THIRD YEAR B. Sc.
COMPUTER SCIENCE

SEMESTER-V

NEW SYLLABUS
CBCS PATTERN

# OPERATING SYSTEMS–I

Dr. Ms. MANISHA BHARAMBE
Mrs. VEENA GANDHI


NIRALI PRAKASHAN
ADVANCEMENT OF KNOWLEDGE

SPPU New Syllabus

*A Book Of*

# OPERATING SYSTEMS - I

**For T.Y.B.Sc. Computer Science : Semester – V**

**[Course Code CS 351 : Credits - 2]**

**CBCS Pattern**

**As Per New Syllabus, Effective from June 2021**


**Dr. Ms. Manisha Bharambe**

*M.Sc. (Comp. Sci.), M.Phil. Ph.D. (Comp. Sci.)*

Vice Principal, Associate Professor, Department of Computer Science

MES's Abasaheb Garware College

Pune

**Mrs. Veena Gandhi**

*M.C.S, M.Phil. (Computer Science), UGC-NET*

Head, Department of BCA Science

Abeda Inamdar Senior College

Pune


**Price ₹ 325.00**


NIRALI PRAKASHAN

ADVANCEMENT OF KNOWLEDGE

N5861

**OPERATING SYSTEMS - I** **ISBN 978-93-5451-169-1**

**Second Edition : August 2022**


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

**Polyplate**

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

## DISTRIBUTION CENTRES

### PUNE

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

**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

### MUMBAI

**Nirali Prakashan**
Rasdhara Co-op. Hsg. Society Ltd., 'D' Wing Ground Floor, 385 S.V.P. Road
Girgaum, Mumbai 400004, Maharashtra
Mobile : 7045821020, Tel : (022) 2385 6339 / 2386 9976
Email : niralimumbai@pragationline.com

## DISTRIBUTION BRANCHES

### DELHI

**Nirali Prakashan**
Room No. 2 Ground Floor
4575/15 Omkar Tower, Agarwal Road
Darya Ganj, New Delhi 110002
Mobile : 9555778814/9818561840
Email : delhi@niralibooks.com

### BENGALURU

**Nirali Prakashan**
Maitri Ground Floor, Jaya Apartments,
No. 99, 6th Cross, 6th Main,
Malleswaram, Bengaluru 560003
Karnataka; Mob : 9686821074
Email : bengaluru@niralibooks.com

### NAGPUR

**Nirali Prakashan**
Above Maratha Mandir, Shop No. 3,
First Floor, Rani Jhanshi Square,
Sitabuldi Nagpur 440012 (MAH)
Tel : (0712) 254 7129
Email : nagpur@niralibooks.com

### KOLHAPUR

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

### JALGAON

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

### SOLAPUR

**Nirali Prakashan**
R-158/2, Avanti Nagar, Near Golden
Gate, Pune Naka Chowk
Solapur 413001, Maharashtra
Mobile 9890918687
Email : solapur@niralibooks.com

**marketing@pragationline.com | www.pragationline.com**

**Also find us on www.facebook.com/niralibooks**

# Preface ...

We take an opportunity to present this Text Book on **"Operating Systems-I"** to the students of Third Year B.Sc. (Computer Science) Semester-V as per the New Syllabus, June 2021.

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. The book covers theory of Introduction to Operating Systems, Processes and Threads, Process Scheduling, Synchronization and Memory Management.

A special word of thank to Shri. Dineshbhai Furia, and Mr. Jignesh Furia for showing full faith in us to write this text book. We also thank to Mr. Amar Salunkhe and Mr. Akbar Shaikh of M/s Nirali Prakashan for their excellent co-operation.

We also thank Ms. Chaitali Takle, Mr. Ravindra Walodare, Mr. Sachin Shinde, Mr. Ashok Bodke, Mr. Moshin Sayyed and Mr. Nitin Thorat.

Although every care has been taken to check mistakes and misprints, any errors, omission and suggestions from teachers and students for the improvement of this text book shall be most welcome.

**Authors**

■■■

# Syllabus ...

1. **Introduction to Operating Systems** **(6 Lectures)**
   - Operating Systems Overview - System Overview and Functions of Operating Systems
   - What does an OS do?
   - Operating System Operations
   - Operating System Structure
   - Protection and Security
   - Computing Environments - Traditional, Mobile, Distributed, Client/Server, Peer to Peer Computing
   - Open Source Operating System
   - Booting
   - Operating System Services
   - System Calls, Types of System Calls and their Working
2. **Processes and Threads** **(6 Lectures)**
   - Process Concept - The Processes, Process States, Process Control Block
   - Process Scheduling - Scheduling Queues, Schedulers, Context Switch
   - Operations on Process - Process Creation with Program Using fork(), Process Termination
   - Thread Scheduling - Threads, Benefits, Multithreading Models, Thread Libraries
3. **Process Scheduling** **(7 Lectures)**
   - Basic Concept - CPU-I/O Burst Cycle, Scheduling Criteria, CPU Scheduler, Preemptive Scheduling, Dispatcher
   - Scheduling Algorithms - FCFS, SJF, Priority Scheduling, Round-Robin Scheduling, Multiple Queue Scheduling, Multilevel Feedback Queue Scheduling
4. **Synchronization** **(5 Lectures)**
   - Background
   - Critical Section Problem
   - Semaphores: Usage, Implementation
   - Classic Problems of Synchronization: The Bounded Buffer Problem, The Reader Writer Problem, The Dining Philosopher Problem
5. **Memory Management** **(12 Lectures)**
   - Background - Basic Hardware, Address Binding, Logical Versus Physical Address Space, Dynamic Loading, Dynamic Linking and Shared Libraries
   - Swapping
   - Contiguous Memory Allocation - Memory Mapping and Protection, Memory Allocation, Fragmentation
   - Paging - Basic Method, Hardware Support, Protection, Shared Pages
   - Segmentation - Basic Concept, Hardware
   - Virtual Memory Management - Background, Demand Paging, Performance of Demand Paging, Page Replacement - FIFO, Optimal, LRU, MFU

■■■

# Contents ...



■■■

CHAPTER 1

# Introduction to Operating Systems

## *Objectives ...*

- To understand Basic Concepts of Operating System
- To learn Services and Functions of an Operating System
- To study various types of Operating Systems
- To learn Operating System Structure
- To study Operating System Operations and Operating System Components
- To understand Concept of Protection and Security
- To study different Computing Environments
- To learn Concept of System Calls and its Working
- To study different Types of System Calls
- To understand the Concept System Booting

## 1.0 INTRODUCTION

- An operating system acts as an interface between the user of a computer system and the computer hardware.
- The purpose of an operating system is to provide an operating environment in which a user can execute programs in a convenient and efficient manner.
- An operating system is the most essential system software that manages the operations of a computer. Without an operating system, it is not possible to use the computer.
- An Operating System (OS) is a system program that acts as an intermediary between the user and the computer hardware and controls the execution of all kinds of programs.
- The operating system provides certain services to programs and to the users of those programs in order to make their tasks easier.
- The basic task of an operating system is to control and co-ordinate the use of hardware among the various application programs for various users.
- In short, an operating system is a program which takes control of the operation of the computer system.

- The purpose of this control is to make the computer system operate in the way intended by the user and in a systematic, reliable and efficient manner.
- Various types of operating systems are UNIX, MS-DOS, MS-Windows 95/98/ME//XP/Vista/7/8/10, Linux, Windows-NT/2000, OS/2, Android, Windows Mobile, iOS, Symbian OS, Mac OS, ZyNOS and so on.

## 1.1 OPERATING SYSTEM OVERVIEW

- An operating system is system software that is used to manage the various resources and overall operations of a computer system.
- An operating system acts as an intermediary/interface (Refer Fig. 1.1) between the user of a computer and computer hardware.
- The main task of an operating system is to control, co-ordinate and supervise of hardware to the all the users.
- An operating system links the computer user and the computer hardware and resides within the computer system.

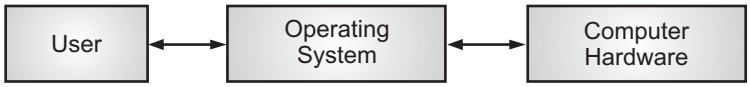



**Fig. 1.1: Operating System Interface**

- The purpose of an operating system is to provide an environment in which a user can execute programs in a convenient and efficient manner.
- An operating system is an integrated set of specialized programs used to manage overall resources and operations of the computer.

**Role of Operating System:**

1. To provide an environment in which the user is able to run application software. The applications that users run rely on services provided by the operating system to perform tasks while they execute.
2. To manage the functions of the computer hardware. It helps the user to assess various functions, running the applications programs, serving as an interface between the computer and the user and allocating computer resources to various functions.

- The operating system creates an operating environment for the user's application program to run properly and systematic in computer system.
- An operating system is an integrated set of system programs whos major function is to manage resources (CPU, Memory etc.), control Input/Output (I/O), schedule jobs, handle errors, provide security and protection and so on.
- An operating system is a set of programs which control all the computer's resources and provide an operational environment to a computer user to execute programs.
- The objectives or goals of operating systems are listed below:

1. **Convenience:** An operating system is designed or constructed in such a way that it makes the computer system more convenient to use.

2. **Efficiency:** An operating system allows the computer system resources to be used in an efficient manner.
3. **Ability to Evolve:** An operating system should be designed or constructed in such a way that the testing, development as well as addition of new system components could be performed effectively without interfering with existing services.

- Operating systems are classified into following categories, depending on their capability of processing:
  1. **Single-user, Single-tasking OS:** As the name implies, this operating system is designed to manage the computer so that one user can effectively do one thing at a time. The palm OS for palm handheld computers is a good example of a modern single-user, single-task operating system. Another example includes MS-DOS and Windows 95.
  2. **Single-user, Multi-tasking OS:** This type of OS which allows a single user to execute two or more tasks at a time. Single-user, multi-tasking is the type of operating system most people use on their desktop and laptop computers today. Both Windows 98 and the Macintosh OS are examples of operating systems that will let a single user have several programs in operation at a time. Another example includes Windows 2000 and UNIX.
  3. **Multi-user, Multi-tasking OS:** A multi-user operating system allows many different users to take advantage of the computer's resources simultaneously. UNIX, Linux, VMS (Virtual Memory System) are examples of multi-user operating systems.
  4. **Real Time Operating System (RTOS):** Real time operating system is used mainly for real-time systems. These OS used by systems where a precise timing and reliability are the first priority. RTOS is managing the resources of the computer so that a particular operation executes in precisely the same amount of time every time it occurs. The response time of a real-time operating system is very quick. The operating system which guarantees the maximum time for these operations are commonly referred to as **hard real-time systems**, while operating systems that can only guarantee a maximum of the time are referred to as **soft real-time systems**. Examples of real-time OS are QNX, RTLINUX etc.
  5. **Distributed Operating System:** Distributed means data is stored and processed on multiple locations like in the network. In a distributed operating system, the users access remote resources including both hardware and software in same manner as they do local resources. Examples of distributed operating systems include LOCUS Distributed OS (based on UNIX), OSF/1 OS, IRIX OS, Solaris etc.
  6. **Time Sharing OS:** In time sharing OS, a number of simultaneous users are there and each user is given a trivial amount of time (a quantum/time slice) in which he/she processes interactively or conversationally. UNIX, Windows NT and Windows XP are examples of this type of operating system.
  7. **Multi-programming OS:** Multi-programming is a term given to a system that may have several (multiple) processes in the 'state of execution' at the same time.

8. **Batch-processing OS:** In this type of OS, there is no direct interaction between user and the computer. The user has to submit a job (written on cards or tape) to a computer operator. Then computer operator places a batch of several jobs on an input device. Jobs are batched together by type of languages and requirement. Then a special program, the monitor, manages the execution of each program in the batch. The monitor is always in the main memory and available for execution.
9. **Multi-threading OS:** It allows different parts of a single program to run concurrently. Multithreading operating systems allow different parts of a software program to run concurrently. Operating systems that would fall into this category are Linux, UNIX and Windows 2000.
10. **Multi-processing OS:** In a multi-processing OS, a single CPU has more than one processor. All these processors may or may not be equally powerful and may or may not prefer same operation.
11. **Multi-tasking OS:** It allows executing more than one task at the same time. An operating system that is capable of allowing multiple software processes to run at the same time. Some examples of multi-tasking operating systems are UNIX and Windows 2000 Windows XP, Windows 7, Windows Vista etc.
12. **Network Operating System (NOS):** A network operating system is a collection of software and associated a protocol that allows a set of autonomous computers, which are interconnected by a computer network to be used together in a convenient and cost-effective manner. Examples of network operating system are BSD (Berkeley System Distribution), MS LAN Manager, Windows-NT, UNIX, Microsoft Windows Server 2003, Novell's NetWare, Microsoft Windows Server 2008 and so on.
13. **Embedded OS:** Embedded operating systems are designed to work on semiconductor memory with limited processing power. This OS is embedded in a device in the ROM. They are specific to a device and are less resource intensive. Examples of embedded operating systems includes embedded Windows XP, embedded Linux etc. They are used in appliances like microwaves, washing machines, traffic control systems etc.
14. **Mobile Operating System:** A mobile operating system also called a mobile OS. It is an operating system that is specifically designed to run on mobile devices such as mobile phones, smartphones, PDAs, tablet computers and other handheld devices. Examples of mobile operating systems include Symbian OS, Windows CE, Apple iOS, Windows Phone and Google Android.

## 1.1.1 Definition of Operating System

- An operating system is defined as, "an organized collection of programs that controls the execution of the application programs and acts as interface between computer system and computer hardware". **OR**

- An operating system is defined as, “a program that acts as an intermediary between the user and the computer hardware and controls the execution of all kinds of programs”. **OR**
- Operating system is a, "set of programs which control/manage all the computer's resources (hardware and software) and provide an operating environment in which users can executes programs".

## 1.1.2 Computer System Overview

- A computer is an electronic device which is used to store, retrieve and process the data.
- A computer system is a collection of hardware and software components designed to provide an effective tool for computation.
- Hardware refers to the electrical, mechanical and electronic parts (physical computing equipment) that make up the computer. Software refers to the programs written to provide services to the system.
- Fig. 1.2 shows the conceptual view of a general computer system.

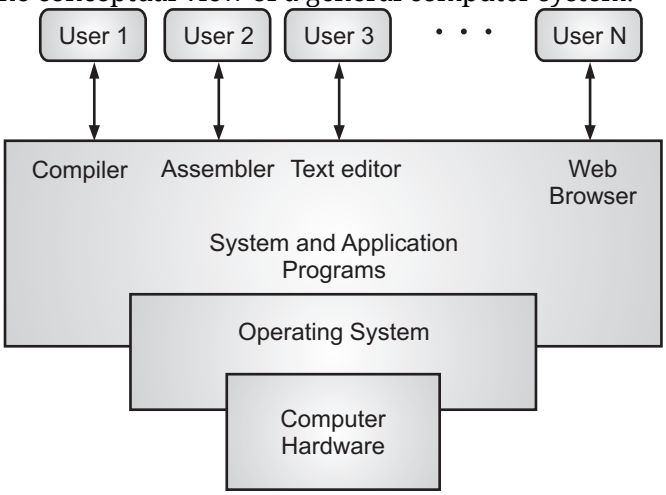



**Fig. 1.2: Conceptual View of Computer System**

- Every computer system is divided into the hardware, system and application programs, the operating system and users.
- The **hardware** consists of Central Processing Unit (CPU), memory and Input/Output (I/O) devices which are altogether used to provide the basic computing resources for the computer system.
- An **operating system** is core software that controls the execution of an application program and acts as an interface between the user of a computer and computer hardware.
- The operating system mainly coordinates the use of the hardware among the various system programs and application programs for various users.
- **Application programs** define the ways in which the system resources are used to solve the computing problems of the users (word processor, video games, spreadsheet etc.).

- **System programs** are the set of utility programs supplied with an operating system to provide basic services for the computer users (compilers, text editors, assembler etc.).
- **Users** are the people (like system analyst, programmers, and operational users etc.) who interact with the computer system.

## 1.1.3 Functions of Operating Systems

- There are many functions those are performed by the operating system but the main goal of operating system is provides the interface for working on the system by the user.
- The various functions those are performed by the operating system are as explained below:
  1. **Memory Management:** An operating system deals with the allocation of main memory and other storage areas to the system programs as well as user programs and data.
  2. **Processor Management:** An operating system deals with the assignment of processor to different tasks being performed by the computer system.
  3. **Device Management:** An operating system deals with the co-ordination and assignment of the different output and input device while one or more programs are being executed. Operating System manages device communication via their respective drivers.
  4. **File Management:** An operating system deals with the storage of file of various storage devices to another. It also allows all files to be easily changed and modified through the use of text editors or some other files manipulation routines.
  5. **Error Detecting Aids:** Production of dumps, traces, error messages and other debugging and error detecting aids.
  6. **Security:** Computer security is a very important aspect of any operating system. The OS keeps the system and programs safe and secure through authentication. By means of password and similar other techniques, preventing unauthorized access to programs and data.
  7. **Control over System Performance:** An OS performs recording delays between request for a service and response from the system.
  8. **Coordination between other Softwares and Users:** Coordination and assignment of compilers, interpreters, assemblers and other software to the various users of the computer systems.
  9. **Job Accounting:** An OS keeping the track of time and resources used by various jobs and users.

## 1.1.4 Types of Operating Systems

- There are many operating systems those have been developed for performing the operations (single operation and multiple operations at a time) those is requested by the user.

**Batch Operating System:**

- It is one of the oldest operating system. In batch operating system, jobs with similar need batched together by operator and run as a group on a computer system.

- A batch is a sequence of user jobs formed for the purpose of processing by a batch processing operating system.
- Batch operating system is one where programs and data are collected together in a batch before processing starts.
- A job is predefined sequence of commands, programs, instructions and data that are combined in to a single unit called job.
- Memory management in batch system is very simple. Memory is usually divided into two areas namely, operating system and user program area as shown in Fig. 1.3.

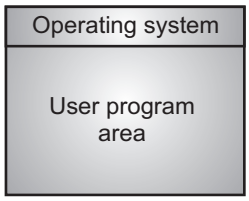



**Fig. 1.3: Memory Layout for a Simple Batch System**

- Batch processing is implemented by the kernel (also called the batch monitor), which resides in one part of the computer's memory. The remaining memory is used for servicing a user job - the current job in the batch.
- When the operator gives a command or instruction to initiate the processing of a batch, the batch monitor sets up the processing of the first job of the batch. At the end of the job, it performs job termination processing and initiates execution of the next job.
- At the end of the batch, it performs batch termination processing and awaits initiation of the next batch by the operator. Thus, the operator needs to intervene only at the start and end of a batch.
- Fig. 1.4 shows a schematic of a batch processing system. The batch consists of n jobs, $job_1$, $job_2$, …, $job_n$, one of which is currently in execution.
- The Fig. 1.4 depicts a memory map showing the arrangement of the batch monitor and the current job of the batch in the computer's memory.
- The part of memory occupied by the batch monitor is called the System area, and the part occupied by the user job is called the User area.

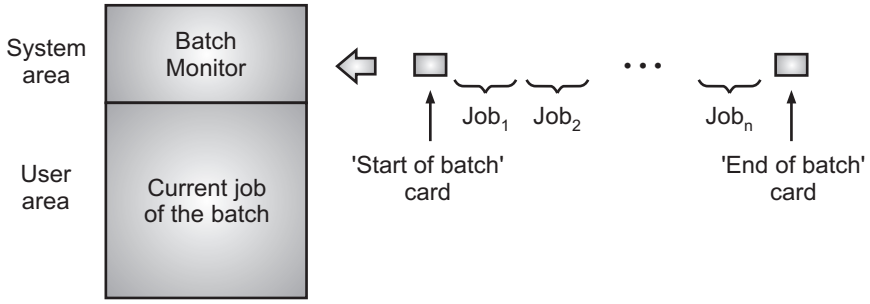



**Fig. 1.4: Schematic of a Batch Processing System**

**Advantages of Batch Operating System:**

1. In batch operating system overhead per program got reduced which improves the system utilization.
2. Increased performance since it was possible for job to start as soon as the previous job finished.
3. In batch operating system huge amount of a data can be processed efficiently.
4. In batch operating system the execution of job becomes fast and well managed.

**Disadvantages of Batch Operating System:**

1. No interaction is possible with the user while the job is being executed.
2. Batch systems are costly.
3. It is difficult to debug program in batch operating system.
4. Due to lack of protection scheme, one batch job can affect pending jobs.
5. As monitor needs to be in main memory, it results in certain wastage of memory.

**Concept of Spooling:**

- SPOOLing stands for Simultaneous Peripheral Operation On Line. Spooling refers to putting data of various I/O jobs in a buffer. This buffer is a special area in memory or hard disk which is accessible to I/O devices.
- Programs and data are punched on cards and cards being read from the card reader directly into memory. The location of card images is recorded in a table kept by operating system.
- When job is executed, the operating system satisfies its requests for card reader input by reading from the disk.
- Similarly when 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 as Spooling.

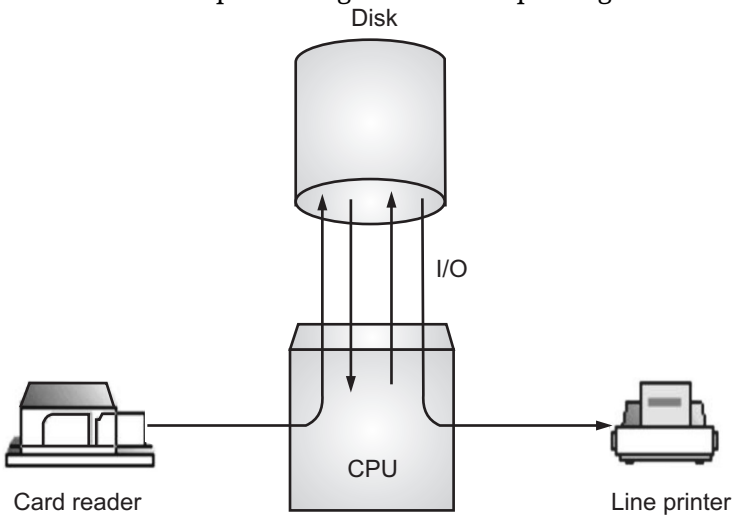



**Fig. 1.5: Concept of Spooling**

- The most common spooling application is print spooling. In print spooling, documents are loaded into a buffer and then the printer pulls them off the buffer at its own rate.

- Because the documents are in a buffer where they can be accessed by the printer, we can perform other operations on the computer while the printing takes place in the background.
- Spooling also lets we place a number of print jobs on a queue instead of waiting for each one to finish before specifying the next one.
- Spooling is also used for processing data at remote sites. The CPU sends the data via communication paths to remote printer.

**Advantage of Spooling:**

1. The spooling operation uses a disk as a very large buffer.
2. Spooling is however capable of overlapping I/O operation for one job with processor operations for another job.

**Disadvantages:**

1. There is an extra overhead of maintaining table of card images.
2. An extreme of spooling is called as staging of tapes. To avoid the wear of the magnetic tape, it is copied onto the disk and CPU will access the disk and not the tape.

**Multiprogramming Operating System:**

- In multiprogramming operating system more than one program/job can be execute at the same time.
- The term multiprogramming means a situation in which a single CPU divides its time between more than on job. In short multiprogramming operating systems allow concurrent execution of multiple programs or jobs.
- In multiprogramming, more than one programs or jobs are present in the memory at a time. Multiprogramming means to maintain two or more jobs concurrently in execution.
- The basic idea of multiprogramming is as follows:
  - The job pool consists of number of jobs. The operating system picks jobs from job pool. Many jobs are kept in memory simultaneously.
  - The number of jobs present in memory at a time is called degree of Multiprogramming. The operating system start execution of a job. (Only one job is executed at a time).
  - If a job requires I/O, then operating system switches to another job and executes it. CPU would not sit idle. When that job requires to wait for an I/O, the operating system switches to another job and so on.
- The Fig. 1.6 shows the multiprogramming system with job execution. Fig. 1.6 shows that job 1 started its execution and if it requires I/O then, operating system switch to job 2 which is executed next and CPU would not remain idle.
- Fig. 1.7 shows that several jobs are ready to run, the system must keep them in memory at a time.
- As there will always be several jobs in memory all the time, operating system requires some form of memory management.
- If many jobs are ready to run, then operating system must select among them. Operating system has to perform some CPU scheduling to schedule only one job at a time.

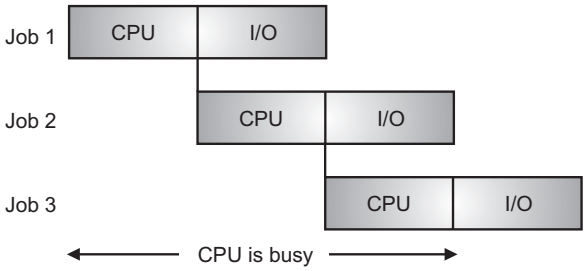



Fig. 1.6: Job Execution in Multiprogramming

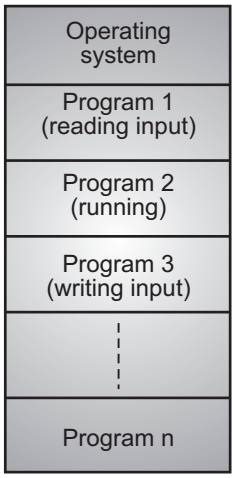



Fig. 1.7: Memory Layout in Multiprogramming

**Advantages:**

1. Efficient memory utilization.
2. High CPU utilization as it is never idle.
3. More CPU throughout.
4. Supports multiple simultaneous interactive user (terminals).

**Disadvantages:**

1. The user cannot interact with the job when it is being executed.
2. The programmer cannot modify a program to study its behavior while the program is being executed.
3. Jobs may have different sizes, so some powerful memory management policy is desired to accommodate them in memory.
4. CPU scheduling is must because now many jobs are ready to be run on the CPU.

