accessed and manipulated only by means of the SIGNAL and WAIT operations.
- The two primitives take one argument each the semaphore variable and may be defined as follows.
  - **Wait(s):** Decrements the value of its argument semaphore, s, as soon as it would become non-negative. Completion of WAIT operation, once the decision is made to decrement its argument semaphore, must be indivisible.
  - **Signal(s)**: Increments the value of its argument semaphore, s, as an indivisible operation. The logic of busy wait versions of the WAIT and SIGNAL operations is given in Fig. 5.2.

```
struct semaphore
   { int count;
    queue Type queue;
   };
void wait(semaphore s)
{ s.count--;
   if(s.count<0)
   {
   place a process P in s.queue;
    block this process.
   }
}
```

```
void signal(semaphore s)
 {
   s.count++;
   if(s.count<=0)
   {
    remove a process P from s.queue;
    place process P on ready list;
   }
}
```

Fig. 5.2: A Busy – wait Implementation of WAIT and SIGNAL

5.3.2 Implementation

- The main disadvantage of semaphore is that it requires busy waiting. It waste CPU cycles that some other process might be able to use productively.
- This type of semaphore is also called as Spinlock because the process spins while waiting for the clock. To overcome the need of busy waiting, we can modify the definition of wait() and signal() semaphore operations.
- When a process executes the wait() operation and finds that semaphore value is positive. It must wait, however, rather than engaging in busy waiting, the process can block itself.
- The block operation places a process into a waiting queue associated with semaphore, and the state of the process is switched to the waiting state.
- Then control is transferred to the CPU scheduler, which selects another process to execute. A process that is blocked, waiting on a semaphore S, should be restarted when some other process executes a signal() operation.
- The process is restarted by wakeup() operation, which changes process from waiting to ready state. The process is then placed in ready queue.
- To implement semaphore under this definition, we have define semaphore C Struct as:

```
typedef struct
{
   int value;
   struct process *list;
}semaphore;
```

- Each semaphore is an integer value and a list of processes list. When a process must wait on semaphore, it is added to the list of processes.

Semaphore Operations:

- Wait() operation can be defined as:

```
wait(semaphore *s)
{
s→value --;
    if(s→value<0)
    {
    add this process to s→list;
    block();
    }
}
```

- A signal() operation removes one process from the list of waiting processes and awakens the process.
- signal() operation can be defined as,

```
signal(semaphore *s)
{
s→value++;
    if(s→value<=0)
    {
    remove a process p from s→list;
    wakeup (p);
    }
}
```

- The block() operation suspends the process that invokes it.
- The wakeup() operation resumes the execution of blocked process p.

5.3.3 Types of Semaphores [S-19]

- Semaphores can be implemented with two types: counting semaphores and binary semaphores.

1. Binary Semaphore:

- In binary semaphores, the value of semaphore can range only between 0 and 1. Initially the value is set to 1 and if some process wants to use resource then the wait() function is called and the value of the semaphore is changed from 0 to 1.
- The process uses the resource and when it releases the resource then signal() function is called and the value of semaphore variable is increased to 1. If at a particular instant time the value of semaphore variable is 0 and some other process wants to use the same resource then it has to wait for the release of resource by previous process.
- For both semaphores a queue is used to hold processes waiting on semaphore. The process that has been blocked the longest is released from the queue first. A semaphore whose definition includes this policy is called a Strong Semaphore.
- A semaphore that does not specify the order in which processes are removed from the queue is a Weak Semaphore.

Counting Semaphore:

- Counting semaphore can take an integer value ranging between 0 and an arbitrarily large number. The Initial value represents the number of units of the critical resources that are available. This is also known as general semaphore.
- A counting semaphore is helpful to coordinate the resource access, which includes multiple instances.
- In counting semaphore, there is no mutual exclusion:
- In counting semaphores, firstly, the semaphore variable is initialized with the number of resources available. After that, whenever a process needs some resource, then the wait() function is called and the value of the semaphore variable is decreased by one.
- The process then uses the resource and after using the resource, the signal() function is called and the value of the semaphore variable is increased by one. So, when the value of the semaphore variable goes to 0 that is all the resources are taken by the process and there is no resource left to be used, then if some other process wants to use resources then that process has to wait for its turn. In this way, we achieve the process synchronization.

5.3.4 Deadlocks and Starvation

- **Deadlocks:** The implementation of a semaphore with a waiting queue may result in a situation where more than two processes are waiting indefinitely for an event that can be caused only by one of the waiting processes.
- The event in question is the execution of a signal() operation.
- When such a state is reached, these processes are said to be deadlocked.
- To illustrate this approach, we consider a system consisting of two processes, P0 and P1, each accessing two semaphores, S and Q, set to the value 1:

P0	P1

wait(S);
wait(Q);
 	
signal(S);
signal(Q);
	wait(Q);
	wait(S);
	
	signal(Q);
	signal(S);

1. Suppose that P0 executes wait(S) and then P1 executes wait(Q).
2. When P0 executes wait(Q), it must wait until P1 executes signal(Q). Similarly, when P1 executes wait(S), it must wait until P0 executes signal(S).
3. Since, these signal() operations cannot be executed, P0 and P1 are deadlocked.
4. We say that a set of processes is in a deadlock state when every process in the set is waiting for an event that can be caused only by another process in the set.
5. The events with which we are mainly concerned here are resource acquisition and release.

- **Starvation:** Another problem related to deadlocks is indefinite blocking, or starvation, a situation in which processes wait indefinitely within the semaphore. Indefinite blocking may occur if we remove processes from the list associated with a semaphore in LIFO (Last-In, First-Out) order.

5.4 CLASSICAL PROBLEMS OF SYNCHRONIZATION

- Semaphore can also be used to solve various synchronization problems. In this section, we present some classical problems of synchronization and use semaphores for synchronization in the solutions of these problems.

5.4.1 Bounded Buffer Problem

- Bounded buffer problem is commonly used to illustrate the power of synchronization primitives.
- In general, the producer/consumer problem may be stated as follows:
  1. Given a set of co-operating processes, some of which 'produce' data item (producers) to be consumed by others (consumers), with possible disparity between production and consumption rates.
  2. Device a synchronization protocol that allows both producers and consumers to operate concurrently as their respective service rates in such a way that produced items are consumed in the exact order in which they are produced (FIFO).

(i) Producer and Consumer with an unbounded buffer:

- Here, we assume that buffer is of unbounded capacity. After the system is initialized, a producer must be the first process to run in order to provide the first item. From that point on, a consumer process may run whenever there is more than one item in the buffer produced but not yet consumed. Given the unbounded buffer, producers may run at any time without restrictions. We also assume that all items produced and subsequently consumed have identical, but unspecified structure. The buffer may be implemented as an array, a linked list, or any other collection of data items. The first solution is as given in Fig. 5.3.
- Since, producer and consumer both accesses same buffer, buffer manipulation procedure must be placed within a critical section protected by a semaphore.

```
/* Program producer – consumer */
int n;
binary semaphore mutex = 1;
general_semaphore produced = 0;
void producer( )
{
     while(true)
     {
        produce( );
        wait(mutex);
```

```
            place_in_buffer;
            signal (mutex);
            signal (produced);
        }
    }
    void consumer( )
    {
        while(true)
        {
            wait(produce);
            wait(mutex);
            take_from_buffer;
            signal(mutex);
            consume;
        }
    }
    void main( )
    {
        initiate producer, consumer;
    }
```

Fig. 5.3: Producer/Consumer Bounded Buffer

- Consequently, a number of producers and consumers may execute concurrently. The initial value of producer is set to 0. When an item is produced, it is placed in the buffer, and the fact is signaled by means of the general semaphore PRODUCED. The nature of the problem implies that the consumer can never get ahead of the producer; since consumer is waiting on PRODUCED semaphore. When item is produced then only consumer can enter in critical section by applying wait on mutex semaphore.
- Adding two semaphores must be handled with care because two semaphores may interact in an undesirable manner. Consider for e.g. reversing the order of WAIT operations in consumer process. As a consequence, waiting on PRODUCED semaphore is moved into critical section controlled by MUTEX. This in turn, may deadlock the system from very start. For instance assume that where a producer is busy preparing in first item, a consumer process becomes scheduled. MUTEX is initially FREE and consumer enters in critical section. It has no item to consume and it is forced to wait on PRODUCED semaphore. However no producer can ever succeed in placing its item in buffer since MUTEX is busy. Consequently consumer remains in critical section forever and system is deadlocked. This tells that although semaphores are powerful tool, their use by no means automatically solve all timing problems.

(ii) Producers and Consumers with a bounded buffer:

- The unbounded – buffer assumption simplifies analysis of producer – consumer problem by allowing unrestricted execution of producers. However this assumption unrealistic since, computer system, which have finite memory capacity.
- The main difference imposed by bounded buffer is that both consumer and producer may be halted under certain circumstances. At any particular time the shared global buffer may be empty, partially filled or full. A producer process may run in either of two former cases, but all producers must be kept waiting when buffer is full similarly when buffer is empty consumer must wait.
- Let i count be the number of items produced but not yet consumed so,

  `icount = produced – consumed.`

  If we have finite capacity then,

  `0 < icount < capacity.`

  Since, producer may run only when there are some empty slots in buffer it can be said,

  `mayproduce: icount < capacity.`

  Consumers can execute only when there is at least one item produced but not yet consumed i.e.

  `mayconsume: icount>0.`
- In practice, buffers are usually implemented in circular fashion, using linked list. Two indices in and out, point to next slot available for a produced item and to the place where the next item is to be consumed from respectively.

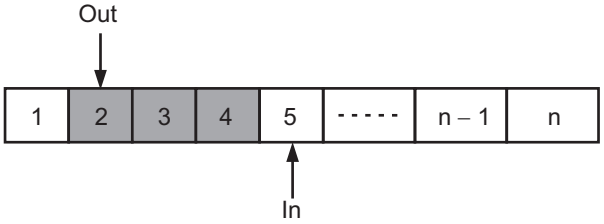


Fig. 5.4: Producer/Consumer Buffer

- Fig. 5.5 shows a solution to bounded buffer producer/consumer problem. General semaphores *mayproduce* and *mayconsume* represent two conditions introduced earlier to control execution of producer and consumer. Two binary semaphores *pmutex* and *cmutex* protect buffer and index manipulations of producers and consumers.

```
buffer → array [1 – capacity];
Semaphore mayproduce, mayconsumer; /* general */
binary semaphore Pmutex, cmutex;
void producer( )
{
   item pitem;
```

```
    while(true)
    {
       wait(mayproduce);
       pitem=produce;
       wait(pmutex);
       buffer[in] = pitem;
       In=(in mod capacity) +1;
       Signal(pmutex);
       Signal(mayconsume);
    }
}
void consumer( )
{
    item citem;
    {
    while(true)
    {
    wait(mayconsume);
    wait(cmutex);
    citem = buffer[out];
    out=(out mod capacity) +1;
    signal(cmutex);
    signal(mayproduce);
    consume (item);
    }
    }
void paraent process( )
{
    in = 1;
    out = 1;
    signal(pmutex);
    signal(cmutex);
    mayconsume = 0;
    for(i=1 to capacity)
    {signal(mayproduce);
}
}
```

Fig. 5.5: Producer / Consumer Bounded Buffer

- Initially *mayproduce* is set to capacity to indicate producer can produce that many item. Whenever consumer completes its cycle, it implies a slot by removing the consumed item from buffer and signals the fact via the *mayproduce*. The *mayconsume* semaphore indicates availability of produced items, and it functions much the same way as in unbounded buffer.

5.4.2 Readers and Writers Problem [S-18, 19]

- Readers and writers is another classical problem in concurrent programming. It basically resolves around a number of processes using a shared global data structure.
- The processes are categorized depending on their usage of the resource, as either readers or writers.
- A reader never modifies the shared data structure, whereas a writer may both read it and write into it. A number of readers may use the shared data structure concurrently because no matter how they are interleaved, they cannot possibly compromise its consistency.
- Writers, on the other hand, must be granted exclusive access to data.
- The problem may be stated as follows:
  - Given a universe of readers that read a common data structure and a universe of writers that modify the same common data structure.
  - Device a synchronization protocol among the readers and writers that ensure consistency of common data while maintaining as high a degree of concurrency as possible.
- One approach to solve readers/writers problem is as shown in Fig. 5.6. The writer process waits on binary semaphore WRITE to grant the permission to enter the critical section and to use the shared resources.
- A reader, on the other hand, goes through two critical sections, one before and one after using the resource. Their purpose is to allow a large number of readers to execute concurrently while making sure that readers are kept away when writers are active.
- An integer, READERCOUNT is used to keep track of the number of readers actively using the resource.
- In Fig. 5.6, first reader passes through MUTEX, increments the number of readers and waits on writers if any. While reader is reading data, semaphore MUTEX is free and WRITE is busy to allow multiple readers only.
- If there are writers, waiting they are prevented by busy WRITE semaphore. When READERCOUNT reaches to zero writers can enter in critical section.
- Even if this solution has high degree of concurrency it faces starvation of writer by postponing them indefinitely when readers are active.
- A strategy proposed by Hoare (1974) holds promise for both readers and writers to complete in finite time. It suggests that,
  - A new reader should not start if there is writer waiting, (prevent starvation of writer).
  - All readers waiting at the end of a write should have priority over next writer, (prevents starvation of readers).

```
/* Program readers_ writers*/
int readercount;
binary_semaphore mutex, write;
void reader( )
{
     while(true)
     {
        wait(mutex);
        readercount ++;
        If(readercount==1)
        wait(write)
        signal(mutex);
        /* read data */
        wait(mutex)
        readercount – –;
        if(readercount==0)
        signal(write);
     }
  }
  void writer()
  {
     while(true)
     {
        wait (write);
        ……
        /* write data */
        signal(write)
     }
}
void parentprocess()
{
     readercount=0;
     signal(mutex);
     signal(write);
     initiate readers, writers
  }
```

Fig. 5.6: Readers/Writers

5.4.3 Dining Philosophers Problem [W-18]

- To understand the Dining Philosopher's problem, Consider five philosophers share a common circular table surrounded by five chairs, each belonging to one philosopher.
- In the center of table is a bowl of rice, and the table is laid with five single chopsticks (Fig. 5.7).

- When a philosopher thinks, she does not interact with her colleagues. From time to time, a philosopher gets hungry and tries to pick up the two chopsticks that are closest to her, (the chopsticks that are between her and her left and right neighbor).
- A philosopher may pick up only one chopstick at a time. Obviously, she cannot pick up a chopstick that is already on the hand of a neighbour.
- When a hungry philosopher has both her chopsticks at the same time, she eats without releasing her chopsticks. When she is finished eating, she puts down both of her chopsticks and start thinking again.

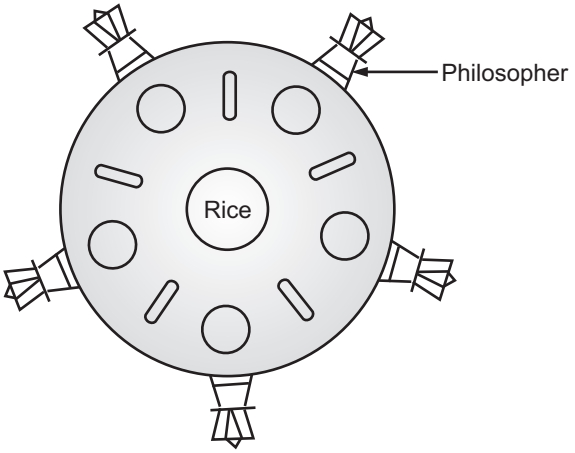


Fig. 5.7: The Situation of Dining Philosophers

- The Dining Philosopher problem is considered a classic synchronization problem.
- One simple solution is to represent each chopstick of a semaphore.
- A philosopher tries to grab the chopstick by executing a wait operation on that semaphore. She releases her chopsticks by executing the signal operation on the appropriate semaphores.
- Thus, the shared data are,

```
Semaphore chopsticks[5];
```

- Where all the elements of chopsticks are initialized to 1. Structure of philosopher i is shown in Fig. 5.8.

```
do
{
     wait (chopsticks[i]);
     wait(chopsticks[(i+1)%5]);
     ……
     eat
     s ……
     signal(chopstick[i]);
     signal(chopstick[(i+1)%5]);
     ……
     think
     ……
  } while(1)
```

Fig. 5.8: Structure of Philosopher

- Although this solution guarantees that no two neighbors are eating simultaneously, it nevertheless must be rejected because it has the possibility of creating a deadlock. Suppose that all five philosophers become hungry simultaneously, and each grabs her left chopstick.
- All elements of chopstick will now be equal to 0. When each tries to grabs right chopstick she will be delayed forever. We can solve this by allow to pick chopsticks only if both are available.

Summary

- Process synchronization means sharing resources by process in such a way that no two processes can share data and resources at the same time. There are two kinds of synchronization: Control synchronization and Data access synchronization.
- A race condition occurs when multiple processes or threads read and write data items so that the final result depends on the order of execution of instructions in the multiple processes.
- Critical section is the part of program which tries to access shared resource, that resource may be any resource like memory location, data structure, CPU or any I/O device.
- The critical section cannot be executed more than one process at same time.
- The critical section problem is used to design a set of protocols which can ensure that the race condition among the processes will never arise.
- Semaphore is defined as a variable that is non-negative and shared between threads. It is a mechanism that can be used to provide synchronization of tasks.
- Counting semaphore uses a count that helps task to be acquired or released numerous times.
- The binary semaphores are quite similar to counting semaphores, but their value is restricted to 0 and 1. Wait operation helps you to control the entry of a task into the critical section.
- Signal semaphore operation is used to control the exit of a task from a critical section. Counting Semaphore has no mutual exclusion whereas Binary Semaphore has Mutual exclusion Semaphore means a signaling mechanism whereas Mutex is a locking mechanism.
- Semaphore allows more than one thread to access the critical section. One of the biggest limitations of a semaphore is priority inversion.
- Classical problems of synchronization we had discusses are bounded buffer problem, readers-writers problem and dining philosopher problems.

Check Your Understanding

1. Which of the following processes can get affected by other process executing in the system?
   (a) child process (b) parent process
   (c) independent process (d) co operating process

2. When process is executing in critical section, no other process can execute in critical section, this condition is called?
   (a) critical section (b) logical section
   (c) mutual exclusion (d) non mutual exclusion
3. Amongst the following which can be considered as synchronization tool?
   (a) socket (b) mutex
   (c) semaphore (d) thread
4. Select the correct statement regarding mutex lock to prevent race condition.
   I. A process must acquire the lock before entering critical section.
   II. A process need to acquire the lock before entering critical section.
   III. It releases the lock when it exits the critical section.
   IV. A process must acquire the lock when it exits critical section
   (a) 1, 4 (b) 2, 4
   (c) 2, 3 (d) 1, 3
5. In the spinlocks _____.
   (a) no context switch is required when a process must wait on a lock
   (b) locks are expected to be held for short times
   (c) employed on multiprocessor systems
   (d) all of above
6. Indefinite blocking or starvation is problem related with _____.
   (a) mutex locks (b) deadlocks
   (c) spinlocks (d) none of the above
7. Which of the following is not classical problem of synchronization?
   (a) reader writer problem (b) bounder buffer problem
   (c) dining philosopher problem (d) customer employer problem
8. A semaphore is a shared integer variable that can not _____.
   (a) go beyond zero (b) more than zero
   (c) it can not more than 1 (d) none of above
9. Process synchronization can be done with following levels.
   (a) hardware (b) software
   (c) both (d) none of above
10. Deadlock can be avoided by _____.
    (a) resource allocation must be done at once
    (b) there must be fixed number of resources to allocate
    (c) all deadlocked processes must be aborted
    (d) inversion technique can be used
11. Which is not part of critical section?
    (a) entry section (b) critical section
    (c) exit section (d) waiting section

Answers

1. (d)	2. (c)	3. (c)	4. (d)	5. (d)	6. (b)	7. (d)	8. (a)	9. (c)	10. (b)	11. (d)

Practice Questions

Q.I Answer the following questions in short:

1. What is process synchronization?
2. What is mutual exclusion?
3. What is meant by binary semaphore?
4. What is meant by deadlock and starvation?
5. List all types of classical problems of synchronization.

Q.II Answer the following questions:

1. Describe Critical Section problem in detail.
2. Explain Readers and Writers problem in detail.
3. With suitable example describe bounded buffer problem in detail.
4. What is semaphore? Explain with its types.
5. Explain bounded buffer problem of synchronization in detail.
6. Explain in detail Dining Philosopher problem.

Q.III Define terms:

1. Strong semaphore
2. Weak semaphore
3. Synchronization
4. Deadlocks
5. Starvation

Previous Exam Questions

Summer 2018

1. What is Semaphores? **[2 M]**

Ans. Refer to Section 5.3.

2. Describe solution for critical section problem. **[4 M]**

Ans. Refer to Section 5.2.

3. Explain the readers and writes problem which is a classical problem of synchronization. **[4 M]**

Ans. Refer to Section 5.4.2.

Winter 2018

1. Describe in detail the 'Dinning Philosopher Problem' synchronization problem. **[4 M]**

Ans. Refer to Section 5.4.3

Summer 2019

1. Define critical section problem and list its solutions. **[2 M]**

Ans. Refer to Section 5.2.

2. Explain Reader's writer's problems. **[4 M]**

Ans. Refer to Section 5.4.2.

3. Explain semaphores and its types in detail. **[4 M]**

Ans. Refer to Sections 5.3.1 and 5.3.3.

4. Explain WAIT and SIGNAL semaphore operations. **[4 M]**

Ans. Refer to Section 5.3.1.

■■■

6... Deadlock

Objectives...

- To learn about concept and principles of Deadlock.
- To study the Deadlock Characterization using Necessary Conditions.
- To learn about different Deadlock handling techniques such as Deadlock Prevention and Avoidance, Detection and Recovery.

6.1 INTRODUCTION [S-18]

- Every process needs some resources to complete its execution. However, the resource is granted in a sequential order.
  1. The process requests for some resource.
  2. Operating system grant the resource if it is available otherwise let the process waits.
  3. The process uses it and release on the completion.
- In an operating system, deadlock state happens when two or more processes are waiting for the same event to happen which never happen, then we can say that those processes are involved in the deadlock.
- For example, Process P1 wants to access the resource R1 while it has access to resource R2. It will release the resource only after it has acquired R2. There is another process P2, which has the access to resource R2 and wants the access to resource R1. This process is also developed in such a way that it will release R2 only when it has acquired R1.

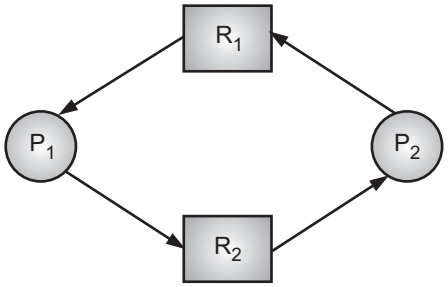


Fig. 6.1: Deadlock state in Operating System

- So, currently in the system there are two processes which have access to one of the system resources and want access to another resource. And they will leave the resource only when they get access to the new resource. This condition is called a deadlock as no process will get to access the required resource and continue execution.

Example of Deadlock State:

(A) Deadlock involving the same resource type:

- To illustrate a deadlock state, we consider a system with three CD-RW drives.
  - Suppose each of the three processes holds one of these CD-RW drives.
  - If each process now requests another drive, the three processes will be in a deadlock state.
  - Each is waiting for the event "CD-RW is released" which can be caused only by one of the other waiting processes.
- This example illustrates a deadlock involving the same resource type.

(B) Deadlock involving the different resource types:

- Deadlocks may also involve different resource types. For example, consider a system with one printer and one tape drive. Suppose process P_i is holding the tape drive and process P_j is holding the printer.
- If P_i requests the printer and P_j requests the tape drive, deadlock occurs.
- A programmer who is developing multithreaded application must pay particular attention to this problem. Multithreaded programs are good candidates for deadlock because multiple threads can compete for shared resources.

6.2 DEADLOCK CHARACTERIZATION

- In a deadlock, processes never finish executing and system resources are tied up, preventing other jobs from starting.
- Before we discuss the various methods for dealing with the deadlock problems, we shall describe features that characterize deadlocks.

6.3 NECESSARY CONDITIONS [W-18, S-19]

- A deadlock situation can arrive if the following four conditions hold simultaneously in a system.
  1. **Mutual Exclusion**: At least one resource is held in a non-sharable mode; that is only one process at a time can use the resource. Example monitor, printer etc. If another process requests that resource, the requesting process must be delayed until the resource has been released. Each resource is either currently assigned to exactly one process or is available.
  2. **Hold and Wait:** A process must be holding at least one resource and waiting to acquire additional resources that are currently being held by other processes.
  3. **No preemption:** The process which once scheduled will be executed till the completion. No other process can be scheduled by the scheduler meanwhile.

4. **Circular Wait:** All the processes must be waiting for the resources in a cyclic manner so that the last process is waiting for the resource which is being held by the first process.

6.4 DEADLOCK HANDLING TECHNIQUES

- There are mainly four methods for handling technique:
  1. Deadlock Prevention
  2. Deadlock Avoidance
  3. Deadlock Detection
  4. Recovery from Deadlock

6.4.1 Deadlock Prevention

- Deadlock prevention is a set of methods for ensuring that at least one of the necessary conditions cannot hold.
- That means that we design such a system where there is no chance of having a deadlock.
- By ensuring that at least one of the following conditions cannot hold, we can prevent the occurrence of a deadlock.
  - **(a) Mutual exclusion:** It can't be resolved as it is the hardware property. For example, the printer cannot be simultaneously shared by several processes. This is very difficult because some resources are not sharable.
  - **(b) Hold and wait:** Hold and wait can be resolved using the conservative approach where a process can start it and only if it has acquired all the resources.
  - **(c) Active approach:** Here the process acquires only requires resources but whenever a new resource requires it must first release all the resources.
  - **(d) Wait time out:** Here there is a maximum time bound until which a process can wait for other resources after which it must release the resources.
  - **(e) Circular wait:** In order to remove circular wait, we assign a number to every resource and the process can request only in the increasing order otherwise the process must release all the high number acquires resources and then make a fresh request.
  - **(f) No preemption:** In no preemption, we allow forceful pre-emption where a resource can be forcefully pre-empted. The pre-empted resource is added to the list of resources where the process is waiting. The new process can be restarted only when it regains its old resources. Priority must be given to a process which is in waiting for state.

6.4.2 Deadlock Avoidance [S-19]

- It requires that the operating system be given in advance additional information concerning which resources a process will request and use during its lifetime. With this additional information, we can decide for each request can be satisfied or must be delayed.
- There are various algorithms are available to handle this situation.

- **A Deadlock Avoidance Algorithm** dynamically examines the resources allocation state to ensure that a circular wait condition case never exists. Where the resources allocation state is defined by the available and allocated resources and the maximum demand of the process.
- **Examples:** Resource Allocation Graph algorithm, Banker's algorithm.

6.4.2.1 Safe State

- There are 3 states of the system: safe, unsafe and deadlock.
- When a system can allocate the resources to the process in such a way so that they still avoid deadlock then the state is called **safe state**.
- A system is in safe state only if there exists a **safe sequence.**
- A sequence of process $\mathbf{P_1, P_2, ..., P_n}$ is a safe sequence for the current allocation state if for each $\mathbf{P_i}$ the resources request that $\mathbf{P_i}$ can still make can be satisfied by currently available resources pulls the resources held by all $\mathbf{P_j}$ with $\mathbf{j < i}$.

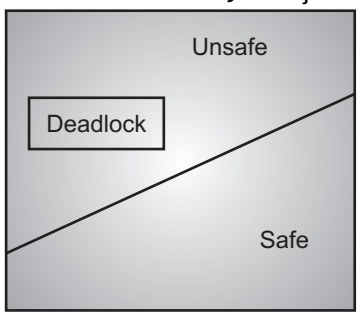


Fig. 6.2: Space of Safe, Unsafe and Deadlock States

- If a safe sequence does not exist, then the system is in an **unsafe** state, which may lead to deadlock. (All safe states are deadlock free, but not all unsafe states lead to deadlocks).

6.4.2.2 Resource Allocation Graph Algorithm [S-18, W-18]

- Deadlock can be described more precisely in terms of a directed graph called a system Resource Allocation Graph.
- In Resource Allocation Graph, processes are represented by circle and resources are represented by boxes. Resource boxes have some number of dots inside indicating available number of that resource that is number of instances.

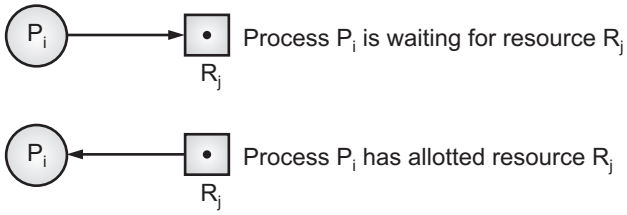


Fig. 6.3: Representation of Process and Resources in the Graph

- This graph consist of a set of vertices V and a set of edges E.
- The set of vertices V is partitioned into two different types of nodes:

$$P = \{P_1, P_2, P_3, \ldots P_n\}$$

- The set consisting of all the active processes in the system, and

$$R = [R_1, R_2, R_3, \ldots R_m\}$$

- The set consisting of all resource types in the system.
- A directed edge from process P_i to resource type R_j is denoted by:

$$P_i = R_j - \text{request edge}$$

- It signifies that process P_i requested an instance of resource type R_j and is currently waiting for that resource.
- A directed edge from resource type R_j to process P_i is denoted by:

$$R_j = P_i - \text{assignment edge}$$

- It signifies that an instance of resource type R_j has been allocated to process P_i.
- When process P_i request an instance of resource type R_j, a request edge is inserted in the resource allocation graph.
- When this request can be fulfilled, the request edge is instantaneously transformed to an assignment edge. When the process no longer needs access to the resource, it release the resource and result is assignment edge is deleted.
- The resource allocation graph is shown in Fig. 6.4.

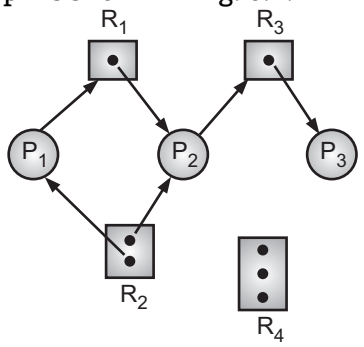


Fig. 6.4: Resource Allocation Graph

- Pictorially process P_i is represented as a circle and each resource, R_j as square since resource type R_j may have more than one instance, we represent each such instance as a dot within the square.
- Also note that a request edge points to only square R_j, whereas an assignment edge must also designate one of the square.
  - The sets P, R and E

$$P = \{P_1, P_2, P_3\}$$

$$R = \{R_1, R_2, R_3, R_4\}$$

$$E = \{P_1 \rightarrow R_1, P_2 \rightarrow R_3, R_1 \rightarrow P_2, R_2 \rightarrow P_2, R_2 \rightarrow P_1, R_3 \rightarrow P_3\}$$

  - Resource instances
    - One instance of resource type R_1.
    - Two instances of resource type R_2.
    - One instance of resource type R_3.
    - Three instances of resource type R_4.