**Difference between Batch Operating System and Multiprogramming Operating System:**

| Sr. No. | Batch Operating System | Multiprogramming Operating System |
|---|---|---|
| 1. | In this OS, a batch of similar jobs that consist of instructions, data and system commands is submitted to the operator and one job is executed at a lime. | In this OS, multiple programs appear to run concurrently by rapidly switching the CPU between the programs. |
| 2. | Access to files is serial/sequential, so simple file management is needed. | A number of processes may attempt to access a file at the same time. This demands for advanced protection and concurrency control methods. |

*contd. ...*

| | | |
|---|---|---|
| 3. | The jobs are executed in the order in which they were submitted, that is, FCFS (First-Come-First-Serve basis). | Multiprogramming systems need more sophisticated scheduling algorithm, as multiple processes reside in the main memory at the same time. |
| 4. | These systems do not tend to achieve efficient CPU utilization. | These systems ensure that the CPU has always something to execute, thus increasing the CPU utilization. |

**Multitasking or Time Sharing Operating System:**

- A running state of a program is called as a process or a task. A CPU handling multiple tasks at a time is known as multitasking.
- A multitasking operating system can handle multiple tasks together by applying multiprogramming technique.
- The time-sharing systems are also called multitasking systems. A multitasking operating system supports two or more active processes simultaneously.
- An instance of a program in execution is called a process or a task. Multitasking operating system supports two or more active processes simultaneously.
- In a multi tasking environment, operating system ensures that multiple tasks run on a single system by sharing the CPU time.
- Though they access the CPU one by one, it appears to the user that all the tasks are running simultaneously.
- For example, while one has opened multiple tabs in a single browser and whenever the user switches from one tab to another, operating system schedules the current process to access the CPU. However, user is never concerned and this is how multi tasking is achieved.
- The two most common methods for sharing the CPU time is either cooperative multitasking or preemptive multitasking.
  1. **Cooperative Multitasking:** The cooperative multitasking lets the programs decide when they wish to let other tasks run. Early versions of the MacOS (uptil MacOS 8) and versions of Windows earlier than Win95/WinNT used cooperative multitasking (Win95 when running old apps).
  2. **Preemptive Multitasking:** Preemptive multitasking moves the control of the CPU to the OS, letting each process run for a given amount of time (a time slice) and then switching to another task. This method prevents one process from taking complete control of the system and thereby making it seem as if it is crashed. This method is most common today, implemented by among others OS/2, Win95/98, WinNT, Unix, Linux, BeOS, QNX, OS9 and most mainframe OS.
- Fig. 1.8 shows multitasking concept. The user gives instructions to the operating system or to a program directly, and receives an immediate response.
- Operating system handles multitasking in the way that it can handle multiple operations/ executes multiple programs at a time.

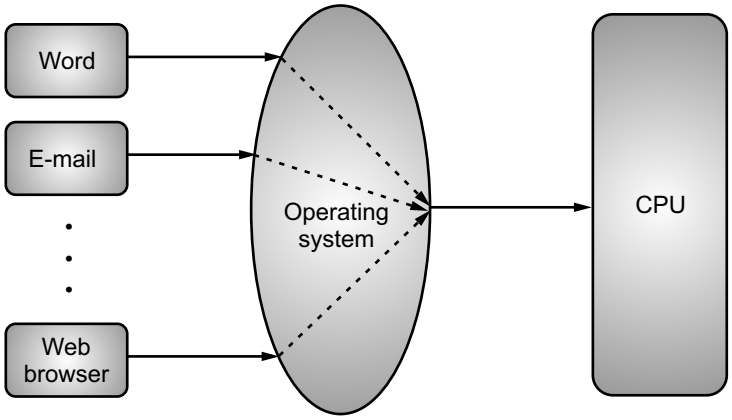



**Fig. 1.8: Multitasking Process**

- 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 time shared operating system allows many users to share the computer simultaneously.
- A time sharing system is one that allows a number of different users to share the single computer system for a variety of applications at the time.
- Fig. 1.9 shows an example of time sharing system which allows many users to simultaneously share the computer resources.
- In the Fig. 1.9, 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.

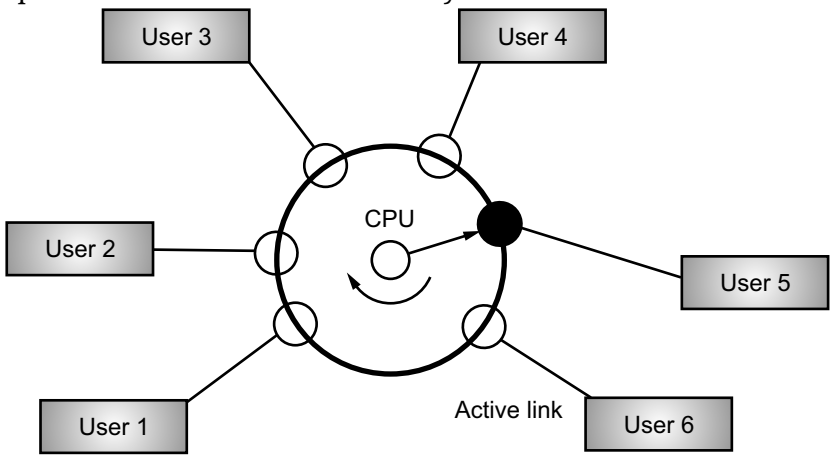



**Fig. 1.9**

**Advantages of Multitasking Operating System:**

1. Multitasking helps in increasing the overall productivity of the user by performing a number of tasks at the same time.
2. It helps in increasing the overall performance of the computer system.
3. Efficient CPU utilization.

**Disadvantages of Multitasking Operating System:**

1. It requires more system resources. For example, large amount of memory is required to execute several programs at the same time.
2. To performing multiple tasks at a single time in multitasking, the CPU speed must be very high.

**Multiprocessor Operating System:** **[April 18]**

- A multiprocessor operating system means the use of two or more processors within a single computer system. The term multiprocessing means multiple CPUs perform more than one job at one time.
- Multiprocessing is the use of two or more Central Processing Units (CPUs) within a single computer system.
- The term multiprocessing also refers to the ability of a system to support more than one processor and/or the ability to allocate tasks between them.
- The basic organization of multiprocessing system is shown in Fig. 1.10.

**Types of Multiprocessor Systems:**

- The two types of multiprocessing system are explained below:
  1. **Asymmetric Multiprocessing:** In this system, a specific task is assigned to each processor. The system has one master processor and others are slave processors. A master processor controls the system and slave processors follow the instructions of master or perform their predefined task.
  2. **Symmetric Multiprocessing:** In symmetric multiprocessing, there is no master-slave concept used. All the processors are peer processors. They perform all tasks within the operating system.

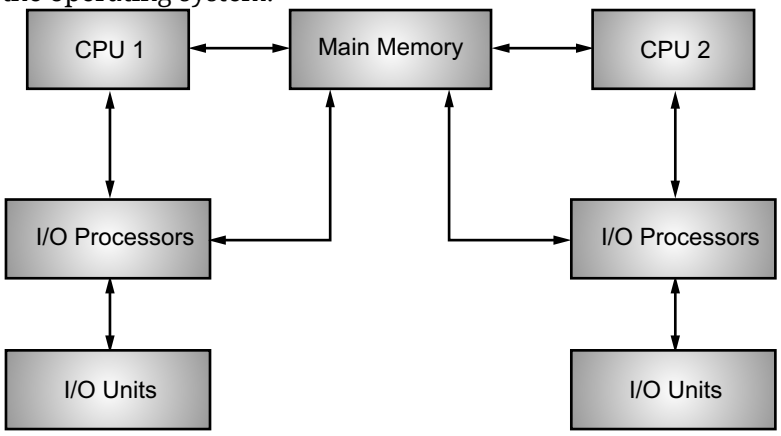



**Fig. 1.10**

**Advantages of Multiprocessor Operating Systems:**

1. It increased throughput by increasing the number of processors, more work done in a shorter period of time.

2. Multiprocessors can also save money (cost) compared to multiple single systems. Because the processors can share peripherals cabinets and power supplies.
3. It increases reliability, if functions can be distributed properly among several processors, then the failure of one processor will not halt the system, but rather will only slow it down.
4. These systems provide higher performance due to parallel processing.

**Disadvantages of Multiprocessor Operating Systems:**

1. If one processor fails then it will affect in the speed.
2. Multiprocessor systems are expensive.
3. Complex OS and large main memory is required.

**Real Time Operating System:**

- Real-time operating systems are the operating systems that are used in real-time applications where the data processing must be done in a fixed interval of time.
- Real time operating system is used in an environment where a large number of events, mostly external to the computer system, must be accepted and processed in a short time or within stipulated deadline.
- A real time computer system may be defined as, "one which controls an environment by receiving data, processing them and taking action or returning results very quickly to affect the functioning of the environment at that time".
- A real-time system is a data processing system in which the time interval required to process and respond to inputs is so small that it controls the environment.
- Examples of real time system include Nuclear Power Plant Control, Medical Imaging Systems, Industrial Control Systems, Weapon Systems, Robots, Air Traffic Control Systems, Space Navigation, and Flight Control System etc.
- Real-time systems are used when there are rigid time requirements on the operation of a processor or the flow of data and real-time systems can be used as a control device in a dedicated application.
- A real-time operating system must have well-defined, fixed time constraints, otherwise the system will fail.

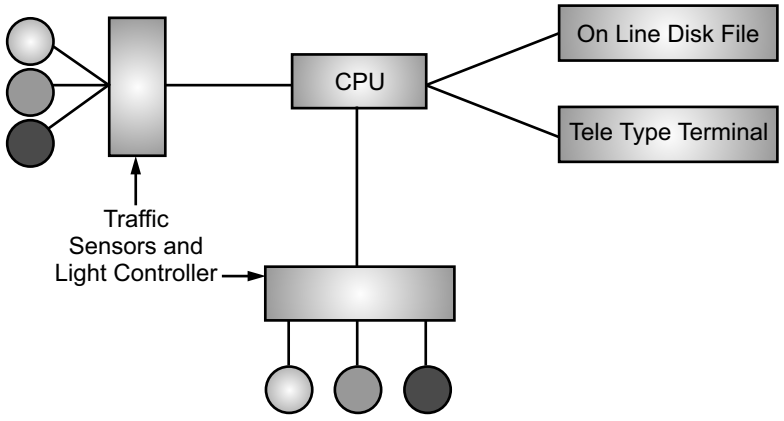



**Fig. 1.11**

- Real time operating systems are of two types namely, hard real time operating systems soft real time operating systems.
- Hard real-time operating systems guarantee that critical tasks complete on time. In hard real time operating systems, the actions must be taken within the specified timeline.
- In soft real time operating systems, it is not mandatory to meet the deadline. A critical real-time task gets priority over other tasks and retains the priority until it completes.

**Advantages:**

1. Useful for real time practical industrial applications.
2. As user gates a higher degree of control the flexibility is setting up the priority of processes is greater.
3. Fast response, the system is immediately updated.
4. It offers better utilization of systems and produces more output from all the resources.

**Disadvantages:**

1. RTOSs are very complex and difficult to maintain.
2. Real Time Operating System (RTOS) is expensive because it is using heavy system resources.
3. It is more difficult to backup these systems when they are in use.

**Difference between Time Sharing System and a Real-Time Operating System:**

| Sr. No. | Time-Sharing Operating System | Real-Time Operating System |
|---|---|---|
| 1. | It is the logical extension of multiprogramming system that supports interactive users and is designed to provide a quick response time. | This OS is designed for environments where a huge number of events must be processed within fixed time constraints. |
| 2. | Round-robin scheduling is used to execute the programs. | Most real-time systems use priority-based pre-emptive scheduling. |
| 3. | This OS tends to reduce the CPU idle time by sharing it among multiple users. | Effective resource utilization and user convenience are of secondary concern in a real-time system. |
| 4. | More sophisticated memory management is needed in time-sharing OSs to provide separation and protection of multiple user programs. | In these OSs, the programs remain in the main memory most of the time and there is a little swapping of programs between main and secondary memory. Thus, memory management is less demanding in real-time systems. |
| 5. | Process deals with more than one application simultaneously. | Process deals with single application at a time. |

**Distributed Operating System:**

- A distributed system is basically a computer network in which two or more autonomous computers are connected their hardware and software interconnections to facilitate communication.
- The computer in distributed system can be interconnected by telephone lines, fiber optic cables, coaxial cables, satellite links, and radio waves and so on.
- A distributed system consists of a collection of autonomous computers, connected through a network and distribution middleware, which enables computers to coordinate their activities and to share the resources of the system
- A distributed environment refers to a collection of autonomous systems, capable of communicating and cooperating with each other through the network, are connected to each other through LAN/WAN.
- 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 OS is one that looks to its users like an ordinary centralized OS but runs on multiple independent CPUs.
- In distributed operating system, users are not be aware of where the programs are located and processed, that should all be handled automatically by the operating system.
- A distributed system comprises a collection of autonomous computers, linked through a computer network and distribution middleware.
- The middleware enables computers to coordinate their activities and to share the resources of the system, so that users perceive the system as a single, integrated computing facility.
- Thus, middleware is the bridge that connects distributed applications across dissimilar physical locations, with dissimilar hardware platforms, network technologies, operating systems, and programming languages.

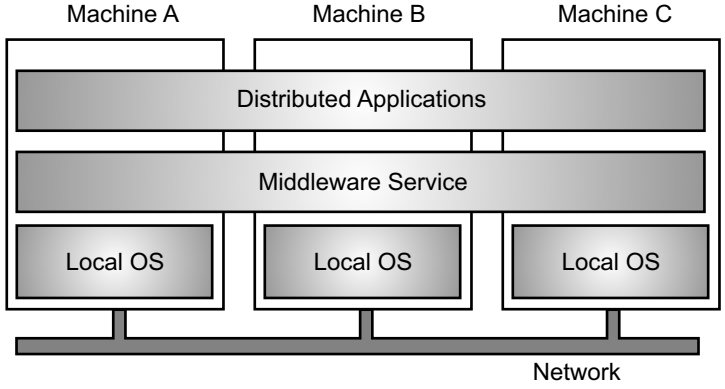



**Fig. 1.12: Distributed Systems**

**Advantages:**

1. **Reliability:** A distributed system is more reliable than a single system. If one machine from system crashes, the rest of the computers remain unaffected.
2. **Speed:** A distributed system can have more computing power than a mainframe. Its speed makes it different than other systems.
3. **Performance:** The collection of processors in the system can provide higher performance than a centralized computer.
4. **Sharing of Resources:** As data or resources are shared in distributed system, it is essential for various applications like banking, ticket reservation system.

**Disadvantages:**

1. **Troubleshooting:** Troubleshooting and diagnosing problems are the most important disadvantages of distributed system.
2. **Networking:** The underlying network in distributed system can cause several problems such as transmission problem, overloading, loss of messages.
3. **Complexity:** Typically, distributed systems are more complex than centralized systems.
4. **Security:** The easy distributed access in distributed system which increases the risk of security.

**Network Operating System:**

- A Network Operating System (NOS) runs on a server and provides the server the capability to manage data, users, groups, security, applications, and other networking functions.
- The primary purpose of the network operating system is to allow shared file and printer access among multiple computers in a network, typically a local area network (LAN), and a private network or to other networks.
- Examples of network operating systems include Microsoft Windows Server 2003, Microsoft Windows Server 2008, UNIX, Linux, Mac OS X, Novell NetWare, and BSD.
- Network operating systems are operated on a server and offer the facility of security, users, applications, data management, and other networking related functions.
- Its objective is to permit file sharing and printer access among various computers in a network, i.e., local area network (LAN), Private network, etc.
- Network Operating System is a computer operating system that facilitates to connect and communicate various autonomous computers over a network.
- An autonomous computer is an independent computer that has its own local memory, hardware, and operating system. NOS can be specialized to serve as client-server operating system and peer-to-peer operating system.
- **Peer-to-peer NOS** is an operating system in which all the nodes are functionally and operationally equal to each other. They all are capable to perform similar kinds of tasks. The nodes are directly connected with each other in the network.
- In peer-to-peer NOS, all computer systems contain equal privileges for consuming the all resources which are presented on the network.

- The advantages of the peer-to-peer NOS includes Easy to install and setup, setup cost is low, sharing of information and resources is fast and easy and so on. Disadvantages of the peer-to-peer NOS includes less secure, no centralized storage system, low performance and so on.
- The **client-server NOS** operate with a single server and multiple client computers in the network (Refer Fig. 1.13).
- The client operating system runs on the client machine, while the NOS is installed on the server machine. The server machine is a centralized hub for all the client machines.
- The client machines generate a request for information or some resource and forward it to the server machine. The server machine, in turn, replies to the client machine by providing appropriate services.

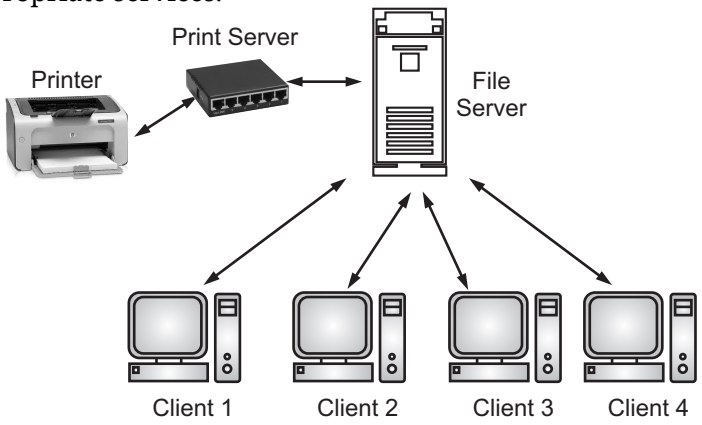



**Fig. 1.13: Network Operating System (NOS)**

**Advantages of Client-Server NOS:**

1. Centralized servers are highly stable.
2. Security is server managed.
3. Upgrades to new technologies and hardware can be easily integrated into the system.
4. Remote access to servers is possible from different locations and types of systems.

**Disadvantages of Client-Server NOS:**

1. High cost of buying and running a server.
2. Dependency on a central location for most operations.
3. Regular maintenance and updates are required.

**Difference between Network OS and Distributed OS:**

| Sr. No. | Network Operating System | Distributed Operating System |
|---|---|---|
| 1. | Network OS provides local services to remote clients. | Distributed OS manages the hardware resources. |
| 2. | Communication is file-based, shared folder based. | Communication is message-based or shared memory-based. |

***contd. ...***

| | | |
|---|---|---|
| 3. | Network OS is highly scalable. A new machine can be added very easily. | Distributed OS is less scalable. The process to add new hardware is complex. |
| 4. | Less fault tolerance as compared to distributed OS. | Distributed OS has very high fault tolerance. |
| 5. | Each machine can acts on its own thus autonomy is high. | Distributed OS has a poor rate of autonomy. |
| 6. | Network OS-based systems are easy to built and maintain. | Distributed OS implementation is difficult. |
| 7. | Network OS-based systems have their own copy of operating systems. | Distributed OS-based nodes have the same copy of the operating system. |
| 8. | Examples include Novell NetWare, BSD, Microsoft Windows Server 2008 and so on. | Examples include LOCUS, Amoeba, Chorus and so on. |

## 1.2 WHAT DOES AN OS DO?

- The operating system controls and coordinates the use of the hardware among the various application programs for the various users.
- The operating system provides proper use of resources like hardware and software in the operation of the computer system.
- An operating system is a software program that enables the computer hardware to communicate and operate with the computer software. Without a computer operating system a computer would be useless.
- The primary objective of an operating system is to make the computer system convenient to use and utilize computer hardware in an efficient manner.
- An Operating System (OS) does the following tasks:
  1. The operating system schedules all the operations and manages all the computing resources.
  2. The operating system provides the proper use of the recourses in the operations of the computer system.
  3. OS provide an environment for running user programs.
  4. An operating system supports operations and services that required communication over network.
  5. OS provide an interface to the user to communicate with the system hardware.
  6. OS manages the computer system resources in an efficient manner.
  7. OS provide security - ensure that only authorize persons can access data and information or perform certain tasks on the computer system.
  8. OS execute user programs and to make solving user problems easier.

9. OS provides file management which refers to the way that the operating system manipulates stores and retrieves data.
10. OS provide an overall control to the computer system.
11. OS provide memory management functions like allocation and deallocation of primary memory.
12. It managing the processes i.e. assigning the processor to a process at a time. OS processor management for multitasking.
13. OS provide job accountability, which helps to track of all time and their resources, which are used by several tasks and multiple users.
14. OS manages all I/O devices communication with their drivers respectively.
15. OS monitors the entire system to identify the errors and bugs and fix immediately without losing any type of data.

## 1.3 OPERATING SYSTEM OPERATIONS

- Modern operating systems are interrupt driven. The operating system sit idle if there is no event occurs such as no processes to execute, no I/O devices to service, no users to whom to respond and so on.
- Events are almost always signaled by the occurrence of an interrupt or a trap.
- A trap (or an exception) is a software-generated interrupt caused either by an error such as, division by zero or invalid memory access or by a specific request from a user program that an operating system service be performed.
- For each type of interrupt, separate segments of code in the operating system determine what action should be taken. An Interrupt Service Routine (ISR) is provided to deal with the interrupt.
- When an event occurs, the operating system analyzes the event and performs appropriate actions.
- A properly designed operating system must ensure that an incorrect (or malicious) program can not cause other programs to execute incorrectly.
- In case of multi-programmed environment the computer resources are shared among several programs simultaneously. Though the sharing of resources improves the resource utilization, it also increases the problems.
- An error in one user program can adversely affect the execution of other programs. It may also happen that the erroneous program modifies another program or data of another program or the operating system itself.
- Without the protection against such type of errors, only one process must be allowed to execute at a time. However, to improve the resource utilization, it is necessary to allow resource sharing among several programs simultaneously.
- Therefore, to handle such environment. the operating system must be designed in such a way that it should ensure that an incorrect program does not affect the execution of other programs or the operating system itself.

## 1.3.1 Dual-Mode Operation and Multimode Operation

- The execution of operating system code and user code must be separated in order to ensure the correct execution of the program. Two separate modes of operations are needed:
  1. User mode, and
  2. Kernel mode (System Mode/Supervisor Mode/Privileged Mode).
- A bit called mode bit is added to the hardware of the computer to indicate the current mode.
  For kernel, the bit is 0.
  For user, the bit is 1.
- The system is in user mode during execution of user application. The system is in kernel mode, when some function is performed by operating system.
- The Fig. 1.14 shows the switching between the users to kernel mode.
- The booting process starts in Kernel mode. Whenever, interrupt occurs, the hardware switches from user mode to kernel mode.
- The system is in user mode when the operating system is running a user application such as handling a text editor.
- The transition from user mode to kernel mode occurs when the application requests the help of operating system or an interrupt or a system call occurs.
- The mode bit is set to 1 in the user mode. It is changed from 1 to 0 when switching from user mode to kernel mode.
- The system starts in kernel mode when it boots and after the operating system is loaded, it executes applications in user mode.
- There are some privileged instructions that can only be executed in kernel mode. These are interrupt instructions, input output management etc. If the privileged instructions are executed in user mode, it is illegal and a trap is generated.
- The mode bit is set to 0 in the kernel mode. It is changed from 0 to 1 when switching from kernel mode to user mode.

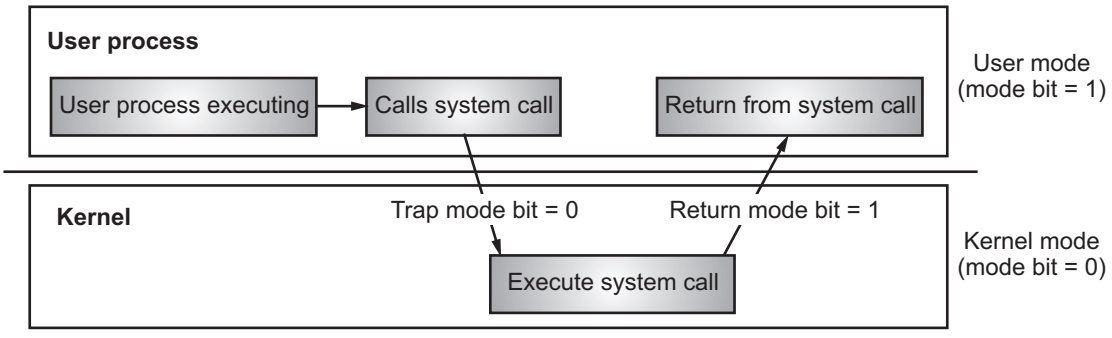



**Fig. 1.14: Transition from User to Kernel Mode**

- The concept of modes of operation in operating system can be extended beyond the dual mode. This is known as the multimode system. In those cases the more than 1 bit is used by the CPU to set and handle the mode.

- An example of the multimode system can be described by the systems that support virtualization. These CPU's have a separate mode that specifies when the Virtual Machine Manager (VMM) and the virtualization management software is in control of the system.
- For these systems, the virtual mode has more privileges than user mode but less than kernel mode.

### 1.3.2 Timer

- If the user program goes in an infinite loop or fail to call system services and never return control to the operating system then we use timer.
- A timer can be set to interrupt the computer after some specified period of time. A timer can be fixed or variable.
- For variable timer, operating system sets a counter to some initial value. It uses fixed rate clock.
- For each clock ticks, the counter is decremented. When counter reaches to zero, an interrupt occurs and control is transferred automatically to the operating system.

## 1.4 OPERATING SYSTEM STRUCTURE [Oct. 18]

- Fig. 1.15 shows the basic structure of an operating system which shows various parts of an operating system and each part consists of a number of programs.
- Structure of an operating systems have a layered structure, with the bottom most layer forming the hardware part of the computer and the outer most (top most) layer forming the User Interface (UI). In between these two layers are kernel layer and shell layer.
- The **kernel** is the innermost layer and is the central controlling part of the operating system. The kernel is the core of the operating system and provides the basic services for all other parts of the operating system.
- The services of a kernel are requested by other parts of the operating system or by application programs through a specified set of program interfaces, sometimes known as system calls.
- A typical kernel contains programs that are used for the basic functions of an operating system like process management, input/output devices management.
- Kernel manages all operations of computer and hardware and acts as a bridge between the user and the resources of the system by accessing various computer resources like the CPU, I/O devices and other resources.
- The **shell** is the next layer to the kernel. A shell is software that provides an interface for users of an operating system access the services of a kernel.
- The shell is the layer of programming that understands and executes the commands a user enters. In some systems, the shell is called a command interpreter.
- Programs in the user interface part either provide a Command Line Interface (CLI) or a Graphical User Interface (GUI) to the user. These programs use facilities by shell.
- A **user** interacts with programs in the User Interface (UI) typically with the command interpreter to request use of resources and services provided by the system.
- The **hardware** consists of Central Processing Unit (CPU), the main memory, I/O devices, etc. provides the basic computing resources.

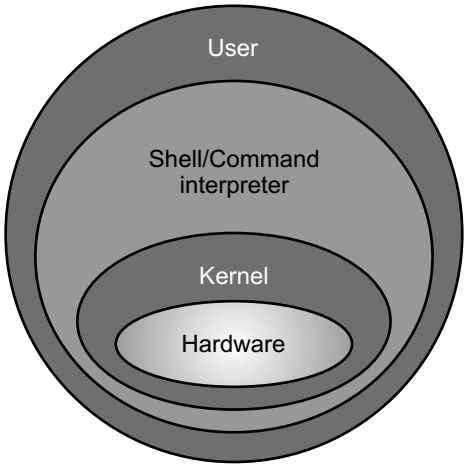



**Fig. 1.15: Basic Structure of an Operating System**

- Operating system can be implemented with the help of various structures like simple structure, monolithic structure, layered structure, microkernel structure and so on.
- Every operating system has its own structure like memory management, file management etc. and the entire performance of the system depends on its structure.
- The structure of the OS depends mainly on how the various common components of the operating system are interconnected and blended into the kernel.
- A system as large and complex as a modern operating system must be designed carefully if it is to function properly and be modified easily.

**1. Simple Structure**:

- In simple structure, the structure of operating systems was not well-defined. Such operating systems started small, simple and limited systems and then grew beyond their original scope.
- Simple structure is the oldest of all the approaches used to design an operating system. MS-DOS is an example of simple structure operating system as shown in Fig. 1.16.

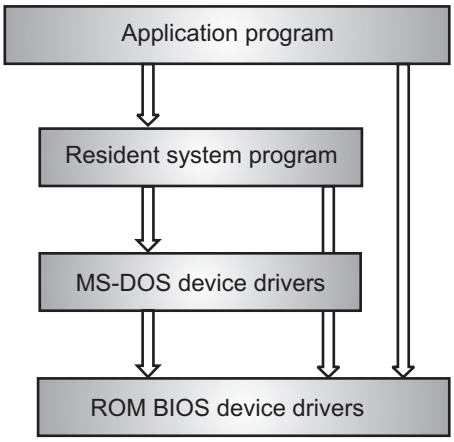



**Fig. 1.16: MS-DOS Layer Structure**

- Initially, MS-DOS was designed as a small-size and simple system, and with limited scope, but grew beyond its scope with time.
- It was designed with the idea of providing more functionality within loss space; therefore, it was not carefully divided into modules.
- Though MS-DOS has a limited structuring, there is no clear separation between the different interfaces and level of functionality.
- For example, application programs can directly call the basic I/O routines to read/write data on disk instead of going through a series of interfaces.
- This exemption makes the MS-DOS system susceptible to malicious programs that may lead to system crash. Moreover, due to the lack of hardware protection and dual-mode operation in the Intel 8088 system (for which MS-DOS system was developed), the base hardware was directly accessible to the application programs.
- The original UNIX operating system also has simple structure like MS-DOS. The UNIX OS consists of following two separable parts:
  (i) Systems programs, and (ii) The kernel.
- Kernel consists of everything below the system-call interface and above the physical hardware.
- Kernel provides the file system, CPU scheduling, memory management, and other operating-system functions; a large number of functions for one level.
- 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. 1.17 shows original UNIX system.

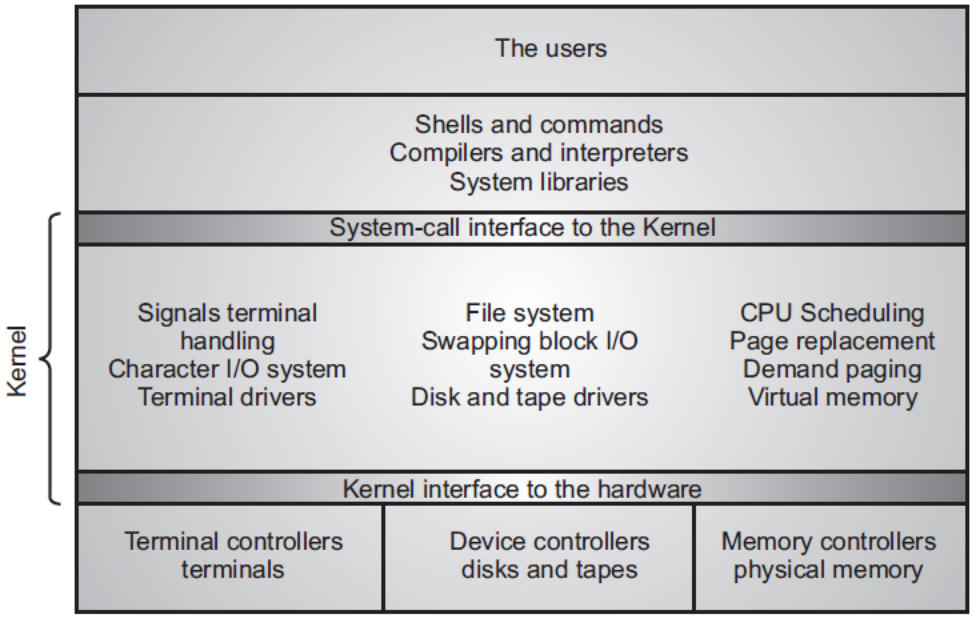



**Fig. 1.17: UNIX Structure**

## 2. Layered Structure:

- In the layered structure, the operating system is organized as a hierarchy of layers or levels with each layer built on the top of the layer below it.
- The top-most layer (layer N) is the User Interface (UI), while the bottom-most layer (layer 0) is the hardware. Each layer has a well defined function and comprises data structures and a set of routines.
- The layers are constructed in such a manner that a typical layer (say, layer N) is able to invoke operations on its lower layers and the operations of layer N can be invoked by its higher layers.
- Fig. 1.18 shows layered structure of an operating system.

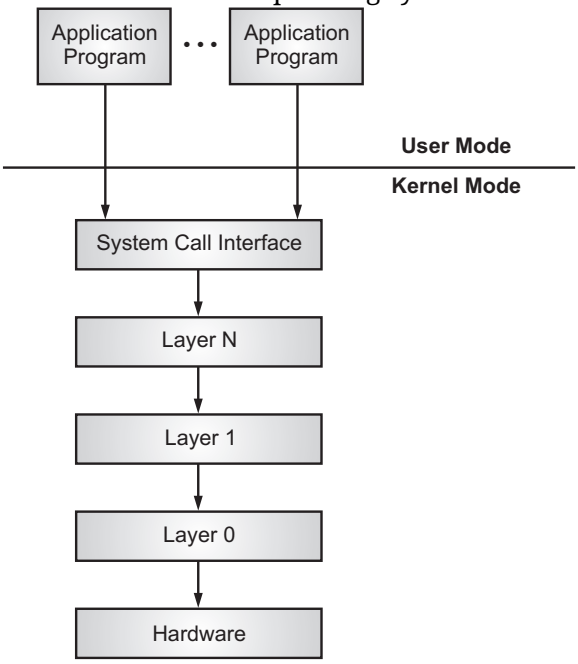



**Fig. 1.18: Layered Structure of Operating System**

- A layered design was first used in the THE (Technische Hogeschool Eindhoven) operating system designed by a team led by Edsger W. Dijkstra.
- Fig. 1.19 shows the structure of THE operating system.

| |
|---|
| **Layer 5**<br>The operator |
| **Layer 4**<br>User program |
| **Layer 3**<br>Input / Output management |
| **Layer 2**<br>Operator process communication |
| **Layer 1**<br>Memory and Drum management |
| **Layer 0**<br>Processor allocation and Multiprogramming |

**Fig. 1.19: Layered Structure of THE Operating System**

- Fig. 1.19 shows t following layers of THE operating system:
  - **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:**

(i) The layered structure gives modularity.
(ii) Easy for debugging and system verification.
(iii) Each layer in layered structure hides the existence of certain data structures, operations and hardware from higher level layers.

**Disadvantages:**

(i) The layered structure requires careful definition of the layers.
(ii) The layered structure is less efficient than other types.

**3. Microkernel Structure:** **[April 17, 19, Oct. 17]**

- The kernel is the core system software that behaves as a bridge between application software and hardware of the system. Kernel manages system resources.
- A microkernel is a piece of software or even code that contains the near-minimum amount of functions and features required to implement an operating system.
- Micro means small so a microkernel is a small kind of kernel in which only basic functionality of kernel is placed.
- A microkernel contains only the core functionalities of the system such as memory management, processor (CPU) scheduling, interprocess communication and so on.

- As UNIX expanded, the kernel becomes large and it is difficult to manage. The microkernel approach is used by operating system.
- In the mid-1980s, researchers at Carnegie Mellon University developed an operating system called Mach that modularized the kernel using the microkernel approach.
- It structures the operating system by moving all non-essential components from the kernel and implementing them as system and user-level programs into user space.
- So, small core operating system is running at kernel level and OS Services built from many independent user-level processes.
- Main function of micro-kernels is to provide communication facility between the client program and various services running in user mode, communication is achieved by message passing
- It provides minimal process and memory management, in addition to a communication facility.
- Examples of microkernel include Tru64 UNIX, Mac OS X kernel (Darwin), QNX (real-time operating system).
- Fig. 1.20 shows structure of a typical microkernel. Communication is provided by message passing.
- In the Fig. 1.20 the microkernel contains basic requirements such as memory management, CPU scheduling and interprocess communication. The only software executing at the privileged level i.e. kernel mode is the microkernel.
- The other functions of the operating system are removed from the kernel mode and run in the user mode. These functions may be device drivers, file servers, application program communication etc.

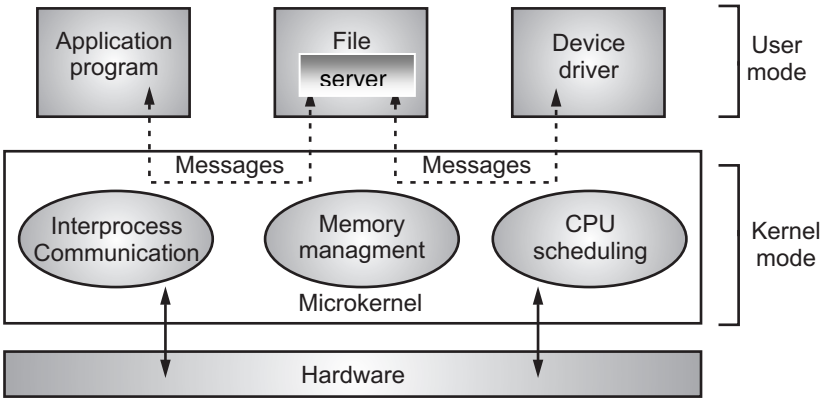



**Fig. 1.20: Structure of Microkernel Operating System**

**Advantages:**

(i) The microkernel structure provides high security and reliability.

(ii) Extending of operating system is easier, since all new services are added to user space and kernel modification is not required.

(iii) The microkernel structure is portable due to small size of kernel, it is easy to port operating system from one hardware to other.

**Disadvantages:**

(i) Performance of micro-kernels suffers due to increased system function overhead.
(ii) Micro kernel does not handle I/O communication directly.
(iii) The execution of the microkernel is slower as compared to the monolithic kernel.

**4. Monolithic Structure:**

- Monolithic structure is the oldest structure of the operating system. The monolithic operating system is also known as the monolithic kernel.
- A monolithic kernel is an operating system structure where the entire operating system is working in kernel space. The kernel can access all the resources present in the system.
- In the monolithic systems, each component of the operating system is contained within the kernel.
- All the basic services of OS like process management, file management, memory management, exception handling, process communication etc. are all present inside the kernel only.
- Monolithic operating system example includes Linux, BSD, OS/360, Open VMS, XTS-400, z/TPF and so on. Fig. 1.21 shows the monolithic structure.

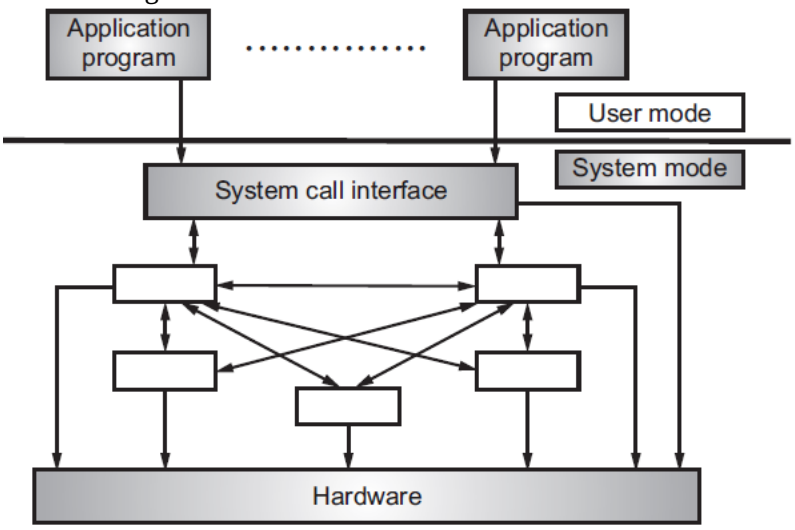



**Fig. 1.21: Structure of Monolithic Operating System**

**Advantages:**

(i) Simple and easy structure to implementation.
(ii) The execution of the monolithic kernel is quite fast due to direct access to all the services.

**Disadvantages:**

(i) If any service fails in the monolithic kernel, it leads to the failure of the entire system.

(ii) To add any new service, the entire operating system needs to be modified by the user.

(iii) Security issues are always there because there is no isolation among various servers' present in the kernel.

## 1.5 PROTECTION AND SECURITY

- Protection and security are the important things in the operating system to prevent interference with use of computer resources.
- Protection involves ensuring controlled access to the system resources. Security involves protecting the system from unauthorized users.
- If a computer system has multiple users and multiple processes, then protection mechanism is required to protect one from the others activities.
- There was need of protecting monitor memory being accessed by user as well as one user accessing memory or data of the other users.
- Protection is a mechanism for controlling the access of programs, processes, or users to the resources defined by a computer system. This mechanism must provide a means for specification of the controls to be imposed, as well as a means of enforcement.
- Security protects the information stored in the system (both data and code), as well as the physical resources of the computer system, from unauthorized access, malicious destruction or alteration, and accidental introduction of inconsistency.
- System protection refers to the mechanism for controlling the access to computer resources by various users and processes.
- Many kinds of protection are provided such as:
  1. **I/O Protection:** One user cannot read/write the data of another user.
  2. **Memory Protection:** Since in multiuser system, many program reside in the main memory. Each program is enclosed in two special fence registers, lower bound register and upper bound registers.
  3. **CPU Protection:** The program should not go in the infinite loop. So timer/counter is set.
- For protection and security, most operating system maintain a list of user names associated with unique user identifier. In Windows Vista, Security ID (SID) is used, which is unique. When the user logs in, the id is determined for authentication.

## 1.6 COMPUTING ENVIRONMENTS

- Operating systems are used in a variety of computing environments. A computing environment consists of a computer system, its interfaces with other systems and the services provided by its operating system to its users and their programs.

### 1.6.1 Traditional Computing

- A few years ago, the computer is connected to network, with servers. Mainframes are used with number of terminals attached.
- Today, web technology is used with traditional computing. Companies are establishing portals, blogs etc. provide web accessibility to their internal servers.

- Handheld systems are used to connect to wireless networks to use the company's web portal. Wireless technology is also used. Due to new technology, faster access with low cost is possible.
- Mobile computers can synchronize with computers to allow very portable use of company data or information.
- Mobile devices can also connect to wireless and cellular networks to use the company's web portals as well as other web resources.
- At home, most users had a single computer with a slow modem connection to the office, the Internet or both.
- Today, network-connection speeds once available only at great cost are relatively inexpensive, giving home users more access to more data.
- These fast data connections are allowing home computers to serve up web pages and to run computer networks that include client computers, servers and so on. Some home users have firewalls to protect their networks from security breaches.

## 1.6.2 Mobile Computing

- Mobile computing deals with human computer interaction using mobile devices. Mobile computing contains mobile hardware and software as well as mobile computing devices.
- Mobile computing refers to computing on handheld smart phones and tablet computers. These mobile device distinguishing physical features of being portable and lightweight.
- Historically, compared with desktop and laptop computers, mobile systems gave up screen size, memory capacity, and overall functionality in return for handheld mobile access to services such as e-mail and web browsing.
- Over the past few years, however, features of mobile devices have been so rich that the distinction in functionality between, say, a consumer laptop and a tablet computer may be difficult to discern.
- In fact, we might argue that the features of a contemporary mobile device allow it to provide functionality that is easier unavailable or impractical on a desktop or a laptop computer.
- Today, mobile systems are used not only for e-mail and web browsing but also for playing music and video, reading digital books, taking photos, and recording high-definition video.
- Two operating systems currently dominate mobile computing are Apple iOS and Google Android.
- iOS was designed to run on Apple iPhone and iPad mobile devices. Android is an open source and Linux-based operating system for mobile devices such as smart phones and tablet computers.
- Other mobile operating system includes Windows Mobile OS, Symbian OS, WebOS, BlackBerry OS and so on.

## 1.6.3 Distributed Computing

- A distributed computing environment contains multiple nodes that are physically separate but linked together using the network.
- All the nodes in this system communicate with each other and handle processes in tandem. Each of these nodes contains a small part of the distributed operating system software.
- A distributed system is managed by a distributed operating system. A distributed operating system manages the system shared resources used by multiple processes, the process scheduling activity, the communication and so on.
- A distributed system is a collection of physically separate, possibly heterogeneous, computer systems that are networked to provide users with access to the various resources that the systems maintain.
- Access to a shared resource increases computation speed, functionality, data availability, and reliability.
- Some operating systems generalize network access as a form of file access, with the details of networking contained in the network interfaces device driver.
- Others make users specifically invoke network functions. Generally, systems contain a mix of two modes – for example, FTP and NFS.
- A network operating system is an operating system that provides features such as file sharing across the network, along with a communication scheme that allows a different process on different computers to exchange messages.
- A computer running a network operating system acts autonomously from all other computers on the network, although it is aware of the network and is able to communicate with other networked computers.
- A distributed operating system provides a less autonomous environment. The different computers communicate closely enough to provide the illusion that only a single operating system controls the network.
- The nodes in the distributed systems can be arranged in the form of client/server systems or peer to peer systems.
    1. **Client-Server Systems:** In client server systems, the client requests a resource and the server provides that resource. A server may serve multiple clients at the same time while a client is in contact with only one server. Both the client and server usually communicate via a computer network and so they are a part of distributed systems.
    2. **Peer to Peer Systems:** The peer to peer systems contains nodes that are equal participants in data sharing. All the tasks are equally divided between all the nodes. The nodes interact with each other as required as share resources. This is done with the help of a network.
- Some of the examples of distributed operating systems are Solaris (for SUN multiprocessor workstations), IRIX (implementation of UNIX System V), AIX (for IBM RS/6000 computers), OSF/1, LOCUS and MICROS.

**Advantages:**

1. All the nodes in the distributed system are connected to each other. So nodes can easily share data with other nodes.
2. More nodes can easily be added to the distributed system i.e. it can be scaled as required.
3. Failure of one node does not lead to the failure of the entire distributed system. Other nodes can still communicate with each other.

**Disadvantages:**

1. It is difficult to provide adequate security in distributed systems because the nodes as well as the connections need to be secured.
2. The database connected to the distributed systems is quite complicated and difficult to handle as compared to a single user system.
3. Overloading may occur in the network if all the nodes of the distributed system try to send data at once.

## 1.6.4 Client-Server Computing

- In recent years there have been significant advances in the development of high performance personal computer and networks.
- There is now an identifiable trend in industry toward downsizing that is replacing expensive mainframe computers with more cost-effective networks of personal computer that achieve the same or even better results. This trend has given rise of the client-server computing.
- In client-server computing the server systems satisfy request generated by clients system. Both the client and server usually communicate via a computer network.
- Fig. 1.22 shows general structure of client-server computing. A client is defined as a requester of services and a server is defined as the provider of services.
- Server systems can be broadly categorized as follows:
  1. **Computer-Server System:** Provides an interface to which a client can send a request to perform an action (for example, read data). In response, the server executes the action and sends the results to the client. A server running a database that responds to client requests for data is an example of such a system.
  2. **File-Server System:** Provides a file-system interface where clients can create, update, read, and delete files. An example of such a system is a web server that delivers files to clients running web browsers.

**Advantages:**

1. It is easy to replace, upgrade or relocate the nodes in the client-server computing because all the nodes are independent and request data only from the server.
2. All the required data is concentrated in a single place i.e. the server. So it is easy to protect the data and provide authorization and authentication.

3. The server need not be located physically close to the clients. Yet the data can be accessed efficiently.

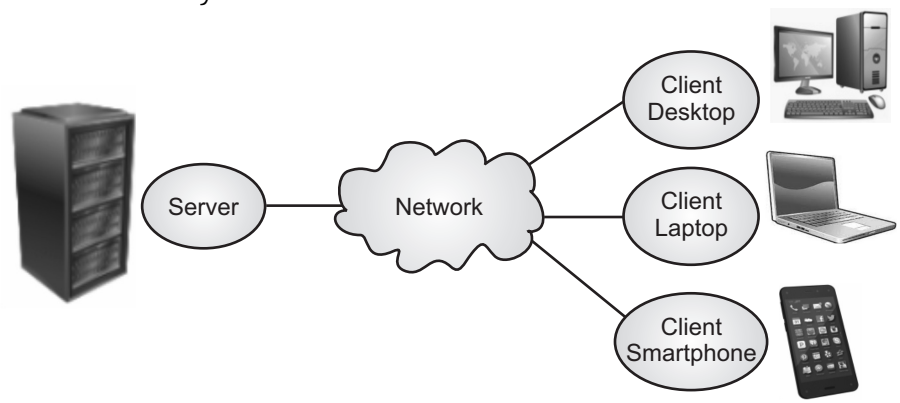



**Fig. 1.22: Client-server Computing**

**Disadvantages:**

1. If all the clients simultaneously request data from the server, it may get overloaded. This may lead to congestion in the network.
2. If the server fails for any reason, then none of the requests of the clients can be fulfilled. This leads of failure of the client server network.
3. The cost of setting and maintaining a client server model are quite high.

## 1.6.5 Peer-to-Peer Computing

- In Peer-to-Peer (P2P) computing, clients and servers are not distinguishing from one another. All nodes in the network are considered peers and they can act as server or client depending upon the service provided.
- The Peer-to-Peer (P2P) computing contains nodes that are equal participants in data sharing.
- In peer-to-peer computing, all the tasks are equally divided between all the nodes. The nodes interact with each other as required as share resources.
- The nodes in peer-to-peer computing both use resources and provide resources. Most modern operating systems such as Windows and Mac OS contain software to implement peer to peer networks.
- Fig. 1.23 shows peer-to-peer computing environment. To participate in peer-to-peer computing, the node must first join the network of peers. Once, a node has joined, it services to or from other node.
- Any node wants to specific service, it first contacts with centralized lookup service to determine which node provides a service.
- The node can broadcast a request for a service on network and finds the response from the peer who is going to provide a service.

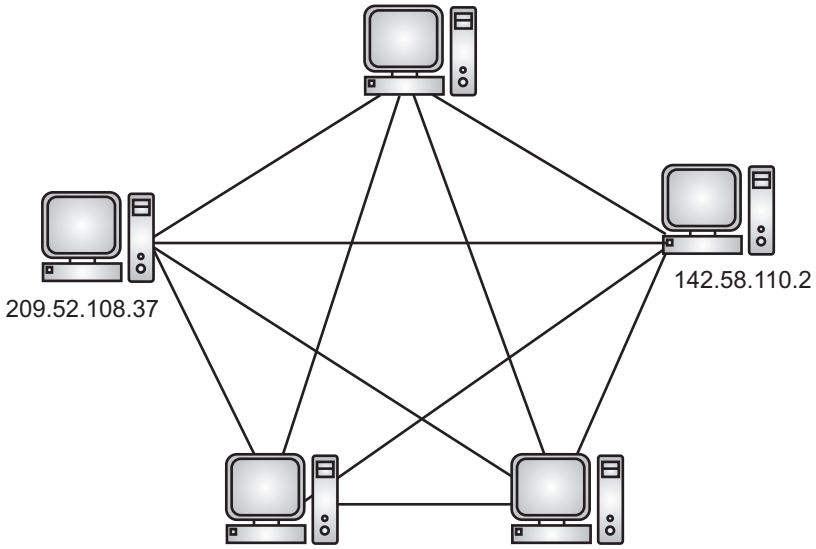



**Fig. 1.23: Peer-to-Peer Computing Environment**

**Advantages:**

1. Each computer in the peer-to-peer computing manages itself. So, the network is quite easy to set up and maintain.
2. It is easy to scale the peer-to-peer network and add more nodes.
3. In peer-to-peer computing the cost of the server is saved.

**Disadvantages:**

1. It is difficult to backup the data as it is stored in different computer systems and there is no central server.
2. It is difficult to provide overall security in the peer-to-peer computing as each system is independent and contains its own data.

## 1.7 OPEN SOURCE OPERATING SYSTEM

- An open-source operating system is an operating system whose code has been made publicly and freely available to anyone who wants to see it and modify it.
- Open source operating systems are available in source code format rather than as compiled binary code.
- An open source operating system is an operating system whose code has been made publicly and freely available to anyone who wants to see it and modify.
- Open source refers to a program or software in which the source code (the form of the program when a programmer writes a program in a particular programming language) is available to the general public for use and/or modification from its original design free of charge.
- Open source allows easy access of source code to the general public that can be edited and re-released in a customized format. Open source is a term that originally referred to Open Source Software (OSS).

- Open source operating systems are released under a license where the copyright holder allows others to study, change as well as distribute the software to other people without any cost or charge.
- GNU/Linux is the most popular open source operating system released under the GNU General Public License (GPL).
- FreeBSD is a free open source OS, a Unix-like operating system that's based on Berkeley Software Distribution (BSD).
- Closed source software is highly guarded. Only the owners of the source code have the legal right to access that code. Microsoft Windows is an example of closed source.
- Closed source code cannot be legally altered or copied, and the user pays only to use the software as it is intended they cannot modify it for new uses nor share it with their communities.