- Process states
  - Process P_1 is holding an instance of resource type R_2, and is waiting for an instance of resource type R_1.
  - Process P_2 is holding an instance of R_1 and R_2 and is waiting for an instance of resource type R_3.
  - Process P_3 is holding an instance of R_3.

- Given the definition of resource allocation graph, it can be shown that if the graph contains no cycles, then no process in a system is deadlocked. If the graph contains a cycle, then a deadlock may exist.
- If each resource type has one instance, then a cycle implies that a deadlock has occurred. If the cycle involves only a set of resource types, each of which has only a single instance, then a deadlock has occurred. Each process involved in a cycle is deadlocked.
- In this case, cycle in the graph is both necessary and a sufficient condition for the existence of deadlock if single instance of each and every resource is there in graph.
- If each resource type has several instances, then a cycle does not necessarily imply that a deadlock has occurred. In this case, cycle is necessary but not sufficient condition for the existence of deadlock.
- To illustrate this concept, let us return to the resource-allocation graph shown in Fig. 6.4. Suppose the process P_3 requests an instance of resource type R_2.
- Since, no resource instance is currently available, a request edge $P_3 \rightarrow R_2$ is added to the graph. At this time, two minimal cycles exist in the system.

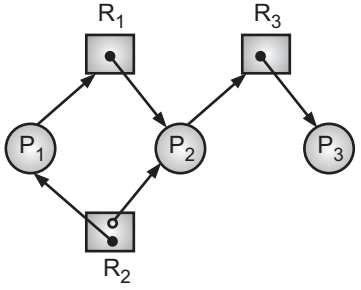
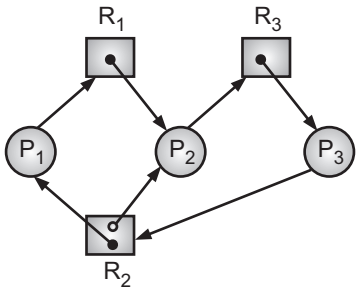


Fig. 6.5: Resource Allocation Graph with a Deadlock

- Process P_1, P_2, P_3 is deadlocked. Process P_2 is waiting for resource R_3, which is held by process P_3, on the other hand, is waiting for either process P_1 or process P_2 to release resource R_2.
- In addition, process P_1 is waiting for process P_2 to release resource R_1.
- Now consider the resource allocation graph in Fig. 6.6. In this example, we have a cycle.



$P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$

$P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_2$

Fig. 6.6: Resource-Allocation Graph with a Cycle but no Deadlock

- However, there is no deadlock because process P_4 may release its instance of resource type R_2. That resource can be allocated to P_3, breaking the cycle.
- In short, if a resource allocation graph does not have a cycle, then the system is not in a deadlock state. On the other hand, if there is a cycle, then the system may or may not be in a deadlock state.

6.4.2.3 Banker's Algorithm [S-19]

- Banker's Algorithm is a **deadlock avoidance algorithm**. It is also used for deadlock detection.
- This algorithm tells that if any system can go into a deadlock or not by analyzing the currently allocated resources and the resources required by it in the future.

Claim Edge:

- Let us introduce a new kind of edge, a *claim edge*, represented by a dashed line. Like a request edge, a claim edge points from a process to a resource type.

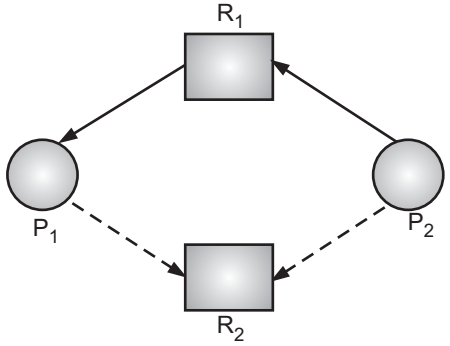


Fig. 6.7: Resource Allocation Graph for Deadlock Avoidance

- A claim edge indicates that a process P_i *may request* the resource R_j some time *in the future*.
- All claim edges for P_i must be present before P_i starts executing.
- Thus, a request edge (P_i, R_j) may be added only if a claim edge (P_i, R_j) is already present.
- A resource-allocation graph with claim edges is called a *maximum-claim graph.* It reflects the projected worst-case future state in resource allocation.
- A state is safe if its corresponding maximum-claim graph is deadlock free.

Data Structures used to implement Banker's Algorithm:

- Several data structures must be maintained to implement the Banker's algorithm. These data structures encode the state of the resource allocation system. Let P_1 be the number of processes in the system and R_1 be the number of resource types.

1. **Available:** It is a 1–D array that tells the number of each resource type (instance of resource type) currently available.

   Example: Available [R_1] = A, means that there are **A** instances of R_1 resources are currently available.

2. **Max:** It is a 2-D array that tells the maximum number of each resource type required by a process for successful execution.

   Example: Max[P_1][R_1] = A, specifies that the process P_1 needs a maximum of **A** instances of resource R_1 for complete execution.

3. **Allocation:** It is a 2-D array that tells the number of types of each resource type that has been allocated to the process.

   Example: Allocation[P_1][R_1] = A, means that **A** instances of resource type R_1 have been allocated to the process P_1.

4. **Need:** It is a 2-D array that tells the number of remaining instances of each resource type required for execution.

   Example: Need [P_1][R_1] = A tells that **A** instances of R_1 resource type are required for the execution of process P_1.

   Need[i][j] = Max[i][j] – Allocation[i][j], where i corresponds any process P(i) and j corresponds to any resource type R(j).

The Bankers Algorithm consists of the following two algorithms:

1. Request-Resource Algorithm
2. Safety Algorithm

1. Resource - Request Algorithm:

- Whenever a process makes a request of the resources then this algorithm checks that if the resource can be allocated or not.
- This algorithm includes following three steps:

Step 1 : The algorithm checks that if the request made is valid or not. A request is valid if the number of requested resources of each resource type is less than the **Need** (which was declared previously by the process). If it is a valid request then Step 2 is executed else aborted.

Step 2 : Here, the algorithm checks that if the number of requested instances of each resource is less than the available resources. If not then the process has to wait until sufficient resources are available else go to Step 3.

Step 3 : Now, the algorithm assumes that the resources have been allocated and modifies the data structure accordingly.

Available = Available – Request(i)

Allocation(i) = Allocation(i) + Request(i)

Need(i) = Need(i) – Request(i)

- After the allocation of resources, the new state formed may or may not be a safe state. So, the **Safety algorithm** is applied to check whether the resulting state is a safe state or not.
- **Safe state:** A safe state is a state in which all the processes can be executed in some arbitrary order with the available resources such that no deadlock occurs.
  1. If it is a safe state, then the requested resources are allocated to the process in actual.
  2. If the resulting state is an unsafe state then it rollbacks to the previous state and the process is asked to wait longer.

2. Safety Algorithm:

- The safety algorithm is applied to check whether a state is in a safe state or not.
- This algorithm involves the following four steps:

Step 1 : Suppose currently all the processes are to be executed. Define two data structures as work and finish as vectors of length m (where m is the length of **Available** vector) and n (is the **number of processes** to be executed).

Work = Available

Finish[i] = false for i = 0, 1, ... , n – 1.

Step 2 : This algorithm will look for a process that has **Need** value less than or equal to the **Work**. So, in this step, we will find an index i such that,

Finish[i] = = false &&

Need[i] <= Work

If no such 'i' is present then go to Step 4 else to Step 3.

Step 3 : The process '**i**' selected in the above step runs and finishes its execution. Also, the resources allocated to it get free. The resources which get free are added to the Work and Finish(i) of the process is set as true. The following operations are performed:

Work = Work + Allocation

Finish[i] = true

After performing the 3rd step, go to step 2.

Step 4 : If all the processes are executed in some sequence then it is said to be a safe state. Or, we can say that

If Finish[i] == true for all i,

Then the system is said to be in a **Safe State**.

Examples of Banker's Algorithm

Example 1: Suppose we have 3 processes (A, B, C) and 3 resource types (R_1, R_2, R_3) each having 5 instances. Suppose at any time t if the snapshot of the system taken is as follows then find the system is in a safe state or not.

Solution:

Total instances of each Resource

R_1	R_2	R_3
5	5	5

Process	Allocation			Max.		
	R_1	R_2	R_3	R_1	R_2	R_3
A	1	2	1	2	2	4
B	2	0	1	2	1	3
C	2	2	1	3	4	1
Total_alloc	5	4	3			

Available = Total – Total_alloc
= [5, 5, 5] – [5, 4, 3]
= [0, 1, 2]

So, the total allocated resources(total_alloc) are [5, 4, 3]. Therefore, the **Available (**the resources that are currently available**)** resources are

Available = [0, 1, 2]

Now, we will make the **Need** Matrix for the system according to the given conditions. As we know, **Need(i) = Max(i) – Allocation(i)**, so the resultant Need matrix will be as follows:

Process	Need		
	R_1	R_2	R_3
A	1	0	3
B	0	1	2
C	1	2	0

Now, we will apply the safety algorithm to check that if the given state is a safe state or not.

1. Work = Available = [0, 1, 2].
2. Also Finish[i] = false, for i = 0, 1, 2, are set as false as none of these processes have been executed.
3. Now, we check that **Need[i] ≤ Work**. By seeing the above **Need** matrix we can tell that only B[0, 1, 2] process can be executed. So, process B(**i = 1**) is allocated the resources and it completes its execution. After completing the execution, it frees up the resources.
4. Again, Work = Work + Available i.e. Work = [0, 1, 2] + [2, 0, 1] = [2, 1, 3] and Finish[1] = true.
5. Now, as we have more instances of resources available we will check that if any other process resource needs can be satisfied. With the currently available

resources [2, 1, 3], we can see that only process A [1, 2, 1] can be executed. So, process A(**i = 0**) is allocated the resources and it completes its execution. After completing the execution, it frees up the resources.

6. Again, Work = Work + Available i.e. Work = [2, 1, 3] + [1, 2, 1] = [3, 3, 4] and Finish[0] = true.
7. Now, as we have more instances of resources available we will check that if the remaining last process resource requirement can be satisfied. With the currently available resources [3, 3, 4], we can see that process C[2, 2, 1] can be executed. So, process C(**i = 2**) is allocated the resources and it completes its execution. After completing the execution, it frees up the resources.
8. Finally, Work = Work + Available i.e. Work = [3, 3, 4] + [2, 2, 1] = [5, 5, 5] and Finish[2] = true.
9. Finally, all the resources are free and there exists a safe sequence B, A, C in which all the processes can be executed. So the system is in a safe state and deadlock will not occur.

This is how Banker's Algorithm is used to check if the system is in a safe state or not.

Example 2: Consider the total amount of resources R_1, R_2 and R_3 are 9, 3 and 6 units. In the current state, allocation have been made to the four processes; leaving 1 unit of resource 2 and 1 unit of resource 3 available. Now, we will find whether the system is in safe state.

Solution: We need to find whether the difference between the current allocation and maximum requirement. In any process be met with the available resources clearly, this is not possible for P_1, which has only 1 unit of R_1 and require 2 more units of R_1, 2 units of R_2 and 2 units of R_3.

However, by assigning one unit of R_3 to process P_2, P_2 has its maximum required resources allocated and can run to completion. Let us assume that this is accomplished.

	R_1	R_2	R_3
P_1	3	2	2
P_2	6	1	3
P_3	3	1	4
P_4	4	2	2

Need Matrix

	R_1	R_2	R_3
P_1	1	0	0
P_2	6	1	2
P_3	2	1	1
P_4	0	0	2

Allocation Matrix

R_1	R_2	R_3
9	3	6

Resource Vector

R_1	R_2	R_3
0	1	1

Available Vector

(a) Initial State

Solution:

	R_1	R_2	R_3
P_1	3	2	2
P_2	0	0	0
P_3	3	1	4
P_4	4	2	2

Need Matrix

	R_1	R_2	R_3
P_1	1	0	0
P_2	0	0	0
P_3	2	1	1
P_4	0	0	2

Allocation Matrix

R_1	R_2	R_3
6	2	3

Available Vector

(b) P_2 runs to completion

	R_1	R_2	R_3
P_1	0	0	0
P_2	0	0	0
P_3	3	1	4
P_4	4	2	2

Need Matrix

	R_1	R_2	R_3
P_1	0	0	0
P_2	0	0	0
P_3	2	1	1
P_4	0	0	2

Allocation Matrix

R_1	R_2	R_3
7	2	3

Available Vector

(c) P_1 runs to completion

	R_1	R_2	R_3
P_1	0	0	0
P_2	0	0	0
P_3	0	0	0
P_4	4	2	2

Need Matrix

	R_1	R_2	R_3
P_1	0	0	0
P_2	0	0	0
P_3	0	0	0
P_4	0	0	2

Allocation Matrix

R_1	R_2	R_3
9	3	4

Available Vector

(d) P_3 runs to completion

Fig. 6.8: Determination of a Safe State

- When P_2 completes, its resources can be returned to the pool of available resources. The resulting state is shown in Fig. 6.8 (b). Now, we can ask again if any of remaining processes can be completed.
- In this case, each of the remaining processes could be completed. Suppose we choose P_1, allocate the required resources, complete P_1, and return all P_1's resources to the available pool.
- We are left in the state shown in Fig. 6.8 (c). Next, we can complete P_3, resulting in state (Fig. 6.8 (d)).
- Finally, we can complete P_4. At this point, all of the processes have been run to completion. Thus, the state defined by Fig. 6.8 (a) is a safe state, and $<P_2, P_1, P_3, P_4>$ is the safe sequence.

Example 3: The operating system contains 3 resources. The number of instances of each resource type are 7, 7, 10. The current resource allocation state is as shown as follows:

Process	Current Allocation			Maximum Need		
	R_1	R_2	R_3	R_1	R_2	R_3
P_1	2	2	3	3	6	8
P_2	2	0	3	4	4	3
P_3	1	2	4	3	3	4

Resources	Instances
R_1	7
R_2	7
R_3	10

(i) Is the current allocation in a safe state?

Solution:

(i) First we calculate available matrix.

Process	Current Allocation			Max			Available		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_1	2	2	3	3	6	8	2	3	0
P_2	2	0	3	4	4	3			
P_3	1	2	4	3	3	4			

Then, the Need matrix will be,

Process	Need		
	R_1	R_2	R_3
P_1	1	4	5
P_2	2	4	0
P_3	2	1	0

- Then, find safe sequence. Currently (2, 3, 0) resources available and need of resources of each process is given in Need matrix. So we can fulfill the requirement of P_3 process first.
- After finishing execution, P_3 will return all the resources to the system so new available resources will resources will be (3, 5, 4).
- Now the request of P_2 process can be satisfied after P_2 finishes it will also return the resources allocated to it.
- Now available resources are (5, 5, 7), then the requirement of P_1 can be easily satisfied therefore, the safe sequence is $<P_3, P_2, P_1>$ and the system is in safe state.

Example 4: Consider the following snapshot of the system.

Process	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P_0	0	0	1	2	0	0	1	2	1	5	2	0
P_1	1	0	0	0	1	7	5	0				
P_2	1	3	5	4	2	3	5	6				
P_3	0	6	3	2	0	6	5	2				
P_4	0	0	1	4	0	6	5	6				

- Is the system safe? Justify?
- If Yes, give safe sequence.

Solution:

(i) Total number of instances,

A	B	C	D
3	14	12	12

(ii) Then Need matrix is,

	A	B	C	D
P_0	0	0	0	0
P_1	0	7	5	0
P_2	1	0	0	2
P_3	0	0	2	0
P_4	0	6	4	2

So the system is in safe state and safe sequence is, $<P_0, P_2, P_3, P_4, P_1>$

Example 5: Consider a system with five processes P_0 through P_4 and three resource types A, B, C. Resource type A has 7 instances, resource type B has 2 instances, and resource type C has 6 instances. Suppose that at time T_0, we have the following resource allocation state.

Process	Allocation			Request			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	0	0	0	0	0	0
P_1	2	0	0	2	0	2			
P_2	3	0	3	0	0	0			
P_3	2	1	1	1	0	0			
P_4	0	0	2	0	0	2			

Solution:

- We claim that the system is not in deadlocked state. Indeed if we execute our algorithm, we will find that sequence $<P_0, P_2, P_3, P_1, P_4>$ will result in

  Finish[i] = true for all i.
- Suppose now that process P_2 makes one additional request for an instance of type C.
- The request matrix is modified as follows:

Process	Request		
	A	B	C
P_0	0	0	0
P_1	2	0	2
P_2	0	0	1
P_3	1	0	0
P_4	0	0	2

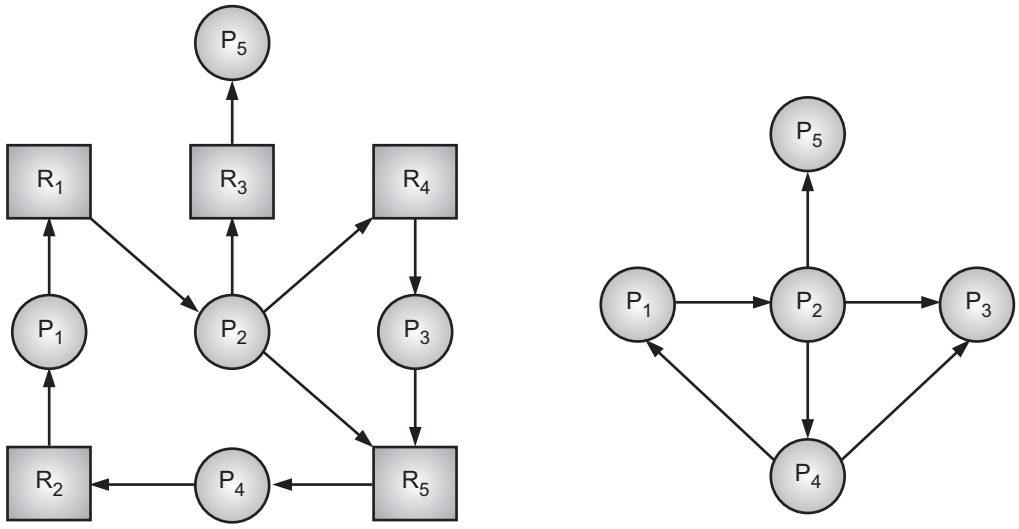
- We claim that the system is now deadlocked. Although we can reclaim the resources held by process P_0 the number of available resources is not sufficient to fulfill the request of other processes. Thus, a deadlock exists, consisting of processes P_1, P_2, P_3 and P_4.

6.4.3 Deadlock Detection

- If a system does not employ either deadlock prevention or a deadlock avoidance algorithm, then a deadlock situation may occur.
- In this environment, the system must provide:
  - An algorithm that examines the state of the system to determine whether a deadlock has occurred.
  - An algorithm to recover from the deadlock.
- In the following sections, we explain on these two requirements as they relate to systems with only a single instance of each resource type, as well as systems with several instances of each resource type.

6.4.3.1 Single Instance of each Resource Type [S-19]

- If each resource category has a single instance, then we can use a variation of the resource-allocation graph known as a wait-for graph.
- A wait-for graph can be obtained from a resource-allocation graph by eliminating the resources and collapsing the associated edges, as shown in the figure below.
- An arc from P_i to P_j in a wait-for graph indicates that process P_i is waiting for a resource that process P_j is currently holding.



(a) Resource Allocation Graph **(b) Corresponding Wait for Graph**

Fig. 6.9

- As before, cycles in the wait-for graph indicate deadlocks.
- This algorithm must maintain the wait-for graph, and periodically search it for cycles.

6.4.3.2 Several Instances of a Resource Type

- The algorithm employs several time-varying data structures that are similar to those used in the banker's algorithm.
  - **Available:** A vector of length m indicates the number of available resources of each type.
  - **Allocation:** An n x m matrix defines the number of resources of each type currently allocated to each process.
  - **Request:** An n x m matrix indicates the current request of each process. If Request $[i_j] = k$, then process Pi is requesting k more instances of resource type R_j.

Detection Algorithm:

- The detection algorithm described here simply investigates every possible allocation sequence for the processes that remain to be completed.

Step 1 : Let Work and Finish be vectors of length m and n, respectively Initialize:

(a) Work = Available

(b) For i = 1, 2, ..., n, if Allocation i $\neq$ 0

then Finish[i] = false; otherwise, Finish[i] = true

Step 2 : Find an index i such that both:

(a) Finish[i] == false

(b) Request i $\leq$ Work

If no such i exists, go to step 4

Step 3 : Work = Work + Allocation i

Finish[i] = true, go to step 2

Step 4 : If Finish[i] == false, for some i, $1 \leq i \leq n$,

Then the system is in deadlock state.

Moreover, if Finish[i] == false, then P_i is deadlocked

Example 6: To illustrate this algorithm, we consider a system with five processes P_0 through P_4 and three resource types A, B, C. Resource type A has 7 instances, resource type B has 2 instances, and resource type C has 6 instances. Suppose that at time T_0, we have the following resource allocation state.

Process	Allocation			Request			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	0	0	0	0	0	0
P_1	2	0	0	2	0	2			
P_2	3	0	3	0	0	0			
P_3	2	1	1	1	0	0			
P_4	0	0	2	0	0	2			

Solution:

- We claim that the system is not in deadlocked state. Indeed if we execute our algorithm, we will find that sequence <P_0, P_2, P_3, P_1, P_4> will result in, Finish[i] = true for all i.
- Suppose now that process P_2 makes one additional request for an instance of type C.
- The request matrix is modified as follows:

Process	Request		
	A	B	C
P_0	0	0	0
P_1	2	0	2
P_2	0	0	1
P_3	1	0	0
P_4	0	0	2

- We claim that the system is now deadlocked. Although we can reclaim the resources held by process P_0, but the number of available resources is not sufficient to fulfill the request of other processes.
- Thus, a deadlock exists, consisting of processes P_1, P_2, P_3 and P_4.

Example 7: Apply the deadlock detection algorithm to the following data and show the results.

$$\text{Available} = (2, 1, 0, 0)$$

$$\text{Request} = \begin{matrix} P_1 \\ P_2 \\ P_3 \end{matrix}\begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix}$$

$$\text{Allocation} = \begin{matrix} P_1 \\ P_2 \\ P_3 \end{matrix}\begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

Solution:

Step 1: $\text{Work} = \text{Available}$

$= (2, 1, 0, 0)$

Step 2: $\text{Request} \leq \text{Work}$

In this Step 2 process P_3 fulfill the condition.

$(2, 1, 0, 0) \leq (2, 1, 0, 0)$

Step 3:

$\text{Work} = \text{Work} + \text{Allocation}$

$\text{Work} = (2, 1, 0, 0) + (0, 1, 2, 0)$

$= (2, 2, 2, 0)$

- In this way, we can finish the execution of P_3 Now we have (2, 2, 2, 0) available resources.
- We have to repeat the procedure and finishes the execution of processes and here we will find that <P_3, P_2, P_1> will result in finish[i] = true for all i.

6.4.4 Recovery from Deadlock [S-18]

- There are three basic approaches to recovery from deadlock:
  1. Inform the system operator, and allow him/her to take manual intervention.
  2. Terminate one or more processes involved in the deadlock
  3. Preempt resources.

6.4.4.1 Process Termination

- To eliminate deadlocks by aborting a process we use one of the two methods. In both methods, the system retains all resources allocated to the terminated processes.
  - **Abort all deadlocked processes:** This method clearly will break the deadlock cycle, but at a great expense, these processes may have computed for a long time, and the result of these partial computations must be discarded and probably recomputed later.
  - **Abort one process at a time until the deadlock cycle is eliminated.**
- This method incurs considerable overhead, since, after each process is aborted, a deadlock-detection algorithm must be invoked to determine whether any processes are still deadlocked.
- Aborting a process may not be easy. If the process was in the midst of updating of file, terminating it will leave that file in an incorrect state. Similarly, if the process was in the middle of printing data on the printer, the system must reset the printer to a correct state before printing the next job.
- If the partial termination method is used, then given a set of deadlocked processes, we must determine which process should be terminated in an attempt to break the deadlock.
- This determination is a policy decision, similar to CPU scheduling problems. The question is basically an economic one we should abort those processes termination of which will incur the minimum cost. Unfortunately, the term minimum cost is not a precise one.
- Many factors may determine which process is chosen including,
  1. What the priority of the process is?
  2. How long the process has computed, and how much longer the process will compute before completing its designated task?
  3. How many and what type of resources the process has used (Example whether the resources are simple to preempt)?
  4. How many more resources the process needs in order to complete?
  5. How many processes will need to be terminated?
  6. Whether the process is interactive or batch?

6.4.4.2 Resource Preemption [S-18]

- When preempting resources to relieve deadlock, there are three important issues to be addressed:
  1. **Selecting a victim:** Deciding which resources to preempt from which processes involves many of the same decision criteria outlined above.
  2. **Rollback:** Ideally one would like to roll back a preempted process to a safe state prior to the point at which that resource was originally allocated to the process. Unfortunately it can be difficult or impossible to determine what such a safe state is, and so the only safe rollback is to roll back all the way back to the beginning. (i.e. abort the process and make it start over.)
  3. **Starvation:** How do you guarantee that a process will not starve because its resources are constantly being preempted? One option would be to use a priority system, and increase the priority of a process every time its resources get preempted. Eventually it should get a high enough priority that it won't get preempted any more.

Solved Examples of Previous Exams

Example 8: Consider the fine processes P_0, P_1, P_2, P_3, P_4 and three resources R_1, R_2, R_3 resource, type R_1 has 10 instances, R_2 has 5 instances and R_3 has 7 instances. Allocation and max matrix is given below:

Process	Allocation			MAX		
	R_1	R_2	R_3	R_1	R_2	R_3
P_0	0	1	0	7	5	3
P_1	2	0	0	3	2	2
P_2	3	0	2	9	0	2
P_3	2	1	1	2	2	2
P_4	0	0	2	4	3	3

Answer the following questions using Banker's algorithm.

(i) What is the content of Need Matrix?

(ii) Is the system in a safe sequence? If yes, give the safe sequence.

Solution: (i) Need [i, j] = Max [i, j] – Allocation [i, j]

So, the content of Need Matrix is:

Process	Need Matrix		
	R_1	R_2	R_3
P_0	7	4	3
P_1	1	2	2
P_2	6	0	0
P_3	0	1	1
P_4	4	3	1

(ii) Then, find safe sequence. Currently (3, 3, 2) resources available and need of resources of each process is given in Need matrix.

Need P_0 <= Work (Available Resources)

So Need P_0 (7, 4, 3) <= (3, 3, 2)

Which is not equal so available resources will not be allocated to P_0.

So Safe sequence = {}

Need P_1 <= Work (Available Resources)

So Need P_1(1, 2, 2) <= (3, 3, 2)

So resources will be allocated to p_1. After successful execution of P_1. Allocated resources of P_1 will be added to available resources.

Available Resources = Allocation of P_1(2, 0, 0)
+ Available Resources (3, 3, 2) = (5, 3, 2)

Safe Sequence = {P_1}

Again Need P_2 <= Work (Available Resources)

So Need P_2 (6, 0, 0) <= (5, 3, 2)

Resources which are required by P_2 are greater than available resources.

So resources will not be given to P_2 process.

Again Need P_3(0, 1, 1) <= (5, 3, 2)

So resources will be allocated to P_3. After completion of P_3 the allocated resources of P_3 will be added to available resources.

So available resources = (7, 4, 3)

Safe Sequence = {P_1, P_3}

Again Need P_4(4, 3, 1) <= (7, 4, 3)

So resources will be allocated to P_4. After completion of P_4 the allocated resources of P_4 will be added to available resources.

So available resources = (7, 4, 5)

Safe Sequence = {P_1, P_3, P_4}

Again Need P_0(7, 4, 3) <= (7, 4, 5)

So resources will be allocated to P_0. After completion of P_0 the allocated resources of P_0 will be added to available resources.

So Available resources = (7, 5, 5)

Safe Sequence = {P_1, P_3, P_4, P_0}

Again Need P_2(6, 0, 0) <= (7, 5, 5)

So resources will be allocated to P_2. After completion of P_2 the allocated resources of P_2 will be added to available resources.

So Available resources = (10, 5, 7)

Safe Sequence = {P_1, P_3, P_4, P_0, P_2}

So if we follow this safe sequence the system will be in safe state and there will be no deadlock.

Example 9: Consider the five processes P_0, P_1, P_2, P_3, P_4 and three resources R_1, R_2, R_3 resource. Type R_1 has 8 instances, R_2 has 4 instances and R_3 has 9 instances. Allocation and max matrix is given below:

Process	Allocation			Max.		
	R_1	R_2	R_3	R_1	R_2	R_3
P_0	1	0	2	5	4	2
P_1	2	1	1	3	2	2
P_2	2	0	3	8	0	4
P_3	1	1	2	2	2	2
P_4	0	1	0	5	2	3

Answer the following questions using Banker's algorithm:

(i) What is the content of need matrix?

(iii) Is the system in a safe sequence? If yes, give the safe sequence.

Solution: First we calculate available matrix.

Process	Current Allocation			Max.					
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_0	1	0	2	5	4	2	2	1	1
P_1	2	1	1	3	2	2			
P_2	2	0	3	8	0	4			
P_3	1	1	2	2	2	2			
P_4	0	1	0	5	2	3			

(i) Then, the Need matrix will be,

Process	Need Matrix		
	R_1	R_2	R_3
P_0	4	4	0
P_1	1	1	1
P_2	6	0	1
P_3	1	1	0
P_4	5	1	3

(ii) Then, find safe sequence. Currently (2, 1, 1) resources available and need of resources of each process is given in Need matrix.

Need P_0 <= Work

So Need P_0 (4, 4, 0) <= (2, 2, 1)

Which are not equal so available resources will not be allocated to P_0.

Need P_1 <= Work

So Need P_1(1, 1, 1) <= (2, 2, 1)

So resources will be allocated to P_1. After successful execution of P_1, allocated resources of P_1 will be added to available resources.

Available Resources = Allocation of P_1(2, 1, 1) + Available Resources (2,1,1)

= (4, 2, 2)

Safe Sequence = $\{P_1\}$

Again Need P_2 <= Work

So Need P_2 (6, 0, 1) <= (4, 2, 2)

Which are not less so resources will not be allocated to P_2.