**Advantages:**

1. Open source operating systems are available freely on internet without any charge.
2. In open source operating system user can modify and edit changes as per his/her choice and requirements.

**Disadvantages:**

1. Constant updates are headaches sometimes as users have to keep update his/her system and code.
2. Frequent modifications are challenge for users to decide which open source system is better.
3. Sometimes hardware does not compatible with open source operating systems.

## 1.8 BOOTING [Oct. 16]

- Booting the system is done by loading the kernel into main memory, and starting its execution. The process of starting a computer by loading the kernel is known as system booting.
- The program called bootstrap program or bootstrap loader is used to locate the kernel and loads it into main memory and starts execution.
- After an operating system is generated, it must be made available for use by the computer hardware. But how does the computer hardware know where the kernel is or how to load that kernel?
- The procedure of starting a computer by loading the kernel is known as booting the computer system.
- On number of computer systems, a small piece of code known as the bootstrap program or bootstrap loader locates the kernel, loads it into main memory, and starts its execution. **[April 16, 19]**
- Some computer systems, such as PCs, use a two-step process in which a simple bootstrap loader fetches a more complex boot program from disk, which in turn loads the kernel.

- When a CPU receives a reset event - for instance, when it is powered up or rebooted - the instruction register is loaded with a predefined memory location, and execution starts there.
- At that location is the initial bootstrap program. This program is in the form of Read Only Memory (ROM), because the RAM is in an unknown state at system startup.
- ROM is convenient because it needs no initialization and cannot be infected by a computer virus.
- Some systems like cellular phones, PDAs, etc. store the entire operating system in ROM. Storing the operating system in ROM is suitable for small operating systems, simple supporting hardware, and rugged operation.
- A problem with this approach is that changing the bootstrap code requires changing the ROM hardware chips.
- Some systems resolve this problem by using Erasable Programmable Read Only Memory (EPROM), which is read only except when explicitly given a command to become writable.
- All forms of ROM are also known as firmware, since their characteristics fall somewhere between those of hardware and those of software.
- A problem with firmware in general is that executing code there is slower than executing code in RAM.
- Some systems store the operating system in firmware and copy it to RAM for fast execution. A final issue with firmware is that it is relatively expensive, so usually only small amounts are available.
- Fig. 1.24 shows booting process of computer system.

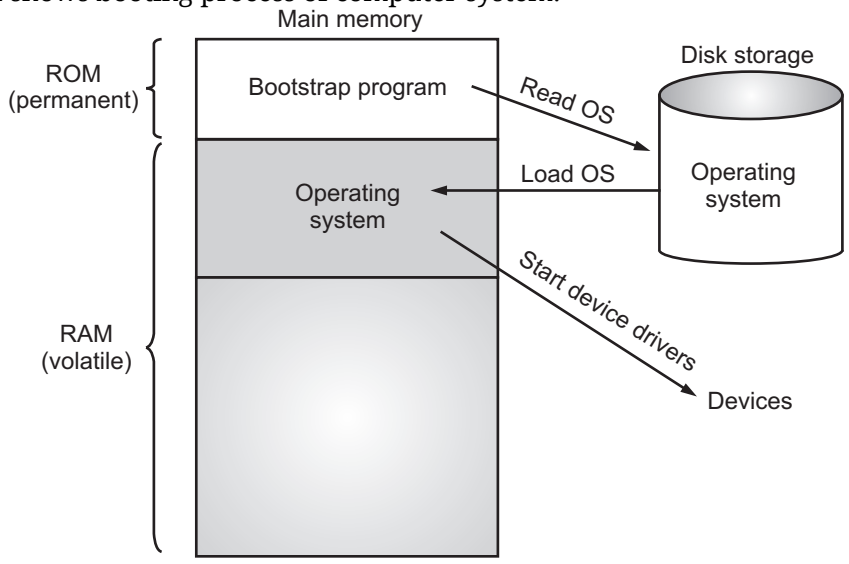



**Fig. 1.24: Booting Procedure in Computer System**

- For large operating systems like Windows, Mac OS X, and UNIX or for systems that change frequently, the bootstrap loader is stored in firmware, and the operating system is on disk.

- In this case, the bootstrap runs diagnostics and has a bit of code that can read a single block at a fixed location (say block zero) from disk into memory and execute the code from that boot block.
- The program stored in the boot block may be sophisticated enough to load the entire operating system into memory and begin its execution.
- More typically, it is simple code (as it fits in a single disk block) and only knows the address on disk and length of the remainder of the bootstrap program.
- All of the disk-bound bootstrap, and the operating system itself, can be easily changed by writing new versions to disk. A disk that has a boot partition is called a boot disk or system disk.
- Now that the full bootstrap program has been loaded, it can traverse the file system to find the operating system kernel, load it into memory, and start its execution. It is only at this point that the system is said to be running.

## 1.9 OPERATING SYSTEM SERVICES

- An operating system provides an environment for the execution of programs. It provides certain services to programs and to the users of those programs.
- Fig. 1.25 shows services provided by operating system.

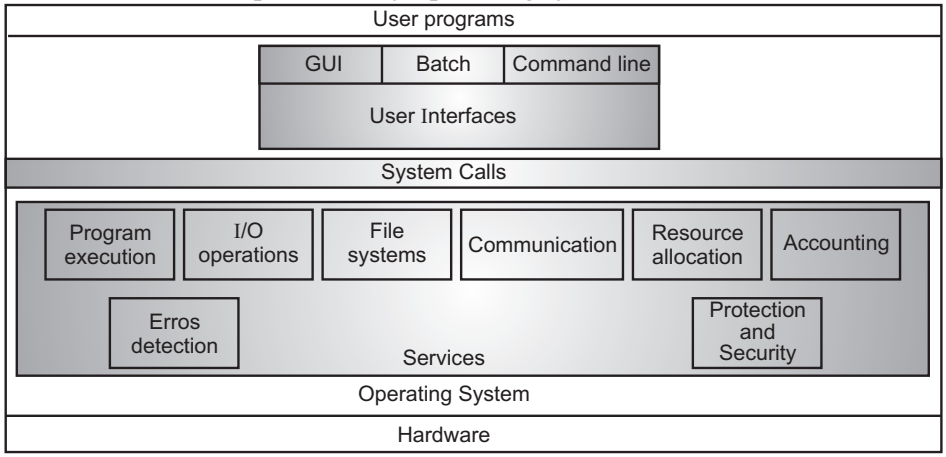



**Fig. 1.25: View of Operating System Services**

- Following are a few common services provided by an operating system:
  1. **User Interface:** Providing a User Interface (UI) to interact with users is essential for an operating system. This interface can be in one of the several forms. One is Command Line Interface (CLI) in which users interact with the operating system by typing commands. Another is batch interface, in which several commands and directives to control those commands are collected into files which are then executed. Another is Graphical User Interface (GUI), in which users interact with the system with a pointing device, such as a mouse.
  2. **Program Execution:** The purpose of computer system is to allow the users to execute programs in an efficient manner. The operating system provides an environment where the user can conveniently run these programs. The user does not have to worry about the memory allocation or de-allocation or any other thing

because these things are taken care of by the operating system. The operating system must be able to load the program into memory and to execute it. The program must be able to terminate its execution, either normally or abnormally.

3. **I/O Operations:** Each program requires an input and after processing the input submitted by user it produces output. This involves the use of I/O devices. The input may be either from a file on the disk or from some other input device. The output may be written to some file on the disk or sent to some output devices such as printer, plotter. Since user programs cannot execute I/O operations directly, the operating system must provide some means to perform I/O.
4. **File System Manipulation:** While working on the computer, generally a user is required to manipulate various types of files like as opening a file, saving a file and deleting a file from the storage disk. Program needs to read a file or write a file. The operating system gives the permission to the program for operation on file. Maintain details of files or directories with their respective details. This is an important task that is also performed by the operating system.
5. **Communications:** Operating system performs the communication among various types of processes in the form of shared memory. In multitasking environment, the processes need to communicate with each other and to exchange their information. These processes are created under a hierarchical structure where the main process is known as parent process and the sub processes are known as child processes.
6. **Error Detection:** Error can occur anytime and anywhere. Error may occur in CPU, in I/O devices or in the memory hardware. Operating system deals with hardware problems. To avoid hardware problems the operating system constantly monitors the system for detecting the errors and fixing these errors (if found). The main function of operating system is to detect the errors like bad sectors on hard disk, memory overflow and errors related to I/O devices. After detecting the errors, operating system takes an appropriate action for consistent computing.
7. **Resource Allocation:** In the multitasking environment, when multiple jobs are running at a time, it is the responsibility of an operating system to allocate the required resources (like as CPU, main memory, tape drive or secondary storage etc.) to each process for its better utilization. For this purpose various types of algorithms are implemented such as process scheduling, CPU scheduling, disk scheduling etc.
8. **Accounting:** Operating system keeps an account of all the resources accessed by each process or user. In multitasking, accounting enhances the system performance with the allocation of resources to each process ensuring the satisfaction to each process.
9. **Protection:** If a computer system has multiple users and allows the concurrent execution of multiple processes, then the various processes must be protected from one another's activities. Protection refers to mechanism or a way to control the access of programs, processes, or users to the resources defined by a computer system. Providing protection to program, data, and files and to ensure data security.

## 1.10 SYSTEM CALLS [Oct. 18]

- System call provides on interface between a running program and an operating system.
- A system call is the programmatic way in which a computer program requests a service from the kernel of the operating system on which it is executed.
- The interface between a process and an operating system is provided by system calls. In general, system calls are available as assembly language instructions. They are also included in the manuals used by the assembly level programmers.
- System calls are usually made when a process in user mode requires access to a resource. Then it requests the kernel to provide the resource via a system call.
- System calls provide an interface to the services made available by an operating system. These calls are generally available as routines written in C and C++, although certain low-level tasks (for example, tasks where hardware must be accessed directly), may need to be written using assembly-language instructions.
- Before we discuss how an operating system makes system calls available, let's first use an example to illustrate how system calls are used: writing a simple program to read data from one file and copy them to another file.
- The first input that the program will need is the names of the two files: the input file and the output file. These names can be specified in many ways, depending on the operating-system design.
- One approach is for the program to ask the user for the names of the two files. In an interactive system, this approach will require a sequence of system calls, first to write a prompting message on the screen and then to read from the keyboard the characters that define the two files.
- On mouse-based and icon-based computer systems, a menu of file names is usually displayed in a window. The user can then use the mouse to select the source name, and a window can be opened for the destination name to be specified. This sequence requires many I/O system calls.
- Once the two file names are obtained, the program must open the input file and create the output file. Each of these operations requires another system call.
- When the program tries to open the input file, it may find that there is no file of that name or that the file is protected against access. In these cases, the program should print a message on the console (another sequence of system calls) and then terminate abnormally (another system call).
- If the input file exists, then we must create a new output file. We may find that there is already an output file with the same name. This situation may cause the program to abort (a system call), or we may delete the existing file (another system call) and create a new one (another system call).

- Another option, in an interactive system, is to ask the user (via a sequence of system calls to output the prompting message and to read the response from the terminal) whether to replace the existing file or to abort the program.
- Now that both files are set up, we enter a loop that reads from the input file (a system call) and writes to the output file (another system call). Each read and write must return status information regarding various possible error conditions.
- On input, the program may find that the end of the file has been reached or that there was a hardware failure in the read (such as a parity error).
- The write operation may encounter various errors, depending on the output device (no more disk space, printer out of paper, and so on).
- Finally, after the entire file is copied, the program may close both files (another system call), write a message to the console or window (more system calls), and finally terminate normally (the final system call).
- Fig. 1.26 shows implementation of a system call. System call offers the services of the operating system to the user programs via API (Application Programming Interface).
- Typically, a number is associated with each system call, and the system-call interface maintains a table indexed according to these numbers.
- The system call interface then invokes the intended system call in the operating system kernel and returns the status of the system call and any return values.
- The caller needs to know nothing about how the system call is implemented or what it does during execution.
- Rather, it just needs to obey the API (Application Programming Interface) and understand what the operating system will do as a result of the execution of that system call.

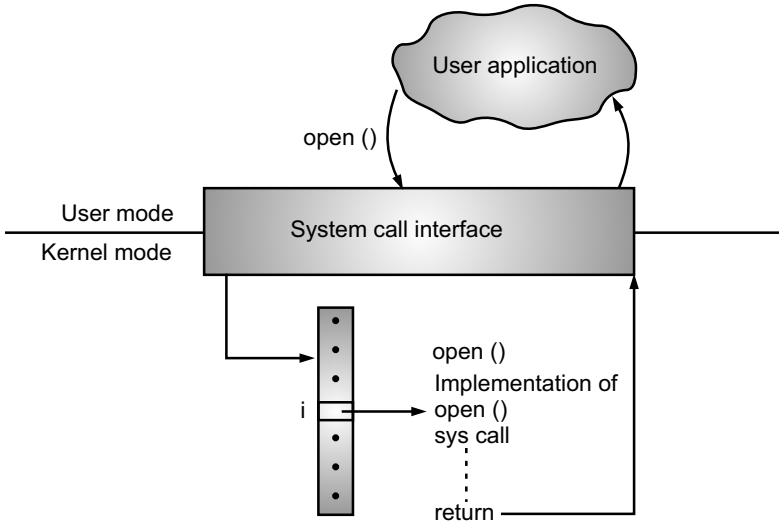



**Fig. 1.26: System Call Implementation or Invoking System Call [open()]**

- Most of the details of the operating-system interface are hidden from the programmer by the API and are managed by the run-time support library. Often, more information is required than simply the identity of the desired system call.
- The standard C language provides a portion of the system call interface for many versions of UNIX and Linux operating systems.
- For example, the C language program invokes the printf() statement. The C language library intercepts the necessary system call(s) in the operating system write() system call.
- The C language library takes the value returned by write() system call and passes it back to the user program as shown in Fig. 1.27.

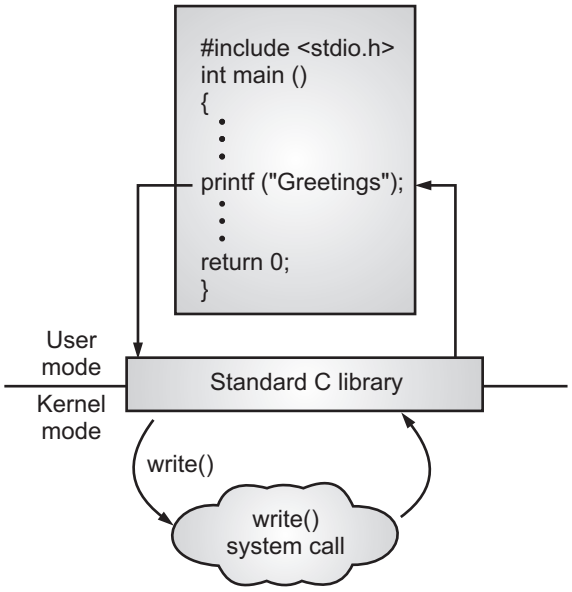



**Fig. 1.27: C Language Library handling write() System Call**

## 1.10.1 Working of System Calls

- Fig. 1.28 shows working of a system call.

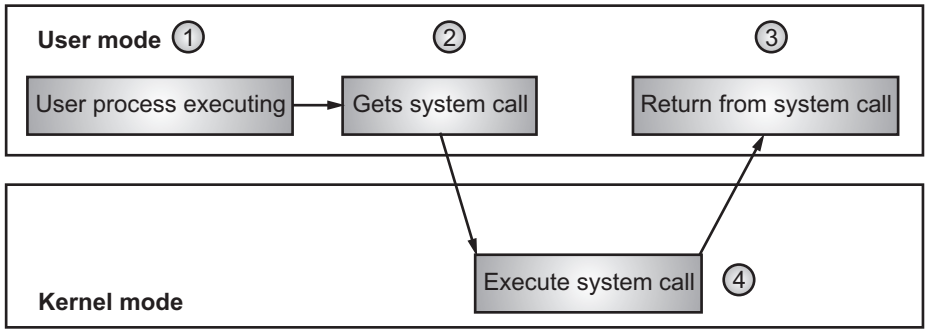



**Fig. 1.28: Working of System Call**

- The steps for system call are given below:
  **Step 1:** The processes executed in the user mode till the time a system call interrupts it.
  **Step 2:** After that, the system call is executed in the kernel-mode on a priority basis.
  **Step 3:** Once system call execution is over, control returns to the user mode.,
  **Step 4:** The execution of user processes resumed in Kernel mode.

**System Call Parameters:**

- Three general methods exist for passing parameters to the OS:
  1. Parameters can be passed in registers.
  2. When there are more parameters than registers, parameters can be stored in a block and the block address can be passed as a parameter to a register.
  3. Parameters can also be pushed on or popped off the stack by the operating system.

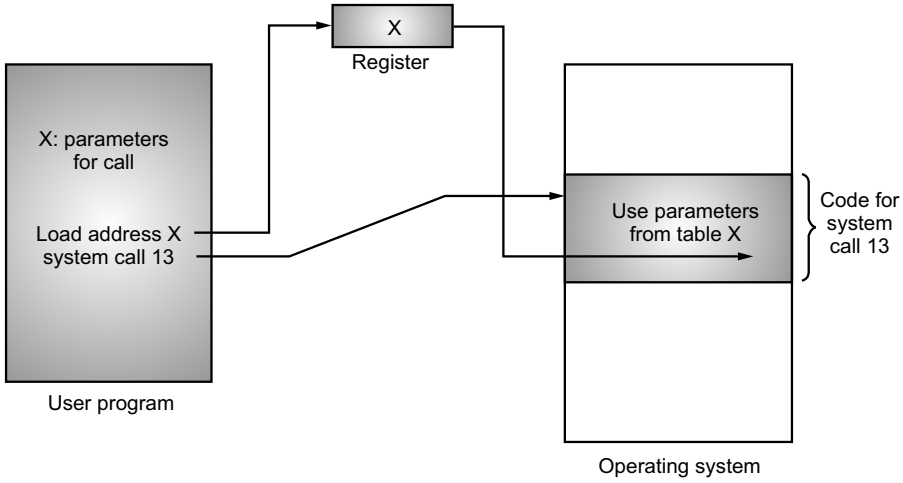



**Fig. 1.29: Passing Parameters in System Calls**

## 1.10.2 Types of System Calls [Oct. 18]

- A system call is a programmatic method in which a computer program requests a service from the kernel of the OS.
- There are mainly five types of system calls. These are explained in detail as follows:
  1. **Process Control:** These system calls deal with processes. Process is nothing but program in execution. Some examples of these system calls are end, abort, load, execute, create new, process, terminate process, setting and retrieving process attributes etc. **[Oct. 18]**
  2. **File Management:** These types of system calls are used to handle or dealing with files. Some examples of these system calls are creating files, delete files, open, close, read, write files etc.
  3. **Device Management:** This system call is used to deal with devices manipulation. Some examples of these system calls are call request device, release device, read and write device operations etc.

4. **Information Maintenance:** These system calls handle information and its transfer between the operating system and the user program. Some examples of these system calls are get time or date, set time or date, get system data, set system data etc.
5. **Communication:** These system calls are useful for interprocess communication (IPC). Examples of these system calls are creating and deleting connection, send and receive messages, read and write messages etc. There are two common models of IPC namely, the message passing model and the shared-memory model. In the **message-passing model**, the communicating processes exchange messages with one another to transfer information. Message-passing model uses a common mailbox to pass messages between processes. In **shared memory model**, processes use shared memory creates and shared memory attaches system calls to create and gain access to regions of memory owned by other processes.

## PRACTICE QUESTIONS

**Q.I Multiple Choice Questions:**

1. What is operating system?
   (a) collection of programs that manages hardware resources
   (b) system service provider to the application programs
   (c) link to interface the hardware and application programs
   (d) All of the above mentioned
2. Which one of the following is not true?
   (a) kernel is the program that constitutes the central core of the operating system
   (b) kernel is the first part of operating system to load into memory during booting
   (c) kernel is made of various modules which cannot be loaded in running operating system
   (d) kernel remains in the memory during the entire computer session
3. What is the main function of the command interpreter?
   (a) to get and execute the next user-specified command
   (b) to provide the interface between the API and application program
   (c) to handle the files in operating system
   (d) none of the mentioned
4. Which of the following is typically a part of the operating system but not the kernel?
   (a) Graphical User Interface (b) Network Management
   (c) Device Driver Management (d) Compiler
5. Which of the following is abstracted by operating system?
   (a) Processor (b) Memory
   (c) Network Cards (d) All of the above mentioned

6. An operating system with multiprogramming capability is one that,
   (a) allows several users to use the same program at once by giving each a slice of time
   (b) loads several independent processes into memory and switches the CPU from one job to another as required
   (c) runs programs over more than one processor
   (d) None of the above
7. Shell is used because of,
   (a) Each user needs protection from other users.
   (b) Users need exclusive environment to work on a system.
   (c) To protect OS from inadvertent unsafe access to kernel space.
   (d) Shell holds all the resources in the system.
8. Which of the following statements are true?
   (a) An exclusively single user system can not be a multiprogramming system.
   (b) A uni-programming system will have higher processor utilization because it does not suffer from switching overheads witnessed in multi-programming systems.
   (c) A multiprogramming system enhances primary memory utilization.
   (d) None of above
9. A time sharing system is always a multiprogramming system.
   (a) True (b) False
   (c) can't say (d) None
10. A system is a multi-programming system because,
   (a) There are multiple devices connected to the system
   (b) There are multiple persons who have an account on that system.
   (c) There are multiple memory resident ready-to-run programs in the system
   (d) None
11. Identify which of these are real-time applications scenarios:
   (a) An on-line bus ticketing system
   (b) Printing of annual report of a company's annual report
   (c) Reconciling a day's transactions in an account book of a small company
   (d) An aircrafts' yaw control system
12. Which of the following describes the RTOS design philiosophybest?
   (a) Maximize the throughput of the system
   (b) Maximize the processor utilization
   (c) Minimizing the response time
   (d) Response within certain stipulated time period

13. Which is the first program run on a computer when the computer boots up?
    (a) System software (b) Operating system
    (c) System operations (d) None of the above
14. Which system contains multiple nodes that are physically separate but linked together using the network?
    (a) Monolithic (b) Layered
    (c) Distributed (d) Client-server
15. The initial program that is run when the computer is powered up is called as,
    (a) boot program (b) bootloader
    (c) initializer (d) bootstrap program
16. In the layered approach of operating systems,
    (a) Bottom Layer(0) is the User interface
    (b) Highest Layer(N) is the User interface
    (c) Bottom Layer(N) is the hardware
    (d) Highest Layer(N) is the hardware
17. Operating system provides the different types of services to the user and for accessing these services, the interface is provided by the an OS is,
    (a) system calls (b) Application Programming Interface
    (c) Native library (d) None of the above
18. The goal of an operating system is to,
    (a) Execute user programs and make solving user problems easier
    (b) Make the computer system convenient to use
    (c) Use the computer hardware in an efficient manner
    (d) All of the mentioned
19. Information maintenance is a type of,
    (a) input (b) output
    (c) system call (d) processes
20. Which is a program that acts as an intermediary between a user of a computer and the computer hardware?
    (a) System Manager (b) Application program
    (c) Operating system (d) Source code
21. System calls of the operating system provide interface to,
    (a) programs (b) processes
    (c) services (d) both (a) and (b)
22. Which controls and coordinates use of hardware among various applications and users?
    (a) System Manager (b) Operating system
    (c) Application program (d) Source code

23. Which system call is used to access data from a file that is stored in the file system?
    (a) close() (b) write()
    (c) open() (d) read()
24. An operating system may be interrupted by,
    (a) Hardware interrupts
    (b) Software interrupts
    (c) Both hardware interrupts as well as software interrupts
    (d) Neither hardware interrupts nor software interrupts
25. Which is changes the system from user mode to kernel mode and vice versa?
    (a) Operating System (b) Interrupt
    (c) System Call (d) Hardware Manager
26. Which of the following is an example of spooled device?
    (a) A graphic display device
    (b) A line printer used to print the output of a number of jobs
    (c) A terminal used to enter input data to a running program
    (d) A secondary storage device in a virtual memory system
27. Which of the following requires a device driver?
    (a) Register (b) Cache
    (c) Main memory (d) Disk
28. Which of the following operating system reads and reacts in actual time?
    (a) Quick Response System (b) Real Time System
    (c) Time Sharing System (d) Batch Processing System
29. Logical extension of multiprogramming of operating system is,
    (a) time sharing (b) multi tasking
    (c) single programming (d) both (a) and (b)
30. Unix OS is an,
    (a) Time Sharing Operating System (b) Multi-User Operating System
    (c) Multi-tasking Operating System (d) All the Above
31. Which of the following is not advantage of multiprogramming?
    (a) Increased throughput
    (b) Shorter response time
    (c) Decreased operating system overhead
    (d) Ability to assign priorities to jobs
32. Which is the most essential system software that manages the operations of a computer?
    (a) Operating System (b) Booting System
    (c) Hardware System (d) None of the Above

33. The primary objectives of an operating system is,
    (a) to make computer system convenient to use
    (b) to utilize computer hardware in an efficient manner
    (c) expected to meet is their ability to evolve
    (d) All of Above
34. The role of operating system is to provide an _____ in which user user is able to run application software.
    (a) interface (b) environment
    (c) both (a) & (b) (d) None of the Above
35. The logical view of operating system describes the _____ functional organization of an operating system.
    (a) external (b) basic
    (c) internal (d) None of the Above
36. Operating system used for processing a set of jobs without any human intervention is called as,
    (a) code processing operating system
    (b) batch processing operating system
    (c) data processing operating system
    (d) All of the Above
37. The process of transferring data intended for a peripheral device on to a disk for some period of time is known as,
    (a) spooling (b) caching
    (c) multiprocessing (d) All of the Above
38. An operating system is an,
    (a) application software (b) firmware software
    (c) system software (d) All of the Above
39. Sharing the processor, when two or more programs reside in memory at the same time, is referred as,
    (a) batch processing (b) multiprogramming
    (c) time sharing (d) None of the Above
40. In real time operating system,
    (a) all processes have the same priority
    (b) process scheduling can be done only once
    (c) a task must be serviced by its deadline period
    (d) None of the Above
41. Which is a more than a mere collection of computers connected to a computer network functioning of individual computers to achieve effective utilization of resources in the system?
    (a) distributed system (b) parallel system
    (c) multimedia system (d) None of the Above