Need P_3(1, 1, 0) <= (4, 4, 2)

So resources will be allocated to P_3. After completion of P_3 the allocated resources of P_3 will be added to available resources.

So available resources = (5, 3, 4)

Safe Sequence = $\{P_1, P_3\}$

Again Need P_4(5, 1, 3) <= (5, 3, 4)

So resources will be allocated to P_4. After completion of P_4 the allocated resources of P_4 will be added to available resources.

So available resources = (5, 4, 4)

Safe Sequence = $\{P_1, P_3, P_4\}$

Again Need P_0(4, 4, 0) <= (5, 4, 4)

So resources will be allocated to P_0. After completion of P_0 the allocated resources of P_0 will be added to available resources.

So available resources = (6, 4, 6)

Safe Sequence = $\{P_1, P_3, P_4, P_0\}$

Again Need P_2(6, 0, 1) <= (6, 4, 6)

So resources will be allocated to P_2. After completion of P_2 the allocated resources of P_2 will be added to available resources.

So available resources = (8, 4, 9)

Safe Sequence = $\{P_1, P_3, P_4, P_0, P_2\}$

So if we follow this safe sequence the system will be in safe state and there will be no deadlock.

Summary

- In an operating system, deadlock state happens when two or more processes are waiting for the same event to happen which never happen, then we can say that those processes are involved in the deadlock.
- A deadlock can occur only if four necessary conditions hold simultaneously in the system: Mutual Exclusion, Hold and Wait, No Preemption, and Circular Wait.
- Methods for handling deadlock: Deadlock prevention, Deadlock avoidance, Deadlock detection and recovery.

- A method for avoiding deadlocks, rather than preventing them, requires that the operating system have a priori information about how each process will utilize system resources.
- The banker's algorithm, for example, requires a priori information about the maximum number of each resource class that each process may request. Using this information, we can define a deadlock-avoidance algorithm.
- If a system does not employ a protocol to ensure that deadlocks will never occur, then a detection-and-recovery scheme may be employed.
- Where preemption is used to deal with deadlocks, three issues must be addressed: Selecting a victim, Rollback, and Starvation.

Check Your Understanding

1. What is a reusable resource?
   (a) that can be used by one process at a time and is not depleted by that use.
   (b) that can be used by more than one process at a time.
   (c) that can be shared between various threads.
   (d) none of the mentioned.
2. Which of the following condition is required for a deadlock to be possible?
   (a) mutual exclusion
   (b) a process may hold allocated resources while awaiting assignment of other resources.
   (c) no resource can be forcibly removed from a process holding it.
   (d) all of the mentioned.
3. A system is in the safe state if _____.
   (a) the system can allocate resources to each process in some order and still avoid a deadlock.
   (b) there exist a safe sequence.
   (c) all of the mentioned
   (d) none of the mentioned
4. The circular wait condition can be prevented by _____.
   (a) defining a linear ordering of resource types
   (b) using thread
   (c) using pipes
   (d) all of the mentioned
5. Which one of the following is the deadlock avoidance algorithm?
   (a) banker's algorithm (b) round-robin algorithm
   (c) elevator algorithm (d) Karn's algorithm
6. What is the drawback of banker's algorithm?
   (a) in advance processes rarely know how much resource they will need.
   (b) the number of processes changes as time progresses.
   (c) resource once available can disappear.
   (d) all of the mentioned

7. For an effective operating system, when to check for deadlock?
   (a) every time a resource request is made.
   (b) at fixed time intervals.
   (c) every time a resource request is made at fixed time intervals.
   (d) none of the mentioned
8. A problem encountered in multitasking when a process is perpetually denied necessary resources is called _____.
   (a) deadlock (b) starvation
   (c) inversion (d) aging
9. Which one of the following is a visual (mathematical) way to determine the deadlock occurrence?
   (a) resource allocation graph (b) starvation graph
   (c) inversion graph (d) none of the mentioned
10. To avoid deadlock _____.
   (a) there must be a fixed number of resources to allocate.
   (b) resource allocation must be done only once.
   (c) all deadlocked processes must be aborted.
   (d) inversion technique can be used.

Answers

1. (a)	2. (d)	3. (a)	4. (a)	5. (a)	6. (d)	7. (c)	8. (b)	9. (a)	10. (a)

Practice Questions

Q.I Answer the following questions in short:

1. What is meant by deadlock?
2. What are the principles of deadlock?
3. What are the necessary conditions for deadlock occurs.
4. List the methods for deadlock handling.
5. What is safe sequence?
6. What is claim age?
7. What is mean by request edge?

Q.II Answer the following questions:

1. Describe safe state in detail.
2. Describe banker's algorithm with suitable example.
3. Explain the term resource allocation graph with example.
4. Explain working of Banker's algorithm for deadlock avoidance with suitable example.
5. Write and explain deadlock detection algorithm with suitable example.
6. Describe deadlock characterization in detail.
7. Explain resource allocation graph with suitable example.
8. Enlist various deadlock handling methods in short.
9. Write short note on Recovery from deadlock.

10. For the following resource allocation graph, show that four necessary conditions for deadlock indeed hold.

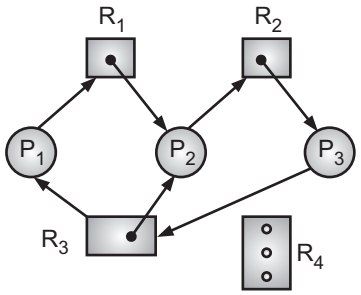


Fig. 6.10

11. Consider the system with 5 processes P = {P_0, P_1, P_2, P_3, P_4} and four resource types {A, B, C, D}. There are 3 instances of type A, 10 instances of type B, 15 instances of type C and 7 instances of type D. The allocation and maximum demand matrix are as follows:

	Allocation				Max				
	A	**B**	**C**	**D**		**A**	**B**	**C**	**D**
P_0	0	1	2	1	P_0	0	8	4	4
P_1	0	1	2	1	P_1	0	6	5	2
P_2	1	0	0	0	P_2	1	6	4	1
P_3	1	3	5	3	P_3	2	3	7	5
P_4	0	0	4	1	P_4	0	5	5	7

Answer the following question using Banker's Algorithm:

(i) Is the system in a Safe State?

(ii) If a request from process P_4 arrives for (0, 2, 0, 2) can it be granted?

12. Consider the system with 5 processes P = {P_0, P_1, P_2, P_3, P_4} and four resources type {A, B, C, D}. There are 3 instances of type A, 14 instances of type B, 12 instances of type C and 12 instance of type D. The allocation and maximum demand matrix are as follows:

	Allocation				Max				
	A	**B**	**C**	**D**		**A**	**B**	**C**	**D**
P_0	0	6	3	2	P_0	0	6	5	2
P_1	0	0	1	2	P_1	0	0	1	2
P_2	1	0	0	0	P_2	1	7	5	0
P_3	1	3	5	4	P_3	2	3	5	6
P_4	0	0	1	4	P_4	0	0	5	6

Answer the following question using Bankers Algorithm:

(i) Is the system in a Safe State?

(ii) If a request from process P_4 arrives for (0, 0, 4, 1) can be the request immediately granted.

Q.III Define terms:

1. Process termination.
2. Resource preemption.
3. Assignment edge in RAG.
4. Request edge in RAG.
5. Deadlock avoidance.

Previous Exam Questions

Summer 2018

1. Define Rollback. **[2 M]**

Ans. Refer to Section 6.4.4.2.

2. Explain different methods for recovery from a deadlock. **[4 M]**

Ans. Refer to Section 6.4.4.

3. Explain Resource Allocation graph in detail. **[4 M]**

Ans. Refer to Section 6.4.2.2.

Winter 2018

1. What is meant by Deadlock? **[2 M]**

Ans. Refer to Section 6.1.

2. Explain Resource Allocation Graph in detail. **[4 M]**

Ans. Refer to Section 6.4.2.2.

3. List and explain necessary conditions for Deadlock occurrence. **[4 M]**

Ans. Refer to Section 6.3.

Summer 2019

1. What is the role of MAX and NEED array used in Banker's Algorithm? **[2 M]**

Ans. Refer to Section 6.4.2.3.

2. Wait for graph is used for deadlock avoidance in the system.
True/False. Justify. **[2 M]**

Ans. Refer to Section 6.4.3.1.

3. List and explain necessary conditions for deadlock. **[4 M]**

Ans. Refer to Section 6.3.

■■■

7...

Memory Management

Objectives...

- To learn basic concepts Memory Management.
- To get information of Swapping, Paging and Segmentation.
- To study about Virtual Memory Management.

7.1 BACKGROUND

- Memory consists of a large array of words or bytes, each with its own address. Both the CPU and I/O system interact with memory. Interaction is achieved through a sequence of reads or writes to specific memory addresses.
- The CPU fetches instructions from memory according to the value of the program counter. These instructions may cause additional loading from and storing to specific memory addresses.
- Memory management is the process of controlling and coordinating computer memory. This process assigning portions called blocks to various running programs to optimize overall system performance.

Major activities of Memory Management:

- Keep track of which parts of memory are currently being used and by whom.
- Decide which processes are to be loaded into memory when memory space becomes available.
- Allocate and deallocate memory space as needed.

7.1.1 Basic Hardware

- Memory accesses to registers are very fast, generally one clock tick. A CPU may be able to execute more than one machine instruction per clock tick.
- A pair of base and limit registers define the logical address space. Every memory access made by a user process is checked against these two registers.

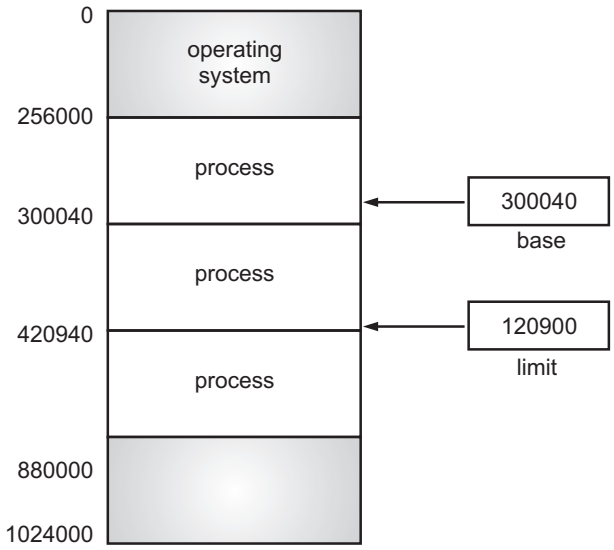


Fig. 7.1: Logical address space with Base and Limit Registers

- CPU must check every memory access generated in user mode to be sure it is between base and limit for that user, otherwise a fatal error is generated.

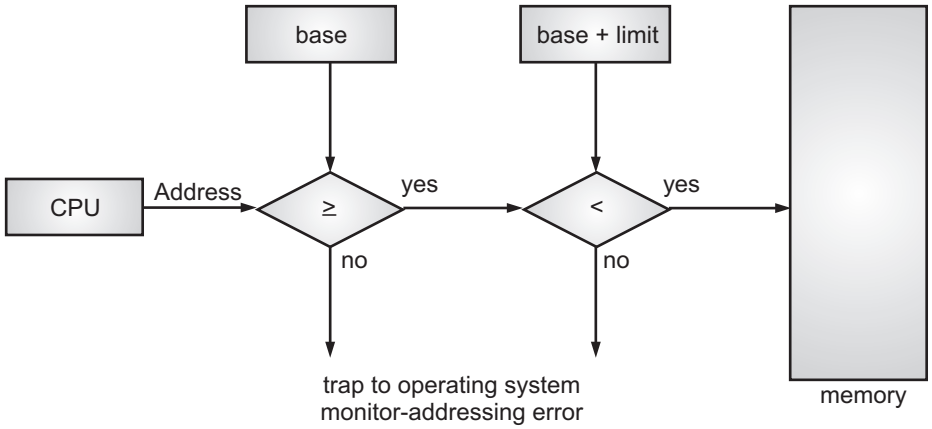


Fig. 7.2: Protection of Hardware Address

7.1.2 Address Binding [W-18]

- Programs are stored on the secondary storage disks as binary executable files. When the programs are to be executed they are brought in to the main memory and placed within a process.
- The collection of processes on the disk waiting to enter the main memory forms the Input Queue.
- One of the processes which are to be executed is fetched from the queue and placed in the main memory. During the execution it fetches instruction and data from main memory.
- After the process terminates it returns back the memory space. During execution the process will go through different steps and in each step the address is represented in different ways.

- In source program the address is symbolic. The compiler converts the symbolic address to re-locatable address. The loader will convert this re-locatable address to absolute address.
- Each binding is a mapping from one address space to another. These absolute addresses loaded into memory to be executed.
- The binding of instructions and data to memory addresses can be done at any step along the following way:
  - **(a) Compile time:** If we know whether the process resides in memory then absolute code can be generated. If the static address changes then it is necessary to re-compile the code from the beginning.
  - **(b) Load time:** If the compiler doesn't know whether the process resides in memory then it generates the re-locatable code. In this the binding is delayed until the load time.
  - **(c) Execution time:** If the process is moved during its execution from one memory segment to another then the binding is delayed until run time. Special hardware is used for this. Most of the general purpose operating system uses this method.

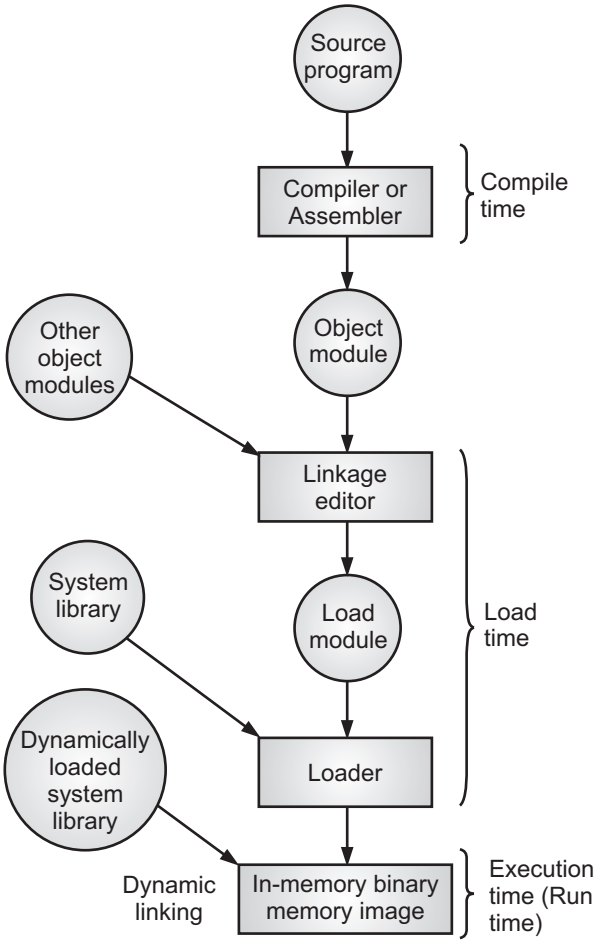


Fig. 7.3: Multistep processing of a user program

7.1.3 Logical versus Physical Address Space [W-18]

- **Logical Address** is generated by CPU while a program is running. The logical address is virtual address as it does not exist physically, therefore, it is also known as Virtual Address.
- This address is used as a reference to access the physical memory location by CPU. The term Logical Address Space is used for the set of all logical addresses generated by a program's perspective.
- The hardware device called Memory-Management Unit is used for mapping logical address to its corresponding physical address.
- **Physical Address** identifies a physical location of required data in a memory. The user never directly deals with the physical address but can access by its corresponding logical address. The user program generates the logical address and thinks that the program is running in this logical address but the program needs physical memory for its execution, therefore, the logical address must be mapped to the physical address by MMU before they are used.
- The term Physical Address Space is used for all physical addresses corresponding to the logical addresses in a Logical address space.

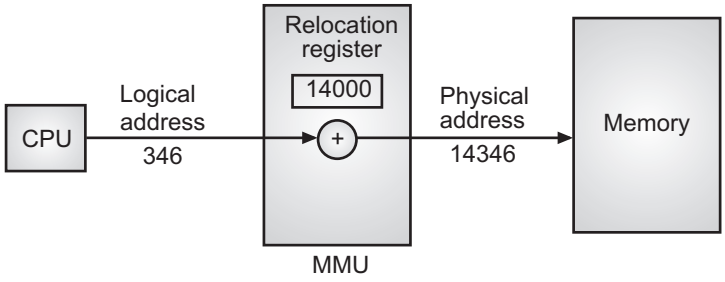


Fig. 7.4: Dynamic relocation using a Relocation Register

Differences between Logical and Physical Address in Operating System:

1. The basic difference between logical and physical address is that logical address is generated by CPU in perspective of a program whereas the physical address is a location that exists in the memory unit.
2. Logical Address Space is the set of all logical addresses generated by CPU for a program whereas the set of all physical address mapped to corresponding logical addresses is called Physical Address Space.
3. The logical address does not exist physically in the memory whereas physical address is a location in the memory that can be accessed physically.
4. Identical logical addresses are generated by Compile-time and Load time address binding methods whereas they differs from each other in run-time address binding method. Please refer this for details.
5. The logical address is generated by the CPU while the program is running whereas the physical address is computed by the Memory Management Unit (MMU).

7.1.4 Dynamic Loading

- For a process to be executed it should be loaded in to the physical memory. The size of the process is limited to the size of the physical memory. Dynamic loading is used to obtain better memory utilization.
- In dynamic loading, the routine or procedure will not be loaded until it is called. Whenever a routine is called, the calling routine first checks whether the called routine is already loaded or not.
- If it is not loaded it cause the loader to load the desired program in to the memory and updates the programs address table to indicate the change. Then the control is passed to newly called routine.

Advantages:

1. It gives better memory utilization.
2. Unused routine is never loaded.
3. It does not need special operating system support.
4. This method is useful when large amount of codes are needed to handle in frequently occurring cases.

7.1.5 Dynamic Linking and Shared Libraries

- Some operating systems support only static linking. In static linking system language libraries are treated like any other object module and are combined by the loader into the binary program image.
- Dynamic linking, in contrast, is similar to dynamic loading. Here, though, linking, rather than loading, is postponed until execution time.
- This feature is usually used with system libraries, such as language subroutine libraries. Without this facility, each program on a system must include a copy of its language library in the executable image. This requirement wastes both disk space and main memory.
- With dynamic linking, a **stub** is included in the image for each library-routine reference.
- The stub is a small piece of code that indicates how to locate the appropriate memory resident library routine or how to load the library if the routine is not already present.
- When the stub is executed, it checks to see whether the needed routine is already in memory. If it is not, the program loads the routine into memory. Either way, the stub replaces itself with the address of the routine and executes the routine.
- Thus, the next time the particular code segment is reached, the library routine is executed directly, incurring to cost for dynamic linking.
- Under dynamic linking, all processes that use a language library execute only one copy of the library code.
- A library may be replaced by a new version, and all programs that reference the library will automatically use the new version.
- Without dynamic linking, all such programs would need to be relinked to gain access the new library, for this reason programs will not accidentally execute new, incompatible versions of libraries, version information is included in both the program and the library.

7.2 SWAPPING [S-18]

- Swapping is a memory management scheme in which any process can be temporarily swapped from main memory to secondary memory so that the main memory can be made available for other processes. It is used to improve main memory utilization. In secondary memory, the place where the swapped-out process is stored is called swap space.
- The purpose of the swapping in operating system is to access the data present in the hard disk and bring it to RAM so that the application programs can use it. The thing to remember is that swapping is used only when data is not present in RAM.
- Although the process of swapping affects the performance of the system, it helps to run larger and more than one process. This is the reason why swapping is also referred to as memory compaction.
- The concept of swapping has divided into two more concepts: Swap-in and Swap-out.
  - Swap-out is a method of removing a process from RAM and adding it to the hard disk.
  - Swap-in is a method of removing a program from a hard disk and putting it back into the main memory or RAM.

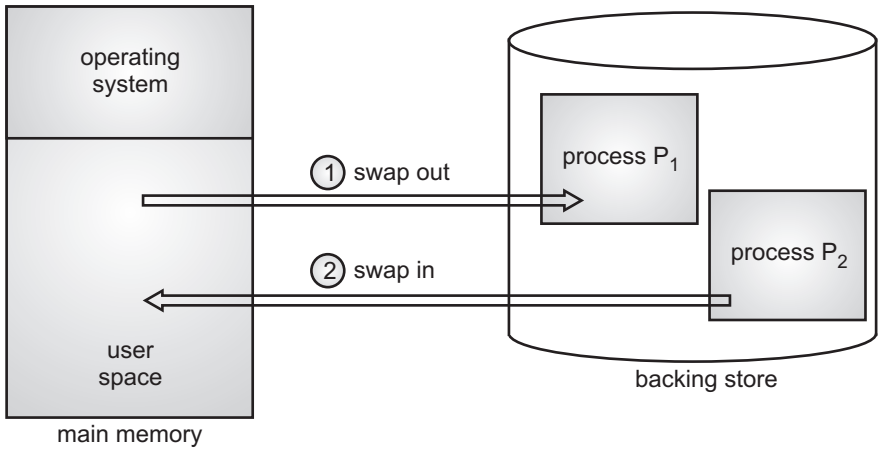


Fig. 7.5: Swapping Method

- **Backing Store:** Swapping requires a backing store. The backing store is commonly a fast drum or disk. It must be large enough to accommodate copies of all memory images for all users, and must provide direct access to these memory images. All memory images are on the backing stores, which are ready to run. Whenever, the CPU scheduler decides to execute a process, it calls the dispatcher. The dispatcher checks to see whether that process is in memory. If not, it swaps out the process currently in memory and swaps in the desired process.
- **Overlapped Swapping:** In this scheme, the objective is to overlap the swapping of one process with the execution of another. Thus, the CPU will not sit idle while swapping is going on.

Example: Suppose the user process's size is 2048KB and is a standard hard disk where swapping has a data transfer rate of 1Mbps. Now we will calculate how long it will take to transfer from main memory to secondary memory.

Solution: User process size is 2048 Kb

Data transfer rate is 1 Mbps = 1024 kbps

$$\text{Time} = \frac{\text{Process size}}{\text{Transfer rate}}$$

= 2048 / 1024

= 2 seconds

= 2000 milliseconds

Now taking swap-in and swap-out time, the process will take 4000 milliseconds.

Advantages of Swapping:

1. It helps the CPU to manage multiple processes within a single main memory.
2. It helps to create and use virtual memory.
3. Swapping allows the CPU to perform multiple tasks simultaneously. Therefore, processes do not have to wait very long before they are executed.
4. It improves the main memory utilization.

Disadvantages of Swapping:

1. If the computer system loses power, the user may lose all information related to the program in case of substantial swapping activity.
2. If the swapping algorithm is not good, the composite method can increase the number of Page Fault and decrease the overall processing performance.

7.3 CONTIGUOUS MEMORY ALLOCATION

- The main memory must accommodate both the operating system and the various user processes. Therefore we need to allocate the parts of the main memory in the most efficient way possible. This section explains one common method, contiguous memory allocation.
- The memory is usually divided into two partitions:
  1. one for the resident operating system.
  2. one for the user processes.
- We can place the operating system in either **low memory** or **high memory**.
- The major factor affecting this decision is the location of the **interrupt vector**. Since the interrupt vector is often in low memory, programmers usually place the operating system in low memory as well. Here, we discuss only the situation where the operating system resides in low memory.
- We usually want several user processes to reside in memory at the same time. We therefore need to consider how to allocate available memory to the processes that are in the input queue waiting to be brought into memory. In this **Contiguous Memory Allocation**, each process is contained in a single contiguous section of memory.

7.3.1 Memory Mapping and Protection [S-19]

- Memory protection means protecting the OS from user process and protecting process from one another. Memory protection is provided by using a re-location register, with a limit register. Re-location register contains the values of smallest physical address and limit register contains range of logical addresses (Re-location = 100040 and limit = 74600).
- The logical address must be less than the limit register; the MMU maps the logical address dynamically by adding the value in re-location register. When the CPU scheduler selects a process for execution, the dispatcher loads the re-location and limit register with correct values as a part of context switch. Since every address generated by the CPU is checked against these register we can protect the OS and other users programs and data from being modified.

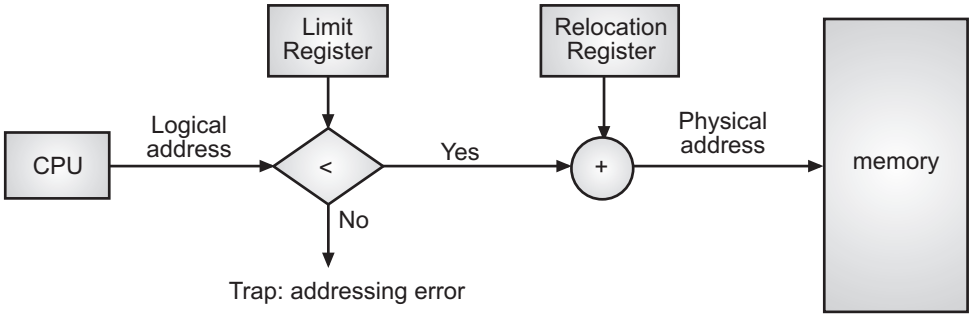


Fig. 7.6: Hardware support for relocation and limit registers

7.3.2 Memory Allocation

- One of the simplest methods for memory allocation is to divide memory in to several fixed partition. Each partition contains exactly one process. The degree of multi-programming depends on the number of partitions.
- In multiple partition method, when a partition is free, process is selected from the input queue and is loaded in to free partition of memory.
- When process terminates, the memory partition becomes available for another process. Batch OS uses the fixed size partition scheme.
- An alternate method is to keep a list of unused (free) memory blocks (holes), and to find a hole of a suitable size whenever a process needs to be loaded into memory.
- The OS keeps a table indicating which part of the memory is free and is occupied. When the process enters the system it will be loaded in to the input queue.
- The OS keeps track of the memory requirement of each process and the amount of memory available and determines which process to allocate the memory.
- When a process requests, the OS searches for large hole for this process, hole is a large block of free memory available.
- If the hole is too large it is split in to two. One part is allocated to the requesting process and other is returned to the set of holes. The set of holes are searched to determine which hole is best to allocate.

- There are three strategies to select a free hole:
  - **(i) First bit:** Allocates first hole that is big enough. This algorithm scans memory from the beginning and selects the first available block that is large enough to hold the process.
  - **(ii) Best bit:** It chooses the hole i.e., closest in size to the request. It allocates the smallest hole i.e., big enough to hold the process.
  - **(iii) Worst fit:** It allocates the largest hole to the process request. It searches for the largest hole in the entire list.

7.3.3 Fragmentation [S-18]

- Memory fragmentation can be of two types:

1. Internal Fragmentation :

- The phenomenon, in which there is wasted space internal to a partition due to the fact that the block of data loaded is smaller than the partition, is referred to as Internal Fragmentation.
- For Example: If there is a block of 50 Kb and if the process requests 40 Kb and if the block is allocated to the process then there will be 10 Kb of memory left.

2. External Fragmentation:

- External Fragmentation exists when there is enough memory space exists to satisfy the request, but it not contiguous i.e., storage is fragmented in to large number of small holes.
- External Fragmentation may be either minor or a major problem.
- One solution for over-coming external fragmentation is compaction. The goal is to move all the free memory together to form a large block. Compaction is not possible always. If the relocation is static and is done at load time then compaction is not possible. Compaction is possible if the re-location is dynamic and done at execution time.
- Another possible solution to the external fragmentation problem is to permit the logical address space of a process to be non-contiguous, thus allowing the process to be allocated physical memory whenever the latter is available.

7.4 PAGING [S-19, W-18]

- Paging is non-contiguous memory allocation technique that permits the physical address space of a process to be non-contiguous. Support for paging is handled by hardware.
- It is used to avoid external fragmentation. Paging avoids the considerable problem of fitting the varying sized memory chunks on to the backing store. When some code or date residing in main memory need to be swapped out, space must be found on backing store.

7.4.1 Basic Method

- Physical memory is broken in to fixed sized blocks called frames (f). Logical memory is broken in to blocks of same size called pages (p). When a process is to be executed its pages are loaded in to available frames from backing store. The blocking store is also divided in to fixed-sized blocks of same size as memory frames.

- The following figure shows paging hardware:

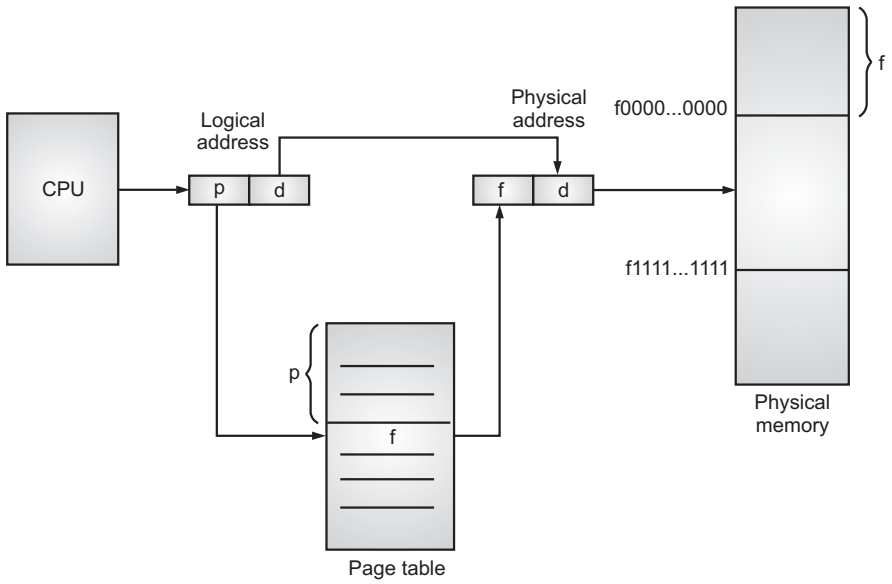


Fig. 7.7: Paging Hardware

- Logical address generated by the CPU is divided in to two parts: page number (p) and page offset (d). The page number (p) is used as index to the page table.
- The page table contains base address of each page in physical memory. This base address is combined with the page offset to define the physical memory i.e., sent to the memory unit.
- The page size is defined by the hardware. The size of a power of 2, varying between 512 bytes and 10Mb per page. If the size of logical address space is 2^m address unit and page size is 2^n, then high order m-n designates the page number and n low order bits represents page offset.