42. A system consisting of more than one processor i.e. tightly coupled is a,
    (a) multimedia system (b) distributed system
    (c) parallel system (d) All of the Above
43. Which is the conceptual design and fundamental operational structure of a computer system?
    (a) processing (b) architecture
    (c) programming (d) None of the Above
44. Which is a system is an operating system with multiple CPUs?
    (a) multiprogramming (b) multitasking
    (c) clustered (d) All of the Above
45. Operating systems for handheld devices that contain computer are called as,
    (a) mobile operating system (b) embedded operating system
    (c) handheld operating system (d) All of the Above
46. Operating system _____ refers to the use of software to allow system hardware to runmultiple instances of different operating systems concurrently, allowing to run different applications requiring different operating systems on one computer system.
    (a) virtualization (b) architecture
    (c) clustering (d) All of the Above
47. Which envolves ensuring controlled access to the system resources while which involves protecting system form unauthorized users?
    (a) Protection, Security (b) Memory, processor
    (c) Buffering, Spooling (d) All of the Above
48. The client-server computing enviroment provides an interface to which a _____ can send a request to perform an action in response, the _____ executes the action and sends back results to the client.
    (a) node, computer (b) client, server
    (c) network, topology (d) None of the Above
49. Which is a type of computing that delivers computing, storage and even apllications as a service across a computer network?
    (a) grid computing (b) network computing
    (c) cloud computing (d) All of the Above
50. Internet is an example of which system.
    (a) client-server (b) multiprogramming
    (c) batch (d) handheld
51. Which is an operating system that helps to run other application software on mobile devices?
    (a) Network operating system (b) Distributed operating system
    (c) Mobile operating system (d) All of the Above

52. Following which tasks are an operating system is performs:
    (a) File management and Memory management
    (b) Storage management and process management
    (c) Handling input and output and Controlling peripheral devices
    (d) All of the above mentioned
53. Which is actually the heart of operating systems?
    (a) Shell (b) Kernel
    (c) Hardware (d) User
54. The client server computing works with a system of,
    (a) Request (b) Response
    (c) Both (a) and (b) (d) None of the above
55. Which refers to a program in which the source code is available to the general public for use and/or modification from its original design free of charge?
    (a) Open source (b) Closed source
    (c) Both (a) and (b) (d) None of the above
56. Following which services provided by an operating system,
    (a) Program execution and I/O operations
    (b) File System manipulation and Communication
    (c) Error Detection and Resource Allocation
    (d) All of the above mentioned
57. Which system call deal with processes such as process creation, process termination?
    (a) Process control system call (b) File management system call
    (c) Communication system call (d) Device management system call
58. Which involves protecting the system from deliberate attacks, either from legitimate users of the system attempting to gain unauthorized access?
    (a) Protection (b) Security
    (c) Password (d) None of the above

## Answers

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1. (d) | 2. (c) | 3. (a) | 4. (a) | 5. (d) | 6. (b) | 7. (c) | 8. (b) | 9. (a) | 10. (c) |
| 11. (d) | 12. (d) | 13. (b) | 14. (c) | 15. (d) | 16. (b) | 17. (a) | 18. (d) | 19. (c) | 20. (c) |
| 21. (c) | 22. (b) | 23. (d) | 24. (c) | 25. (c) | 26. (b) | 27. (d) | 28. (b) | 29. (d) | 30. (d) |
| 31. (c) | 32. (a) | 33. (d) | 34. (c) | 35. (c) | 36. (b) | 37. (a) | 38. (c) | 39. (b) | 40. (c) |
| 41. (a) | 42. (c) | 43. (b) | 44. (c) | 45. (d) | 46. (a) | 47. (a) | 48. (b) | 49. (c) | 50. (a) |
| 51. (c) | 52. (d) | 53. (b) | 54. (c) | 55. (a) | 56. (d) | 57. (a) | 58. (b) | | |

**Q.II Fill in the Blanks:**

1. An operating system is an interface between the _____ and the hardware and enables the interaction of a computer's hardware and software.
2. When the user application requests for a service from the operating system or an interrupt occurs or _____.
3. _____ refers to providing a protection system to computer system resources such as CPU, memory, disk, software programs and most importantly data/information stored in the computer system.
4. An operating system for smart phones, tablets and other mobile devices is called _____ OS.
5. _____ acts as an interface between the applications and actual data processing done at hardware level (CPU, disk memory etc.).
6. _____ is a mobile operating system (OS) currently developed by Google,
7. An example of a layered OS is _____.
8. An OS is _____ driven.
9. Computing environments evolve continuously to provide better _____ to users.
10. In a _____ computing environment multiple nodes are connected together using network but physically they are separated.
11. The _____ computing the client sends a request to the server and the server responds with the desired information.
12. A _____ operating system is an extension of the network operating system that supports higher levels of communication and integration of the machines on the network.
13. Operating system is system software that works as an extended machine as well as a _____ manager.
14. An example of a client-server computing system is a _____ server which returns the web pages to the clients that requested them.
15. In _____ to computing, all the nodes are equal and share data with each other directly.
16. A _____ or exception is a software-generated interrupt caused either by an error or a user request
17. _____ is the best-known and most-used open source operating system.
18. OS is the first software to be loaded into computer memory when the computer is switched on and this is called _____.
19. A software whose source code is freely distributed with a license to study, change and further distributed to anyone for any purpose is called _____ software.
20. A _____ interrupts the computer system after a specified time.
21. _____ is a process of that starts operating systems when the user turns on a computer system.
22. _____ refers to a mechanism or a way to control the access of programs, processes, or users to the resources defined by a computer system.

23. An Operating System provides programs an environment to _____.
24. A system call is a _____ made by a program to the operating system.
25. Booting is a startup sequence that starts the operating system of a computer when it is turned _____.
26. Bootstrap _____ locates the kernel, loads it into main memory and starts its execution.
27. Information _____ system calls handle information and its transfer between the operating system and the user program.
28. Network operating system runs on _____.
29. _____ system calls are useful for interprocess communication.
30. The _____ system call is used to provide access to a file in a file system.
31. Example of _____ time includes Aircraft systems, Medical critical care System, etc.
32. In client/server network operating systems the clients run programs and access data that are stored on the _____.

## Answers

| | | | |
|---|---|---|---|
| 1. user | 2. system call | 3. Security | 4. mobile |
| 5. Kernel | 6. Android | 7. Windows NT | 8. interrupt |
| 9. quality of service | 10. distributed | 11. client-server | 12. distributed |
| 13. resource | 14. web | 15. peer-to-peer | 16. trap |
| 17. Linux | 18. booting | 19. open source | 20. timer |
| 21. Booting | 22. Protection | 23. execute | 24. request |
| 25. on | 26. loader | 27. maintenance | 28. server |
| 29. Communication | 30. open() | 31. hard-real | 32. server |

**Q.III State True or False:**

1. An operating system is an interface that allows the user application programs to interact with the system hardware.
2. When system boots then hardware starts in kernel mode and when operating system is loaded then it start user application in user mode.
3. Windows Phone 7 is the latest mobile OS developed by Apple.
4. A trap is a software generated interrupt caused by an error.
5. Shell lies in the center of the operating ystem which manages the communication between the user level applications and the hardware installed on the system.
6. Examples of the microkernel OS are Linux, SunOS, MULTICS, OpenVMS, BSD etc.
7. A mobile operating system is an operating system that helps to run other application software on mobile devices.
8. A distributed system contains multiple nodes that are physically separate but linked together using the network.
9. In the client server computing the client and server should follow a common communication protocol so they can easily interact or communicate with each other.

10. A computer program makes a system call when it makes a request to the operating system's shell.
11. A distributed operating system is an extension of the network operating system that supports higher levels of communication and integration of the machines on the network.
12. Operating System controls the hardware components and coordinates for an efficient use of these hardware components for the various users.
13. The kernel directly interacts with the computer hardware as well as it controls the use of hardware among different types of application programs.
14. open source refers to a program in which the source code is available to the general public for use and/or modification from its original design free of charge
15. Linux is the best-known and most-used open source operating system.
16. OS is the first software to be loaded into computer memory when the computer is switched on and called booting.
17. A system resource manger is a way for programs to interact with the operating system.
18. Bootstrap loader locates the kernel, loads it into main memory and starts its execution.
19. An kernel supplies different kinds of services to both the users and to the programs as well.
20. System Call is used to show all services offered by the OS. It serves as an interface b/w user program and Operating System. The system calls interface layer includes the entrance point in the kernel code.
21. Security refers to a mechanism or a way to control the access of programs, processes, or users to the resources defined by a computer system.
22. An operating system provides users the services to execute the programs in a convenient manner.
23. System call provides the services of the operating system to the user programs via Application Program Interface (API).
24. File management system calls are responsible for file manipulation such as creating a file, reading a file, writing into a file etc.
25. Peer-to-peer network operating systems allow users to share resources and files located on their computers and to access shared resources found on other computers.

## Answers

| | | | | |
|---|---|---|---|---|
| 1. (T) | 2. (T) | 3. (F) | 4. (T) | 5. (F) |
| 6. (F) | 7. (T) | 8. (T) | 9. (T) | 10. (F) |
| 11. (T) | 12. (T) | 13. (F) | 14. (T) | 15. (T) |
| 16. (T) | 17. (F) | 18. (T) | 19. (F) | 20. (T) |
| 21. (F) | 22. (T) | 23. (T) | 24. (T) | 25. (T) |

**Q.IV Answer the following Questions:**

**(A) Short Answer Questions:**

1. What is operating system?
2. Give the role of operating system.
3. What is the purpose of an operating system.
4. List objectives of an operating system.
5. What are the functions of an operating system?
6. What is batch operating system?
7. Define spooling.
8. What is multiprogramming?
9. Define multitasking operating system.
10. What is network operating system?
11. Enlist operations of an operating system.
12. What is timer?
13. Give structure components of an operating system.
14. What is the function of a Kernel?
15. What is shell?
16. Define protection and security.
17. What is mobile computing?
18. What distributed computing?
19. What is client-server computing?
20. What is meant by open source operating system?
21. Define booting.
22. What are the services provided by operating system?
23. What is system call?
24. What is the function of a system call?
25. List types of system calls.

**(B) Long Answer Questions**

1. Define operating system. Describe types of operating systems.
2. What is batch operating system? State its advantages and disadvantages.
3. Explain multiprogramming system in detail with diagram.
4. Define the term distributed operating system. State its advantages and disadvantages.
5. With the help of diagram describe structure of operating system.
6. Describe the operating system operations with diagram.
7. What is computer system architecture? What are its types? Explain them diagrammatically.
8. Differentiate between client-server and peer-to-peer computing envirnments.
9. Write a note on: Distributed operating system.
10. Some CPU provides for more than two modes of operation. What are two possible uses of these multiple modes?

11. Define differences between symmetric and asymmetric multiprocessing.
12. Explain various functions of operating system.
13. Give any four examples of real-time operating system.
14. Modern Operating Systems are interrupt driven. Justify.
15. Enlist different services provided by an operating system.
16. What is a system call? How to implement it?
17. What is booting? Explain with the help of diagram.
18. What is meant by bootstrap loader? Explain its use.

## UNIVERSITY QUESTIONS AND ANSWERS

**April 2016**

**1.** What is function of bootstrap loader? **[1 M]**

**Ans.** Refer to Section 1.8.

**October 2016**

**1.** Define system boot. **[1 M]**

**Ans.** Refer to Section 1.8.

**April 2017**

**1.** What is the main function of microkernel? **[1 M]**

**Ans.** Refer to Page 1.27.

**October 2017**

**1.** What is the main function of microkernel? **[2 M]**

**Ans.** Refer to Page 1.27.

**April 2018**

**1.** Write advantages of multiprocessor system. **[1 M]**

**Ans.** Refer to Pages 1.13 and 1.14.

**October 2018**

**1.** Explain any two operating system components. **[5 M]**

**Ans.** Refer to Section 1.4.

**2.** What is system call? Explain the system call for process and job control. **[3 M]**

**Ans.** Refer to Sections 1.10 and 1.10.2 Points (1).

**April 2019**

**1.** What is bootstrap loader? **[1 M]**

**Ans.** Refer to Section 1.8.

**2.** Write the advantages of microkernel. **[2 M]**

**Ans.** Refer to Page 1.27.

■■■

CHAPTER 2

# Processes and Threads

## *Objectives ...*

- To understand Concept of Process and Process Model
- To learn Process Control Block (PCB)
- To study Importance of Process Scheduling and Different Types of Schedulers
- To learn various Operation on Processes
- To understand the Basic Thread Concept with its Benefits
- To study different Multithreaded Models

## 2.0 INTRODUCTION

- A process is a smallest unit of work that is scheduled by operating system. A process is basically a program in execution.
- A process needs resources, such as CPU time, memory, files and I/O devices, to accomplish its task. These resources are allocated either when the program is created, or when it is executing.
- Operating system enable processes to share and exchange information protect the resources of each process from other processes and enable synchronization among processes.
- To meet these requirements, process management is required which is integral part of an operating system.
- Thread is the segment of a process means a process can have multiple threads and these multiple threads are contained within a process.
- Multithreading allows the execution of multiple parts of a program at the same time. These parts are known as threads and are lightweight processes available within the process.

## 2.1 PROCESS CONCEPT

- A process is a program in execution. The process concept helps to explain, understand and organize execution of programs in an operating system. It is an instance of an application execution.
- The execution of a process must progress in a sequential fashion. An OS uses processes to organize execution of programs.

- A process is defined as, an entity which represents the basic unit of work to be implemented in the system.
- When a program is loaded into the memory and it becomes a process. The process contains program counter which specify the next instruction to be executed.
- Fig. 2.1 shows a simplified layout of a process inside main memory. The process can be divided into four sections namely, stack, heap, text and data.

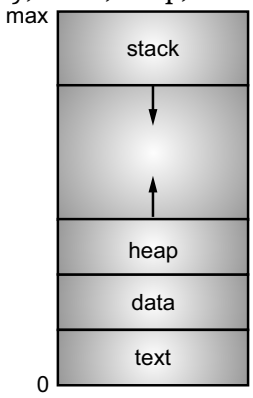



**Fig. 2.1: Process in Memory**

- Each process has following sections:
  1. A **text section** contains the program code.
  2. A **data section** contains global and static variables.
  3. The **heap** section is used for dynamically allocated memory to a process during its run time.
  4. The **stack section** is used for local variables. A process stack which contains the temporary data (such as subroutine parameters, return addresses, and temporary variables).
- Use of the process concept enables an OS to execute both sequential and concurrent programs equally easily.

## 2.1.1 Processes [April 17, 19]

- A process is an instance of an executing program. A process is a program in execution. Process is also called as job, task and unit of work.
- A process is defined as, "an entity which represents the basic unit of work to be implemented in the system". **OR**
- A process is defined as, "a program under execution, which competes for the CPU time and other resources."
- A process is an executing set of machine instructions. It can be either a system process executing the system's code or a user process executing the user's code.

**Process Model:**

- In process model the operating system is organized into a number of sequential processes.

- 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 multi-programming.

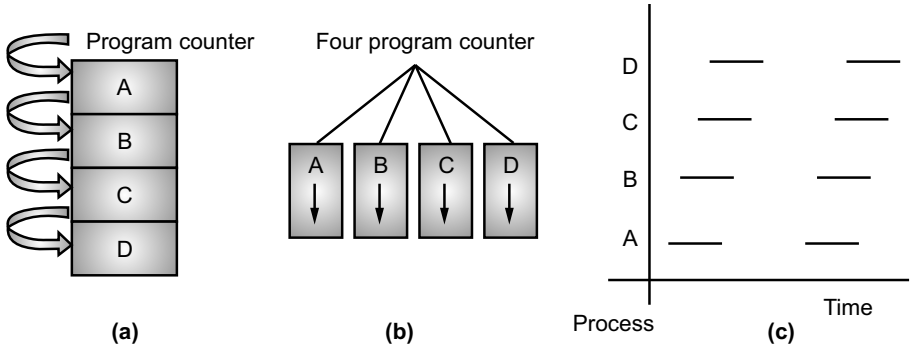



**Fig. 2.2: Process Models**

- In Fig. 2.2 (a), computer multiprogramming four programs in the memory.
- In Fig. 2.2 (b), we can see how this 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 Fig. 2.2 (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.
- 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.

## 2.1.2 Process States [April 17, 19 Oct. 18]

- The current activity of a process is known as its state. As a process executes, it changes state. The process state reflects the current status of the process.
- The process state is an indicator of the nature of the current activity in a process. Fig. 2.3 shows process state diagram.
- Each process in an operating system undergoes change in state during its lifetime. The change in a state of a process is known as state transition of a process.
- A state diagram represents the different states in which a process can be at different times, along with the transitions from one state to another that are possible in the operating system.

- A process may be in one of the following states depending on the current activity of the process:
  1. **New:** A process is said to be new state if it is being created.
  2. **Ready:** A process is said to be ready state if it is ready for the execution and waiting for the CPU to be allocated to it.
  3. **Running State:** A process is said to be in running state if the CPU has been allocated to it and it is currently being executed.
  4. **Waiting or Blocked:** A process is said to be in waiting state if it has been blocked by some event. Unless that event occurs the process cannot continue it execution.
  5. **Terminated:** A process is said to be in waiting state if it has completed its execution normally or it has been terminated abnormally by the OS because of some error or killed by some other processes.

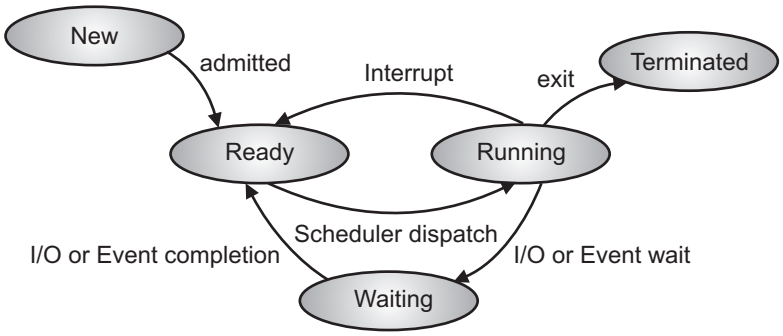



**Fig. 2.3: Process state diagram**

- When the process is selected by the scheduler, it is loaded into memory means the process is in new state or admitted.
- After the process is new state, it is in ready state and the process control block is created by scheduler. The process is waiting for CPU. When the process is allocated to the CPU and executed, then it is in running state.
- During the execution of process, if process requires some input/output operations then it is in wait state if I/O device is not available for the operation. After completion of I/O operation, the process is again in ready state, waiting for the CPU.
- During execution if interrupt occurs then process change running to ready state. After completion of execution the process is in terminated or halted state.

## 2.1.3 Process Control Block (PCB) [April 16]

- In order to execute instructions the operating system has to create processes. Each process is represented in the operating system by a Process Control Block (PCB) also called as Task Control Block (TCB).
- The operating system groups all information that needs about a particular process into a data structure called a PCB or process descriptor.
- When a process is created, operating system creates a corresponding PCB and released whenever the process terminates.

- A PCB stores descriptive information pertaining to a process, such as its state, program counter, memory management information, information about its scheduling, allocated resources, accounting information, etc. that is required to control and manage a particular process.
- The basic purpose of PCB is to indicate the so far progress of a process. Fig. 2.4 shows a typical PCB.
- The PCB is a data structure with fields for recording the various aspects of process execution and the usage of resources.
- In simple words, the operating system maintains the information about each process in a record or a data structure called Process Control Block (PCB).

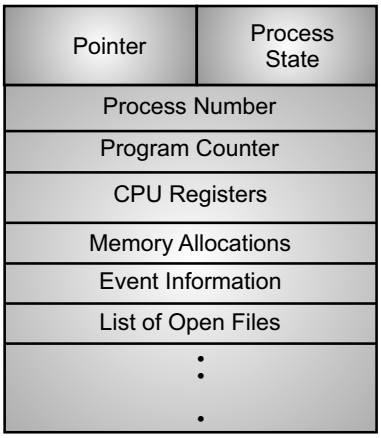



**Fig. 2.4: Process Control Block (PCB)**

- Process Control Block (PCB) is a data structure that stores information about a particular process. A PCB keeps all the information needed to keep track of a process as listed below:
  1. **Pointer:** The pointer has an address of the next PCB, whose process state is ready.
  2. **Process State:** Process state specifies the current state of the process. The current state of the process may be new, ready, running, waiting, terminated and so on.
  3. **Process Number:** Each process is identified by its process number called Process Identification Number (PID). The process-id is provided by the operating system.
  4. **Priority:** The process is assigned the priority at the time of its creation. Process with the highest priority is allocated the CPU first among all the processes.
  5. **Program Counter:** It is a pointer indicates the address of the next instruction to be executed for the process.
  6. **CPU Registers:** Various CPU registers like accumulators, index registers, stack pointers, and general-purpose register etc. where process need to be stored for execution for running state.
  7. **CPU Scheduling Information:** This includes process priority and other scheduling information which is required to schedule the process.

8. **Memory Management Information:** This includes the information of page table, memory limits, Segment table depending on memory used by the operating system.
9. **Accounting Information:** This information includes the amount of CPU and real time used, time limits, account numbers, job or process numbers, and so on.
10. **I/O Status Information:** This information includes the list of I/O devices allocated to the process, a list of open files and so on.
11. **File Management:** It includes information about all open files, access rights etc.

- The PCB simply serves as the repository for any information that may vary from process to process.

## 2.2 PROCESS SCHEDULING [April 18]

- Process scheduling is an essential part of multiprogramming operating systems which allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.
- Scheduling is the method by which work is assigned to resources that complete the work. A scheduler is what carries out the scheduling activity.
- Schedulers are often implemented so they keep all computer resources busy, allow multiple users to share system resources effectively.
- When two or more processes compete for the CPU at the same time then choice has to be made which process to allocate the CPU next. This procedure of determining the next process to be executed on the CPU is called process scheduling and the module of operating system that makes this decision is called the scheduler.
- The process scheduler is the component of the operating system that is responsible for deciding whether the currently running process should continue running and, if not, which process should run next.

### 2.2.1 Scheduling Queue [April 17]

- In multiprogramming, several processes are there in ready or waiting state. These processes form a queue. The operating system maintains all PCBs in process scheduling queues.
- The operating system maintains a separate queue for each of the process states and PCBs of all processes in the same execution state are placed in the same queue.
- When the state of a process is changed, its PCB is unlinked from its current queue and moved to its new state queue.
- When two or more processes compete for the CPU at the same time then choice has to be made which process to allocate the CPU next. This procedure of determining the next process to be executed on the CPU is called process scheduling and the module of operating system that makes this decision is called the scheduler.
- The process scheduler is the component of the operating system that is responsible for deciding whether the currently running process should continue running and, if not, which process should run next.

- Process scheduling is used in determining which processes run when there are multiple runnable processes.
- The operating system maintains the following important process scheduling queues:
  1. **Job queue** keeps all the processes in the system. As the process enter the system, it is put into a job queue.
  2. **Ready queue** keeps a set of all processes residing in main memory, ready and waiting to execute. A new process is always put in this queue. **[April 19]**
  3. A process may have an input/output request, and the device requested may be busy. In such case, the input/output request in maintained is the device queue. The list of processes waiting for a particular I/O device is called a **Device queue**.

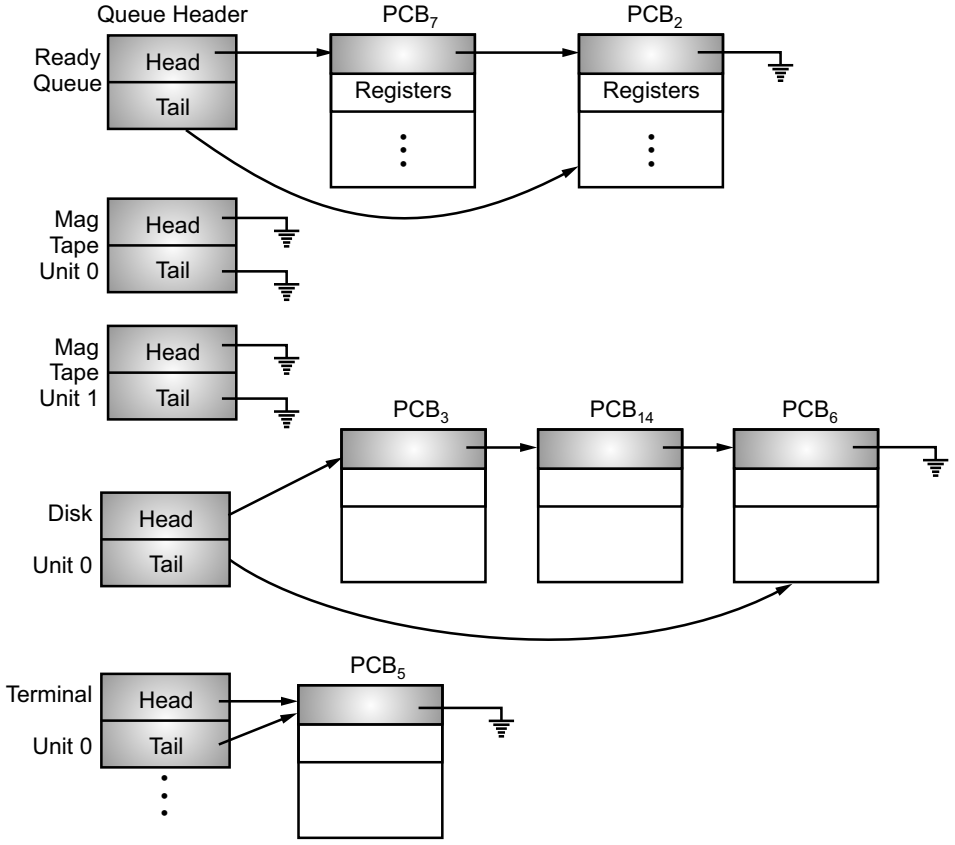



**Fig. 2.5: Ready Queue and various I/O Device Queue**

- Process scheduling is represented by a queuing diagram as shown in Fig. 2.6.
- In Fig. 2.6, queues are represented as rectangles and resources are represented as circles. Directed arrows represent the flow of the processes in the system.
- As new process is initially put in the ready queue. It waits there until it is selected for execution or is dispatched. Once, the process is assigned the CPU and is executing, one of several events could occur:
  1. The process could create a new sub-process and wait for the termination of the sub-process.

2. The process could issue an I/O request and then be placed in an I/O queue.
3. The process could be removed forcibly from the CPU, as a result of an interrupt, and again put in the ready queue.

- In the first two cases, the process transition from the waiting state to ready state occurs. A process continues this cycle until it terminates.
- When the process terminates, it is removed from all queues and its PCB and resources are de-allocated.

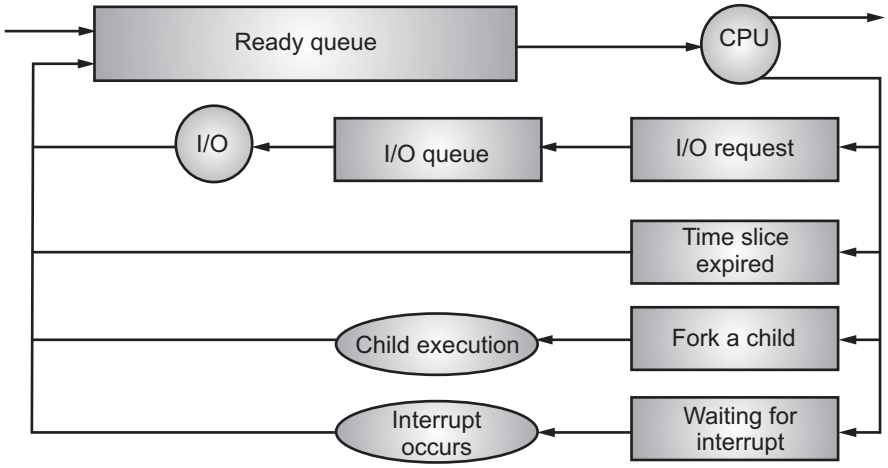



**Fig. 2.6: Queuing Diagram representation of Process Scheduling**

## 2.2.2 Types of Schedulers [Oct. 17, April 19]

- Schedulers are special system software which handles process scheduling in various ways.
- The main task of scheduler is to select the jobs to be submitted into the system and to decide which process to run.
- In other words, the job of process scheduling is done by a software routine (module) called as scheduler.
- There are three different types of scheduler as shown in Fig. 2.7 namely, long-term scheduler, medium-term scheduler and short-term scheduler.
- In Fig. 2.7 initially process is maintained in the Batch queue. Then, job of the long-term scheduler is to fetch the process from batch queue to ready queue (main memory).
- Then, short-term scheduler selects process from the ready queue and assigns them to CPU. When process completes their task, then, process will end or terminate.
- During that period, some processes may ask for I/O device so kept in I/O waiting queue. After that I/O completion, they are moved back to ready queue.
- Now, consider process is executing and due to time slice process is suspended so it will be swap out from CPU and it will be kept in suspended and swapped out queue.
- Once swap out is done, it will be swap in ready queue. Thus scheduling is done by mid-term scheduler and this completes job done in time-sharing system.

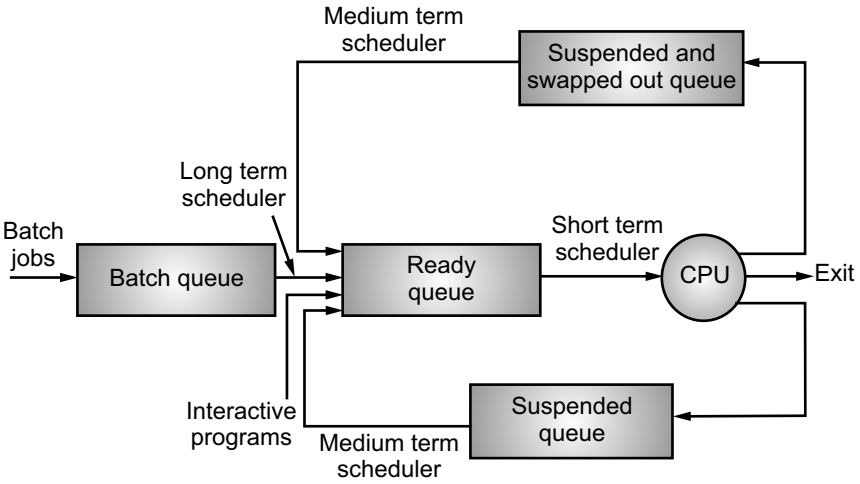



**Fig. 2.7: Types of Schedulers**

- The types of schedulers are explained below:

**1. Long-term Scheduler (Job Scheduler or Admission Scheduler):**

- The long-term scheduler selects the job or process to be executed from job pool on a secondary storage device and loads them into memory for execution. The long-term scheduler executes less frequently.
- The long-term scheduler is invoked when the process leaves the system. Because of the longer duration between executions.
- The long-term scheduler can afford to take more time to decide which process should be selected for execution.
- A long-term scheduler determines which programs are admitted to the system for processing.
- A long-term scheduler selects processes from the queue and loads them into memory for execution. Process loads into the memory for CPU scheduling.
- The primary objective of the job scheduler is to provide a balanced mix of jobs, such as I/O bound and processor bound.
- A long-term scheduler also controls the degree of multiprogramming. If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system.

**2. Short-term Scheduler (CPU Scheduler or Process Scheduler):**

- Short-term scheduler selects a job from ready queue and submits it to CPU. As the short-term scheduler selects only one job at a time, it is invoked very frequently.
- The main objective of short-term scheduler is to increase system performance in accordance with the chosen set of criteria.
- It is the change of ready state to running state of the process. CPU scheduler selects a process among the processes that are ready to execute and allocates CPU to one of them.

- Short-term schedulers, also known as dispatchers, make the decision of which process to execute next. Short-term schedulers are faster than long-term schedulers.
- In case of input/output bound jobs as the ready queue is almost empty, short-term scheduler has very less work to do.
- The systems like time sharing do not have long-term scheduler. The jobs are placed directly in the ready queue for the short-term scheduler. But such some systems have an additional type of scheduler called as medium-term scheduler.

**3. Medium-term Scheduler:** **[Oct. 16]**

- Some OSs, such as time-sharing systems, may introduce an additional, intermediate level of scheduling (medium term scheduler). Medium term scheduling is a part of swapping so it is also known swapper.
- Swapping is term associated with the medium-term scheduler by which processes are temporarily removed and then brought back to the ready queue.
- The medium term scheduler temporarily removes processes from main memory and places them on secondary memory (such as a disk drive) or vice versa. This is commonly referred to as "swapping out" or "swapping in" (Refer Fig. 2.8).
- Medium-term scheduling removes the processes from the memory. It reduces the degree of multiprogramming.

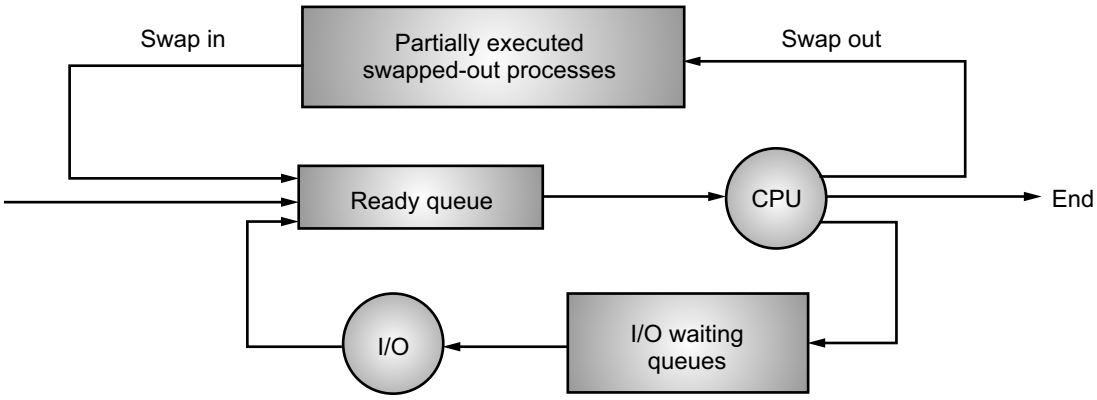



**Fig. 2.8: Addition of Medium Term Scheduling**

**Comparison between Schedulers:**

| Sr. No. | Long Term Scheduler | Short Term Scheduler | Medium Term Scheduler |
|---|---|---|---|
| 1. | It is a job scheduler. | It is a CPU scheduler. | It is a process swapping scheduler or swapper. |
| 2. | It controls the degree of multiprogramming. | It provides lesser control over degree of multiprogramming. | It reduces the degree of multiprogramming. |
| 3. | It is almost absent or minimal in time sharing system. | It is also minimal in time sharing system. | It is a part of Time sharing systems. |

*contd. ...*

| | | | |
|---|---|---|---|
| 4. | Speed is lesser than short term scheduler. | Speed is fastest among other two. | Speed is in between both short and long term scheduler. |
| 5. | It selects processes from pool and loads them into memory for execution. | It selects those processes which are ready to execute. | It can re-introduce the process into memory and execution can be continued. |
| 6. | It deals with main memory for loading process. | It deals with CPU. | It deals with main memory for removing processes and reloading whenever required. |

## 2.2.3 Context Switch [April 17]

- The context switch is an essential feature of a multitasking operating system. 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.
- A context switch is the mechanism to store and restore the state or context of a CPU in Process Control Block (PCB) so that a process execution can be resumed from the same point at a later time.
- Using context switch technique, a context switcher enables multiple processes to share a single CPU.
- The context of a process is represented in the PCB of the process, it includes the value of the CPU registers the process state and memory management information.
- When a context switch occurs, the kernel saves the context of the old process in its PCB and loads the saved context of the new process scheduled to run.
- Fig. 2.9 shows context switching in which, initially Process1 is running. Process1 is switched out and Process2 is switched in because of an interrupt or a system call.
- Context switching involves saving the state of Process1 into PCB1 and loading the state of Process2 from PCB2.
- After some time again a context switch occurs and Process2 is switched out and Process1 is switched in again. This involves saving the state of Process2 into PCB2 and loading the state of Process1 from PCB1.

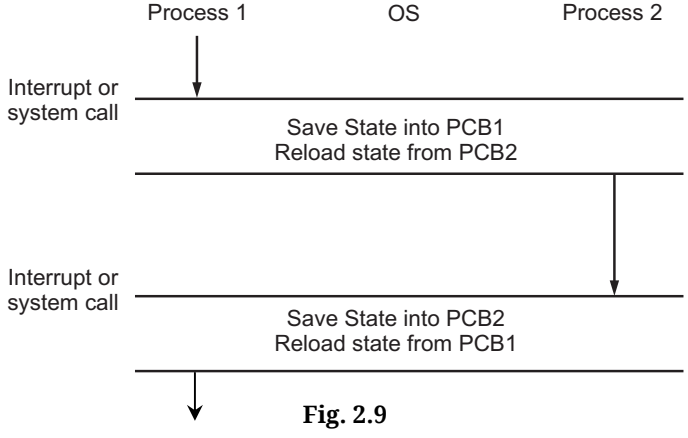



**Fig. 2.9**

- A context switching sometimes referred as a task switch or a process switch.
- Basically task context switching is the switching of the CPU from one process/thread to another one. A process which also sometimes referred as a task is running instance of a program.

**Steps in Context Switching:**

- Fig. 2.10 shows the concept of context switch. There are several steps involves in context switching of the processes.
- Fig. 2.10 represents the context switching of two processes, $P_0$ to $P_1$. As we can see in the diagram, initially, the $P_0$ process is running on the CPU to execute its task, and at the same time, another process, $P_1$, is in the ready state.
- If an error or interruption has occurred or the process requires input/output, the $P_0$ process switches its state from running to the waiting state.
- Before changing the state of the process $P_0$, context switching saves the context of the process $P_0$ in the form of registers and the program counter to the $PCB_0$.
- After that, it loads the state of the $P_1$ process from the ready state of the $PCB_1$ to the running state.
- The following steps are taken when switching Process $P_0$ to Process $P_1$:
  1. First, the context switching needs to save the state of process $P_0$ in the form of the program counter and the registers to the PCB, which is in the running state.
  2. Now update $PCB_0$ to process $P_0$ and moves the process to the appropriate queue, such as the ready queue, I/O queue and waiting queue.
  3. After that, another process gets into the running state, or we can select a new process from the ready state, which is to be executed, or the process has a high priority to execute its task.
  4. Now, we have to update the PCB for the selected process $P_1$. It includes switching the process state from ready to running state or from another state like blocked, exit, or suspend.
  5. If the CPU already executes process $P_1$, we need to get the status of process $P_1$ to resume its execution at the same time point where the system interrupt occurs.
- Similarly, process $P_1$ is switched off from the CPU so that the process $P_0$ can resume execution. $P_0$ process is reloaded from $PCB_0$ to the running state to resume its task at the same point. Otherwise, the information is lost, and when the process is executed again, it starts execution at the initial level.

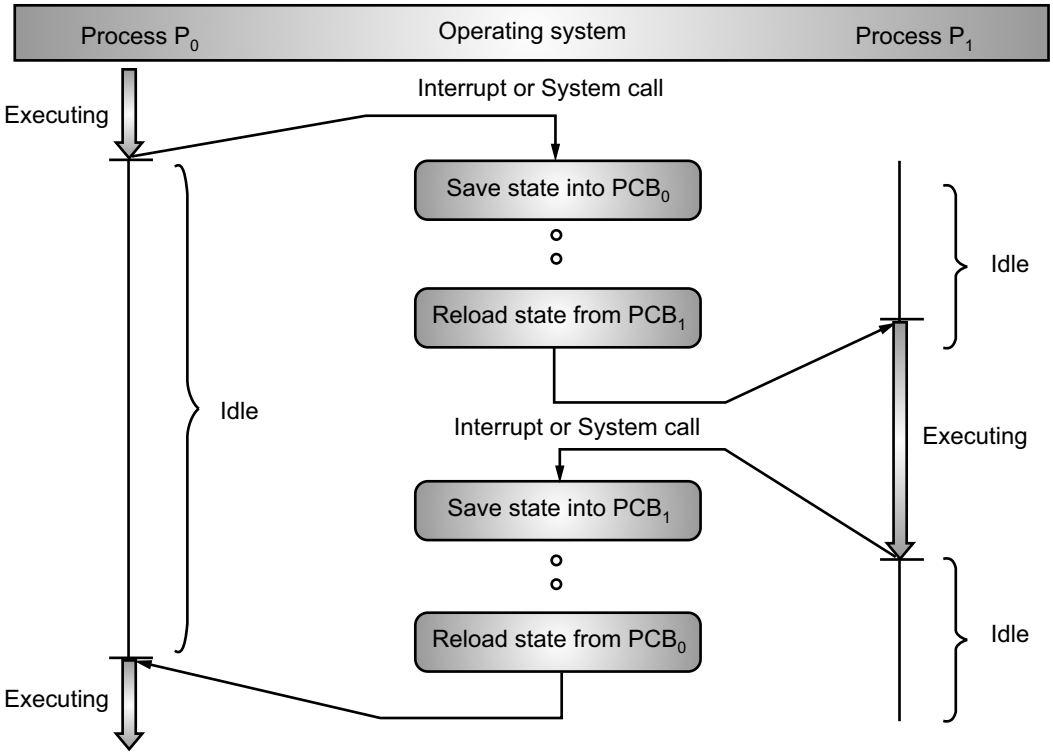



**Fig. 2.10: Context Switch from Process to Process**

## 2.3 OPERATIONS ON PROCESSES

- The processes in the system can execute concurrently and they must be created and deleted dynamically.
- There are two operations provided by operating system on processes:
  1. Process creation, and
  2. Process termination.

### 2.3.1 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.
- 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. Create data structures.

- In UNIX operating system process creation is achieved through the fork() system call. The 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.
- The newly created process is called the child process and the process that initiated it (or the process when execution is started) is called the parent process.
- A child process can have only one parent, but a parent process can have multiple child processes.
- Also, both the parent and child processes have the same memory, open files, and environment strings, but they have distinct address spaces.
- When a process creates a new process, two possibilities exist in terms of execution:
  1. The parent continues to execute concurrently with its children.
  2. The parent waits until some or all of its children have terminated.
- When a process creates a new process, there are two possibilities in terms of address space of the new process.
  1. The child process has the same program and data as parent.
  2. The child process has new program loaded into it.
- Fig. 2.11 shows process creation using fork() system call.

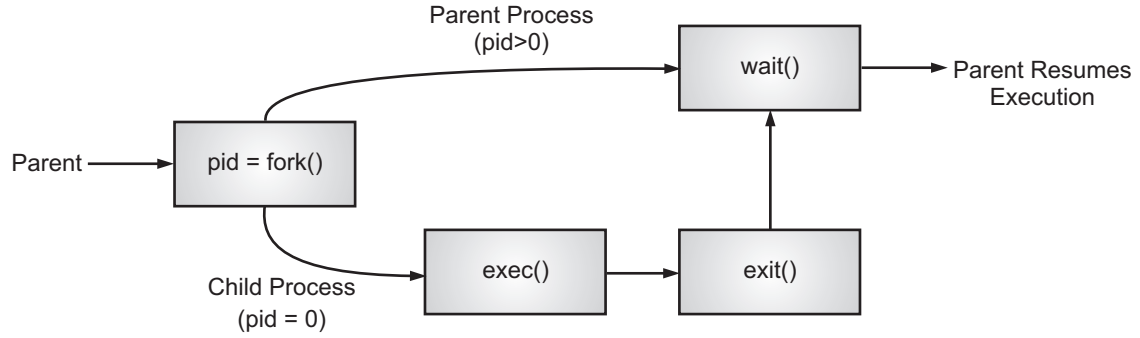



**Fig. 2.11: Creating a Process using the UNIX fork() System Call**

- After creation of the child process, let us see the fork() system call details.

```
#include <sys/types.h>
#include <unistd.h>
pid_t fork(void);
```

- Creates the child process. After above call, there are two processes, the existing one is called the parent process and the newly created one is called the child process.
- The fork() system call returns either of the following three values:
  1. Negative value to indicate an error, i.e., unsuccessful in creating the child process.
  2. Returns a zero for child process.
  3. Returns a positive value for the parent process. This value is the process ID of the newly created child process.

- Let us consider a simple program.

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main() {
   fork();
   printf("Called fork() system call\n");
   return 0;
}
```

**Output:**

```
Called fork() system call
Called fork() system call
```

- Usually after fork() call, the child process and the parent process would perform different tasks. If the same task needs to be run, then for each fork() call it would run 2 power n times, where n is the number of times fork() is invoked.
- In the above case, fork() is called once, hence the output is printed twice (2 power 1). If fork() is called, say 3 times, then the output would be printed 8 times (2 power 3). If it is called 5 times, then it prints 32 times and so on and so forth.
- Having seen fork() create the child process, it is time to see the details of the parent and the child processes.

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
   pid_t pid, mypid, myppid;
   pid = getpid();
   printf("Before fork: Process id is %d\n", pid);
   pid = fork();
   if (pid < 0) {
      perror("fork() failure\n");
      return 1;
   }
   // Child process
```

```
    if (pid == 0) {
        printf("This is child process\n");
        mypid = getpid();
        myppid = getppid();
        printf("Process id is %d and PPID is %d\n", mypid, myppid);
    } else { // Parent process
        sleep(2);
        printf("This is parent process\n");
        mypid = getpid();
        myppid = getppid();
        printf("Process id is %d and PPID is %d\n", mypid, myppid);
        printf("Newly created process id or child pid is %d\n", pid);
    }
    return 0;
}
```

**Output:**

```
Before fork: Process id is 166629
This is child process
Process id is 166630 and PPID is 166629
Before fork: Process id is 166629
This is parent process
Process id is 166629 and PPID is 166628
Newly created process id or child pid is 166630
```

**EXECLP System Call:**

- The EXECLP system call is used after a FORK call by one of the two processes to replace the processes memory space with a new program.
- This call loads a binary file into memory and starts its execution. So two processes can be easily communicates with each other.

```
#include<sys/types.h>
#include<stdio.h>
#include<unistd.h>
int main()
{
    int pid;
    /* fork a child process */
    pid = fork();
```

```
    if (pid< 0)
    { /* error occurred */
       fprintf(stderr, "Fork Failed");
    return 1;
    }
    else if (pid == 0){ /* child process */
       execlp("/bin/wc","wc",NULL);
    }
    else{ /* parent process */
    /* parent will wait for the child to complete */
    wait(NULL);
       printf("Child Complete");
    }
    return 0;
}
```

**NICE System Call:**

- Using NICE system call, we can change the priority of the process in multi-tasking system. The new priority number is added to the already existing value.

**Example:**

```
#include<stdio.h>
main()
{
   int pid, retnice
   print("press DEL to stop process \n");
   pid-fork();
   for(::)
   {
      if(pid == 0)
      {
         retnice = nice (– 5);
         print("child gets higher CPU priority %d \n", retnice);
         sleep(1);
      }
```

```
        else
        {
           retnice=nice(4);
           print("Parent gets lower CPU priority %d \n", retnice);
           sleep(1);
        }
     }
  }
```

**SIGCLD System Call:**

- When the process is terminated it should be removed from the process table and shell should know that the process has died.
- Whenever process is died, it intimates the parent by signal sigcld.

  **Example:**

```
#include<stdio.h>
#include<signal.h>
void abc()
int pid
main()
{
   pid = fork();
   if(pid == 0)
   else
   {
      signal (SIGCLD, abc);  //signal function to handle SIGCLD.signal
      for(i=0; i<1000; i++)
         print(%d, i)
         print("parent existing \n");
   }
}
void abc()
{
   print("child died \n");
   getchar();
}
```

**EXEC System Call:**

- The EXEC system call is used to execute a file which is residing in an active process. When EXEC is called the previous executable file is replaced and new file is executed.

- By using exec system call, we will replace the old file or program from the process with a new file or program.
- The entire content of the process is replaced with a new program. The new program is loaded into the same process space.
- The current process is just turned into a new process and hence the process id PID is not changed, this is because we are not creating a new process we are just replacing a process with another process in exec.
- The exec system call is a collection of functions and in C programming language, the standard names for these functions are as follows:
  1. **execl:** l is for the command line arguments passed a list to the function
  2. **execle:** It is an array of pointers that points to environment variables and is passed explicitly to the newly loaded process.
  3. **execlp:** p is the path environment variable which helps to find the file passed as an argument to be loaded into process.
  4. **execv:** v is for the command line arguments. These are passed as an array of pointers to the function.