Example:

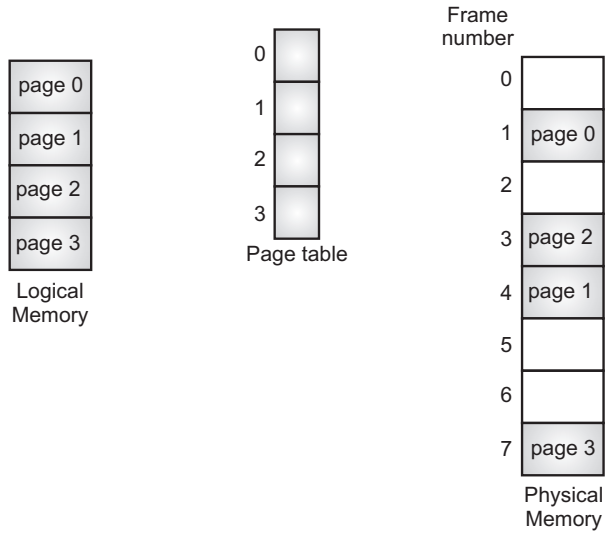


Fig. 7.8: Example for Paging

- To show how to map logical memory into physical memory; consider a page size of 4 bytes and physical memory of 32 bytes (8 pages).
- Logical address 0 is page 0 and offset 0. Page 0 is in frame 5. The logical address 0 maps to physical address 20. [(5*4) + 0].
- Logical address 3 is page 0 and offset 3 maps to physical address 23 [(5*4) + 3].
- Logical address 4 is page 1 and offset 0 and page 1 is mapped to frame 6. So logical address 4 maps to physical address 24 [(6*4) + 0].
- Logical address 13 is page 3 and offset 1 and page 3 is mapped to frame 2. So logical address 13 maps to physical address 9 [(2*4) + 1].

7.4.2 Hardware Support

- The hardware implementation of the page table can be done in several ways:

(i) Page table implemented as dedicated fast registers:

- The simplest method is that the page table is implemented as a set of dedicated registers. These registers must be built with very high speed logic for making paging address translation. Every access to memory must go through the paging map.
- The use of registers for page table is satisfactory if the page table is small.

(ii) Page table implemented in the Main Memory:

- If the page table is large then the use of registers is not visible. So the page table is kept in the main memory and a Page Table Base Register (PTBR) points to the page table. Changing the page table requires only one register which reduces the context switching type.
- The problem with this approach is the time required to access memory location. To access a location [i], first we have to index the page table using PTBR offset. It gives the frame number which is combined with the page offset to produce the actual address. Thus we need two memory accesses for a byte.
- The only solution is to use special, fast, lookup hardware cache called Translation Look- aside Buffer (TLB) or associative register. LB is built with associative register with high speed memory. Each register contains two paths a key and a value.

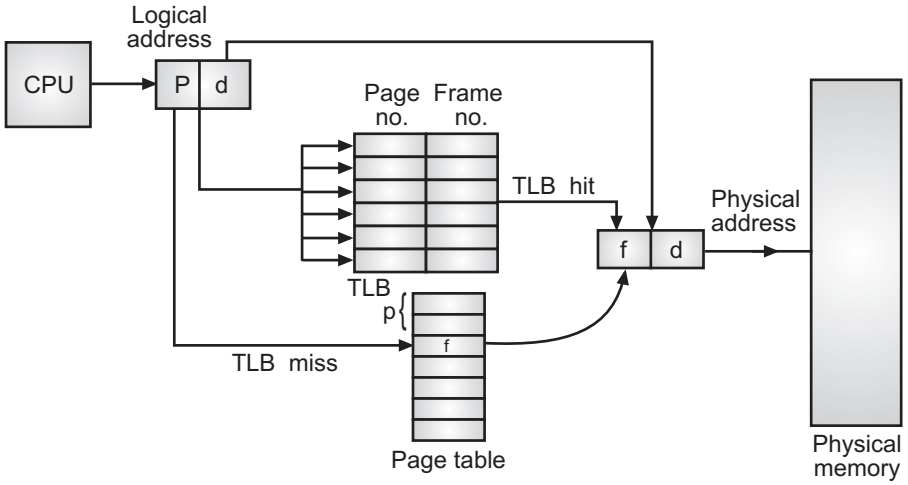


Fig. 7.9: Paging hardware with TLB

- When an associative register is presented with an item, it is compared with all the key values, if found the corresponding value field is return and searching is fast.
- TLB is used with the page table as follows:
  - TLB contains only few page table entries. When a logical address is generated by the CPU, its page number along with the frame number is added to TLB.
  - If the page number is found, its frame memory is used to access the actual memory.
  - If the page number is not in the TLB (TLB miss) the memory reference to the page table is made. When the frame number is obtained, we can use it to access the memory.
  - If the TLB is full of entries the OS must select anyone for replacement. Each time a new page table is selected the TLB must be flushed /erased to ensure that next executing process do not use wrong information.
- The percentage of time that a page number is found in the TLB is called HIT ratio.

7.4.3 Protection

- Memory protection in paged environment is done by protection bits that are associated with each frame these bits are kept in page table.
- One bit can define a page to be read-write or read-only. To find the correct frame number every reference to the memory should go through page table. At the same time physical address is computed. The protection bits can be checked to verify that no writers are made to read-only page. Any attempt to write in to read-only page causes a hardware trap to the OS.
- This approach can be used to provide protection to read-only, read-write or execute-only pages.
- One more bit is generally added to each entry in the page table: a valid-invalid bit.

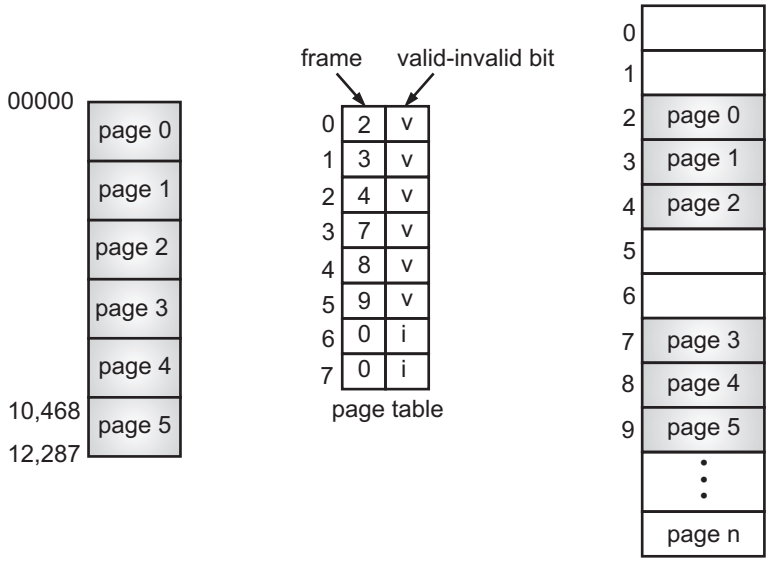


Fig. 7.10: Memory Protection using Valid-invalid bits

- A valid bit indicates that associated page is in the processes logical address space and thus it is a legal or valid page.
- If the bit is invalid, it indicates the page is not in the processes logical addressed space and illegal. Illegal addresses are trapped by using the valid-invalid bit.
- The OS sets this bit for each page to allow or disallow accesses to that page.

7.4.4 Shared Pages

- An advantage of paging is the possibility of sharing common code. This is useful in Timesharing environment.
- For Example, consider a system with 40 users, each executing a text editor. If the text editor is of 150k and data space is 50k, we need (150k+50k)*40 users = 8000k for 40 users.
- If the code is re-entrant, it can be shared. Consider the following Fig. 7.11.

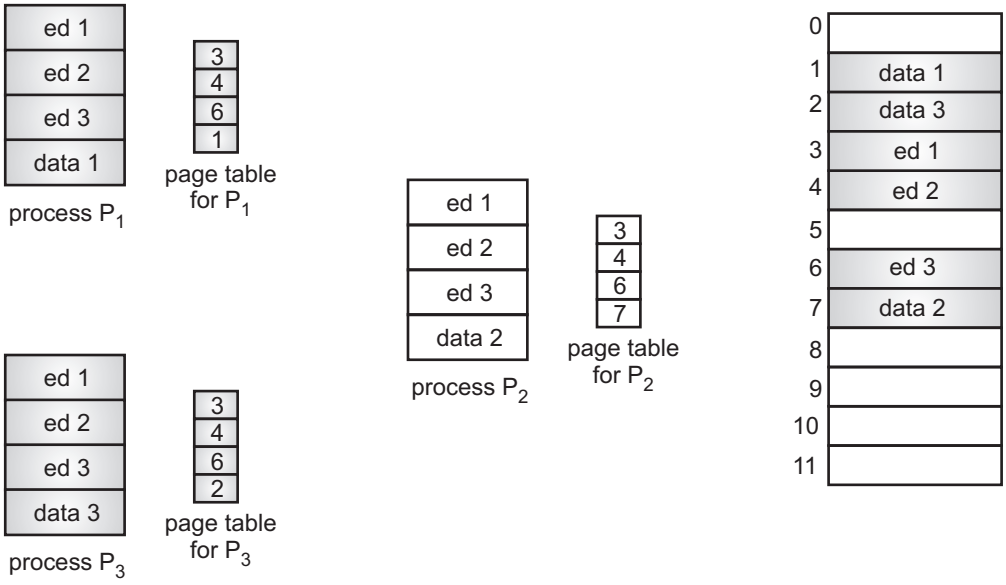


Fig. 7.11: Sharing of code in a Paging

- If the code is re-entrant then it never changes during execution. Thus two or more processes can execute same code at the same time.
- Each process has its own copy of registers and the data of two processes will vary. Only one copy of the editor is kept in physical memory.
- Each user's page table maps to same physical copy of editor but date pages are mapped to different frames. So to support 40 users we need only one copy of editor (150k) plus 40 copies of 50k of data space i.e., only 2150k instead of 8000k.

7.4.5 Advantages and Disadvantages of Paging

Advantages of Paging:

1. It permits physical address space of process to be non-contiguous and support virtual memory concept.
2. It maintains clear separation between user's view of memory and actual physical memory.

3. It does not require any support for dynamic relocation because paging itself is a form of dynamic relocation. Every logical address is bound by paging hardware to physical address.
4. It does not suffer from external fragmentation. Any free frames can be allocated to process that it needs.
5. We can share common code, so memory space is properly utilized.

Disadvantages of paging:

1. Paging suffers from internal fragmentation. Internal fragmentation especially on last page of process; average of 50% on last page.
2. Smaller frames create less fragmentation but increase number of frames and size of page table.
3. Overhead to maintain and update page table.
4. Paging hardware increases the cost of operating system.

7.5 SEGMENTATION

- Like Paging, Segmentation is another non-contiguous memory allocation technique.
- The user's view is mapped onto the physical storage. This mapping permits differentiation between logical memory and physical memory.
- Segmentation is a memory management technique which supports user's view of memory.

7.5.1 Basic Concept

- Most users do not think memory as a linear array of bytes rather the users thinks memory as a collection of variable sized segments which are dedicated to a particular use such as code, data, stack, heap etc.
- A logical address is a collection of segments. Each segment has a name and length. The address specifies both the segment name and the offset within the segments.
- The users specify address by using two quantities: a segment name and an offset. For simplicity the segments are numbered and referred by a segment number. So the logical address consists of <segment number, offset>.

7.5.2 Hardware

- Although the programmer can now refer to objects in the program by a two dimensional address, the actual physical memory is still a one dimensional sequence of bytes. Thus, we must define an implementation to map 2D user defined address in to 1D physical address. This mapping is affected by a segment table. Each entry in the segment table has a segment base and segment limit. The segment base contains the starting physical address where the segment resides and limit specifies the length of the segment.

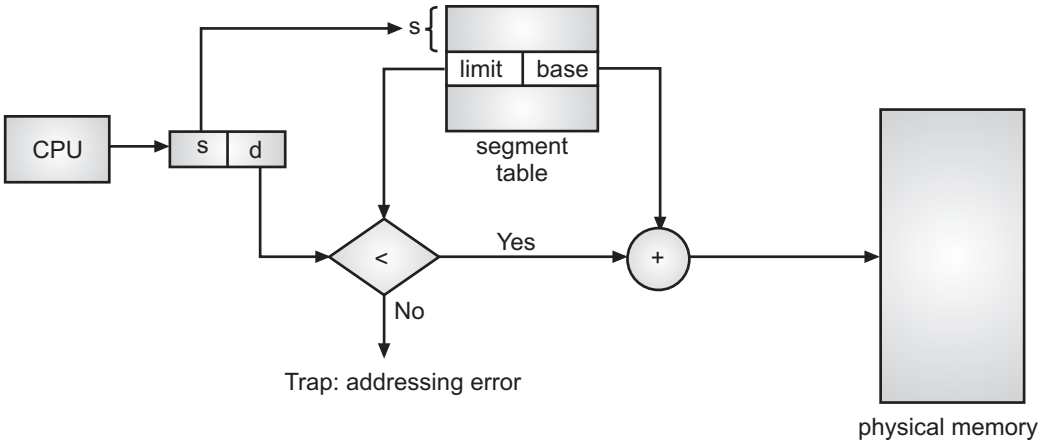


Fig. 7.12: Segmentation Hardware

- The use of segment table is shown in the above figure: Logical address consists of two parts: segment number 's' and an offset 'd' to that segment.
- The segment number is used as an index to segment table.
- The offset 'd' must be in between 0 and limit, if not an error is reported to OS. If legal the offset is added to the base to generate the actual physical address.
- The segment table is an array of base limit register pairs.

7.6 VIRTUAL MEMORY MANAGEMENT

- Virtual Memory is a storage scheme that provides user an illusion of having a very big main memory. This is done by treating a part of secondary memory as the main memory.

7.6.1 Background [S-19]

- Virtual memory is a technique that allows the execution of processes that may not be completely in the physical memory.
- In this technique, the operating system loads only those parts of program in memory that are currently needed for execution of the process. The rest part of the disk and isloaded into the memory only when needed.
- Virtual memory is commonly implemented by demand paging or segmentation. Out of these two ways, demand paging is commonly used as it is easier to implement.
- Virtual memory provides following benefits:
  - Only part of the program needs to be in memory for execution.
  - Logical address space can therefore be much larger than physical address space.
  - Allows address spaces to be shared by several processes.
  - Allows for more efficient process creation.

7.6.2 Demand Paging

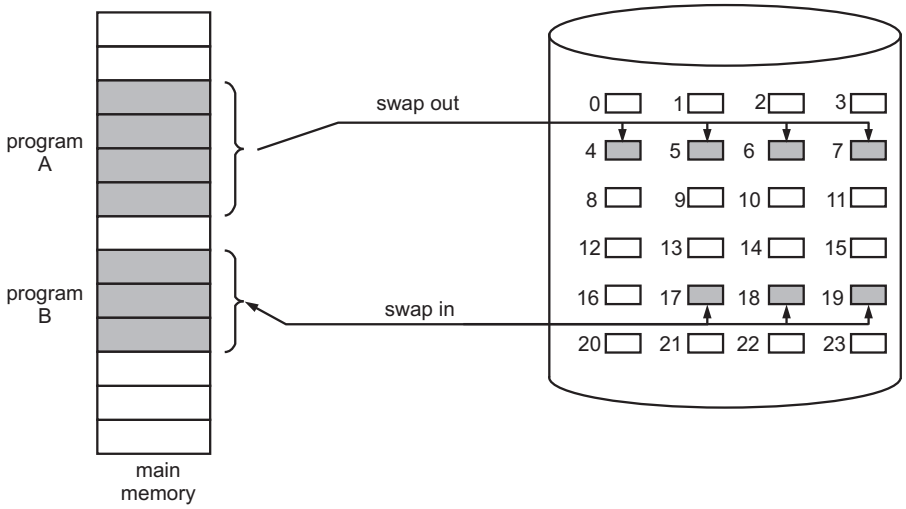


Fig. 7.13: Transfer of a paged memory to contiguous disk space

- The basic idea behind paging is that when a process is swapped in, the pager only loads into memory those pages that it expects the process to need (right away).
- Pages that are not loaded into memory are marked as invalid in the page table, using the invalid bit. (The rest of the page table entry may either be blank or contain information about where to find the swapped-out page on the hard drive).
- If the process only ever accesses pages that are loaded in memory (memory resident pages), then the process runs exactly as if all the pages were loaded in to memory.

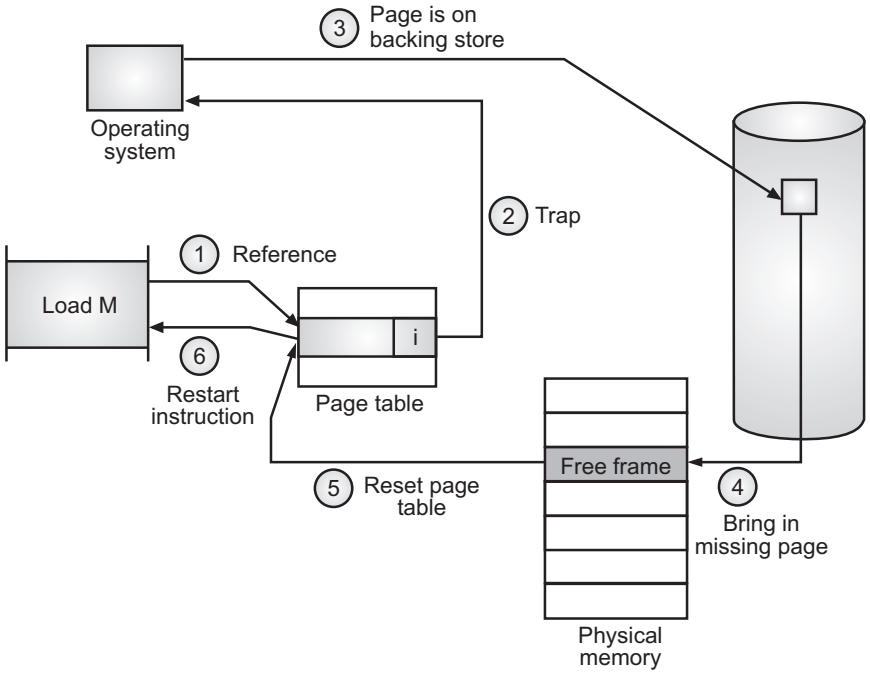


Fig. 7.14: Steps for handling Page Fault

- **Page fault:** A page fault occurs when a program attempts to access data or code that is in its address space, but is not currently located in the system RAM. The fault notifies the operating system that it must locate the data in virtual memory, then transfer it from the storage device, such as an HDD or SSD, to the system RAM.
- **Steps for handling Page Fault:**
  1. Check the location of the referenced page in the Page Map Table(PMT).
  2. If a page fault occured, call on the operating system to fix it.
  3. Using the frame replacement algorithm, find the frame location.
  4. Read the data from disk to memory.
  5. Update the page map table for the process.The instruction that caused the page fault is restarted when the process resumes execution.

7.6.3 Performance of Demand Paging [S-19]

- Obviously there is some slowdown and performance hit whenever a page fault occurs and the system has to go get it from memory, but just how big a hit is it exactly?
- There are many steps that occur when servicing a page fault and some of the steps are optional or variable. Suppose that a normal memory access requires 200 nano-seconds, and that servicing a page fault takes 8 milliseconds. (8,000,000 nanoseconds, or 40,000 times a normal memory access).
- With a **page fault rate** of p, (on a scale from 0 to 1), the effective access time is now:
- Effective access time = p * time taken to access memory in page fault+ (1-p)* time taken to access memory
  = p * 8000000 + (1 – p) * (200)
  = 200 + 7,999,800 * p
  which depends heavily on p!
- Even if only one access in 1000 causes a page fault, the effective access time drops from 200 nanoseconds to 8.2 microseconds, a slowdown of a factor of 40 times. In order to keep the slowdown less than 10%, the page fault rate must be less than 0.0000025, or one in 399,990 accesses.
- The swap space is faster to access than the regular file system, because it does not have to go through the whole directory structure. For this reason some systems will transfer an entire process from the file system to swap space before starting up the process, so that future paging all occurs from the (relatively) faster swap space.

7.6.4 Page Replacement Algorithms

- When a page fault occurs, the operating system has to choose a page to remove from memory to make room for the page that has to be brought in. This is known as **Page Replacement**.
- Page replacement algorithms should have the lowest page fault rate.
- For a given page size, we need to consider only the page number, not the entire address.
- If we have a reference to a page p, then any immediately following references to page p will never cause a page fault. Page p will be memory after the first reference; hence, the immediately following references will not cause a page fault.

- As the number of frames increases, the numbers of page faults are minimized.
- Page replacement is basic to demand paging. It completes the separation between logical memory and physical memory. With this mechanism an enormous virtual memory can be provided for programmers on a smaller physical memory.
- Every operating system usually has its own page replacement scheme.

7.6.4.1 FIFO (First In First Out) [S-18, W-18]

- The simplest page replacement algorithm is a FIFO.
- A FIFO replacement algorithm associates with each page the time when that page was brought into the memory. When a page must be replaced, the oldest page is chosen.
- FIFO queue is created to hold all pages in the memory. We replace the page at the head of the queue. When a page is brought into memory, we insert it at the tail of the queue.
- The FIFO page replacement algorithm is easy to understand and program.
- Its performance is not always good.
- For example consider the following reference string:

  7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0,1,7,0,1

- Our three frames are initially empty. The first three references (7, 0, 1) cause page faults and are brought into these empty frames.
- The next reference (2) replaces page 7, because page 7 was brought in first. Since, 0 is the next reference and 0 is already in memory, we have no fault for this reference.
- The first reference to 3 results in page 0 being replaced, since it was the first of the three pages in memory (0, 1 and 2) to be brought in.
- Because of this replacement, the next reference 0 will cause page fault. Page 1 is then replaced by page 0. This process continues. There are 15 faults altogether.

Reference String:

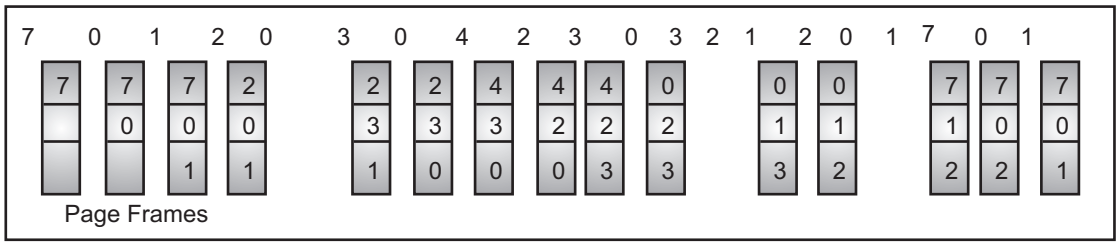


Fig. 7.15: Reference string in FIFO

- The number of faults for four frames (10) is greater than the number of faults for three frames (9).
- This most unexpected result is known as Belady's anomaly. For some page replacement algorithms, the page fault rate may increase as the number of allocated frames increases.

7.6.4.2 Optimal Page Replacement (OPT) [W-18]

- An optimal page replacement algorithm has the lowest page fault rate of all algorithms and does not suffer from Belady's anomaly.

- Optimal replacement algorithm state that replaces the page which will not be used for the longest period of time.
- For example, consider the following reference string.

  7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1
- The first three reference cause faults which fill the three empty frames.
- The reference to page 2 replaces page 7, because 7 will be used until reference 18, whereas page 0 will be used at 5 and page 1 at 14, the reference to page 3 replaces page 1, as page 1 will be the last of the three pages in memory to be referenced again.
- With only nine page faults, optimal replacement is much better than a FIFO algorithm, which had 15 faults. The optimal page replacement algorithm is difficult to implement, because it requires future knowledge of the reference string.

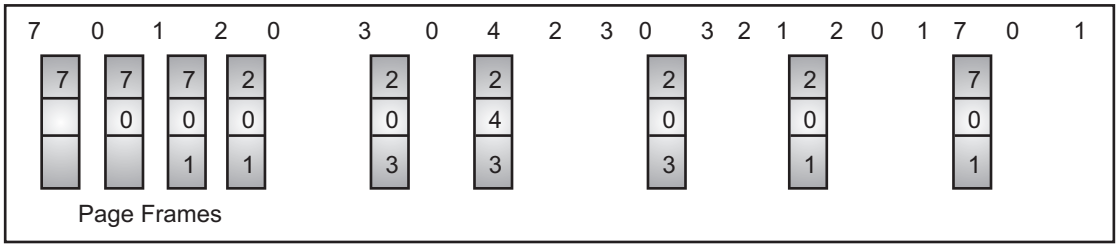


Fig. 7.16: Optimal Page Replacement Algorithm

7.6.4.3 LRU (Last Recently Used)

- If we use the recent past as an approximation of the near future, then LRU replaces the page which has not been used for the longest period of time. This is the least recently used algorithm.
- LRU replacement associates with each page the time of its last use. When a page is to be replaced, LRU chooses that page which has not been used for the longest period or time.
- For example, consider the following reference string

  7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1
- By applying LRU, the first five faults are same as the optimal replacement. When the reference to page 4 occurs, LRU sees out of the three frames in memory, page 2 was used least recently.
- The most recently used page is page 0, and just before that page 3 was used. Thus, LRU replaces page 2, not knowing that it is about to be used.
- When fault for page 2 occurs, LRU replaces page 3. Since of the three pages in memory (0, 3, 4) page 3 is the least recently used. No. of page faults = 12.

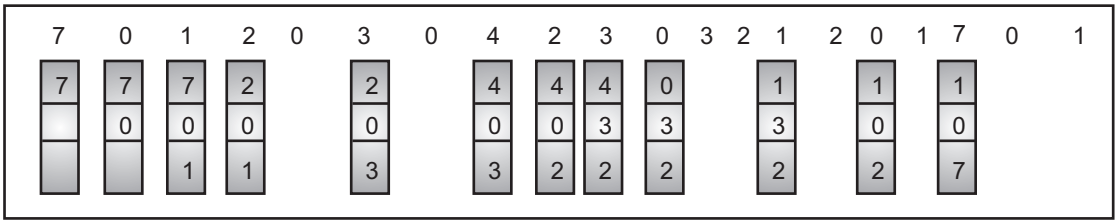


Fig. 7.17: LRU

7.6.4.4 Second-Chance Replacement Algorithm

- The basic algorithm of second-chance replacement is a FIFO replacement algorithm.
- When a page has been selected, however, we inspect its reference bit. If the value is 0, we proceed to replace this page, but if the reference bit is set to 1, we give the page a second chance and move on to select the next FIFO page.
- When a page gets a second chance, its reference bit is cleared and its arrival time is reset to the current time for this reason, a page that is given a second chance will not be replaced until all other pages have been replaced.
- If a page is used often enough to keep its reference bit set, it will never be replaced.
- One way to implement the second-chance algorithm is as a circular queue.
- A pointer indicates which page is to be replaced next. When a frame is needed, the pointer advances until it finds a page with a 0 reference bit.
- Following Fig. 7.18 shows Second-chance (clock) Page-Replacement Algorithm.

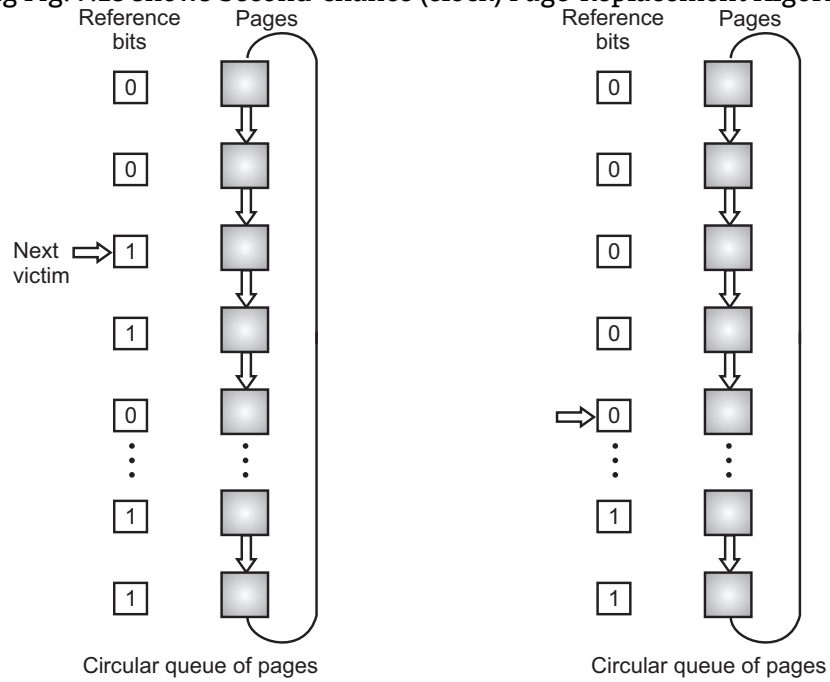


Fig. 7.18: Second-chance (clock) Page-Replacement Algorithm

- As it advances, it clears the reference bits as shown in Fig. 7.18.
- Once, a victim page is found, the page is replaced, and the new page is inserted in the circular queue in that position.
- Second-chance replacement degenerates to FIFO replacement if all bits are set.

Enhanced Second-Chance Algorithm:

- We can enhance the second-chance algorithm by considering the reference bit and the modify bit as an ordered pair. With these two bits, we have the following four possible classes:
  1. **(0, 0) neither recently used nor modified:** best page to replace.
  2. **(0, 1) not recently used but modified:** Not quite as good, because the page will need to be written out before replacement.

3. **(1, 0) recently used but clean:** Probably will be used again soon.
4. **(1, 1) recently used and modified:** Probably will be used again soon and the page will need to be written out to disk before it can be replaced.

7.6.4.5 MFU [S-18]

- The **Most Frequently Used (MFU) page-replacement algorithm** is based on the argument that the page with the smallest count was probably just brought in and has yet to be used.
- As you might expect, neither MFU nor LFU page replacement is common. The implementation of these (MFU and LFU) algorithms is expensive and they do not approximate OPT replacement well.

7.6.4.6 LFU

- The **Least Frequently Used (LFU) page-replacement algorithm** requires that the page with the smallest count be replaced. The reason for this selection is that an actively used page should have a large reference count.
- A problem arises, however, when a page is used heavily during the initial phase of a process but then it is never used again.
- Since, it was used heavily; it has a large count and remains in memory even though it is no longer needed. One solution of this problem is to shift the counts right by 1 bit at regular intervals, forming an exponentially decaying average usage count.

Solved Examples of Previous Exam

Example 1: Consider the following page reference string.

7,5,4,9,4,7,8,5,2,3,4,7,9,7,4

Find the number of page fault for the following algorithm with 3 frames.

(i) LFU, (ii) FIFO.

Solution:

(i) **LFU:** Initially three memory frame slots are empty, so the first three memory references (7, 5, 4) cause page faults and are brought into these empty frames.