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
    printf("PID of example.c = %d\n", getpid());
    char *args[] = {"Hello", "C", "Programming", NULL};
    execv("./hello", args);
    printf("Back to example.c");
    return 0;
}
```

## 2.3.2 Process Termination

- After execution ends, the process is terminated. The operating system terminates the process using exit() system call.
- 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.

- Consider the following example program with exit() system call:

```
#include <stdio.h>
#include <stdlib.h>
void exitfunc() {
   printf("Called cleanup function - exitfunc()\n");
   return;
}
int main() {
   atexit(exitfunc);
   printf("Hello, World!\n");
   exit (0);
}
```

**Output:**

```
Hello, World!
Called cleanup function - exitfunc()
```

- The process may return a status value (typically an integer) to its parent process (via the wait() system call). It also can terminate a child process by parent process issuing kill() system call.
- The kill() system call can occur when the child has exceeded allocated resources, task assigned to the child is no longer required, or the parent is exiting and the operating system does not allow a child to continue if its parent terminates.

## 2.4 THREAD SCHEDULING

- A process is a program that performs a single thread of execution. For example, if 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 a time. Nowadays many modern operating systems have extended the process concept to allow a process to have multiple threads of execution.
- Multithreading allows the execution of multiple parts of a program at the same time. These parts are known as threads and are lightweight processes available within the process.
- A thread is a flow of execution through the process code, with its own program counter, system registers and stack.
- A thread is a path of execution within a process. A process can contain multiple threads.
- Threads are implemented using User Level Threads (user managed threads) and Kernel Level Threads (operating system managed threads acting on kernel).
- On operating systems that support them, it is kernel-level threads - not processes - that are being scheduled by the operating system.
- User-level threads are managed by a thread library, and the kernel is unaware of them.
- To run on a CPU, user-level threads must ultimately be mapped to an associated kernel-level thread, although this mapping may be indirect and may use a lightweight process (LWP).
- In this section, we explore scheduling issues involving user-level and kernel-level threads and offer specific examples of scheduling for Pthreads.

**Contention Scope:**

- The word contention here refers to the competition or fight among the User level threads to access the kernel resources. Thus, this control defines the extent to which contention takes place.
- One distinction between user-level and kernel-level threads lies in how they are scheduled.
- The thread library schedules user-level threads to run on an available LightWeight Process (LWP), a scheme known as Process-Contention Scope (PCS), since competition for the CPU takes place among threads belonging to the same process.
- When we say the thread library schedules user threads onto available LWPs, we do not mean that the thread is actually running on a CPU; this would require the operating system to schedule the kernel thread onto a physical CPU.
- To decide which kernel thread to schedule onto a CPU, the kernel uses system-Contention Scope (SCS).

- Competition for the CPU with SCS scheduling takes place among all threads in the system. Systems using the one-to-one model (such as Windows XP, Solaris 9, and Linux) schedule threads using only SCS.
- Typically, PCS is done according to priority—the scheduler selects the runnable thread with the highest priority to run.
- User-level thread priorities are set by the programmer and are not adjusted by the thread library, although some thread libraries may allow the programmer to change the priority of a thread.

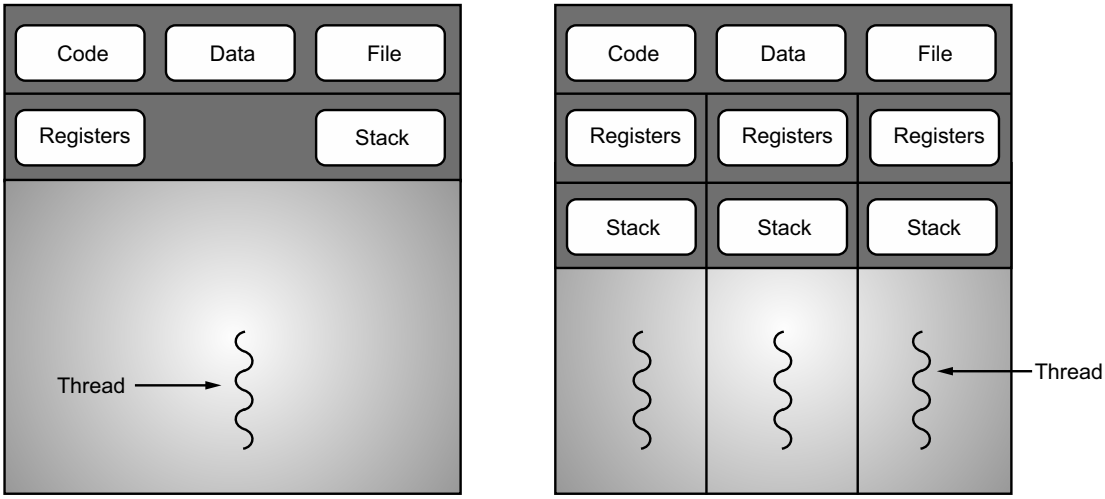
**Pthread Scheduling:**

- Now, we highlight the POSIX Pthread API that allows specifying either PCS or SCS during thread creation.
- POSIX (Portable Operating System Interface) is a set of standard operating system interfaces based on the Unix operating system.
- Pthread identifies the following contention scope values:
  - PTHREAD_SCOPE_PROCESS schedules threads using PCS scheduling.
  - PTHREAD_SCOPE_SYSTEM schedules threads using SCS scheduling.
- On systems implementing the many-to-many model, the PTHREAD_SCOPE_PROCESS policy schedules user-level threads onto available LWPs.
- The number of LWPs is maintained by the thread library, perhaps using scheduler activations.
- The PTHREAD_SCOPE_SYSTEM scheduling policy will create and bind an LWP for each user-level thread on many-to-many systems, effectively mapping threads using the one-to-one policy.

## 2.4.1 Threads [Oct. 18]

- A thread, sometimes called a Light Weight Process (LWP). A thread is a basic unit of CPU utilization.
- A thread is a flow of execution through the process code, with its own program counter that keeps track of which instruction to execute next, system registers which hold its current working variables, and a stack which contains the execution history.
- A thread is a program execution that uses the resources of a process.
- A thread is defined as, "a unit of concurrency within a process and had access to the entire code and data parts of the process". Thus, thread of the same process can share their code and data with one another.
- A traditional or heavyweight process has a single thread of control comprises a single thread of control i.e., it can execute one task at a time and thus, is referred to as a single-threaded process.
- If a process has multiple threads of control, it can perform more than one task at a time, such a process is known as multithreaded process.

- The Fig. 2.12 shows the single threaded and multithreaded processes.




**(a) Single Threaded Process** **(b) Multithreaded Process**

**Fig. 2.12: Threads Process**

**Difference between Process and Thread:**

| Sr. No. | Parameters | Process | Thread |
|---|---|---|---|
| 1. | Weight of process | It is a heavy weight process. | It is a light weight process. |
| 2. | Definition/ Meaning | An executing instance of a program is called a process. | A thread is a subset of the process. |
| 3. | Process | It has its own copy of the data segment of the parent process. | It has direct access to the data segment of its process. |
| 4. | Communication | Processes must use Inter-Process Communication (IPC) to communicate with sibling processes. | Threads can directly communicate with other threads of its process. |
| 5. | Overheads | Processes have considerable overhead. | Threads have almost no overhead. |
| 6. | Creation | New processes require duplication of the parent process. | New threads are easily created. |
| 7. | Control | Processes can only exercise control over child processes. | Threads can exercise considerable control over threads of the same process. |
| 8. | Changes | Any change in the parent process does not affect child processes. | Any change in the main thread may affect the behavior of the other threads of the process. |

***contd. ...***

| | | | |
|---|---|---|---|
| 9. | Memory | Run in separate memory spaces. | Run in shared memory spaces. |
| 10. | File descriptors | Most file descriptors are not shared. | It shares file descriptors. |
| 11. | File system | There is no sharing of file system context. | It shares file system context. |
| 12. | Signal | It does not share signal handling. | It shares signal handling. |
| 13. | Controlled by | Process is controlled by the operating system. | Threads are controlled by programmer in a program. |
| 14. | Dependence | Processes are independent. | Threads are dependent. |

- The threads are implemented using two approaches namely, Kernel level threads and User level threads.
- A library which provides the programmers with an Application Programming Interface (API) for thread creation and management is referred to as a thread library.
- The thread library maps the user threads to the kernel threads. A thread library can be implemented either in the user space or in the kernel space.

**1. User Level Threads:**

- The threads implemented at the user level are known as user threads. In user level thread, thread management is done by the application; the kernel is not aware of existence of threads.
- User threads are supported above the kernel and are implemented by a thread library at the user level. The library provides support for thread creation, scheduling and management with no support from the kernel.
- Because the kernel us unaware of user level threads, all thread creation and scheduling are down in user space without the need for kernel intervention. Therefore, user level threads are generally fast to create and manage.
- The thread library contains code for creating and destroying threads, for passing message and data between threads, for scheduling thread execution and for saving and restoring thread contexts.
- User thread libraries include POSIX PThreads, Mach C-threads, and Solaris 2 UI-threads.

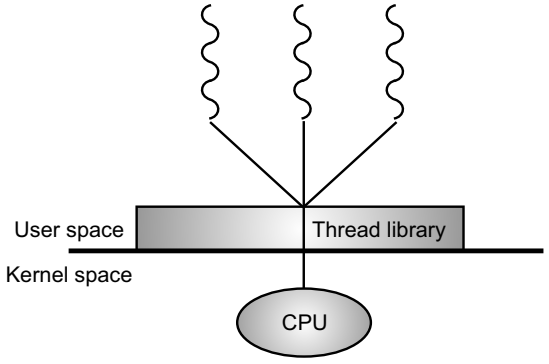



**Fig. 2.13: User Level Thread**

**Advantages of User Level Threads:**

(i) User level thread can run on any operating system.

(ii) A user thread does not require modification to operating systems.

(iii) User level threads are fast to create and manage.

(iv) User thread library easy to portable.

(v) Low cost thread operations are possible.

**Disadvantages of User Level Threads:**

(i) Multithreaded applications cannot take advantage of multiprocessing.

(ii) At most, one user level thread can be in operation at one time, which limits the degree of parallelism.

(iii) If a user thread is blocked in the kernel, the entire processes (all threads of that process) are blocked.

**2. Kernel Level Threads:**

- The threads implemented at kernel level are known as kernel threads. The thread management is done by the Kernel.
- There is no thread management code in the application area. Kernel threads are supported directly by the operating system.
- The kernel performs thread creation, scheduling and management in kernel space. Because thread management is done by the operating system, kernel threads are generally slower to create and manage than are user threads.
- However, since the kernel is managing the threads, if a thread performs a blocking system call, the kernel can schedule another thread in the application for execution.
- Also in a multiprocessor environment, the kernel can schedule threads on different processors.
- Most contemporary operating systems - including Windows NT, Windows 2000, Solaris 2, BeOS and Tru64 UNIX - support kernel threads.

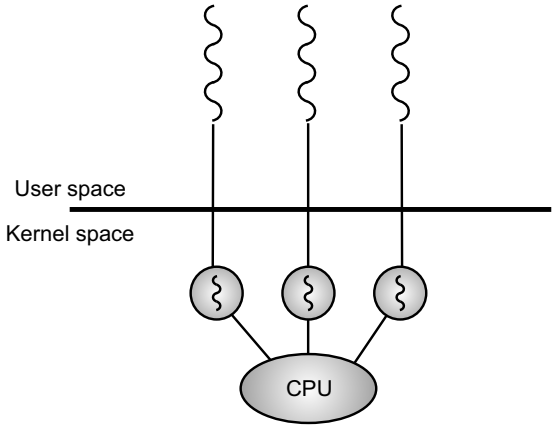



**Fig. 2.14: Kernel Level Threads**

**Advantages of Kernel Level Threads:**

(i) Kernel can simultaneously schedule multiple threads from the same process on multiple processes.

(ii) If one thread in a process is blocked, the Kernel can schedule another thread of the same process.

(iii) Kernel routines themselves can multithreaded.

**Disadvantages of Kernel Level Threads:**

(i) The kernel-level threads are slow and inefficient. For instance, threads operations are hundreds of times slower than that of user-level threads.

(ii) Transfer of control from one thread to another within same process requires a mode switch to the Kernel.

**Difference between User Level Thread and Kernel Level Thread:** **[Oct. 17]**

| Sr. No. | User Level Threads | Kernel Level Threads |
|---|---|---|
| 1. | User level threads are faster to create and manage. | Kernel level threads are slower to create and manage. |
| 2. | Implemented by a thread library at the user level. | Operating system support directly to kernel threads. |
| 3. | User level thread can run on any operating system. | Kernel level threads are specific to the operating system. |
| 4. | Multithread application cannot take advantage of multiprocessing. | Kernel routines themselves can be multithreaded. |

## 2.4.2 Benefits **[April 17, 19, Oct. 17, 18]**

- Threads offers following benefits:
  1. **Resource Sharing:** All the threads of a process share its resources such as memory, data, files etc. A single application can have different threads within the same address space using resource sharing.
  2. **Responsiveness:** Program responsiveness allows a program to run even if part of it is blocked using multithreading. This can also be done if the process is performing a lengthy operation. For example, a web browser with multithreading can use one thread for user contact and another for image loading at the same time.
  3. **Proper Utilization of Multiprocessor Architecture:** In a multiprocessor architecture, each thread can run on a different processor in parallel using multithreading. This increases concurrency of the system. This is in direct contrast to a single processor system, where only one process or thread can run on a processor at a time.

4. **Economical:** Thread creation is more economical than process creation. It is more economical to use threads as they share the process resources. Comparatively, it is more expensive and time-consuming to create processes as they require more memory and resources. The overhead for process creation and management is much higher than thread creation and management.
5. **Efficient Communication:** Communication between multiple threads is easier, as the threads shares common address space.
6. **Computational Speedup:** On a single processor system, a process can be executed speedily by the creating multiple threads in the process.

## 2.4.3 Multithreading Models

- There are three common ways to establish the relationship between user level threads and kernel level threads each resulting in a specific multithreading model.
- The three multithreading models are Many-to-one, One-to-one and Many-to-many explained in this section.

## 2.4.3.1 One-to-One Model [Oct. 17]

- In one-to-one thread model, one-to-one relationship between a user-level thread to a kernel-level thread.
- The one-to-one model maps each user thread to a kernel thread. The Fig. 2.15 shows one-to-one model.
- Example, OS/2, Windows NT and windows 2000 use one-t- one relationship model.

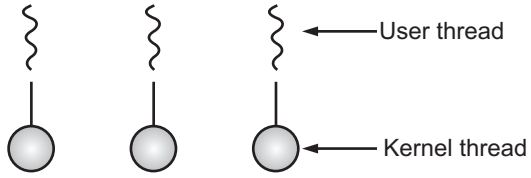



**Fig. 2.15: One-to-One Model**

**Advantages:**

1. This model provides more concurrency than the many to one model.
2. It supports multiple threads to execute in parallel on microprocessors.

**Disadvantages:**

1. Each user thread, the kernel thread is required.
2. Creating a kernel thread is overhead.
3. It reduces the performance of system.

## 2.4.3.2 Many-to-One Model

- The many-to-one model maps many user level threads to one kernel thread.
- Thread management is done in user space, so it is efficient, but the entire process will block if a thread makes a blocking system call.

- Only one thread can access the kernel at a time. Multiple threads are unable to run in parallel on multiprocessors.
- Green threads a thread library available for Solaris 2- uses many-to-one thread model.
- Fig. 2.16 shows many-to-one model concept.

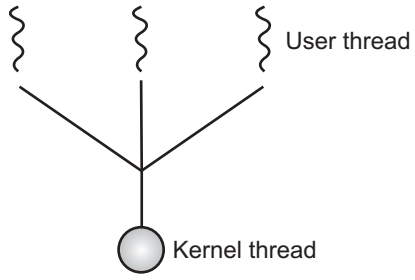



**Fig. 2.16: Many-to-one model**

**Advantages:**

1. One kernel thread controls multiple user threads.
2. It is efficient because thread management is done by thread library.
3. Used in language systems, portable libraries.

**Disadvantages:**

1. The disadvantage is entire process will block if a thread makes blocking system call.
2. Multiple threads are not able to run in parallel since only one thread can be accessed by kernel at a time.

## 2.4.3.3 Many-to-Many Model [April 16]

- In this model, many user level threads multiplex to the Kernel thread of smaller or equal numbers.
- The number of Kernel threads may be specific to either a particular application or a particular machine.
- Solaris 2, IRIX, HP-UX and Tru64 UNIX support many to many thread model.
- The Fig. 2.17 shows many-to-many model which maps many user held threads to a smaller or equal number of kernel threads.

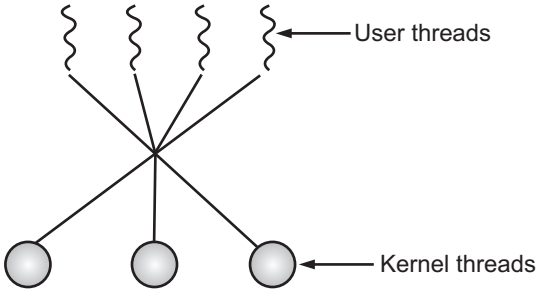



**Fig. 2.17: Many-to-many model**

**Advantages:**

1. Many threads can be created as per user's requirement.
2. Provides the best accuracy on concurrency.
3. When a thread performs a blocking system call, the kernel can schedule another thread for execution.
4. Multiple kernel or equal to user threads can be created.

**Disadvantages:**

1. Multiple threads of kernel are an overhead for OS.
2. Low performance.
3. True concurrency cannot be achieved.

## 2.4.4 Thread Libraries [Oct. 17]

- A thread library provides the programmer an API for creating and managing threads.
- A library which provides the programmers with an Application Programming Interface (API) for thread creation and management is referred to as a thread library.
- A thread library can be implemented either in the user space or in the kernel space. A thread library provides the programmer an API for creating and managing threads.
- There are following two primary ways of implementing a thread library:
  1. To provide a library entirely in user space with no kernel support. All code and data structures for the library exist in user space. This means that invoking a function in the library results in a local function call in user space and not a system call.
  2. To implement a kernel-level library supported directly by the operating system. In this case, code and data structures for the library exist in kernel space. Invoking a function in the API for the library typically results in a system call to the kernel.
- Three main thread libraries are in use today are POSIX Pthreads, Win32 threads, and Java threads.
- Pthreads refers to the POSIX standard (IEEE 1003.1c) defining an API for thread creation. Numerous systems implement the Pthreads specification, including Solaris, Linux, Mac OS X, and Tru64 UNIX.
- The technique for creating threads using the Win32 thread library is similar to the Pthreads technique in several ways.
- We must include the windows.h header file when using the Win32 API. Threads are created in the Win32 API using the CreateThread() function.
- The Java thread API allows thread creation and management directly in Java programs. However, because in most instances the JVM is running on top of a host operating system, the Java thread API is typically implemented using a thread library available on the host system.

# PRACTICE QUESTIONS

**Q.I Multiple Choice Questions:**

1. Which is a program in execution?
   (a) Thread (b) Process
   (c) Stack (d) Heap
2. Which is the segment of a process?
   (a) Thread (b) Process
   (c) Stack (d) Heap
3. The process of saving the context of one process and loading the context of another process is known as,
   (a) CPU scheduling
   (b) Context switching
   (c) Link to interface the hardware and application programs
   (d) All of the above mentioned
4. Which of the following information is stored in process control block to support process context switching?
   (a) The date and time of switching
   (b) UID
   (c) PID
   (d) Terminal from where the process was initiated
5. The address of the next instruction to be executed by the current process is provided by the,
   (a) CPU registers (b) Program counter
   (c) Process stack (d) Pipe
6. A Process Control Block (PCB) does not contain which of the following?
   (a) Code (b) Stack
   (c) Bootstrap program (d) Data
7. In Unix Which system call creates the new process?
   (a) fork (b) create
   (c) new (d) none of the mentioned
8. A process can be terminated due to,
   (a) Normal exit (b) fatal error
   (c) Killed by another process (d) all of the mentioned
9. What is the ready state of a process?
   (a) When process is scheduled to run after some execution
   (b) When process is unable to run until some task has been completed
   (c) When process is using the CPU
   (d) None of the above mentioned
10. A set of processes is deadlock if,
    (a) Each process is blocked and will remain so forever
    (b) Each process is terminated
    (c) All processes are trying to kill each other
    (d) None of the above mentioned

11. A process stack does not contain,
    (a) Function parameters (b) Local variables
    (c) Return addresses (d) PID of child process
12. Which system call can be used by a parent process to determine the termination of child process?
    (a) wait (b) exit
    (c) fork (d) get
13. The number of processes completed per unit time is known as _____.
    (a) Output (b) Throughput
    (c) Efficiency (d) Capacity
14. The state of a process is defined by _____.
    (a) the final activity of the process
    (b) the activity just executed by the process
    (c) the activity to next be executed by the process
    (d) the current activity of the process
15. Which of the following is not the state of a process?
    (a) New (b) Old
    (c) Waiting (d) Running
16. What is a Process Control Block?
    (a) Process type variable (b) Data Structure
    (c) A secondary storage section (d) A Block in memory
17. The entry of all the PCBs of the current processes is in _____.
    (a) Process Register (b) Program Counter
    (c) Process Table (d) Process Unit
18. What is the degree of multiprogramming?
    (a) the number of processes executed per unit time
    (b) the number of processes in the ready queue
    (c) the number of processes in the I/O queue
    (d) the number of processes in memory
19. A single thread of control allows the process to perform _____.
    (a) only one task at a time (b) multiple tasks at a time
    (c) only two tasks at a time (d) all of the mentioned
20. Which of the following do not belong to queues for processes?
    (a) Job Queue (b) PCB queue
    (c) Device Queue (d) Ready Queue
21. When the process issues an I/O request _____.
    (a) It is placed in an I/O queue (b) It is placed in a waiting queue
    (c) It is placed in the ready queue (d) It is placed in the Job queue
22. What will happen when a process terminates?
    (a) It is removed from all queues
    (b) It is removed from all, but the job queue
    (c) Its process control block is de-allocated
    (d) Its process control block is never de-allocated

23. What is a long-term scheduler?
    (a) It selects processes which have to be brought into the ready queue
    (b) It selects processes which have to be executed next and allocates CPU
    (c) It selects processes which heave to remove from memory by swapping
    (d) None of the mentioned
24. If all processes I/O bound, the ready queue will almost always be _____ and the Short term Scheduler will have a _____ to do.
    (a) full, little (b) full, lot
    (c) empty, little (d) empty, lot
25. What is a medium-term scheduler?
    (a) It selects which process has to be brought into the ready queue
    (b) It selects which process has to be executed next and allocates CPU
    (c) It selects which process to remove from memory by swapping
    (d) None of the above mentioned
26. What is a short-term scheduler?
    (a) It selects which process has to be brought into the ready queue
    (b) It selects which process has to be executed next and allocates CPU
    (c) It selects which process to remove from memory by swapping
    (d) None of the above mentioned
27. The primary distinction between the short term scheduler and the long term scheduler is _____.
    (a) The length of their queues (b) The type of processes they schedule
    (c) The frequency of their execution (d) None of the mentioned
28. The only state transition that is initiated by the user process itself is _____.
    (a) block (b) wakeup
    (c) dispatch (d) None of the mentioned
29. In a time-sharing operating system, when the time slot given to a process is completed, the process goes from the running state to the _____.
    (a) Blocked state (b) Ready state
    (c) Suspended state (d) Terminated state
30. In a multiprogramming environment _____.
    (a) the processor executes more than one process at a time
    (b) the programs are developed by more than one person
    (c) more than one process resides in the memory
    (d) a single user can execute many programs at the same time
31. Suppose that a process is in “Blocked” state waiting for some I/O service. When the service is completed, it goes to the _____.
    (a) Running state (b) Ready state
    (c) Suspended state (d) Terminated state
32. The context of a process in the PCB of a process does not contain _____.
    (a) the value of the CPU registers (b) the process state
    (c) memory-management information (d) context switch time

33. Which of the following need not necessarily be saved on a context switch between processes?
    (a) General purpose registers (b) Translation look aside buffer
    (c) Program counter (d) All of the above mentioned
34. Which of the following does not interrupt a running process?
    (a) A device (b) Timer
    (c) Scheduler process (d) Power failure
35. A thread is also called as _____.
    (a) Heavy Weight Process (HWP) (b) Light Weight Process (LWP)
    (c) Both (a) & (b) (d) None of the above mentioned
36. In which model one to one relationship of user level thread to the kernel level thread i.e. each user threads maps to a kernel thread.
    (a) One to Many (b) Many to One
    (c) Many to Many (d) One to One
37. Which of the following operating system not uses the kernel level thread?
    (a) Windows XP (b) MS-DOS
    (c) Linux (d) Solaris
38. Process Control Block (PCB) contains which of the following,
    (a) list of open and close files (b) process state
    (c) process number (d) All of the mentioned
    (e) All of the above mentioned
39. The address of the next instruction to be executed for the current process is stored in _____.
    (a) CPU registers (b) program counter
    (c) process state (d) process number
40. A process's current activity is represented by _____.
    (i) program counter (ii) contents of the processor's registers
    (iii) stack (iv) data section
    (a) 1 (b) 1, 2
    (c) 1, 2, 3 (d) 1, 2, 3, 4
41. A process generally also includes the process _____, which contains temporary data (such as function parameters, return addresses and local variables).
    (a) text section (b) program counter
    (c) stack (d) data section
42. A process generally also includes the process _____, which contains global variables.
    (a) heap (b) program counter
    (c) stack (d) data section

43. Which of the following is TRUE for an I/O bound process?
    (i) is one that spends more of its time doing I/O.
    (ii) is one that spends more of its time doing computations.
    (iii) if all process are I/O bound, the ready queue will almost always be empty.
    (iv) if all process are I/O bound, the ready queue will almost always be full.
    (a) (i) only (b) (i) and (iii) only
    (c) (ii) and (iv) only (d) (ii) and (iii) only
44. Which of the following is TRUE for CPU bound process?
    (i) is one that spends more of its time doing I/O.
    (ii) is one that spends more of its time doing computations.
    (iii) if all process are CPU bound, the I/O waiting queue will almost always be empty.
    (iv) if all process are CPU bound, the I/O waiting queue will almost always be full.
    (a) (i) only (b) (i) and (iii) only
    (c) (ii) and (iii) only (d) (ii) and (iv) only
45. The list of process waiting for a particular I/O device is called a _____.
    (a) device queue (b) ready queue
    (c) job queue (d) None of the mentioned
46. Once the process is allocated the CPU and is executing, which of several events could occur _____.
    (i) The process could issue an I/O request and then be placed in an I/O queue.
    (ii) The process could created a new child process and wait for the child's termination.
    (iii) The process could be removed forcibly from the CPU, as a result of an interrupt and be put back in the ready queue.
    (a) Only (i) (b) (i) and (ii)
    (c) All (i), (ii), (iii) (d) (ii) and (iii)
47. Thread shares with other threads belonging to the same process its _____.
    (a) thread ID (b) program counter
    (c) register set and a stack (d) code section and data section
48. A process can be,
    (a) single threaded
    (b) multithreaded
    (c) both single threaded and multithreaded
    (d) None of the above mentioned
49. Which is a path of the execution within a process?
    (a) process (b) stack
    (c) thread (d) None of the above mentioned
50. User threads _____.
    (a) are supported above the kernel and are managed without kernel support.
    (b) are supported below the kernel and are managed without kernel support.
    (c) are supported above the kernel and are managed with kernel support.
    (d) are supported below the kernel and are managed with kernel support.

51. Kernel threads _____.
    (a) Cannot be supported and managed directly by the operating system.
    (b) can be supported and managed directly by the operating system.
    (c) are supported below the kernel and are managed without kernel support.
    (d) None of the above mentioned
52. Which of the following multithreading model maps many user-level threads to one kernel thread?
    (a) Many to One model (b) One to Many model