In Least Frequently used memory technique the page which has least memory count is replaced with the new page which has to brought in.

So initially 7 has page count 1, 5 has page count 1 also 4 has page count 1 when new page 9 comes it will be replaced with 7 because all has same count so according to first in first out 7 will be replaced with 9, again 4 is the page references which is already in memory so no page fault will occur. And page 4 has count 2 now. Similarly 7 page reference will get replace with 5 because it is the page which is not accessed recently.

7	5	4	9		7	8	5	2	3	4	7	9
7	7	7	9		9	8	5	5	5	5		9
	5	5	5		7	7	7	7	7	7		7
		4	4		4	4	4	2	3	4		4
1	2	3	4		5	6	7	8	9	10	11	12

So on all pages will be mapped to memory frames and total 11 page faults will occur using LFU page replacement Technique

(ii) FIFO: Our three frames are initially empty. The first three references (7, 5, 4) cause page faults and are brought into these empty frames.

The next reference (9) replaces page 7, because page 1 was brought in first. Since, 4 is the next reference and 4 is already in memory, we have no fault for this reference.

Page 7 will be replaced with 5 since 5 is the oldest page in frame.

So this process continues. There are 12 faults altogether.

Reference String:

7	5	4	9		7	8	5	2	3	4	7	9	
7	7	7	9		9	9	5	5	5	4	4	4	
	5	5	5		7	7	7	2	2	2	7	7	
		4	4		4	8	8	8	3	3	3	9	
1	2	3	4		5	6	7	8	9	10	11	12	

Using FIFO page fault algorithm total 12 page faults will be occurred.

Example 2: Consider the following page reference string.

1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 3

Find the number of page fault for the following algorithm with 3 frames.

(i) FIFO, (ii) LRU.

Solution: (i) FIFO:

1		2		3		4	2	1		5		6		2		1		3
1		1		1		4		4		4		6		6		6		3
		2		2		2		1		1		1		2		2		2
				3		3		3		5		5		5		1		1
1		2		3		4		1		5		6		2		1		3

Using FIFO page fault algorithm the page will be replace with the new page. So total 10 page faults will be occurred.

Our three frames are initially empty. The first three references (1, 2, 3) cause page faults and are brought into these empty frames.

The next reference (4) replaces page 1, because page 1 was brought in first. Since, 2 is the next reference and 2 is already in memory, we have no fault for this reference.

Page 1 will be replaced with 2 since 2 is the oldest page in frame.

So this process continues. There are 10 faults altogether.

Reference String: Using FIFO page fault algorithm total 10 page faults will be occurred.

(ii) LRU:

1		2		3		4	2	1		5		6		2		1		3
1		1		1		4		4		5		5		5		1		1
		2		2		2		2		2		6		6		6		3
				3		3		1		1		1		2		2		2
1		2		3		4		1		5		6		2		1		3

Total number of page faults = 10.

Reference String: Our three frames are initially empty. The first three references (1, 2, 3) cause page faults and are brought into these empty frames.

The next reference (4) replaces page 1, because page 1 was brought in first. Since, 2 is the next reference and 2 is already in memory, we have no fault for this reference.

Page 1 will be replaced with 3 since 2 is the page which has been accessed in frame. So this process continues. There are 10 faults altogether.

Example 3: Consider the following page reference string:

4, 6, 7, 8, 4, 6, 9, 6, 7, 8, 4, 6, 7, 9

The number of frames is 3. Show page trace and calculate page fault for the following page replacement schemes:

(i) FIFO, (ii) LRU

Solution: (i) FIFO

Reference String: 4, 6, 7, 8, 4, 6, 9, 6, 7, 8, 4, 6, 7, 9

4	6	7	8	4	6	9	6	7	8	4	6	7	9
4	4	4	8	8	8	9		9	9	4	4	4	9
	6	6	6	4	4	4		7	7	7	6	6	6
		7	7	7	6	6		6	8	8	8	7	7
1	2	3	4	5	6	7		8	9	10	11	12	13

So on all pages will be mapped to memory frames and total 13 page faults will occurs using FIFO page replacement Technique.

(ii) LRU:

Reference String:

4	6	7	8	4	6	9	6	7	8	4	6	7	9
4	4	4	8	8	8	9		9	8	8	8	7	7
	6	6	6	4	4	4		7	7	7	6	6	9
		7	7	7	6	6		6	6	4	4	4	4
1	2	3	4	5	6	7		8	9	10	11	12	13

Using LRU Page Replacement Algorithm page fault algorithm total 13 page faults will be occurred.

Summary

- Computer memory can be defined as a collection of some data represented in the binary format.
- Memory management is the process of controlling and coordinating computer memory. This process assign portions called blocks to various running programs to optimize overall system performance.
- The memory-management algorithms (contiguous allocation, paging, segmentation, and combinations of paging and segmentation) differ in many aspects.
- In comparing different memory-management strategies, we use the following considerations:
- **Hardware support:** A simple base register or a base – limit register pair is sufficient for the single- and multiple-partition schemes, whereas paging and segmentation need mapping tables to define the address map.
- **Performance:** If memory-management algorithm becomes more complex, the time required to map a logical address to a physical address increases.
- **Fragmentation** is a multi-programmed system that performs more efficiently if it has a higher level of multiprogramming. For a given set of processes, we can increase the multiprogramming level only by packing more processes into memory. To accomplish this task, we must reduce memory waste, or fragmentation. Systems with fixed-sized allocation units, such as the single-partition scheme and paging, suffer from internal fragmentation. Systems with variable-sized allocation units, such as the multiple-partition scheme and segmentation, suffer from external fragmentation.
- **Relocation:** One solution to the external-fragmentation problem is compaction. Compaction involves shifting a program in memory in such a way that the program does not notice the change.
- **Swapping:** Processes are copied from main memory to a backing store and later are copied back to main memory.
- **Sharing:** Another means of increasing the multiprogramming level is to share code and data among different processes. Sharing generally requires that either paging or segmentation be used to provide small packets of information (pages or segments) that can be shared. Sharing is a means of running many processes with a limited amount of memory, but shared programs and data must be designed carefully.
- **Protection:** If paging or segmentation is provided, different sections of a user program can be declared execute-only, read-only, or read – write.
- **Virtual Memory** is a storage scheme that provides user an illusion of having a very big main memory. This is done by treating a part of secondary memory as the main memory.
- The basic idea behind paging is that when a process is swapped in, the pager only loads into memory those pages that it expects the process to need.

- When a page fault occurs, the operating system has to choose a page to remove from memory to make room for the page that has to be brought in. This is known as Page Replacement.
- The process of replacement is sometimes called swap out or write to disk.
- **Types of Page Replacement Algorithms:** Optimal Page Replacement (OPT) Algorithm, First-in-First-Out **(**FIFO), Least recent used (LRU), Second chance page replacement, MFU, LFU.

Check Your Understanding

1. In which of the following page replacement policies Belady's anomaly occurs?
   (a) FIFO (b) OPT
   (c) MFU (d) program status word
2. _____ allocates the largest hole (free fragment) available in the memory.
   (a) Best fit (b) Worse Fit
   (c) First fit (d) None of the above
3. Which one of the following is the address generated by CPU?
   (a) physical address (b) absolute address
   (c) logical address (d) none of the mentioned
4. Run time mapping from virtual to physical address is done by _____.
   (a) Memory management unit (b) CPU
   (c) PCI (d) None of the mentioned
5. Memory management technique in which system stores and retrieves data from secondary storage for use in main memory is called?
   (a) fragmentation (b) paging
   (c) mapping (d) none of the mentioned
6. The address of a page table in memory is pointed by _____.
   (a) stack pointer (b) page table base register
   (c) page register (d) program counter
7. Program always deals with _____.
   (a) logical address (b) absolute address
   (c) physical address (d) relative address
8. The page table contains _____.
   (a) base address of each page in physical memory
   (b) page offset
   (c) page size
   (d) none of the mentioned

9. What is compaction?
   (a) a technique for overcoming internal fragmentation.
   (b) a paging technique.
   (c) a technique for overcoming external fragmentation.
   (d) a technique for overcoming fatal error.
10. Operating System maintains the page table for _____.
   (a) each process (b) each thread
   (c) each instruction (d) each address

Answers

1. (a)	2. (b)	3. (c)	4. (a)	5. (b)	6. (b)	7. (a)	8. (a)	9. (c)	10. (a)

Practice Questions

Q.I Answer the following questions in short:

1. What is meant by memory management?
2. What is swapping?
3. Describe the basic concept of paging.
4. What is segmentation?
5. What is external fragmentation?
6. Define Belady's Anomaly.

Q.II Answer the following questions:

1. Explain page replacement algorithms with their advantages and disadvantages.
2. Define NRU, MFU, LFU page replacement algorithms.
3. Explain the term virtual memory in detail.
4. Explain FIFO, LRU page replacement algorithms for the reference string 7 0 1 2 0 3 0 4 2 3 1 0 3.
5. With the help of diagram describe demand paging.
6. Describe second chance algorithm with example.
7. State advantages and disadvantages of segmentation.
8. Differentiate between Internal and External fragmentation.
9. Explain optional page replacement algorithm with example.
10. Consider the given reference string: 2 3 2 1 5 2 4 5 3 2 5 2
    Calculate the page fault for number of frames = 3 for the following algorithms.
    (i) LRU
    (ii) FIFO
11. Compare paging and segmentation.
12. What is meant by memory partition?

13. Consider the given page reference string: 4, 3, 2, 1, 4, 3, 5, 4, 3, 2, 1, 5.
    The number of frames is 3. Show page trace and calculate page faults for the following page replacement schemes:
    (i) FIFO
    (ii) MFU

Q.III Define terms:

1. Fragmentation.
2. Page fault.
3. Page table.
4. Dynamic loading.
5. Dynamic linking.
6. Swap time.
7. Logical address.
8. Physical address.

Previous Exam Questions

Summer 2018

1. Define Swap Time. **[2 M]**

Ans. Refer to Section 7.2.

2. Consider the following page reference string: **[4 M]**
7, 5, 4, 9, 4, 7, 8, 5, 3, 4, 7, 9, 7, 4
Find the number of page fault for the following algorithm with 3 frames:
(i) FIFO
(ii) MFU

Ans. Refer to Section 7.6. 3.1 and 7.6.3.5.

3. What is fragmentation? Explain types of fragmentation in details. **[4 M]**

Ans. Refer to Section 7.3.3.

Winter 2018

1. What is meant by Address Binding? **[2 M]**

Ans. Refer to Section 7.1.2.

2. What do you mean by Paging? List the advantages and disadvantages of Paging. **[4 M]**

Ans. Refer to Section 7.4.

3. Define the terms: **[4 M]**
(i) Logical Address
(ii) Physical Address.

Ans. Refer to Section 7.3.1.

4. Consider the following page reference string:

9, 2, 3, 4, 2, 5, 2, 6, 4, 5, 2, 5, 4, 3, 4, 2, 3, 9, 2, 3

The number of frames is 4. Calculate the page faults for the following page replacement schemes:

(i) FIFO

(ii) Optimal **[4 M]**

Ans. Refer to Section 7.6.3.1 and 7.6.3.2.

Summer 2019

1. What is demand paging? **[2 M]**

Ans. Refer to Section 7.6.1.

2. Write the steps of calculate the physical address by operating system. Explain with example. **[4 M]**

Ans. Refer to Section 7.3.1.

3. What is a page fault? Explain the different steps in handling a page fault. **[4 M]**

Ans. Refer to Section 7.6.2.

■■■

8... File System

Objectives...

- To introduce concept of file system.
- To know about various file access and allocation methods.
- To get knowledge of file structure.
- To learn free space management techniques.

8.1 INTRODUCTION AND FILE CONCEPTS

- A file is a named collection of related information that is recorded on secondary storage such as magnetic disks, magnetic tapes and optical disks.
- In general, a file is a sequence of bits, bytes, lines or records whose meaning is defined by the file's creator and user.
- The information in a file is defined by its creator. Many different types of information may be stored in a file: Source programs, Object programs, Executable programs, Numeric Data, Text, Payroll records, Graphic Images, Sound recording and so on. A file has a certain defined structure according to its type.

8.1.1 File Attributes [S-19]

- A file has certain attributes, which vary from one operating system to another, but typically consist of these:
  1. **Name**: Every file carries a name by which the file is recognized in the file system. One directory cannot have two files with the same name.
  2. **Identifier**: Along with the name, Each File has its own extension which identifies the type of the file. For example text files have an extension .txt or .doc, spreadsheets have extension .xlsx and audio/video files have an extension .mp3 and .mp4 etc.
  3. **Type**: In a File System, the Files are classified in different types such as video files, audio files, text files, executable files, etc.
  4. **Location**: In the File System, there are several locations on which, the files can be stored. Each file carries its location as its attribute.

5. **Size**: The Size of the File is one of its most important attribute. By size of the file, we mean the number of bytes acquired by the file in the memory.
6. **Protection**: The Admin of the computer may want the different protections for the different files. Therefore each file carries its own set of permissions to the different group of Users.
7. **Time and Date**: Every file carries a timestamp which contains the time and date on which the file is last modified.

8.1.2 Operations on Files [W-18]

- There are various operations which can be implemented on a file. We will see all of them in detail:

1. **Create**: Creation of the file is the most important operation on the file. Different types of files are created by different methods. For example, text editors are used to create a text file, word processors are used to create a word file and Image editors are used to create the image files.
2. **Write**: Writing the file is different from creating the file. The OS maintains a write pointer for every file which points to the position in the file from which, the data needs to be written.
3. **Read**: Every file is opened in three different modes: Read, Write and Append. A Read pointer is maintained by the OS, pointing to the position up to which, the data has been read.
4. **Re-position**: Re-positioning is simply moving the file pointers forward or backward depending upon the user's requirement. It is also called as **Seeking**.
5. **Delete**: Deleting the file will not only delete all the data stored inside the file. It also deletes all the attributes of the file. The space which is allocated to the file will now become available and can be allocated to the other files.
6. **Truncate**: Truncating is simply deleting the file except deleting attributes. The file is not completely deleted although the information stored inside the file gets replaced.

8.1.3 Types of Files

- It refers to the ability of the operating system to differentiate various types of files like text files, binary, and source files. However, Operating systems like MS-DOS and UNIX has the following type of files:

(i) Character Special File:

- It is a hardware file that reads or writes data character by character, like mouse, printer, and more.

(ii) Ordinary Files:

- These types of files stores user information.
- It may be text, executable programs, and databases.
- It allows the user to perform operations like add, delete, and modify.

(iii) Directory Files:

- Directory contains files and other related information about those files. It is basically a folder to hold and organize multiple files.

(iv) Special Files:

- These files are also called device files. They represent physical devices like printers, disks, networks, flash drive, etc.

Table 8.1: File Types and Functions

File Type	Usual Extensions	Function
Executable	`exe, com, bin or none`	Ready-to-run machine language program.
Object	`obj, o`	Compiled, machine language, not linked.
Source Code	`c, cc, java, pas, asm, a`	Source code in various languages.
Text	`txt, doc`	Textual data, documents.
Batch	`bat, sh`	Commands to the command interpreter.
Word Processor	`wp, rtf, tex, doc`	Various word-processor formats.
Library	`lib, a,dll`	Libraries of routines for programmers.
Print or View	`ps, gif, dvi`	ASCII or binary file in a format for printing or viewing.
Multimedia	`Mov,mpeg,rm,mp3,avi`	Binary file containing audio or A/V information.
Archive	`arc, zip, tar`	Related files grouped into one file, sometimes compressed, for archiving or storage.

8.1.4 Functions of File

- Create file, find space on disk, and make an entry in the directory.
- Write to file, requires positioning within the file.
- Read from file involves positioning within the file.
- Delete directory entry, regain disk space.
- Reposition: move read/write position.

8.1.5 Commonly used terms in File Systems

- **Field:** This element stores a single value, which can be static or variable length.
- **Database:** Collection of related data is called a database. Relationships among elements of data are explicit.
- **Files:** Files is the collection of similar record which is treated as a single entity.

- **Record:** A Record type is a complex data type that allows the programmer to create a new data type with the desired column structure. It groups one or more columns to form a new data type. These columns will have their own names and data type.

8.2 ACCESS METHODS

- There are various types of file access methods in the operating system:
  - Sequential Access Method
  - Direct Access Method
  - Index Sequential Access Method

8.2.1 Sequential Access

- Sequential Access method is one of the simplest file access methods. Most of the OS uses a sequential access method to access the file.
- In this method, the OS read the file word by word and we have a pointer that points the file's base address.
- If we need to read file's first word, then there is a pointer that offers the word in which we want to read and increment value of the word by 1. This process continues till the end of the file.
- The sequential access method is one of the mostly used methods because more files like text files, audio files, and video files require to be sequentially accessed.

Key Points:

- In the sequential access method, if we use read command, then the pointer is moved ahead by 1.
- If the write command is used, then the memory will be allocated, and at the end of the file, a pointer will be moved.

Advantages:

1. Easy to access the next record.
2. Data organization is very simple.
3. Absence of auxiliary data structures.
4. Sequential files are typically used in batch applications where they are involved in the processing of all the records such as payroll, billing etc.
5. Sequential Access Method is reasonable for tape.
6. Automatic backup copy is created.

Disadvantages:

1. Record deletion creates a waste space. It requires some type of record reorganization.
2. The sequential file provides poor performance for interactive applications that involve queries and/or updates of individual records.

8.2.2 Direct Access [S-18]

- For direct access, the file is viewed as a number of sequential blocks of records.
- A block is generally a fixed length quantity, defined by an operating system.

- A block may be 512 bytes long, 1024 bytes long or some other length, depending upon the system.
- A direct access file allows arbitrary blocks to be read or written. Thus, we may read block 14, then read block 50 and then write block 7. There are no restrictions on the order of reading or writing for a direct access file.
- Direct access files are of great use for immediate access to large amounts of information.
- When a query concerning a particular subject arrives, we compute which block contains the answer and then read the block directly to provide the desired information.
- For example, in an Airline Reservation System, we store all the information about a particular flight in the block identified by the flight by number (flight 710). Thus, the number of available seats for flight 710 is stored in block 710 of the reservation file.
- The block number provided by the user to the operating system is normally relative block number.
- A relative block number is an index relative to the beginning of the file. Thus, the first relative block of the file is 0; the next is 1 and so on.
- The use of relative block numbers allows the operating system to decide where the file should be placed.
- Not all the operating system support both sequential and direct access of files. Some systems allow only sequential file access, others allow only direct access.

Advantages of a direct access file:

1. Data access is direct and fast.
2. Centrally maintained data can be kept up-to-date.

Disadvantages of a direct access file:

1. As the entire data is to be stored on disk but disk (hardware) is expensive.
2. There is no backup if a file gets destroyed. This is so because files are updated directly and no transaction files are maintained.

8.2.2.1 Hashing

- The basic idea of Hash addressing is that each record is placed in the database at a location whose Stored Record Address(SRA) may be computed as some function (Called as Hash function) of a value usually the primary key value.
- Thus, to store the record initially, the DBMS computes SRA and instructs the access method to place the occurrence at that position. To retrieve the occurrence, the DBMS performs the same computation as before and then requests the access method to fetch the occurrence at the computed position.
- What address is generated by the Hashing function? There are a number of ways of converting a key to a numeric value. Most of the keys are numeric, but if the keys are alphabetic or alpha numeric we can use the bit representation of the alphabet to generate the numeric equivalent key.

- A number of simple hashing methods are given below:
  1. Use the **low order part of the key**.
  2. For long keys, we identify start, middle and end regions, such that the sum of the lengths of the start and end regions equals the length of the middle region. The start and end digits are combined and the combined string of digits is added to the middle region digits. The (new number) mod (the upper limit of the hash function) gives the **bucket address**.
  3. **Square** all or part of the key and take a part from the result. The whole or some defined part of the key is squared and a number of digits are selected from the square as being part of the hash result.
  4. **Division** can be used to form the address. The key can be divided by a number (consider it as a prime number) and the remainder is taken as the bucket address usually a prime number is used for division because if keys are in some multiples, this would produce a poor result.

Hash Function:

- There are a number of possible methods for generating a hash function but it has been found that hash functions using division or multiplication perform quite well under most conditions.
- We will take a simple hashing function:

  $$H(k) = k \bmod s$$

  where, H(k) produces an address and H is a hash function that maps the key value k to the value H(k) and k is the numeric representation of the key.

Advantages:

1. This method provides very fast direct access on the basis of values of the hashed field.

Disadvantages:

1. The sequence of stored record occurrence will not be a primary key sequence i.e. it has no particular sequence.
2. The possibility of collision i.e. two distinct stored record occurrences whose keys may hash to the same SRA.

8.2.3 Index Sequential Access Method (ISAM)

- It is different method of accessing file which is built on the top of sequential access method. These methods construct an index for the file. It controls the pointer by using index.
- The index like an index in the book that contains the pointer to the various blocks. To search a record in the file, we first search the index and then by the help of pointer we access the file directly.
- In this method, an index value is generated for each primary key and mapped with the record. This index contains the address of the record in the file.
- If any record has to be retrieved based on its index value, then the address of the data block is fetched and the record is retrieved from the memory.

Advantages of ISAM:

1. Records are processed efficiently in both sequential and random order.
2. Data access is direct and fast.
3. Centrally maintained data can be kept up-to-date.

Disadvantages of ISAM:

1. As file grows, its performance deteriorates rapidly because of overflows and thus reorganization.
2. ISAM lowers the system's efficiency.

8.3 FILE STRUCTURE

- Files can be structured in any of the several ways. The three common possibilities are described in Fig. 8.1.

1. **Stream of Bytes (See Fig. 8.1 (a))**
   - OS (Operating System) considers a file to be unstructured.
   - Simplifies file management for the OS. Applications can impose their own structure.
   - Used by UNIX, Windows and most modern Operating Systems.
2. **Records (See Fig. 8.1 (b))**
   - A file is a sequence of fixed length record, each with some internal structure.
   - Collection of bytes is treated as a unit.
   - For example: employee record.
   - Operations are at the level of records (read_rec, write_rec).
   - File is a collection of similar records. OS can optimize operations on records.
3. **Tree of Records (see Fig. 8.1 (c))**
   - A file consists of a tree of records, not necessarily all the same length.
   - Records can be of variable length.
   - Each record has an associated key. The record retrieval is based on the key.

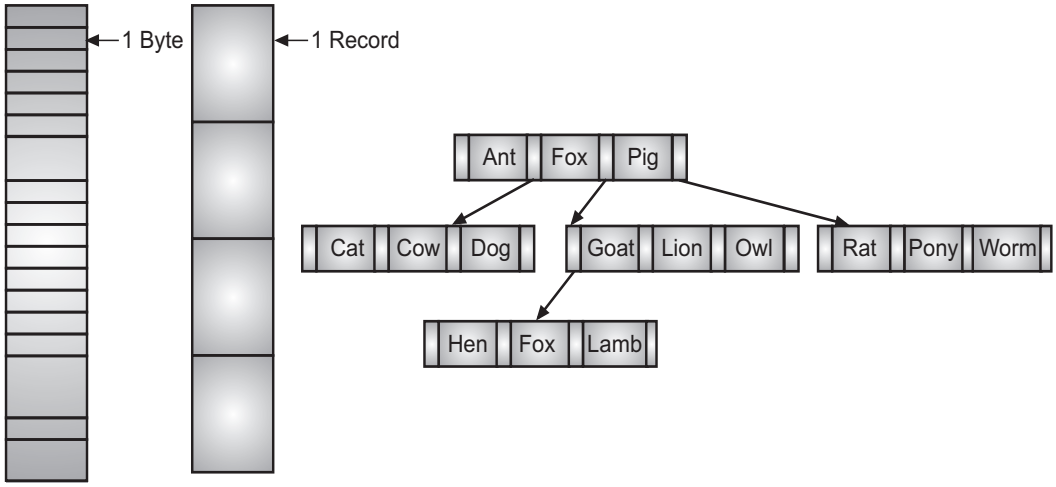


Fig. 8.1: Various File structures

8.3.1 Directory Structure [W-18]

- File systems usually consist of files separated into groups called directories (also called folders). Directories can contain files or additional directories.

Operations performed on directory:

- When we are considering a particular directory structure, the operations that are performed on a directory are:
  - **Search for a file:** We need to be able to search a directory structure to find the entry for particular file.
  - **Create a file:** New files need to be created and added to directory.
  - **Delete a file:** When a file is deleted, an entry must be removed from directory.
  - **List directory:** We need to be able to list the files in directory and contents of the directory entry for each file in the list.
  - **Update directory:** Because some file attributes are stored in the directory a change in one of these attributes requires a change in the corresponding directory entry.

Types of directory structure:

- The directory can be viewed as a symbol table translates file names into their directory entries. If we take such a view, we see that the directory itself can be organized in many ways.

1. Single level directory:

- The simplest directory structure is the single level directory. All files are contained in a same directory, which is easy to support and understand.

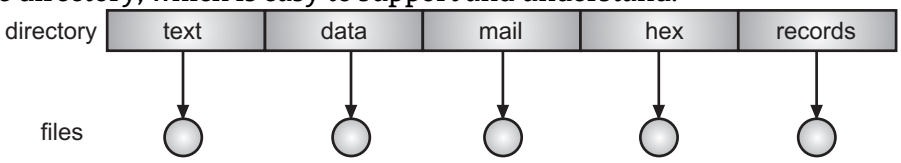


Fig. 8.2: Single level directory

- A single-level directory has significant limitation, however, when number of files increases or when the system has more than one user. Since all files are in the same directory, they must have unique names.
- A single-level directory have problem of filenames using by different users. The solution is to create separate directory for each user.

2. Two-level directory:

- In the two-level directory structure, each user has own User File Directory (UFD) when user log in, the system's Master File Directory (MFD) is searched. The MFD is indexed by user name or account number and each entry points to the UFD for that user.

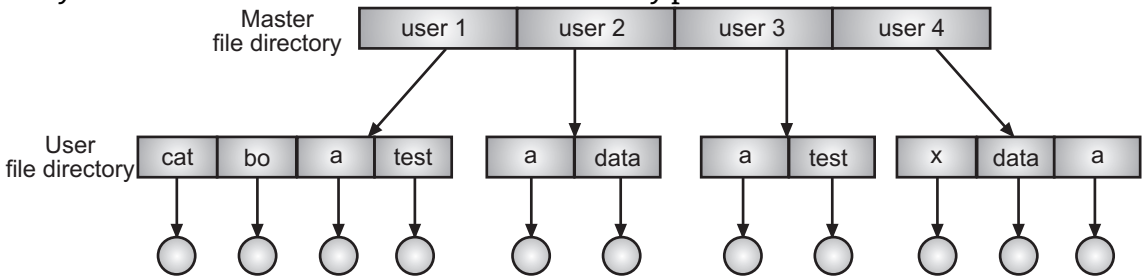


Fig. 8.3: Two level directory

- To create file for a user, the operating system searches only that user's UFD to ascertain whether another file of that name exists. To delete file, the operating system continues its search to the local UFD. The user directories themselves must be created and deleted as necessary.
- The two-level directory solves the name-collision problem, it still has disadvantage. This structure isolates one user from another. This isolation is an advantage when the user is completely independent but is disadvantage when the user wants to co-operate on same task and to access one another's file.
- The two-level hierarchy eliminates name conflicts among users but is not satisfactory for user with large number of files. We have to extend the two-level tree into the directory structure to tree of arbitrary height.

3. Tree-structured directory:

- It is an inverted tree structure with a single root. The topmost directory in the hierarchy is called the root directory. A directory is called the parent directory of the subdirectories and files in it. It can contain any number of directories. A directory can contain any number of files and subdirectories.
- The MS-DOS system is structured as a tree. The tree has a root directory. Every file in the system has a unique path name.
- A path name is the path from the root, through all the subdirectories to specified file.

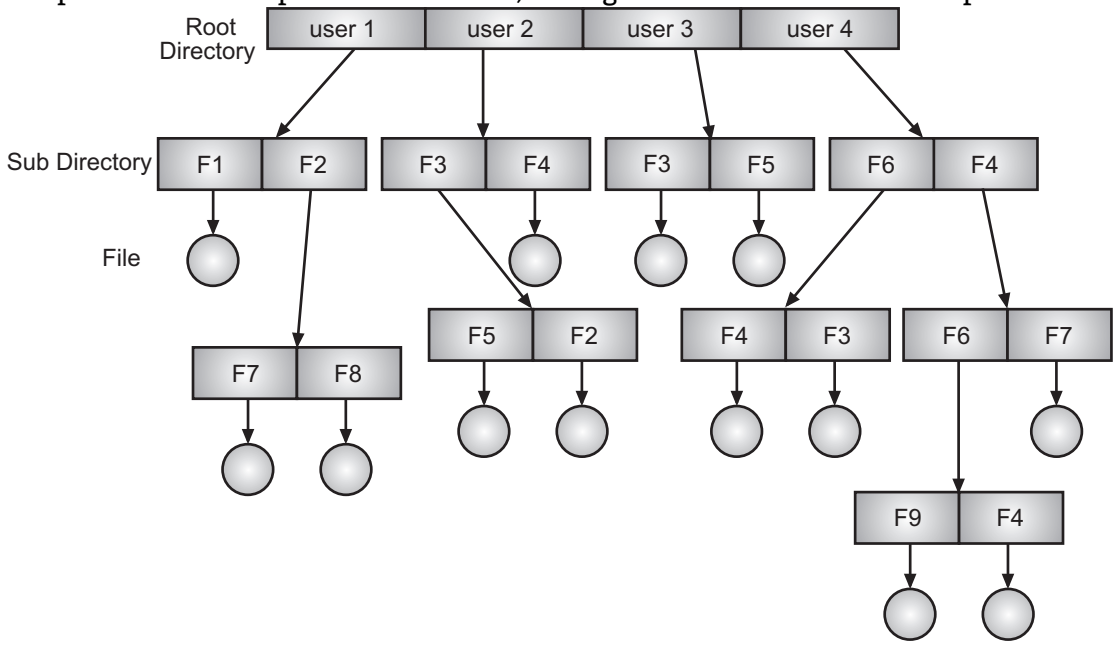


Fig. 8.4: Tree-structured directory

4. Acyclic graph Directory:

- Consider two programmers who are working on a joint project. The files associates with that project can be stored in a subdirectory, separating them from other projects and files of the two programmers.
- But since both the programmers are equally responsible for the project both want the subdirectory to be in their own directories. The common subdirectory should be shared.
- A shared directory or file will exist in two different places at once.

- A tree structure prohibits the sharing of files or directories. But an acyclic graph allows directories to have shared subdirectories of files, (Fig. 8.5).

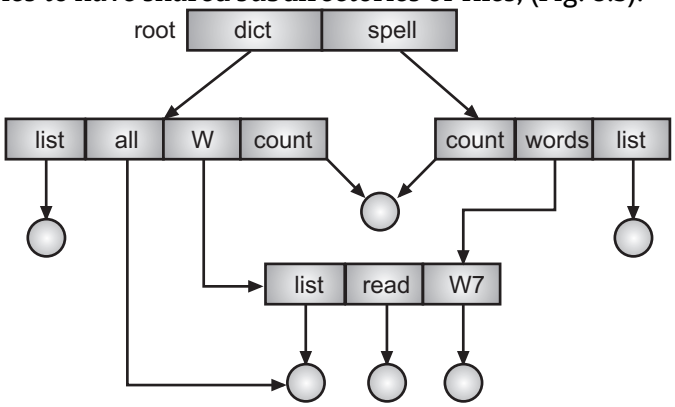


Fig. 8.5: Acyclic Graph Directory

- The same file subdirectory may be in two different directories. An acyclic graph, that is, a graph with no cycles is a natural generation of tree structured directory schemes.

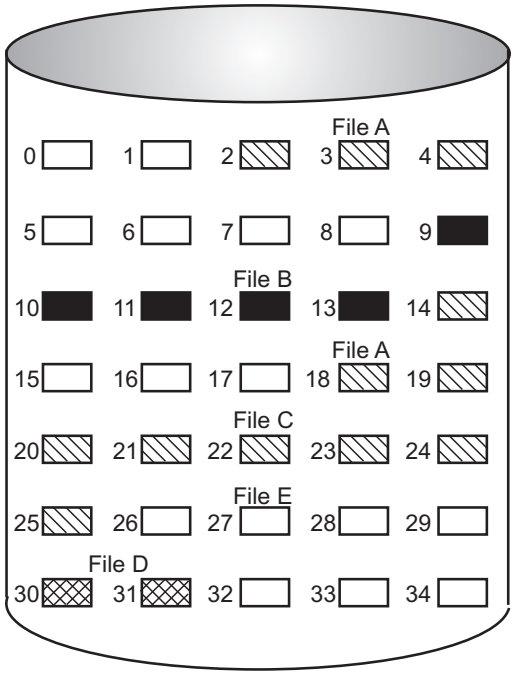
8.4 ALLOCATION METHODS

- The file allocation methods define how the files are stored in the disk blocks.
- Several issues are involved in file allocation:
  1. When a new file is created, is the maximum space required for the file allocated at once?
  2. Space is allocated to file as one or more contiguous units. What size of portion should be used for file allocation?
  3. What sort of data structure is used to keep track of the blocks to a file? Such a table is typically referred to as File Allocation Table (FAT).
- A static allocation requires that the maximum size of file be declared at the time of file creation request. In number of cases the value can be reliably estimate, but many cases, it is difficult.
- In those cases users and application programmers would tend to overestimate file size. This is clearly wasteful from the point view of secondary storage allocation.
- Thus, there are advantages to the use of dynamic allocation, which allocates space to a file in blocks are needed.
- There are different types of file allocation methods, but we mainly use three types of file allocation methods.
  - Contiguous allocation
  - Linked list allocation
  - Indexed allocation
- These methods provide quick access to the file blocks and also the utilization of the disk space in an efficient manner.

8.4.1 Contiguous Allocation [S-18]

- Contiguous allocation is one of the most used methods for allocation.
- With this method a single contiguous set of blocks is allocated to a file creation (Fig. 8.6). Thus, this is the static allocation method, using variable-size portion.
- The FAT needs just a one entry. For each file, showing starting block and the length of file. Contiguous allocation is best for sequential file.

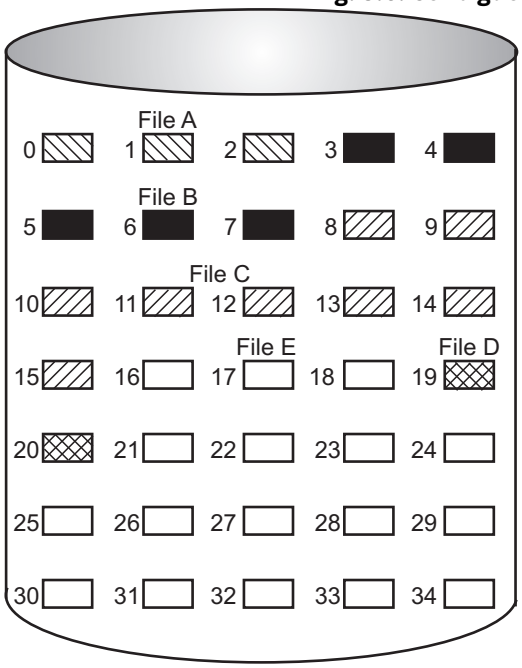
- Contiguous allocation has some problems. External fragmentation will occur, making it difficult to find contiguous blocks of space of sufficient length.
- For time to time, it is necessary to do a compaction algorithm to free up additional space of disk.



File Allocation Table

File Name	Start Block	Length
File A	2	3
File B	9	5
File C	18	8
File D	30	2
File E	26	3

Fig. 8.6: Contiguous File Allocation



File Allocation Table

File Name	Start Block	Length
File A	0	3
File B	3	5
File C	8	8
File D	19	2
File E	16	3

Fig. 8.7: Contiguous File Allocation (After Compaction)

Advantages:

1. Supports both sequential and direct access methods.
2. Contiguous allocation is the best form of allocation for **sequential files**. Multiple blocks can be brought in at a time to improve I/O performance for sequential processing.
3. It is also easy to retrieve a single block from a file. For example, if a file starts at block 'n' and the i^{th} block of the file is wanted, its location on secondary storage is simply n + i.

Disadvantages:

1. Suffers from external fragmentation.
2. Very difficult to find contiguous blocks of space.
3. Also with per-allocation, it is necessary to declare the size of the file at the time of creation which many times is difficult to estimate.

8.4.2 Linked Allocation [S-19]

- Linked allocation solves all the problem of contiguous allocation (Fig. 8.8). With chained allocation, each file is a linked list of disk blocks; the blocks may be scattered anywhere on the disk.
- Again, the file allocation table needs just a single entry for each file, showing starting block and length of file.
- Although static allocation is possible, it is more common simply to allocate block as needed. There is no external fragmentation because only one block at a time is needed.
- The major problem is that it can be used efficiently only for sequential file. To find its block, we must start at the beginning of that file and follow the pointer until we get to the i^{th} block.
- Linked allocation is inefficient to support a direct access files. Another disadvantage to chained allocation is the space required for the pointers.

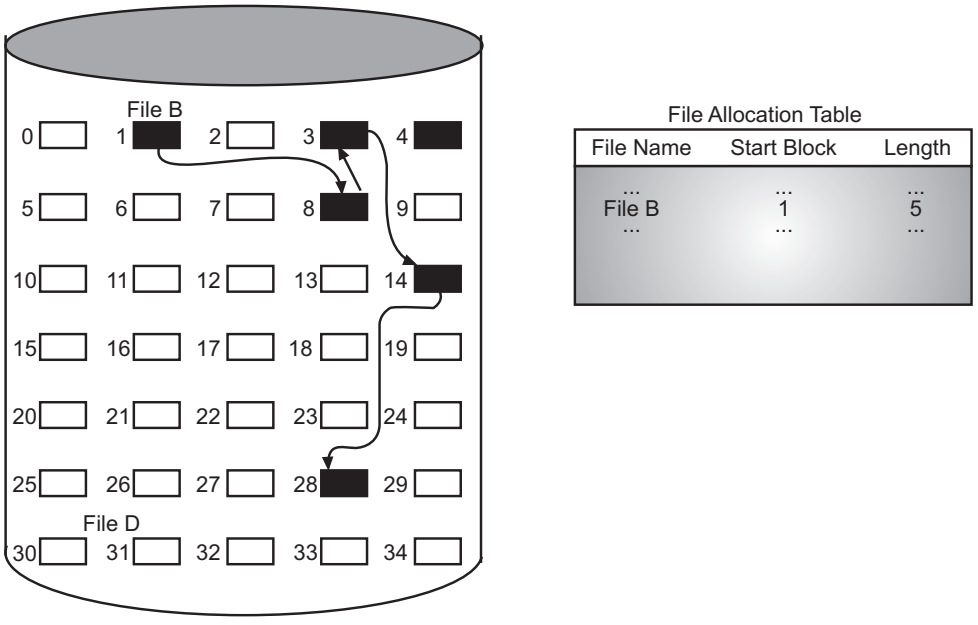


Fig. 8.8: Linked/Chained Allocation

Advantages:

1. Any free blocks can be added to a chain.
2. There is no external fragmentation.
3. Best suited for sequential files that are to be processed sequentially.

Disadvantages:

1. There is no accommodation of the principle of locality that is series of accesses to different parts of the disk are required.
2. Space is required for the pointers. 1.5% of disk is used for the pointers and not for information. If a pointer is lost or damaged or bug occurs in operating system or disk hardware failure occur, it may result in picking up the wrong pointer.
3. This method cannot support direct access.

8.4.3 Indexed Allocation [W-18]

- Linked allocation solves the external fragmentation and size declaration problem and contiguous allocation.
- However, chained allocation cannot support efficient direct access, since pointers are scattered with the blocks themselves all over the disk and need to be retrieved in order.
- Indexed allocation solves this problem by bringing all the pointer is together into one location: the index block. In this case, the FAT contains a separate one-level index for each file, the index has one entry for each portion allocated to file.
- File indexes are not physically stored as part of the FAT, but it is kept in a separate block and entry for the file in the FAT points to that block.
- Allocation may be on the basis of either fixed-sized blocks or variable-size partitions. Allocation by blocks eliminates external fragmentation, whereas allocation by variable size portions improve locality.
- Indexed allocation supports both sequential and direct access to the file and thus is the most popular form of file allocation.

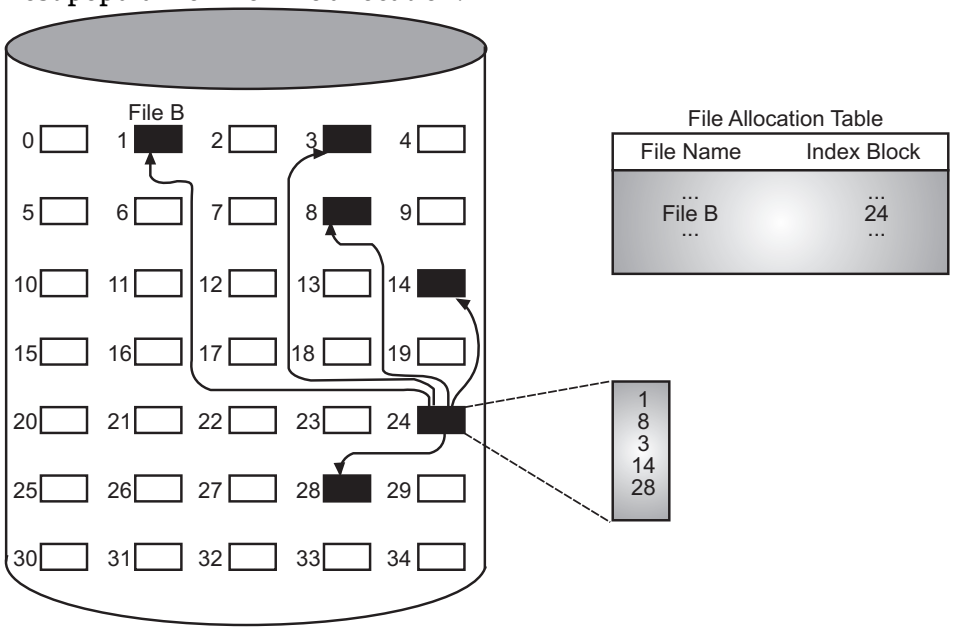


Fig. 8.9: Indexed Allocation with Block Portions

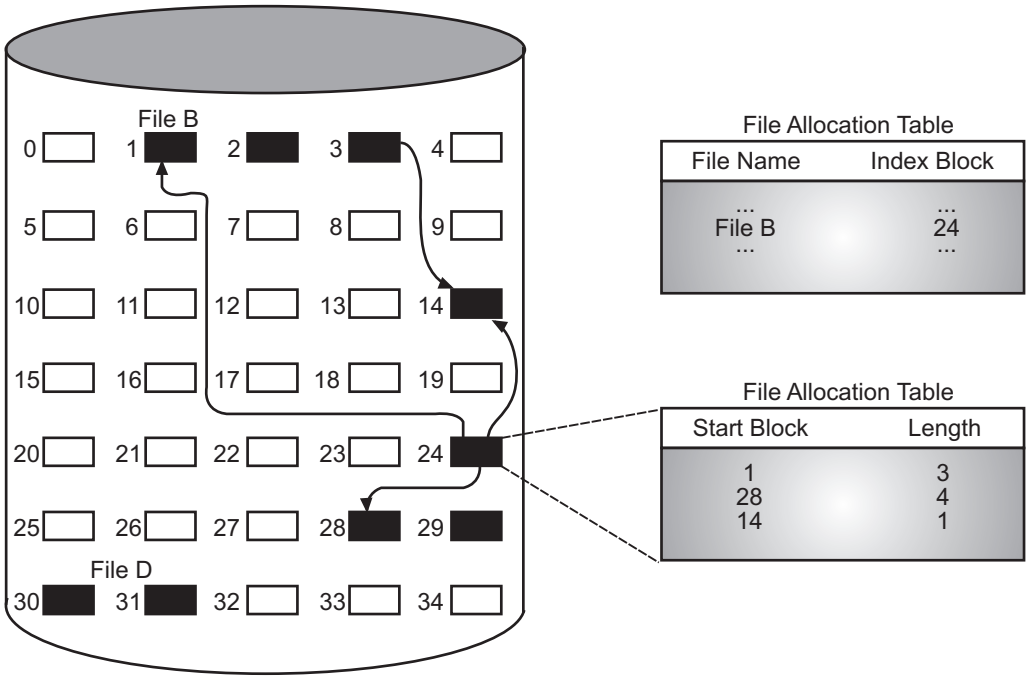


Fig. 8.10: Indexed Allocations with Variable-Length Portions

Advantages:

1. Does not suffer from external fragmentation.
2. Support both sequential and direct access to the file.

Disadvantages:

1. In index allocation, pointer overhead is more.
2. We can lose the entire file if an index block is not correct.
3. It is totally wastage to create an index for a small file.

8.5 FREE SPACE MANAGEMENT [W-18]

- As we know that the memory space in the disk is limited. So we need to use the space of the deleted files for the allocation of the new file. One optical disk allows only one write at a time in the given sector and thus it is not physically possible to reuse it for other files.
- The system maintains a free space list by keep track of the free disk space.
- The free space list contains all the records of the free space disk block. The free blocks are those which are not allocated to other file or directory.
- When we create a file we first search for the free space in the memory and then check in the free space list for the required amount of space that we require for our file. If the free space is available then allocate this space to the new file.
- After that, the allocating space is deleted from the free space list. Whenever we delete a file then its free memory space is added to the free space list.
- The process of looking after and managing the free blocks of the disk is called Free Space Management.

- There are some methods or techniques to implement a free space list. These are as follows:
  1. Bitmap
  2. Linked list
  3. Grouping
  4. Counting

8.5.1 Bit Vector (Bitmap)

- This technique is used to implement the free space management. When the free space is implemented as the bitmap or bit vector then each block of the disk is represented by a bit. When the block is free its bit is set to 1 and when the block is allocated the bit is set to 0.
- The main advantage of the bitmap is it is relatively simple and efficient in finding the first free block and also the consecutive free block in the disk. Many computers provide the bit manipulation instruction which is used by the users.
- The calculation of the block number is done by the formula:

  (Number of bits per words) × (Number of 0 value word) + Offset of first 1-bit
- Apple Macintosh operating system uses the bitmap method to allocate the disk space.

Example 1: Assume the following blocks are free. Rest of the blocks are allocated:

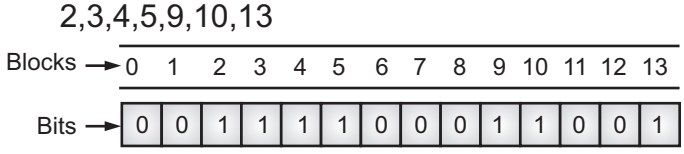


Fig. 8.11: Bit Vector Technique

Example 2: Consider a disk where blocks 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, and 27 are free and the rest of the blocks are allocated. The free-space bitmap would be: 00111100111111000110000001110000 0.

Advantages:

1. This technique is relatively simple.
2. This technique is very efficient to find the free space on the disk.

Disadvantages:

1. This technique requires a special hardware support to find the first 1 in a word it is not 0.
2. This technique is not useful for the larger disks.

8.5.2 Linked list

- This is another technique for free space management. In this linked list, the entire free block is maintained. In this, there is a *head pointer* which points the first free block of the list which is kept in a special location on the disk.
- This block contains the pointer to the next block and the next block contains the pointer of another next and this process is repeated. By using this list it is not easy to search the free list.

- This technique is not sufficient to traverse the list because we have to read each disk block that requires I/O time. So traversing in the free list is not a frequent action.

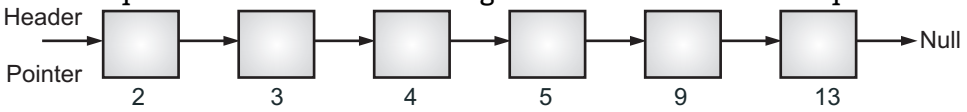


Fig 8.12: Example of Linked List

- In this example, we see that keep block 2 is the first free block which points to another block which contains the pointer of the 3 blocks and 3 blocks contain the pointer to the 4 blocks and this contains the pointer to the 5 block then 5 block contains the pointer to the next block and this process is repeated at the last .

Advantages:

1. Whenever a file is to be allocated a free block, the operating system can simply allocate the first block in free space list and move the head pointer to the next free block in the list.

Disadvantages:

1. Searching the free space list will be very time consuming; each block will have to be read from the disk, which is read very slowly as compared to the main memory.
2. Not Efficient for faster access.

8.5.3 Grouping

- This is also the technique of free space management. In this, there is a modification of the free-list approach which stores the address of the n free blocks. In this the first n-1 blocks are free but the last block contains the address of the n blocks.
- When we use the standard linked list approach the addresses of a large number of blocks can be found very quickly. In this approach, we cannot keep a list of n free disk addresses but we keep the address of the first free block.

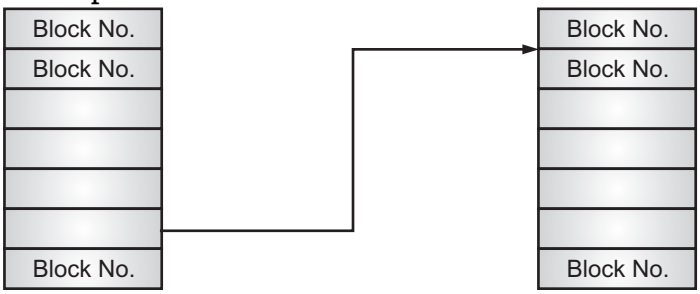


Fig. 8.13: Grouping Technique

8.5.4 Counting

- Counting is another approach for free space management. Generally, some contiguous blocks are allocated but some are free simultaneously.
- The free space is allocated to a process according to the contiguous allocation algorithm or clustering.
- So, we cannot keep the list of n free block address but we can keep the address of the first free block and then the numbers of n free contiguous block which follows the first block.

- There is an entry in the free space list that consist the address of the disk and a count variable. This method of free space management is similar to the method of allocating blocks.
- We can store these entries in the B-tree in place of the linked list. So the operations like lookup, deletion, insertion are efficient.

Summary

- A file is a named collection of related information that is recorded on a secondary storage. Commonly, files represent programs (source and object forms) and data.
- The file may have attributes like name, creator, date, type, permissions etc.
- There are various operations which can be implemented on a file such as create, write, delete, truncate, read, re-position etc.
- There are three ways to access a file into a computer system: Sequential-Access, Direct Access, Index sequential Method.
- In Sequential-Access, information in the file is processed in order, one record after the other.
- Direct access is very inefficient with linked allocation. Indexed allocation may require substantial overhead for its index block.
- File allocation methods define how the files are stored in the disk blocks.
- There are three main file allocation methods: Contiguous Allocation, Linked Allocation, Indexed Allocation.
- Contiguous allocation can suffer from external fragmentation.
- Free-space allocation methods also influence the efficiency of disk-space use, the performance of the file system, and the reliability of secondary storage.
- The methods used include bit vectors and linked lists. Optimizations include grouping, counting, and the FAT, which places the linked list in one contiguous area.

Check Your Understanding

1. _____ is a unique tag usually a number identifies the file within the file system.
   (a) File identifier (b) File name
   (c) File type (d) None of the mentioned
2. To create a file _____.
   (a) allocate the space in file system
   (b) make an entry for new file in directory
   (c) allocate the space in file system & make an entry for new file in directory
   (d) none of the mentioned
3. By using the specific system call, we can _____
   (a) open the file (b) read the file
   (c) write into the file (d) all of the mentioned

4. File type can be represented by _____
   (a) file name (b) file extension
   (c) file identifier (d) none of the mentioned
5. Which file is a sequence of bytes organized into blocks understandable by the system's linker?
   (a) object file (b) source file
   (c) executable file (d) text file
6. What is the mounting of file system?
   (a) creating of a filesystem.
   (b) deleting a filesystem.
   (c) attaching portion of the file system into a directory structure.
   (d) removing the portion of the file system into a directory structure.
7. Which one of the following explains the sequential file access method?
   (a) random access according to the given byte number.
   (b) read bytes one at a time, in order.
   (c) read/write sequentially by record.
   (d) read/write randomly by record.
8. When will file system fragmentation occur?
   (a) unused space or single file are not contiguous.
   (b) used space is not contiguous.
   (c) unused space is non-contiguous.
   (d) multiple files are non-contiguous.
9. The larger the block size, the _____ the internal fragmentation.
   (a) greater (b) lesser
   (c) same (d) none of the mentioned
10. In the sequential access method, information in the file is processed _____.
    (a) one disk after the other, record access does not matter.
    (b) one record after the other.
    (c) one text document after the other.
    (d) none of the mentioned.
11. Sequential access method _____ on random access devices.
    (a) works well
    (b) does not work well
    (c) maybe works well and does not work well
    (d) none of the mentioned
12. The direct access method is based on a _____ model of a file, as _____ allow random access to any file block.
    (a) magnetic tape, magnetic tapes (b) tape, tapes
    (c) disk, disks (d) all of the mentioned

13. For a direct access file _____.
    (a) there are restrictions on the order of reading and writing.
    (b) there are no restrictions on the order of reading and writing.
    (c) access is restricted permission wise.
    (d) access is not restricted permission wise.
14. A relative block number is an index relative to _____.
    (a) the beginning of the file (b) the end of the file
    (c) the last written position in file (d) none of the mentioned
15. The index contains _____.
    (a) names of all contents of file (b) pointers to each page
    (c) pointers to the various blocks (d) all of the mentioned
16. For large files, when the index itself becomes too large to be kept in memory?
    (a) index is called
    (b) an index is created for the index file
    (c) secondary index files are created
    (d) all of the mentioned

Answers

1. (a)	2. (c)	3. (d)	4. (b)	5. (a)	6. (c)	7. (b)	8. (a)	9. (a)	10. (b)
11. (a)	12. (c)	13. (b)	14. (a)	15. (c)	16. (b)				

Practice Questions

Q.I Answer the following questions in short:

1. Define the term file.
2. State various attributes of files.
3. What are the types of files?
4. What is meant by file allocation?
5. List basic operations on file.
6. What are the different types of access methods?
7. What is counting technique?

Q.II Answer the following questions:

1. With suitable diagram describe direct access method.
2. Describe the term Indexed allocation in detail.
3. With the help of diagram describe file structure.
4. With the help of diagram describe sequential access method.
5. What is meant by free space management?
6. Write a short note on Linked Allocation in file.
7. Compare sequential access and direct access.

8. Explain Direct Access method with advantages and disadvantages.
9. Explain contiguous Allocation method in detail.
10. List and explain any two operations that can be performed on file.
11. List and explain different attributes related to file.

Q.III Define terms:

1. Bit vector
2. Linked list
3. Grouping
4. File attributes
5. Directory

Previous Exam Questions

Summer 2018

1. Explain Direct Access method with advantages and disadvantages. **[4 M]**

Ans. Refer to Section 8.2.2.

2. Explain contiguous memory allocation method in detail. **[4 M]**

Ans. Refer to Section 8.4.1

Winter 2018

1. List various operations on files. **[2 M]**

Ans. Refer to Section 8.1.2.

2. Write a short note on File Directories. **[4 M]**

Ans. Refer to Section 8.3.1.

3. Explain Indexed Allocation briefly. **[4 M]**

Ans. Refer to Section 8.4.3.

4. What is meant by free space management? Define Bit vector and Grouping. **[4 M]**

Ans. Refer to Section 8.5.

Summer 2019

1. List any four attributes on files. **[2 M]**

Ans. Refer to Section 8.1.1.

2. Explain advantages and disadvantages of Linked allocation Method. **[4 M]**

Ans. Refer to Section 8.4.2.

■■■

9... I/O System

Objectives...

- To introduce concept of I/O System.
- To learn about I/O Hardware.
- To study Application of I/O Interface and Kernel I/O Subsystem.
- To get information of Disk Scheduling algorithms such as FCFS, Shortest Seek Time First, SCAN (Elevator Disk Algorithm), C-SCAN, C-LOOK.

9.1 INTRODUCTION

- Operating system controls all the computers I/O devices, it issues command to devices, catch interrupts and handler errors. It should provide the interface between device and rest of the system.
- The control of devices connected to the computer is a major concern of operating-system designers because I/O devices vary so widely in their function and speed varied methods are needed to control them and these methods form the I/O subsystem of the kernel, which separates the rest of the kernel from the complexities of managing I/O devices.
- The role of the operating system in computer I/O is to manage and control I/O operations and I/O devices.
- I/O device technology exhibits two conflicting trends one hand, we see increasing standardization of software and hardware interfaces. This trend helps us to incorporate improved device generations into existing computers and operating systems and on the other hand, we see an increasingly broad variety of I/O devices.
- Some new devices are so unlike previous devices that it is a challenge to incorporate them into our computers and operating systems. This challenge is met by a combination of hardware and software techniques.
- The basic I/O hardware elements, such as buses, ports and device controllers, accommodate a wide variety of I/O devices. To encapsulate the details and oddities of different devices, the kernel of an operating system is structured to use device driver modules.
- The device drivers present a uniform device-access interface to the I/O subsystem, much as system calls provide a standard interface between the application and the operating system.

- The I/O devices with computer system can be roughly group into three categories:
  - **Human readable:** Use for communicating with the computer uses.
    Examples: printer, keyboard.
  - **Machine readable:** Use for communicating with electronic equipment.
    Examples: disk and tape drives, sensors, controllers.
  - **Communication:** Use for communicating with remote devices.
    Example: modems.
- These are three techniques for performing I/O:
  - **Programmed I/O:** The processor issues an I/O command, on behalf of process, to an I/O module; that process then busy waits for the operation to be completed before proceeding.
  - **Interrupt-driven I/O:** The processor issues an I/O command, on behalf of process, continues to execute next instructions, and is interrupted by the I/O module when the latter has completed its work.
  - **Direct Memory Access (DMA):** A DMA module controls the exchange of data between main memory and an I/O module. The processor sends a request for the transfer of a block of data to the DMA module and is interrupted only after the entire block has been transferred I/O system organization.
- In modern operating system, device management is implemented either through the interaction of device driver and interrupt routine called interrupt driven I/O or wholly within device driver if interrupt are not used called direct I/O with polling.
- Fig. 9.1 shows the unit involved in I/O operation for both approaches.

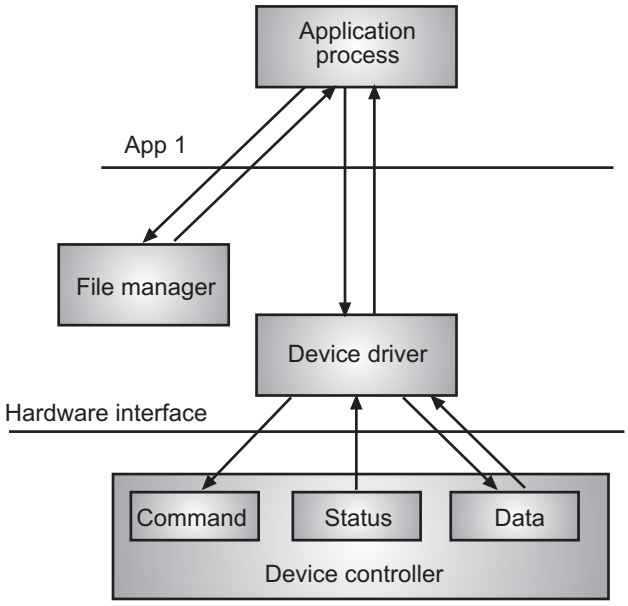


Fig. 9.1: I/O Operations

- An application process uses device by issuing commands and exchanging data with the device management device drives. The device driver has two major responsibilities.

- Implement an abstract application programming interface (API) to the application process.
- Provide device dependent operations to issue appropriate command to implement functions defined on the API.

9.2 I/O HARDWARE [S-18]

- Basically, computers operate a great many kinds of devices. Most fit into the general categories of storage devices such as disks, tapes etc. transmission devices such as network cards, modems etc. and human-interface devices such as screen, keyboard, mouse and so on.
- Devices communicate with the computer via signals sent over wires or through the air.
- Devices connect with the computer through **ports**, e.g. a serial or parallel port.
- **Bus:** If a device uses common set of wires it is called as bus. A Bus is nothing but set of wires with protocol that specifies set of messages that can be sent over the wires.
- Buses are widely used in computer architecture, and they vary in signaling methods, their speed, throughput and connection methods.
- A Common PC bus structure is given in Fig. 9.2.
- **PCI bus** connects processor-memory subsystem to the fast devices and expansion bus that connects relatively slow devices like keyboard, serial and USB ports.
- Disks are connected with **SCSI bus** plugged into SCSI Controller. Other buses are also used to connect main parts of computer like PCI-X bus and PCI-e (express) bus.
- A **Controller** is collection of electronics that operate on port, bus or device.
- A **Serial Port controller** is simple device controller. It is a single chip in computer that controls signals on wire of serial port.
- **SCSI controllers** are complex; it is often implemented as separate circuit board, that is plugged in to computer. It typically contains processor, microcode and some private memory to enable it to process the SCSI protocol message.
- **Disk controller** has one or more registers for data and control signals. The processor communicates with the controller by reading and writing bit patterns in these registers. One way in which this communication can occur is through the use of special I/O instructions that specify the transfer of a byte or word to an I/O port address.
- Another technique for communicating with devices is ***memory-mapped I/O.***
- In this case, the I/O instruction triggers bus lines to select the proper device and to move bits into or out of a device register. Alternatively, the device controller can support memory-mapped I/O. In this case, the device-control registers are mapped into the address space of the processor. The CPU executes I/O requests using the standard data-transfer instructions to read and write the device-control registers.
- Some computer systems use both techniques. For instance, PCs use I/O instructions to control some devices and memory-mapped I/O to control others.

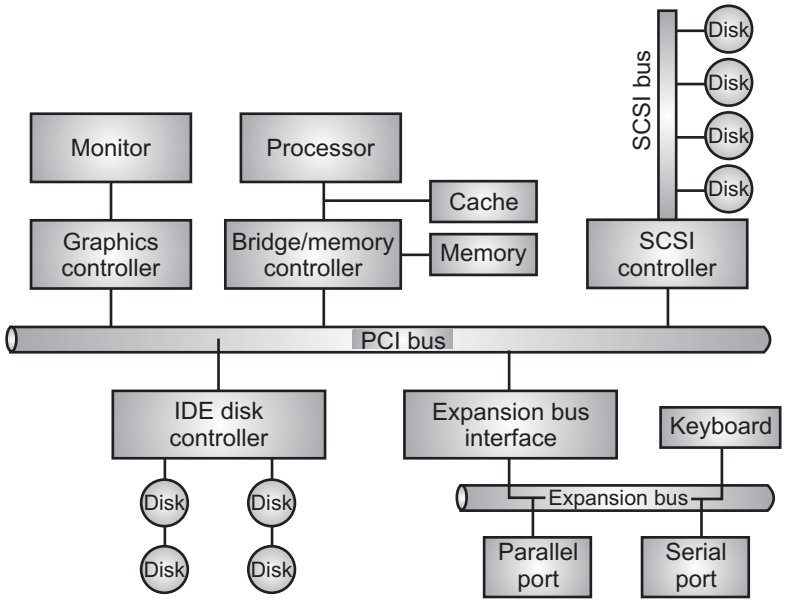


Fig. 9.2: A typical PC bus structure

- Table 9.1 shows the usual I/O port addresses for PCs. The graphics controller has I/O ports for basic control operations, but the controller has a large memory mapped region to hold screen contents.
- The process sends output to the screen by writing data into the memory-mapped region then the controller generates the screen image based on the contents of this memory. This technique is simple and easy to use.

Table 9.1: Most common Device I/O Port Address Ranges

I/O address range (hexadecimal)	Device
000-00F	DMA controller
020-021	Interrupt controller
040-043	Timer
200-20F	Game controller
2F8-2FF	Serial port (Secondary)
320-32F	Hard-disk controller
378-37F	Parallel port
3D0-3DF	Graphics controller
3F0-3F7	Diskette-drive controller
3F8-3FF	Serial port (primary)

- An I/O port typically consists of four registers called, status, control, data-in, and data-out registers.
  1. **The data-in register:** It is read by the host to get input.
  2. **The data-out register:** It is written by the host to send output.

3. **The status register:** It contains bits that can be read by the host. These bits indicate states.
4. **The control register:** It can be written by the host to start a command or to change the mode of a device.

- The data registers are typically 1 to 4 bytes in size. Some controllers have FIFO chips that can hold several bytes of input or output data to expand the capacity of the controller beyond the size of the data register.

Direct I/O with Polling:

- In direct I/O method, to do I/O, the CPU is responsible for transferring the data between the primary memory and the device controller data registers.
- While managing I/O, the device manager may poll the device busy-done flags or use interrupts to detect the operations completion.
- Steps required to do an input operation using polling (Refer Fig. 9.3):
  (i) The application process requests a read operation.
  (ii) The device driver queries the status register to determine if the device is idle. If the device is busy, the driver waits for it to become idle.
  (iii) The driver stores an input command into the controller's command register, thereby starting the device.
  (iv) The driver repeatedly reads the status register while waiting for the device to complete its operations.
  (v) The driver copies the contents of the controller's data registers into the user process space.
- The steps to perform an output operation are as follows:
  1. The application process requests a write operation.
  2. The device driver queries the status register to determine if the device is idle. If device is busy, the driver waits for it to become idle.

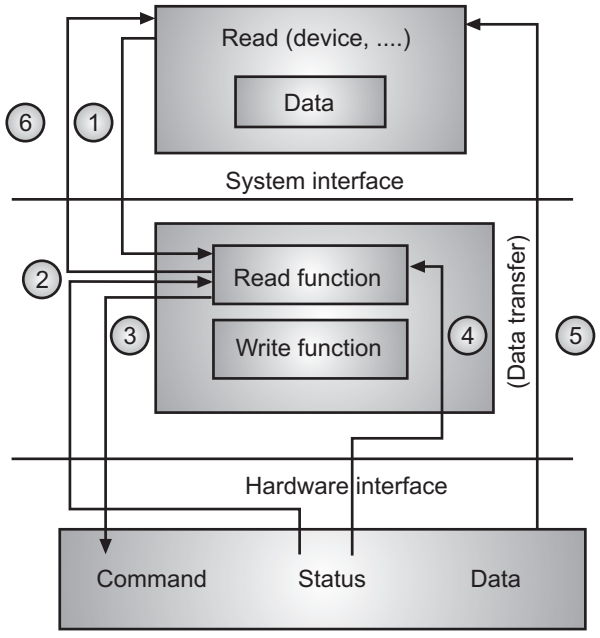


Fig. 9.3: I/O operations using Polling

3. The driver copies data from user space memory to the controller's data registers.
4. The driver stores an O/P command into the command register, thereby starting the device.
5. The driver repeatedly read the status register while waiting for the device to complete its operation.

Interrupts: **[S-19]**

- The motivation for incorporating interrupt into the computer hardware is to eliminate the need for the device driver to constantly poll the controller status register.
- In the scenario using interrupts, the device management functionality is partitioned into four different parts:
  1. The device driver that initiates the I/O operation.
  2. Device status table.
  3. Interrupt handler.
  4. Device handler.
- The following are the steps for performing an input instruction in a system by using interrupts, (Refer Fig. 9.3).
  1. The application process requests a read operation.
  2. The top half of device driver queries the status register to determine if device is idle. If the device is busy, the driver waits for the device to become idle.
  3. The driver stores an input command into the controllers command register, thereby starting the device.
  4. When the top half of the device driver completes it work, it saves information regarding the operation that it began in the Device Status Table. This table contains an entry for each device in the system. The top half of the driver writes information into the entry for the device it is using such as the return address of the original call and any special parameters for the I/O operation. The CPU then can be used by another program, so the device manager invokes the scheduler past of the process manager. It then terminates.
  5. Eventually the device completes the operation and interrupts the CPU, thereby causing the interrupt handler to run.
  6. The interrupt handler determines which device causes the interrupt. It then branches to the device handler for that device.
  7. The device handler retrieves the pending I/O status information from the device status table.
  8. The device handler copies the contents of the controller's data registers into the user processes space.
  9. The device handler invoked by the application process return control to the application process. The output process behaves similarly.

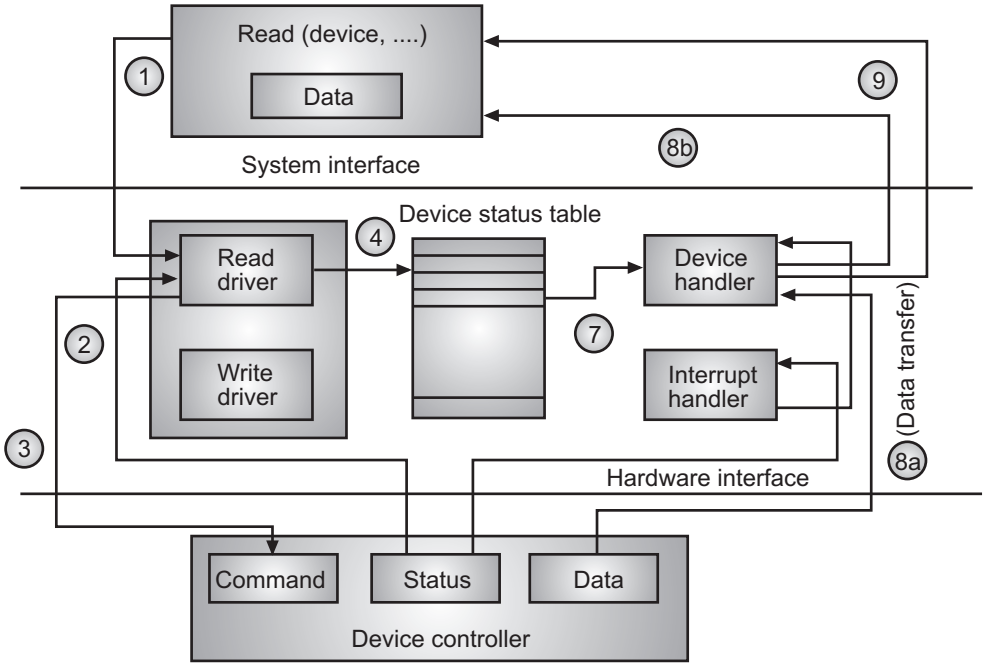


Fig. 9.4: I/O operations using interrupt

Direct Memory Access (DMA): **[S-19]**

- For a device that transfers large quantities of data, such as disk drive, it is convenient to use an expensive general purpose processor to watch status bit and to feed data into controller register one byte at a time.
- Many computers avoid burdening the main CPU by off-loading some of this work to a special purpose processor called **Direct Memory Access (DMA) controller**.
- The DMA controller has access to system bus independent of the CPU as shown in Fig. 9.5.

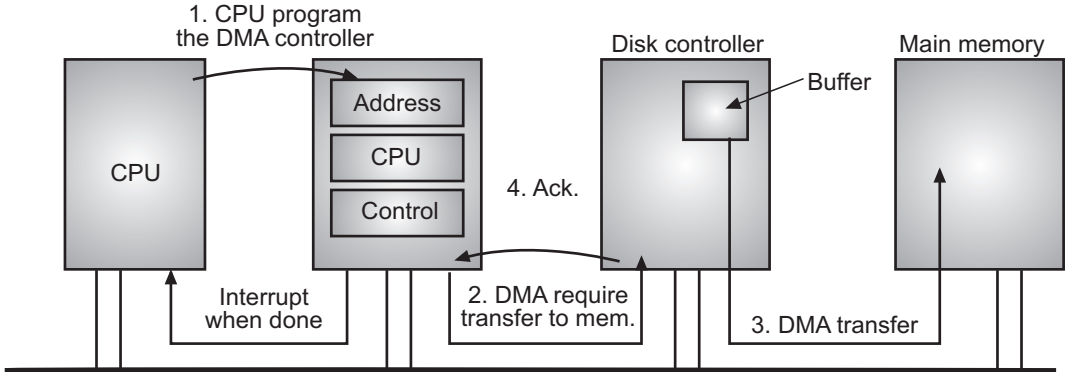


Fig. 9.5: DMA Controller

- DMA contains several registers that can be written and read by the CPU.
- This includes a memory address register, a byte count register and one or more control registers.
- The control registers specify the I/O port to use, the direction of the transfer, the transfer unit and the number of bytes to transfer in one burst.

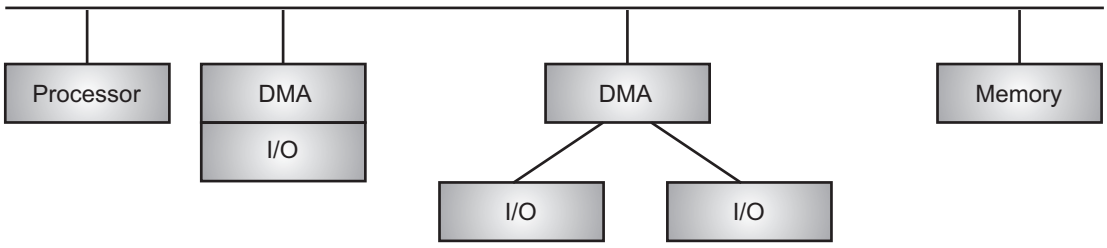
Steps in DMA transfer:

1. The CPU programs the DMA controller by setting its registers, so it knows what to transfer. It also issues a command to the disk controller to read data from the disk into it internal buffer and verify the checksum. When valid data are in the disk controller's buffer, DMA can begin.
2. The DMA controller initiates the transfer by issuing read request over the bus to the disk controller.
3. The memory address to write to is on the bus address lines so when the disk controller fetches the next word from its internal buffer. It knows where to write it. The write to memory is another standard bus cycle.
4. When the write is complete, the disk controller sends an acknowledgment signal to the disk controller, also over the bus.
5. The DMA controller then increments the memory address to use and decrements the byte count. If the byte count is still greater than 0, step 2 through 4 are repeated until the count reaches 0.
6. At that time, the DMA controller interrupts the CPU to let it know that the transfer is now complete.

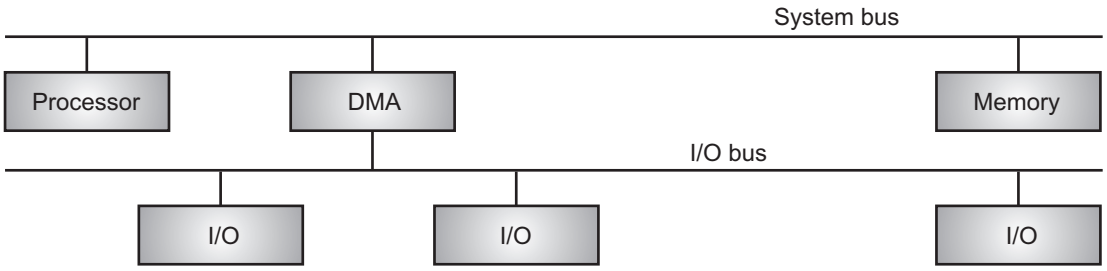
DMA Mechanism:

- The DMA mechanism can be configured in a variety of ways. Some possibilities are shown in Fig. 9.6.
- In the first example (Fig 9.6(a)), all modules share the same system bus. The DMA module, acting as a surrogate processor, uses programmed I/O to exchange data between memory and I/O module through DMA module.
- This configuration may be inexpensive but is inefficient. As with processor controlled programmed I/O, each transfer of a word consumes two bus cycles.
- The number of required bus cycles can be substantially by integrating the DMA and I/O functions.
- As in Fig. 9.6(b) indicates, this means that there is a path between the DMA module and one or more I/O modules that does not include the system bus.
- The DMA logic may actually be a part of an I/O module, or it may be separate module that controls one or more I/O modules.
- This concept can be taken on one step further by connecting I/O modules to the DMA module using an I/O bus, (Fig. 9.6(c)). This reduces the number of I/O interfaces in the DMA module to one and provides for an easily expandable configuration.

(a) Single bus, detached DMA



(b) Single bus, integrated DMA - I/O system bus



(c) I/O bus

Fig. 9.6: Configuration of DMA Mechanism

9.3 APPLICATION I/O INTERFACE

- Application I/O Interface represents the structuring techniques and interfaces for the operating system to enable I/O devices to be treated in a standard, uniform way.
- The actual differences lies kernel level modules called **device drivers** which are custom tailored to corresponding devices but show one of the standard interfaces to applications. The purpose of the device-driver layer is to hide the differences among device controllers from the I/O subsystem of the kernel, such as the I/O system calls.
- Fig. 9.7 shows the I/O-related portions of the kernel are structured in software layers.
- The purpose of the device-driver layer is to hide the differences among device controllers from the I/O subsystem of the kernel, much as the I/O system calls encapsulate the behavior of devices in a few generic classes that hide hardware differences from applications.
- Making the I/O subsystem independent of the hardware simplifies the job of the operating-system developer. It also benefits the hardware manufacturers.
- They either design new devices to be compatible with an existing host controller interface or they write device drivers to interface the new hardware to popular operating systems for this reason we can attach new peripherals to a computer without waiting for the operating-system vendor to develop support code.

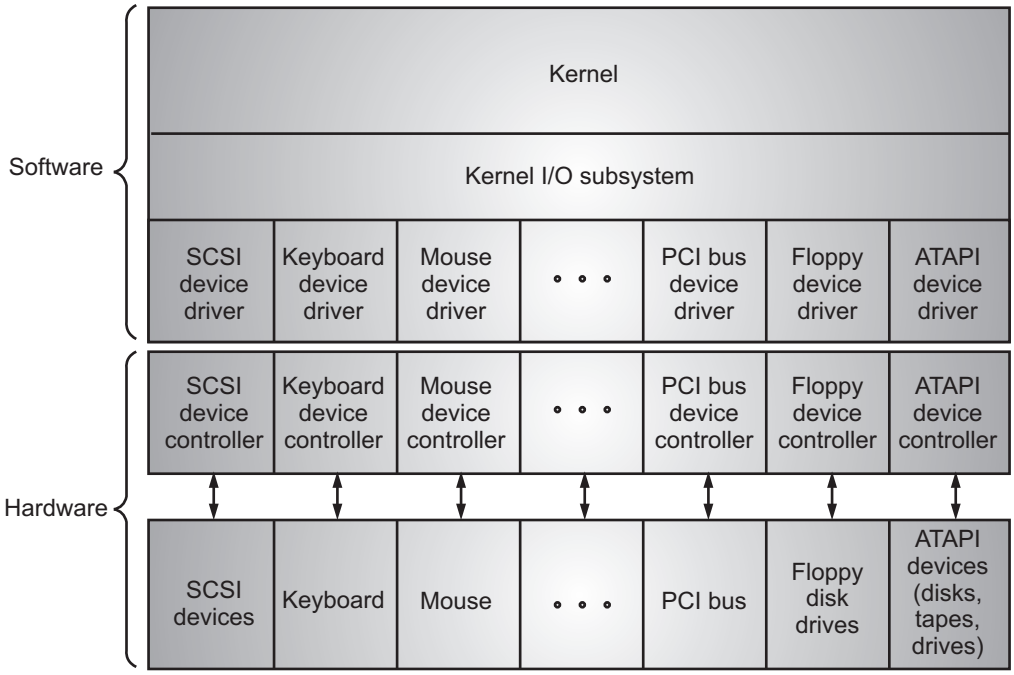


Fig. 9.7: A Kernel I/O structure

- Unfortunately, for device-hardware manufacturers, each and every type of operating system has its own standards for the device-driver interface.
- A given device may ship with multiple device drivers for instance, drivers for Windows NT/2000, MS-DOS, Windows 95/98/2000 and Solaris. Devices vary on many dimensions, as shown in Table 9.2.
  - **Character-stream or Block:** This device transfers bytes one by one, whereas a block device transfers a block of bytes as a unit.
  - **Sequential or Random access:** This device transfers data in a fixed order determined by the device, whereas the user of a random-access device can instruct the device to seek to any of the available data storage locations.
  - **Synchronous or Asynchronous:** This device performs data transfers with predictable response times. An asynchronous device exhibits irregular response times.
  - **Sharable or Dedicated:** This device can be used concurrently by several processes or threads; a dedicated device cannot.
  - **Speed of operation:** Device speeds range from a few bytes per second to a few gigabytes per second.
  - **Read-write, Read only or Write only:** Some devices perform both input and output, but others support only one data transfer direction.

Table 9.2: Characteristics of I/O devices

Aspect	Variation	Example
data-transfer mode	character block	terminal disk
access method	sequential random	modem CD-ROM
transfer schedule	synchronous asynchronous	tape keyboard
device speed	latency seek time transfer rate delay between operations	
I/O direction	read only write only read-write	CD-ROM graphics controller disk

- Most devices can be characterized as either Block I/O, Character I/O, Memory mapped file access, or Network sockets.
- Operating systems also provide special system calls to access a few additional devices, such as a time-of-day clock and a timer. Some operating systems provide a set of system calls for graphical display, video and audio devices.
- Most operating systems also have an escape or back door, which allows applications to send commands directly to device drivers if needed.

9.4 KERNEL I/O SUBSYSTEM

- Kernel provides different services related with I/O such as scheduling, caching, spooling, device reservation, and error handling.
- The I/O sub system is also responsible for protecting itself from errant processes and malicious users.

1. I/O Scheduling:

- It means to determine a good order in which to execute input/output requests called **Disk Scheduling**. Disk scheduling is also known as I/O scheduling.
- Scheduling I/O requests can greatly improve overall efficiency. Priorities can also play important role in request scheduling. It can reduce average waiting time for I/O completion process.
- By this device access, permissions will be fair to all jobs, when application issues a blocking I/O system call, the request is placed on the job queue for that device. The order of the queue rearranges by the kernel I/O scheduler to improve the overall system efficiency.

2. Buffering:

- A buffer is an area of memory that stores data when they are moved between two devices or between a device and an application.
- Buffering is done for following three reasons:
  - **Speed differences between two devices:** A slow device may write data into a buffer, and when the buffer is full, the entire buffer is sent to the fast device all at once. So that the slow device still has somewhere to write while this is going on, a second buffer is used, and the two buffers alternate as each becomes full. This is known as **double buffering.**
  - **Data transfer size differences:** To provide adaptation for data that have different data-transfer sizes.
  - **To support copy semantics:** Third use of buffering is to support copy semantics for the application I/O, "copy semantic" means, suppose that an application wants to write data on a disk that is stored in its buffer. It calls the write() system's call, providing a pointer to the buffer and the integer specifying the number of bytes to write.

3. Caching:

- A cache is a fast memory area that contains copy data. Access to a cached copy is more efficient than access to the original data. For example, instructions from the current process are stored on disk, cached in physical memory, and then copied again into the secondary and primary cache of the CPU.
- The difference between a buffer and an cache is that the buffer can store the only data information whereas a cache by definition only stores a data from a place to be accessed faster.
- Caching and buffering are two different functions, but sometimes a memory area can be used for both. For example, to save on semantics copy and make scheduling I/O efficient, the operating system uses a buffer in main memory to store data.
  - This buffer is also used as cache, to improve the efficiency of I/O for files that are shared by several applications, or that are being read and written repeatedly.
  - When the kernel receives an I/O file request, the kernel accesses the buffer cache to see whether memory area is available in main memory.
  - If it is available, a physical disk I/O can be avoided or it is not used. Disk writing also accumulates into the buffer cache for few seconds, so large transfer will be collected to streamline the writing schedule.

4. Device Spooling and Reservation:

- A spool is a buffer that stores output for a device, such as a printer, that cannot accept interleaved data streams. Even though the printer can only serve one job at a time, some applications can require the printer to print, without having to get their output mixed together.

- The operating system will solve this problem by intercepting all output to the printer. Each application output has been spooled to a different disk file. When an application finishes printing, the spooling system will continue to the next queue. In some operating systems, spooling is handled by a process daemon system. On other operating systems, this system is handled by an in-kernel thread.
- For some devices, such as screen drives and printers, it cannot multiplex I/O requests from some applications. Spooling is one way to overcome this problem. Another way is to divide coordination for multiple concurrent.
- Some operating systems provide support for exclusive device access, by allocating processes to idle devices and removing devices that are no longer needed. Other operating systems impose limits on a file to handle this device. Many operating systems provide functions that make the process of handling coordinates exclusive access among them.

5. Error Handling:

- An operating system that uses protected memory can guard against many possible errors due to hardware or applications. Devices and I/O transfers can fail in many ways, because of transient reasons, such as overloaded on the network, or permanent reasons such as damage to the disk controller.
- Operating systems can often compensate for transient errors. Like, a read error on the disk will cause a reread and a transmission error on the network will result in a re transmission if the protocol is known. However, for permanent errors, the operating system in general will not be able to restore the situation to normal.
- A general rule, an I/O system will return one bit of information about the status of the call, which will indicate whether the process is successful or failed.
- The operating system on UNIX uses an additional integer called errno to return an error code of about 1 out of 100 values that indicate the cause of the error. However, some hardware can provide detailed error information, even though many operating systems do not support this facility.

6. I/O Protection:

- Errors and the issue of protection are closely related. A user process may attempt to issue illegal I/O instructions to disrupt the normal function of a system. We can use the various mechanisms to ensure that such disruption cannot take place in the system. To prevent illegal I/O access, we define all I/O instructions to be privileged instructions. The user cannot issue I/O instruction directly.

9.5 DISK SCHEDULING [W-18]

- Over many years, the increase in the speed of processor and main memory put impact on disk access.
- This gap between speed of processor, main memory and disk is expected to continue into the foreseeable future. Thus, the performance of disk storage subsystem is important and much research has goes into schemes for improving that performance.
- Because the performance of the disk system is tied closely to file system design issues, discuss in next sections.

Disk Performance Parameters:

- The actual details of disk I/O operation depend on the computer system, the operating system and the nature of the I/O channel and disk controller hardware.
- A general timing diagram of disk I/O transfer is shown in Fig. 9.8.

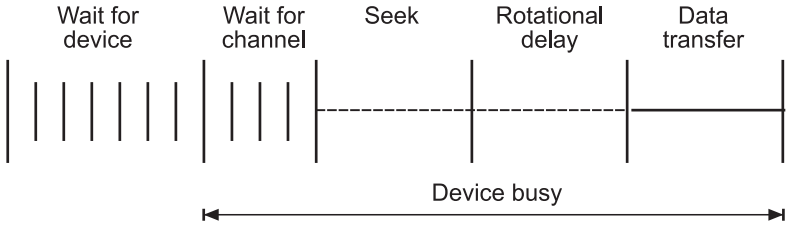


Fig. 9.8: Timing of a Disk I/O Transfer

- When the disk drive is operating, the disk is rotating at constant speed. To read or write, the head must position at desired track and at beginning of desire sector on the track.
- Track selection involves moving the head in movable head system or electronically selecting one head on a fixed-head system. On a movable-head system, the time it takes to position at track is known as seek time.
- In either case, once the track is selected the disk controller waits until the appropriate sector rotates to line up with the head. The time it takes for beginning of the sector to reach the head is known as rotational delay or rotational latency.
- The sum of the seek time, if any and the rotational delay is the access time.
- Once, head is positioned, the read or write operation is then performed as the sector moves under the head; this is the data transfer portion of the operation.
- In addition to the access time and transfer time, there are several queuing delay associate with I/O operation. When a process issues an I/O request, it must first wait in a queue for the device to be available.
- At that time a device is assigned to the process. If device shares a single channel or set of I/O channels with other disk drives, then there may be an additional wait for the channel to be available. At that point, seek is performed to begin disk access.

Seek Time:

- Seek time is the time required to move the disk arm to the required tack. The seek time consists of two key component – the initial startup time and the time taken to traverse the tracks that have to be crossed once the access is up to speed.
- Unfortunately, the traversal time is not linear and settling time. Much improvement comes from smaller and lighter disk components.

Rotational Delay:

- Magnetic disk, other than floppy disk, have rotational speed in the range 5400 to 10,000 r.p.m., the latter equivalent to one revolution per 6 ms. Thus, at 10,000 r.p.m., the average rotational delay will be 3 ms.
- Floppy disk rotates at between 300 and 600 r.p.m. The average delay will be between 100 and 200 ms.

Transfer Time:

- The transfer time to or from the disk depends on the rotation speed of the disk in the following manner:

$$T = \frac{b}{rN}$$

where, T = Transfer time

b = Number of bytes to be transferred

N = Number of bytes on a track

r = Rotation speed in revolution per second

Thus, the Total Average Access Time can be expressed as,

$$T_a = T_s + \frac{1}{2r} + \frac{1}{rN}$$

where, T_s is the average seek time.

Disk Access Time:

Disk access time is given as,

Disk Access Time = Rotational Latency + Seek Time + Transfer Time

Disk Response Time:

- It is the average of time spent by each request waiting for the I/O operation.

Disk Scheduling Algorithm:

- In Multiprogramming environment, the operating system maintains a queue of requests for each I/O device. So, for a single disk, there will be a number of I/O request from various processes in queue.
- If we selected request from the queue in random order, then tracks to be visited will occur randomly, giving the worst possible performance. Thus, there is the necessity of good algorithms.

9.5.1 FCFS (First Come First Serve) [W-18]

- The simplest scheduling algorithm is FCFS (First Come First Serve) which simply means that process request from queue in sequential order. This algorithm is quite simple.

Example 1: Fig. 9.9 illustrates the disk arm movement with FCFS. In this example, we assume a disk with 200 tracks and then the disk request queue has random requests in it. The requested tracks, in the order received are 55, 58, 39, 19, 90, 160, 150, 38, 184. Initial position of head is at 100 tracks.

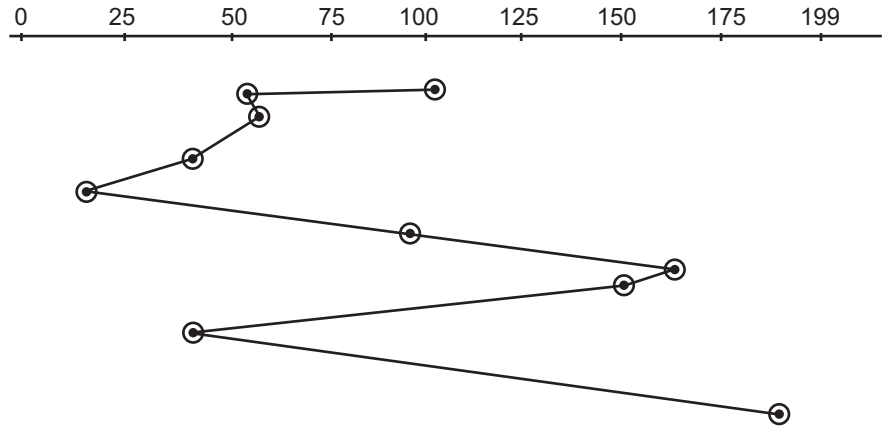


Fig. 9.9: FCFS (Starting at Track 100)

Solution: The average seek length is,

Next Track Accessed	Number of Track Traversed
55	45
58	3
39	19
18	21
90	72
160	70
150	10
38	112
184	146
Average Seek Length	**55.3**

With FIFO, if there are few processes the require access and if many of the requests are to clustered file sectors, then we can hope for good performance.

However, this technique will often approach random scheduling in performance, if these are many processes competing for the disk. Thus, there is a necessity of more sophisticated scheduling policy.

Example 2: Assume there are total 0-199 tracks that are present on each surface of the disk. If request queue is: 68, 172, 4, 178, 130, 40, 118, 136 and initial position of the head is 25. Apply FCFS disk scheduling algorithm and calculate total head movement.

Solution: The requested tracks, in the order received are 68, 172, 4, 178, 130, 40, 118, 136. Initial position of head is 25 tracks.

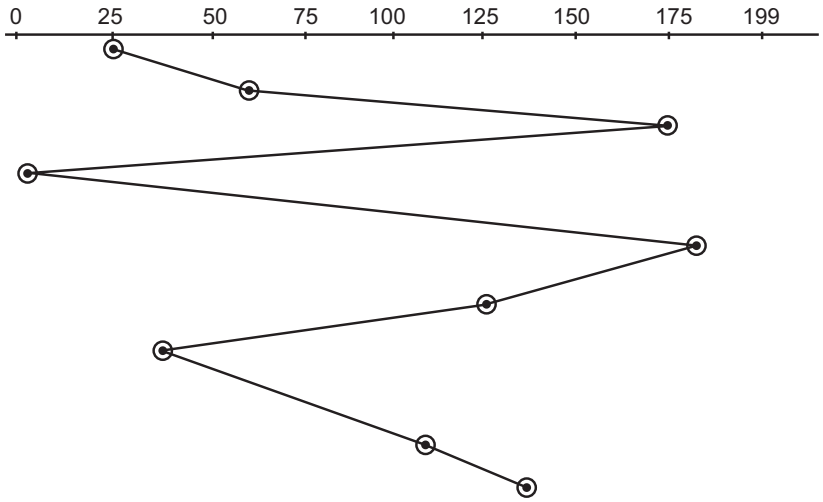


Fig. 9.10

The Total Head Movements:

Next Track Accessed	Number of Track Traversed
25	0
68	43
172	104
04	168
178	174
130	48
40	90
118	78
136	18
Total Head Movements	**723**

Example 3: Suppose the order of request is- (82,170,43,140,24,16,190) And current position of Read/Write head is : 50.

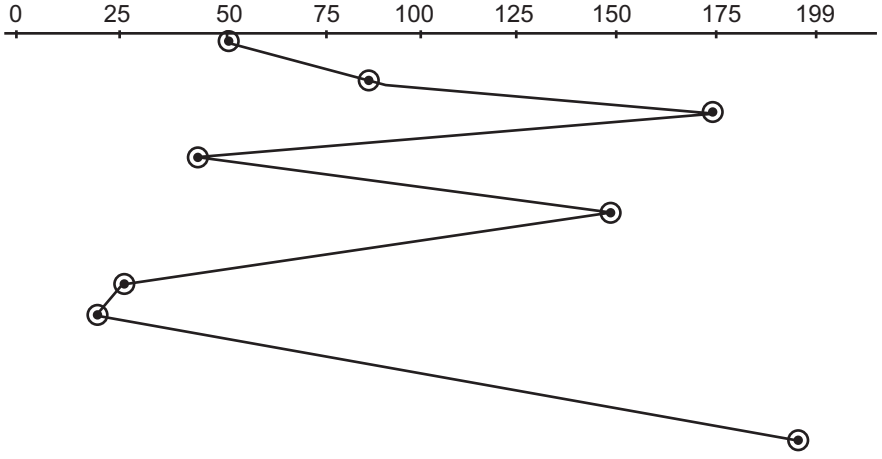


Fig. 9.11

So, total seek time:

= (82 –50) + (170 – 82) + (170 – 43) + (140 – 43) + (140 – 24) + (24 – 16) + (190 – 16)

= 642

Example 4: The request queue is as follows:

28, 136, 15, 185, 50, 197

Number of tracks = 0 to 199.

Starting position or current head position = 134.

Find total head movement by applying FCFS (First Come First Serve) disk scheduling algorithm.

Solution: The requested tracks, in the order received are 28, 136, 15, 185, 50, 197. Initial position of head is at 134 tracks.

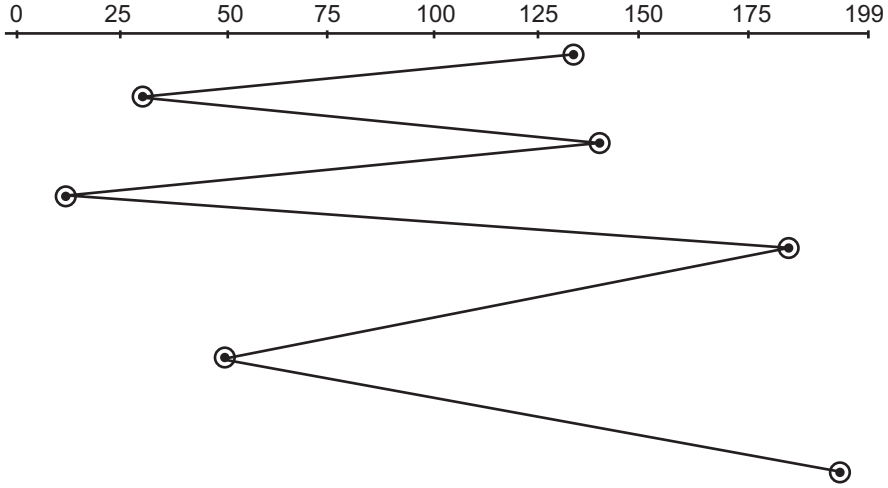


Fig. 9.12: FCFS disk scheduling

Next Tract Accessed	Number of Tracks Traversed
28	106
136	108
15	121
185	170
50	135
197	147
Total Head Movements	**787**

9.5.2 Shortest-Seek-Time-First (SSTF) Algorithm [S-18]

- SSTF selects the request with the minimum seek time from the current head position.
- SSTF scheduling is a form of SJF scheduling; may cause starvation of some requests.

Example 5: Suppose the order of queue is (98, 183, 37, 122, 14, 124, 65, 67) and current position of Read/Write head is : 53

Solution: (53 – 65) + (65 – 67) + (67 – 37) + (37 – 14) + (98 – 14) + (122 – 98) + (124 – 122) + (183 – 124)

Total head movement = 236 cylinders.

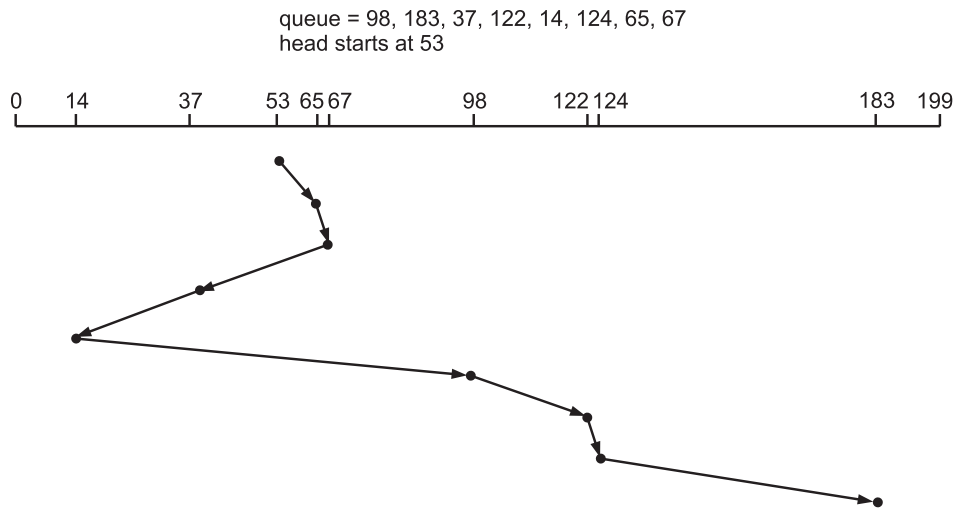


Fig. 9.13: SSTF

Example 6: The request queue is as follows: 15, 27, 137, 18, 150, 65, 194

Number of tracks = 0 to 199.

Starting position or current head position = 128

Find total head movement by applying SSTF (Shortest Seek Time First) disk scheduling algorithm.

Solution: The requested tracks, in the order received are 15, 27, 137, 18, 150, 65, 194. Initial position of head is at 128 tracks.

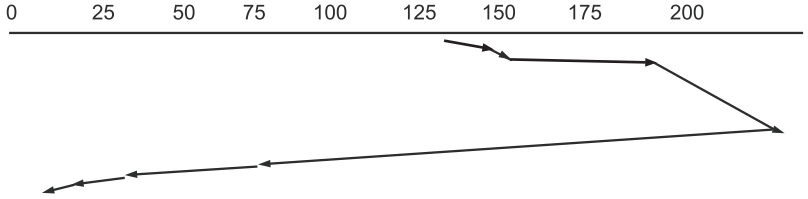


Fig. 9.14

The Total Head Movements:

Next Track Accessed	Number of Track Traversed
128	0
137	9
150	13
194	44
65	129
27	38
18	9
15	3
Total Head Movements	**245**

Example 7: The request queue is as follows:

87, 148, 92, 171, 96, 131, 103, 71

Number of tracks = 0 to 199.

Starting position or current head position = 125

Find total head movement by applying SSTF (Shortest Seek Time First) disk scheduling algorithm.

Solution: The requested tracks, in the order received are 87, 148, 92, 17, 96, 131, 103, 71
Initial position of head is at 125 tracks.

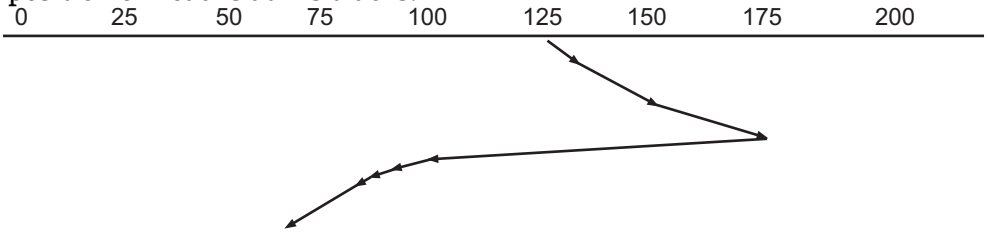


Fig. 9.15

The Total Head Movements:

Next Track Accessed	Number of Track Traversed
125	0
131	6
148	17
171	23
103	68
96	07
92	04
87	05
71	16
Total Head Movements	**146**

In SSTF algorithm, select the disk I/O request that requires the list movement of the disk arm from the current position. Thus, we incur the minimum seek time. Of course, always choosing the minimum seek time does not guarantee that the average seek time over a number of arm movement will be minimum.

This should provide better performance than FIFO. Because the arm can move in the two directions. FIFO algorithm may be used to resolve cases of equal distances.

Example 8: Fig. 9.16 illustrates the disk arm movement with SSTF, on the same example as was used for FIFO.

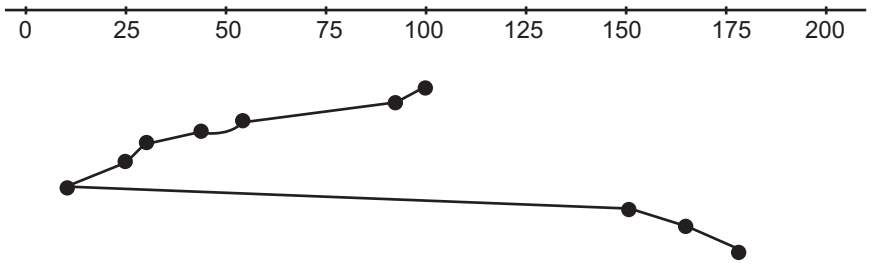


Fig. 9.16: SSTF (Starting at track 100)

Solution: The average seek length is,

Next track accessed	Number of track traversed
90	10
58	32
55	3
39	16
38	1
18	20
150	132
160	10
184	24
Average Seek Length	**27.5**

Unfortunately, SSTF has a problem. Suppose more requests keep coming in while the request of Fig. 9.16 is being processed.

For example, if after going to track 38, a new request for 32 is present, that request will have priority over track 18. If a request for track 25 then comes in, the arm will next go to 25 instead of 18.

With a heavily loaded disk, the arm will tend to stay in middle of disk most of the time, so request at either extreme will have to wait. Requests far from the middle may get poor service.

The goals of minimal response time and fairness are in conflict here. A simple alternative that prevents this sort of starvation is the SCAN algorithm.

9.5.3 SCAN (Elevator Disk Algorithm) [S–19]

- In SCAN, the disk arm starts at 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.
- Sometimes, called the **Elevator Algorithm.** This algorithm does not give uniform wait time.
- Fig. 9.17 illustrates total head movement of 208 cylinders.
  Total head movement = 208 cylinders.

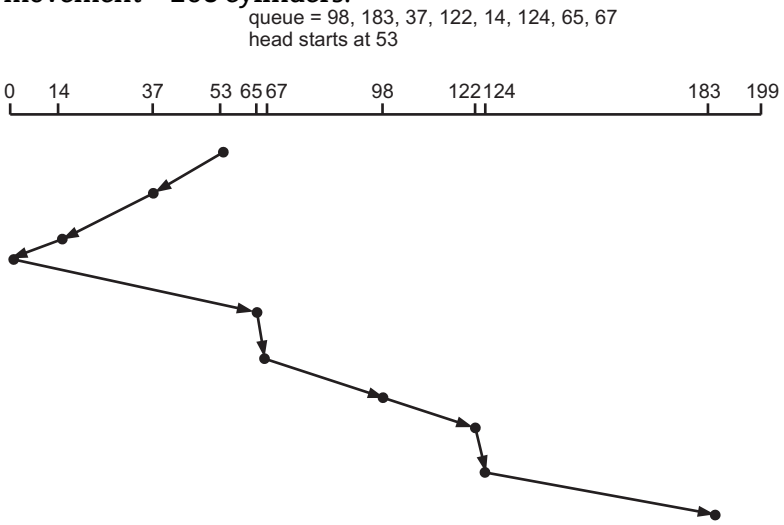


Fig. 9.17: SCAN Disk Algorithm

- In SCAN algorithm, the arm is required to move in one direction only, satisfying all outstanding request in route, until it reaches the last track in that direction or until there are no more requests in that direction.

- The service direction is then reversed and the scan proceed is opposite direction, again picking up all requests in order Fig. 9.18 illustrates the disk arm movement with SCAN, on previous example.

Example 9:

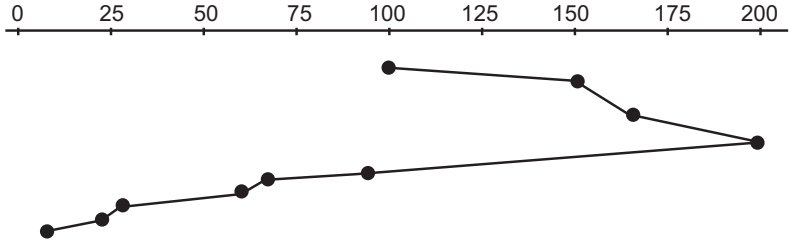


Fig. 9.18: SCAN (Starting at track 100, in the direction of increasing track number)

Solution: The average seek length is,

Next track accessed	Number of track traversed
150	50
160	10
184	24
90	94
58	32
55	3
39	16
38	1
18	20
Average Seek Length	**27.8**

Example 10: The request queue is as follows:

86, 147, 91, 170, 95, 130, 102, 70

Number of Tracks: 0 to 199

Starting position or current head position = 125

Find total head movement by applying SCAN disk scheduling algorithm.

Solution:

The requested tracks, in the order received are 86, 147, 91, 170, 95, 130, 102, 70

Initial position of head is at 125 tracks.

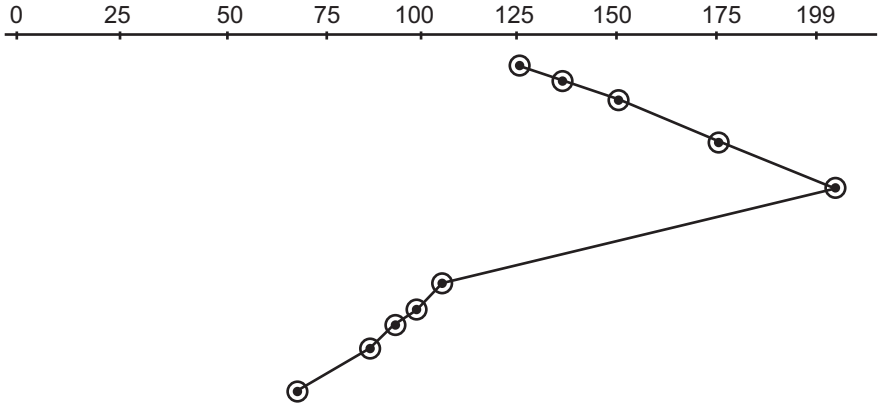


Fig. 9.19: C-SCAN Scheduling

The Total Head Movements:

Next Track Accessed	Number of Track Traversed
125	0
130	5
147	17
170	23
102	68
95	07
91	04
86	05
70	16
Total Head Movements	**145**

SCAN algorithm behaves almost identically with SSTF policy. However, this is a static example in which no new items are added to the queue even when the queue is dynamically changing.

SCAN will be similar to SSTF unless the request pattern is unusual.

SCAN policy favours jobs where requests use for tracks nearest to both innermost and outermost tracks. The problem can be avoided via C-SCAN algorithm.

9.5.4 C-SCAN

- C-SCAN scheduling algorithm provides a more uniform wait time than SCAN algorithm.
- Treats the cylinders as a circular list that wraps around from the last cylinder to the first one.

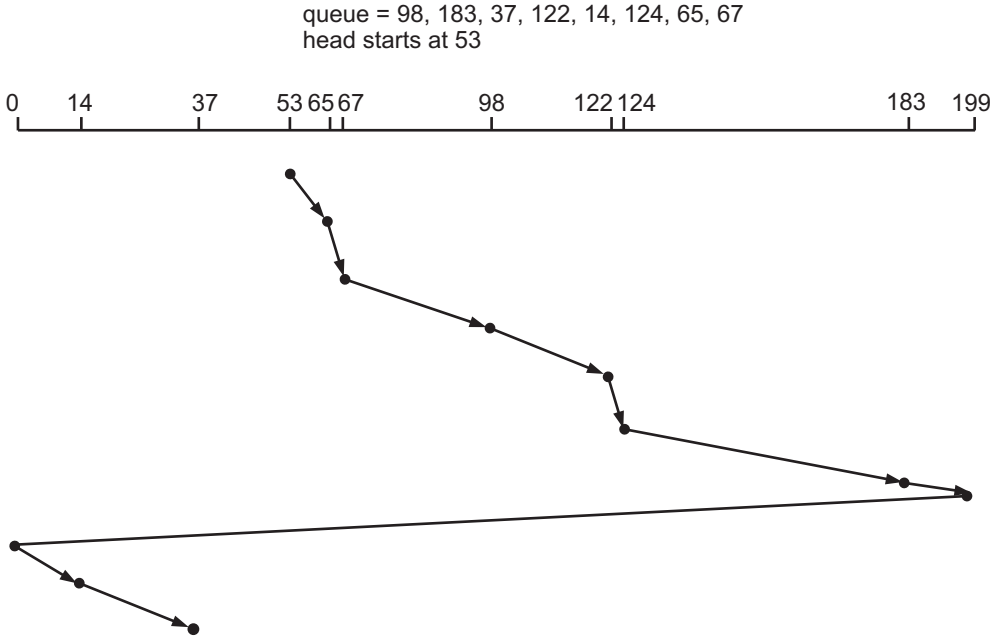


Fig. 9.20: C-SCAN Scheduling

- In C-SCAN algorithm head moves from one end of the disk to the other servicing requests as it goes. When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip.
- The C-SCAN (Circular SCAN) algorithm restricts scanning to one direction only. Thus, when the last track has been visited in one direction, the arm is returned to the opposite end of the disk and the scan begins again.
- This reduces the maximum delay experienced by new requests. Fig. 9.20 illustrates the disk arm movement.

Example 11:

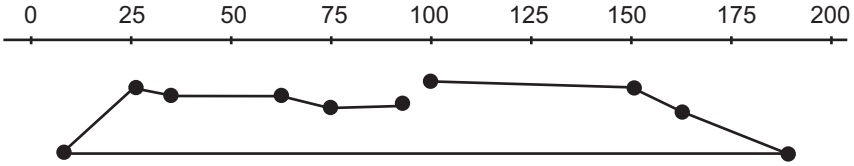


Fig. 9.21: C-SCAN (starting at track 100, in the direction of increasing track number

Solution: The Average seek length is,

Next track accessed	Number of track traversed
150	50
160	10
184	24
18	166
38	20
39	1
55	16
58	3
90	32
Average Seek Length	**35.8**

9.5.5 C-LOOK

- C-LOOK is version of C-SCAN algorithm.
- In C-SCAN algorithm, arm only shows as far as the last request in each direction, then reverses direction immediately, without first going all the way to the end of the disk.
- Fig. 9.22 shows C-LOOK disk scheduling.

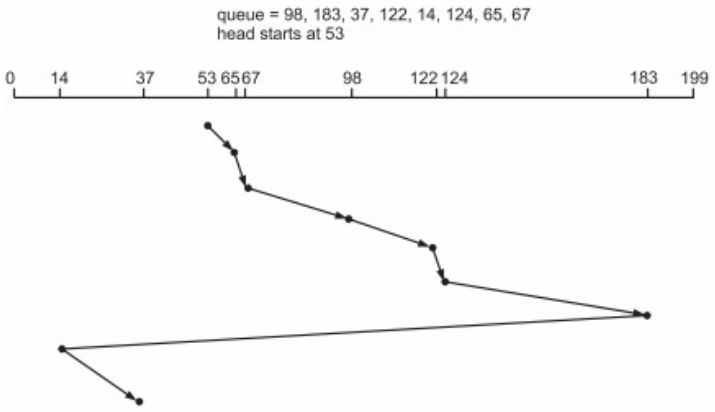


Fig. 9.22: C-LOOK Scheduling

Example 12: Suppose the requests to be addressed are 82,170,43,140,24,16,190. And the Read/Write arm is at 50, and it is also given that the disk arm should move "towards the larger value".

Solution:

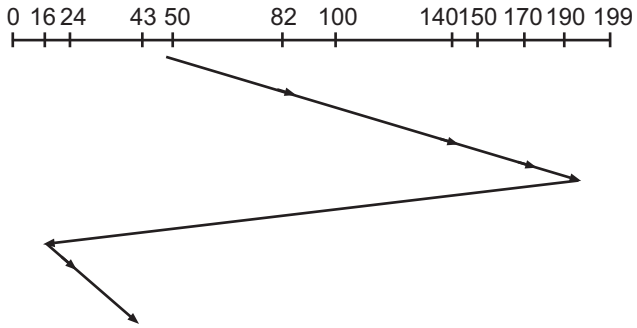


Fig. 9.23

So, the seek time is calculated as:

= (190 – 50) + (190 – 16) + (43 – 16) = 341

Summary

- The basic hardware elements involved in I/O are buses, device controllers, and the devices themselves. The work of moving data between devices and main memory is performed by the CPU as programmed I/O or is offloaded to a DMA controller.
- The system-call interface provided to applications is designed to handle several basic categories of hardware, including block devices, character devices, memory-mapped files, network sockets, and programmed interval timers.
- The kernel's I/O subsystem provides numerous services. These are I/O scheduling, buffering, caching, spooling, device reservation, and error handling.
- Requests for disk I/O are generated by the file system and by the virtual memory system. Each request specifies the address on the disk to be referenced, in the form of a logical block number.
- Disk-scheduling algorithms can improve the effective bandwidth, the average response time, and the variance in response time.
- Algorithms such as SSTF, SCAN, C-SCAN, LOOK, and C-LOOK are designed to make such improvements through strategies for disk-queue ordering. Performance of disk-scheduling algorithms can vary greatly on magnetic disks.
- In contrast, because solid-state disks have no moving parts, performance varies little among algorithms, and quite often a simple FCFS strategy is used.

Check Your Understanding

1. Buffering is done to _____.
   (a) cope with device speed mismatch
   (b) cope with device transfer size mismatch
   (c) maintain copy semantics
   (d) all of the mentioned
2. Caching is _____ spooling.
   (a) same as (b) not the same as
   (c) all of the mentioned (d) none of the mentioned
3. Caching _____.
   (a) holds a copy of the data (b) is fast memory
   (c) holds the only copy of the data (d) holds output for a device

4. Spooling _____.
   (a) holds a copy of the data (b) is fast memory
   (c) holds the only copy of the data (d) holds output for a device
5. The _____ keeps state information about the use of I/O components.
   (a) CPU (b) OS
   (c) kernel (d) shell
6. The kernel data structures include _____.
   (a) process table (b) open file table
   (c) close file table (d) all of the mentioned
7. Windows NT uses a _____ implementation for I/O.
   (a) message – passing (b) draft – passing
   (c) secondary memory (d) cache
8. A _____ is a full duplex connection between a device driver and a user level process.
   (a) Bus (b) I/O operation
   (c) Stream (d) Flow
9. I/O is a _____ in system performance.
   (a) major factor (b) minor factor
   (c) does not matter (d) none of the mentioned
10. If the number of cycles spent busy – waiting is not excessive, then _____.
    (a) interrupt driven I/O is more efficient than programmed I/O
    (b) programmed I/O is more efficient than interrupt driven I/O
    (c) both programmed and interrupt driven I/O are equally efficient
    (d) none of the mentioned
11. Whenever a process needs I/O to or from a disk it issues a _____.
    (a) system call to the CPU (b) system call to the operating system
    (c) a special procedure (d) all of the mentioned
12. If a process needs I/O to or from a disk, and if the drive or controller is busy then _____.
    (a) the request will be placed in the queue of pending requests for that drive
    (b) the request will not be processed and will be ignored completely
    (c) the request will be not be placed
    (d) none of the mentioned
13. Consider a disk queue with requests for I/O to blocks on cylinders.
    98 183 37 122 14 124 65 67
    Considering FCFS (first cum first served) scheduling, the total number of head movements is, if the disk head is initially at 53 is?
    (a) 600 (b) 620
    (c) 630 (d) 640
14. Consider a disk queue with requests for I/O to blocks on cylinders.
    98 183 37 122 14 124 65 67
    Considering SSTF (shortest seek time first) scheduling, the total number of head movements is, if the disk head is initially at 53 is _____.
    (a) 224 (b) 236
    (c) 245 (d) 240
15. Random access in magnetic tapes is _____ compared to magnetic disks.
    (a) fast (b) very fast
    (c) slow (d) very slow

16. Magnetic tape drives can write data at a speed _____ disk drives.
    (a) much lesser than (b) comparable to
    (c) much faster than (d) none of the mentioned
17. On media that use constant linear velocity (CLV), the _____ is uniform.
    (a) density of bits on the disk (b) density of bits per sector
    (c) the density of bits per track (d) none of the mentioned
18. SSTF algorithm, like SJF _____ of some requests.
    (a) may cause starvation (b) will cause starvation
    (c) does not cause starvation (d) causes aging
19. In the _____ algorithm, the disk arm starts at one end of the disk and moves toward the other end, servicing requests till the other end of the disk. At the other end, the direction is reversed and servicing continues.
    (a) LOOK (b) SCAN
    (c) C-SCAN (d) C-LOOK
20. In the _____ algorithm, the disk head moves from one end to the other, servicing requests along the way. When the head reaches the other end, it immediately returns to the beginning of the disk without servicing any requests on the return trip.
    (a) LOOK (b) SCAN
    (c) C-SCAN (d) C-LOOK
21. In the _____ algorithm, the disk arm goes as far as the final request in each direction, then reverses direction immediately without going to the end of the disk.
    (a) LOOK (b) SCAN
    (c) C-SCAN (d) C-LOOK

Answers

1. (d)	2. (b)	3. (a)	4. (c)	5. (c)	6. (b)	7. (a)	8. (c)	9. (a)	10. (b)
11. (b)	12. (a)	13. (d)	14. (b)	15. (d)	16. (b)	17. (c)	18. (a)	19. (b)	20. (c)
21. (a)									

Practice Questions

Q.I Answer the following questions in short:

1. List categories of I/O devices.
2. List techniques for performing I/O.
3. List Disk Scheduling algorithms.
4. What is purpose of Disk Scheduling?
5. Define the term response time.
6. Define the term transfer time.

Q.II Answer the following questions:

1. Explain the term I/O hardware.
2. What is meant by disk scheduling?
3. With the help of diagram describe FCFS disk scheduling algorithm.
4. Describe the term application of I/O interface in detail.
5. With the help of example describe C-LOOK scheduling algorithm.
6. Explain C-SCAN disk scheduling, algorithm with example.
7. Write short note on Kernel I/O subsystem.
8. Compare FCFS and SCAN disk scheduling algorithms.
9. Explain the term SSTF in detail.

10. Assume there are total 200 tracks that are present on each surface of the disk. If request queue is 30, 140, 20, 170, 60, 190 and initial position of the head is 120. Apply FCFS Disk Scheduling and calculate total head movement.
11. Assume there are total 200 tracks that are present on each surface of the disk. If request queue is 70, 120, 10, 180, 90, 50, 100 and initial position of the head is 105. Apply FCFS Disk Scheduling Algorithm and calculate total head movement.
12. Explain DMA with the help of Block Diagram.
13. A disk drive has 540 cylinders numbered 0–539. The drive is currently serving the request at cylinder 54. The queue is in order: 98, 183, 47, 125, 10, 126, 380, 200, 79.
14. Starting from the current head position what is the total distance that the disk arm moves to satisfy all the pending request for the following Disk Scheduling Algorithm?
    (i) FCFS (ii) SCAN

Q.III Define terms:

1. I/O hardware, 2. Buffering, 3. Caching, 4. Spooling, 5. Seek time

Previous Exam Questions

Summer 2018

1. Describe I/O Hardware with its type of I/O devices. **[4 M]**

Ans. Refer to Section 9.2.

2. The request queue is a follows: **[4 M]**
87, 148, 92, 171, 96, 131, 103, 71
Number of tracks = 0 to 199
Starting position or current head position = 125. Find total head movement by applying SSTF (Shortest Seek Time First) Disk Scheduling Algorithm.

Ans. Refer to Section 9.5.2.

Winter 2018

1. What do you mean by Seek Time in Disk Scheduling? **[2 M]**

Ans. Refer to Section 9.5.

2. Assume there are total 200 tracks present on each surface of the disk, if request queue is 57, 60, 41, 20, 92, 162, 152, 40, 186. Initial position of head is at 99 track. Apply FCFS disk scheduling Algorithm and calculate total head movement? **[4 M]**

Ans. Refer to Section 9.5.1.

Summer 2019

1. Write a note on interrupts. **[4 M]**

Ans. Refer to Section 9.2.

2. Assume there are total 0-199 tracks that are present on each surface of the disk. If request queue is: 84, 145, 89, 168, 93, 128, 100, 68 and initial position of the head is 125. Apply SCAN disk scheduling algorithm and calculate total head movement. **[4 M]**

Ans. Refer to Section 9.5.3.

■■■