    (c) Many to Many model (d) One to One model
53. Which of the following multithreading model in which the entire process will block if a thread makes a block system call?
    (a) Many to One model (b) One to Many model
    (c) Many to Many model (d) One to One model
54. Which of the following multithreading model, only one thread can access the kernel at a time, multiple threads are unable to run in parallel on multicore systems?
    (a) Many to One model (b) One to Many model
    (c) Many to Many model (d) One to One model
55. Which of the following multithreading model maps each user thread to a kernel thread?
    (a) Many to One model (b) One to Many model
    (c) Many to Many model (d) One to One model
56. Which of the following multithreading model provides more concurrency that the many-to-one model by allowing another thread to run when a thread makes a block system call?
    (a) Many to One model (b) One to Many model
    (c) Many to Many model (d) One to One model
57. Which of the following multithreading model also allows multiple threads to run in parallel on multithreaded processors?
    (a) Many to One model (b) One to Many model
    (c) Many to Many model (d) One to One model
58. Which of the following multithreading model has drawback "that creating a user thread requires creating the corresponding kernel thread".
    (a) Many to One model (b) One to Many model
    (c) Many to Many model (d) One to One model
59. Which of the following multithreading model multiplexes many user-level threads to a smaller or equal number of kernel threads?
    (a) Many to One model (b) One to Many model
    (c) Many to Many model (d) One to One model

60. Which of the following components of program state are shared across threads in a multithreading process?
    (a) Register values (b) Heap memory
    (c) Global variables (d) Stack memory
    (e) Both (b) and (c)

## Answers

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1. (b) | 2. (a) | 3. (b) | 4. (c) | 5. (b) | 6. (c) | 7. (a) | 8. (d) | 9. (a) | 10. (a) |
| 11. (d) | 12. (a) | 13. (b) | 14. (d) | 15. (b) | 16. (b) | 17. (b) | 18. (d) | 19. (a) | 20. (b) |
| 21. (a) | 22. (a) | 23. (a) | 24. (c) | 25. (c) | 26. (b) | 27. (c) | 28. (a) | 29. (b) | 30. (c) |
| 31. (b) | 32. (d) | 33. (b) | 34. (c) | 35. (b) | 36. (d) | 37. (b) | 38. (e) | 39. (b) | 40. (b) |
| 41. (c) | 42. (d) | 43. (d) | 44. (c) | 45. (a) | 46. (c) | 47. (d) | 48. (c) | 49. (c) | 50. (a) |
| 51. (a) | 52. (a) | 53. (a) | 54. (a) | 55. (d) | 56. (d) | 57. (d) | 58. (d) | 59. (c) | 60. (e) |

**Q.II Fill in the Blanks**

1. A _____ is a program in execution.
2. The execution of a process must progress in a _____ fashion.
3. The current activity of a process is known as its _____.
4. The operating system maintains all PCBs in process _____ queues.
5. A PCB is a _____ in an operating system kernel that contains the information needed to keep track of a process.
6. A _____ is also referred as job of batch system or task of time-sharing system.
7. The PCB is identified by an integer _____.
8. A process is a program that performs a _____ thread of execution.
9. _____ queue helps to store all the processes in the system.
10. The _____ scheduler, or admission scheduler, decides which jobs or processes are to be admitted to the ready queue (in main memory).
11. In general, most processes can be described as either I/O bound or _____ .
12. An _____ process is one that spends more of its time doing I/O than it spends doing computations.
13. A _____ process, in contrast, generates I/O requests infrequently, using more of its time doing computations.
14. The threads implemented at kernel level are known as _____ threads.
15. The _____ scheduler temporarily removes processes from main memory and places them in secondary memory (such as a hard disk drive)
16. The _____ scheduler (also known as the CPU scheduler) decides which of the ready, in-memory processes is to be executed (allocated a CPU) after a clock interrupt, an I/O interrupt, an operating system call or another form of signal.
17. A _____ is an execution unit that has its own program counter, a stack and a set of registers that reside in a process.
18. A _____ switch is the process of storing the state of a process.
19. When more than one process is runnable, the operating system must decide which one first. The part of the operating system concerned with this decision is called the _____.

20. The _____ scheduler is the component of the operating system that decide which process should run next, which process should continue in running state.
21. _____, the threads extension of the POSIX standard, may be provided as either a user- or kernel-level library.
22. 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 _____ switch.
23. The context of a process is represented in the Process Control Block (PCB) of a _____.
24. A process may be created by another process using the _____ method.
25. The _____ system call is used to terminate a process.
26. Scheduling of threads involves two boundary scheduling using _____ level threads to _____ level threads.
27. The threads implemented at the user level are known as _____ threads.
28. A thread _____ provides the programmer an API for creating and managing threads.
29. The model in which one kernel thread is mapped to many user-level threads is called as _____ model.
30. The model in which one user-level thread is mapped to many kernel level threads is called as _____ model.

## Answers

| | | | |
|---|---|---|---|
| 1. process | 2. sequential | 3. state | 4. scheduling |
| 5. data structure | 6. process | 7. process ID (PID) | 8. single |
| 9. Job | 10. long term | 11. CPU bound | 12. I/O-bound |
| 13. CPU-bound | 14. kernel | 15. medium term | 16. short term |
| 17. thread | 18. context | 19. scheduler | 20. process |
| 21. Pthreads | 22. context | 23. process | 24. fork() |
| 25. exit() | 26. user, kernel | 27. user | 28. library |
| 29. Many to one | 30. One to many | | |

**Q.III State True or False:**

1. A process is an instance of a program execution.
2. The thread contains text section, data section, heap and stack.
3. Ready queue is a type of queue helps you to set every process residing in the main memory, which is ready and waiting to execute.
4. The process scheduler is a part of the operating system that decides which process runs at a certain point in time.
5. The medium term scheduler also called as CPU Scheduler.
6. An efficient scheduling system will select a good process mix of CPU-bound processes and I/O bound processes.
7. Processes waiting for a device to become available or to deliver data are placed in device queues.

8. Process scheduling is determining which process moves from ready state to the running state.
9. A thread library provides the programmer an API for creating and managing threads.
10. The Win32 thread library is a kernel-level library available on Windows systems.
11. A long term scheduler is typical of a batch system.
12. A context switch enables multiple processes to share a single CPU.
13. A process is a program in execution.
14. Each process is represented in the operating system by a process context switch.
15. The processes that are residing in main memory and are ready and waiting to execute are kept on a list called the ready queue.
16. A process may be created by another process using the exit() method.
17. Switching the CPU to another process requires performing a state save of the current process and a state restore of a different process. This task is know as a context switch.
18. Switching the CPU from one process to another process requires saving the state of old process and loading the saved state of new process. This task is known as Context Switch.
19. The exit() system call is used to terminate a process.
20. Context of a process represented in the scheduler.
21. Thread library schedules which thread of a process to run on which Light-Weight Process (LWP) and how long.
22. The long term scheduler selects which process to remove from memory by swapping.

## Answers

| | | | | |
|---|---|---|---|---|
| 1. (T) | 2. (F) | 3. (T) | 4. (T) | 5. (T) |
| 6. (F) | 7. (T) | 8. (T) | 9. (T) | 10. (T) |
| 11. (T) | 12. (T) | 13. (T) | 14. (F) | 15. (T) |
| 16. (F) | 17. (T) | 18. (T) | 19. (T) | 20. (F) |
| 21. (T) | 22. (F) | | | |

**Q.IV Answer the following Questions:**

**(A) Short Answer Questions:**

1. What is process?
2. What is thread?
3. What is PCB?
4. What is context switch?
5. What is scheduler
6. What is meant by scheduling queue?
7. List types of scheduling queue.
8. Define scheduler.
9. List operations on processes.

10. State the role of medium term process scheduler.
11. Give purpose of CPU scheduler.
12. What is thread library?
13. What is meant by thread scheduling?
14. What is process scheduling?

**(B) Long Answer Questions**

1. Define process and thread. Compare them with any four points.
2. With the help of diagram describe process model.
3. Describe PCB with its all fields.
4. What is process scheduler? List types of process schedulers.
5. Which activities carried by process scheduling?
6. Describe context switch diagrammatically.
7. Compare long term, short term and medium term schedulers.
8. Explain operations on processes with example.
9. Explain user and kernel level threads diagrammatically.
10. Explain scheduling queue in detail.
11. "Any executable file on a disk is called as a process" True/False – Justify)
12. Define short term scheduler with queuing diagram.
13. What are user and kernel level threads? Differentiate them.
14. Explain many-to-one model of multithreading
15. Compare process and thread.
16. What are the benefits of multithreading programming?
17. Explain the following multithreading models.
    (i) One-to-one model
    (ii) Many-to-many model.

## UNIVERSITY QUESTIONS AND ANSWERS

**April 2016**

**1.** List any two examples of many to many model. **[1 M]**

**Ans.** Refer to Section 2.4.3.3.

**2.** Explain PCB with proper diagram. **[5 M]**

**Ans.** Refer to Section 2.1.3.

**October 2016**

**1.** Write a primary function of medium term scheduler. **[1 M]**

**Ans.** Refer to Section 2.2.2 Point (3).

**2.** State scalability and Responsiveness benefits of multithreading. **[1 M]**

**Ans.** Refer to Section 2.4.2.

**April 2017**

**1.** State two benefits of multi-threaded programming. **[1 M]**

**Ans.** Refer to Section 2.4.2.

**2.** What is process? State and explain in brief different types of process states. **[5 M]**

**Ans.** Refer to Sections 2.1.1 and 2.1.2.

**October 2017**

**1.** List any two operating system examples that uses one to one model. **[1 M]**

**Ans.** Refer to Section 2.4.3.1.

**2.** Give examples of operating system with user Pthread. **[1 M]**

**Ans.** Refer to Section 2.4.4.

**3.** What is context switching? State conditions when context switching occurs. **[5 M]**

**Ans.** Refer to Section 2.2.3.

**4.** State two benefits of multithreaded programming. **[2 M]**

**Ans.** Refer to Section 2.4.2.

**April 2018**

**1.** Differentiate between user level thread and kernel level thread. **[2 M]**

**Ans.** Refer to Page 2.26.

**October 2018**

**1.** List any two advantages of multithreaded programming. **[1 M]**

**Ans.** Refer to Section 2.4.2.

**2.** Write a short note on process states. **[5 M]**

**Ans.** Refer to Section 2.1.2.

**3.** What do you understand by a thread? **[1 M]**

**Ans.** Refer to Section 2.4.1.

**April 2019**

**1.** What is ready queue? **[1 M]**

**Ans.** Refer to Section 2.2.1 Point (2).

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

**Ans.** Refer to Section 2.2.2.

**3.** What is process? Explain the different types of process states. **[5 M]**

**Ans.** Refer to Sections 2.1.1 and 2.1.2.

**4.** Explain many-to-many multithreading model. **[2 M]**

**Ans.** Refer to Section 2.4.2.

■■■

CHAPTER 3

# Process Scheduling

***Objectives ...***

- To understand Concept of CPU Scheduling
- To learn Concept of Scheduler and Criteria for Scheduling Algorithm
- To study various CPU Scheduling Algorithms like FCFS, SJF, RR, etc.
- To learn Multiple Queue Scheduling and Multilevel Feedback Queue Scheduling

## 3.0 INTRODUCTION

- The assignment of physical processors to processes allows processors to accomplish work. In multi-programming environment, many processes are in memory.
- The problem of determining when processors should be assigned and to which processes is called processor scheduling or CPU scheduling.
- When more than one process is run able, the operating system must decide which one first. The part of the operating system concerned with this decision is called the scheduler, and algorithm it uses is called the scheduling algorithm.
- Process scheduling is an essential part of multiprogramming operating systems which allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.

## 3.1 BASIC SCHEDULING CONCEPT

- Scheduling is an important function of an operating system. CPU scheduling is to determine when and on what processor each process has to run.
- In other words, CPU scheduling is the process of selecting a process and allocating the processor to the selected process for execution.
- CPU scheduling is important because the overall system utilization and performance as well as the response time of processes depend on how the processes are scheduled to run.
- The main objectives of scheduling are:
  1. Enforcement of fairness in allocating resources to processes.
  2. Enforcement of priorities to processes.
  3. Make best use of available system resources.
  4. Give preference to processes holding key resources.

- The scheduler is the kernel component (module/program) responsible for deciding which program should be executed on the CPU. A scheduler selects a job/task which is to be submitted next for execution.
- When it receives control, the scheduler the program and gives the CPU to another program. The function of deciding which program should be given the CPU, and for how long, is called scheduling.
- Fig. 3.1 shows a schematic for CPU scheduling. Several programs await allocation of the CPU.
- The scheduler selects one of these programs for execution on the CPU. A preempted program is added to the set of programs waiting for the CPU.

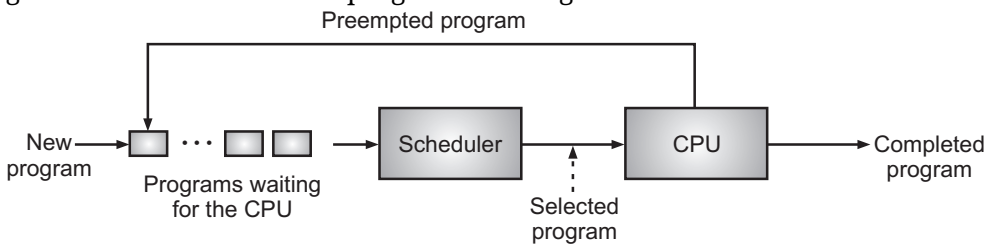



**Fig. 3.1: A Schematic of Scheduling**

**Scheduling Model:**

- Fig. 3.2 shows a simple model of CPU scheduling.
- CPU scheduling is the process of selecting a process from the ready queue and assigning the CPU to it. CPU scheduling algorithms decides which of the processes in the ready queue is to be allocated to the CPU.
- Every process in OS that requests CPU service carries out the following sequence of actions:
  1. In first action, join the ready queue and wait for CPU service.
  2. In second action, execute (receive CPU service) for the duration of the current CPU burst or for the duration of the time slice (timeout).
  3. In third action, join the I/O queue to wait for I/O service or return to the ready queue to wait for more CPU service.
  4. In fourth action, terminate and exit if service is completed i.e., if there are no more CPU or I/O bursts. If more service is required, return to the ready queue to wait for more CPU service.
- The CPU scheduler is the part of the operating system that selects the next process to which the CPU will be allocated, de-allocates the CPU from the process currently executing, and allocates the CPU to the newly selected process.

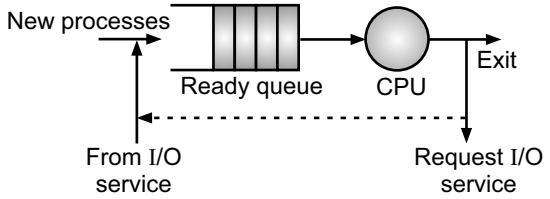



**Fig. 3.2: Model of CPU Scheduling**

- The algorithm used by the scheduler to carry out the selection of a process for execution is known as scheduling algorithm.
- A number of scheduling algorithms are available for CPU scheduling such as FCFS, RR etc. Each and every scheduling algorithm influences the resource utilization, overall system performance, and quality of service provided to the user.

## 3.1.1 CPU and I/O Burst Cycle [April 16, 18, Oct. 17]

- CPU scheduling is greatly affected by how a process behaves during its execution. Almost all the processes continue to switch between CPU (for processing) and I/O devices (for performing I/O) during their execution.
- The success of CPU scheduling depends upon the observed property of processes such as process execution is a cycle of CPU execution and I/O waits.
- Processes alternate back and forth between these two states. Process execution begins with a CPU burst. It is followed by an I/O burst, which is followed by another CPU burst, then another I/O burst and so on.
- Eventually the last CPU burst will end with a system request to terminate execution, rather than another I/O burst.

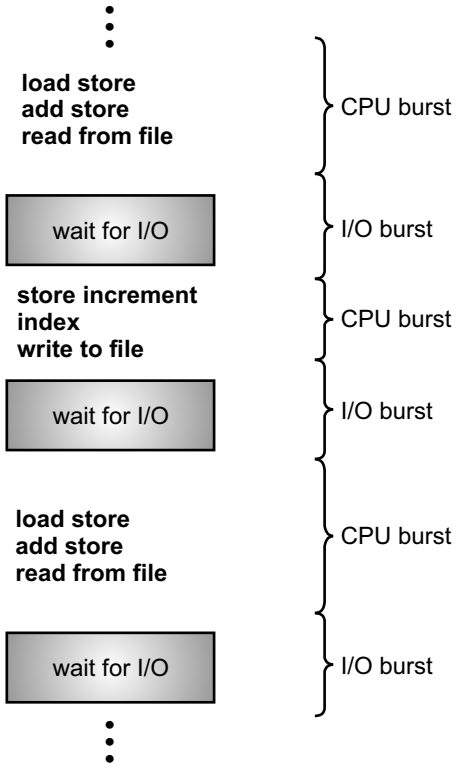



**Fig. 3.3: Execution is an Alternating Sequence of CPU and I/O Bursts**

- In short, we can say that the process execution comprises alternate cycles of CPU burst (the time period elapsed in processing before performing the next I/O operation) and I/O burst (the time period elapsed in performing I/O before the next CPU burst).

- To realize a system's scheduling objectives, the scheduler should consider process behavior.
  1. **CPU Bound Process:** The process which spends more time in computations or with CPU and very rarely with the I/O devices is called as CPU bound process.
  2. **I/O Bound Process:** The process which spends more time in I/O operation than computation (time spends with CPU) is I/O bound process.
- Fig. 3.3 shows the sequence of CPU and I/O bursts.

## 3.1.2 Scheduling Criteria [April 16, Oct. 17]

- For scheduling purposes, the scheduler may consider some performance measures and optimization criteria. Different CPU scheduling algorithms have different properties and may favor one class of processes over another.
- In choosing which algorithm to use in a particular situation/condition, the properties of various algorithms must be considered. Scheduling algorithm is one which selects some job from ready queue based on some criteria and submits it to the CPU.
- Many criteria have been suggested for comparing CPU scheduling algorithms criteria, which are used include CPU utilization, balanced utilization, throughput, waiting time, turnaround time and response time.
  1. **CPU Utilization:** CPU utilization is defined as, the percentage of time the CPU is busy in executing processes. For higher utilization, CPU must be kept as busy as possible, that is, there must be some process running at all times. CPU Utilization may range from 0 to 100 percent. In a real system it should range from 40 percent to 90 percent.
  2. **Throughput:** If the CPU is busy executing processes, then work is being done. One measure of work is the number of processes that are 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. Throughput is defined as, the total number of processes that a system can execute per unit of time.
  3. **Turnaround Time (TAT):** It is the difference between the time a process enters the system and the time it exits the system. From the point of view of a particular 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. Turnaround Time is the sum; of the periods spent waiting to get into memory writing in the ready queue, executing on the CPU and doing I/O. Turnaround time is defined as, the amount of time that has rolled by from the time of creation to the termination of a process. **[April 16]**
  4. **Waiting Time:** The CPU scheduling algorithm does not affect the amount of time during which a process executes of does I/O, if affects only the amount of time that a process spends waiting in the ready queue. Waiting time is the sum of the periods spent waiting in the ready queue. Waiting time is defined as, the time spent by a process while waiting in the ready queue.

5. **Balanced Utilization:** Balanced utilization is defined as, the percentage of time all the system resources are busy. It considers not only the CPU utilization but the utilization of I/O devices, memory, and all other resources. To get more work done by the system, the CPU and I/O devices must he kept running simultaneously. For this, it is desirable to load a mixture of CPU-bound and I/O-bound processes in the memory.
6. **Response Time:** Response time is defined as, the time elapsed between the moment when a user initiates a request and the instant when the system starts responding to this request. In an interactive system TAT (Turnaround Time) may not be best criterion. Often a process can produce some output early and can continue computing new results while previous results are being output to the user. Thus another measure is the time from submission of a request until the first response is produced. This measure called response time is the amount of time it takes to start responding, but not the time that it takes to output that response. The Turnaround Time is generally limited by the speed of the output device.

## 3.1.3 CPU Scheduler

- A scheduler is an operating system module (program) that selects a job which is to be admitted next for execution.
- CPU scheduling is the process of selecting a process and allocating the processor to the selected process for execution.
- CPU scheduler or short-term scheduler selects the process from ready queue allocates to CPU for execution. This selection of process is not in FIFO order but it depends on CPU scheduling algorithms.
- CPU scheduling in operating system is a fundamental function, almost all computer resources scheduled before use.
- The CPU is also one of the primary resources. So, CPU scheduling before use. The Scheduling Algorithms determine how the CPU will be allocated to the process.

## 3.1.4 Types of Scheduling [Oct. 16]

- CPU scheduling is the process of selecting a process and allocating the processor to the selected process for execution.
- Scheduling types are depending upon how the process is executed (i.e., whether the CPU is provided to a process completely or provided in cycles with breaks in between i.e. the process is in the waiting or suspended state after certain period of time).
- CPU scheduling decisions may take place under the following four conditions:
  1. When a process switches from the running state to the waiting state (for example, I/O request, or invocation of wait for termination of one of the child processes).
  2. When a process switches from the running state to the ready state (for example, when an interrupt occurs).
  3. When a process switches from the waiting state to the ready state (for example, completion of I/O).
  4. When a process terminates.
- Depending on above situations, there are two types of scheduling namely Pre-emptive Scheduling and Non-Preemptive Scheduling.

## 3.1.4.1 Pre-emptive Scheduling [Oct. 16]

- A pre-emptive scheduling allows a higher priority process to replace a currently running process, even if its time slot is not completed or it has not requested for any I/O.
- If a higher priority process is enters the system, the currently running process is stopped and the CPU transfers the control to the higher priority process.
- The currently running process may be interrupted and moved to the ready state by the operating system.
- Window-95 introduces pre-emptive scheduling and all subsequent versions of Windows operating systems have used pre-emptive scheduling. The Mac OS X operating system for the Macintosh also uses preemptive scheduling.
- The pre-emptive scheduling algorithms are based on priority where a scheduler may preempt a low priority running process anytime when a high priority process enters into a ready state.
- The advantage of pre-emptive scheduling algorithms is they allow real multiprogramming.
- The disadvantages of pre-emptive scheduling algorithms are they are complex and they lead the system to race condition.

## 3.1.4.2 Non-Preemptive Scheduling [Oct. 16]

- In non-preemptive scheduling once the CPU is assigned to a process, the processor do not release until the completion of that process.
- It means that a running process has the control of the CPU and other allocated resources until the normal termination of that process.
- In non-pre-emptive 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 waiting state. This scheduling method was used by Microsoft Windows 3.
- In Microsoft Windows 3 case, 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 i.e., if a higher priority process enters the system, the running process cannot be forced to release the control.
- Non-preemptive algorithms are designed so that once a process enters the running state, it cannot be preempted until it completes its allotted time.
- The advantages of non-preemptive scheduling algorithms are they are simple and they cannot lead the system to race condition.
- The disadvantage of non-preemptive scheduling algorithms is they do not allow real multi-programming.

**Difference between Pre-emptive and Non-preemptive Scheduling:**

| Sr. No. | Preemptive Scheduling | Non-preemptive Scheduling |
|---|---|---|
| 1. | In the preemptive schedule, the processes with higher priorities are executed first. | 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 waiting state. |
| 2. | The CPU utilization is high in this type of scheduling. | The CPU utilization is less efficient. |
| 3. | Waiting and response time of preemptive scheduling is less. | Waiting and response time of the non-preemptive Scheduling method is higher. |
| 4. | In this process, the CPU is allocated to the processes for a specific time period. | In this process, CPU is allocated to the process until it terminates or switches to the waiting state. |
| 5. | Only the processes having higher priority are scheduled. | Processes having any priority can get scheduled. |
| 6. | It does not treat all processes as equal. | It treats all process as equal. |
| 7. | It has the overhead of switching the process from the ready state to the running state. | It has no such overhead of switching the process from running into the ready state. |
| 8. | Preemptive scheduling is flexible. | Non-preemptive scheduling is rigid. |
| 9. | Examples of pre-emptive scheduling are Round Robin (RR) and Shortest Remaining Time First (SRTF). | Examples of non-preemptive scheduling are First Come First Serve (FCFS) and Shortest Job First (SJF). |
| 10. | Preemptive scheduling can be pre-empted that is the process can be scheduled. | In non-preemptive scheduling, process cannot be Scheduled. |
| 11. | The system resources are used efficiently. | The system resources are not used efficiently. |

## 3.1.5 Dispatcher

**[April 16, 18, 19 Oct. 16]**

- Dispatcher is a component which involves in the CPU scheduling. When the scheduler completes its job of selecting a process, it is the dispatcher which takes that process to the desired state/queue.
- The dispatcher is the module that actually gives control of the CPU to the process selected by the short term scheduler.
- The module of the operating system that performs the function of setting up the execution of the selected process on the CPU is known as dispatcher.
- The CPU scheduler only selects a process to be executed next on the CPU but it cannot assign CPU to the selected process.

- The function of setting up the execution of the selected process on the CPU is performed by another module of the operating system, known as dispatcher.
- The following are functions performed by dispatcher:
  1. Loading the register of the process.
  2. Switching operating system to the user mode.
  3. Restart the program by jumping to the proper location in the user program.

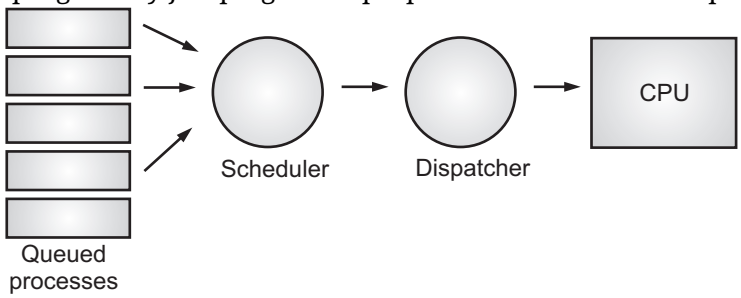



**Fig. 3.4: Concept of Dispatcher**

- The time taken by dispatcher to stop one process and start another process to run is called dispatch latency time. **[April 16, 18, 19 Oct. 16]**

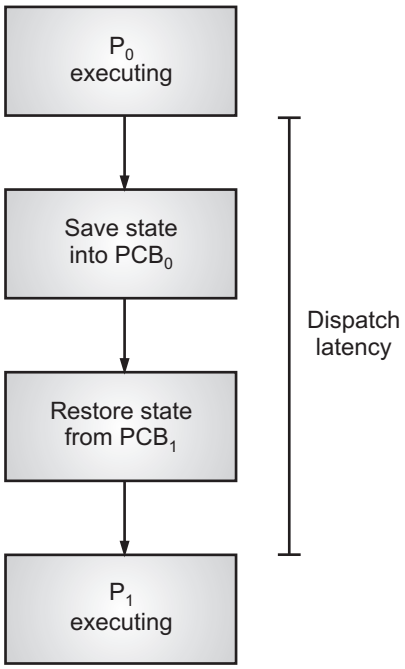



**Fig. 3.5: Concept of Dispatch Latency**

- The idea of multiprogramming is relatively simple if a process (job) is waiting for an I/O request, then the CPU switches from the process to another through a dispatcher, so the CPU is always busy in multiprogramming.

**Difference between Dispatcher and Scheduler:**

- The scheduler works independently, while the dispatcher has to be dependent on the scheduler i.e. the dispatcher transfers only those processes that are selected by the schedule
- The scheduler selects a process from a list of processes by applying some process scheduling algorithm. The dispatcher transfers the process selected by the short-term scheduler from one state to another.
- The only duty of a scheduler is to select a process from a list of processes. But apart from transferring a process from one state to another, the dispatcher can also be used for switching to user mode. Also, the dispatcher can be used to jump to a proper location when the process is restarted.
- Dispatcher is a module that provides control of the CPU to the process. CPU scheduler is also a module selects one of the processes in memory that are ready for execution.
- For selecting a process, the scheduler uses some process scheduling algorithm like FCFS, Round-Robin, SJF, etc. But the dispatcher doesn't use any kind of scheduling algorithms.

## 3.2 SCHEDULING ALGORITHMS

- CPU scheduling is the process of selecting a process from the ready queue and assigning the CPU to it. CPU scheduling algorithms decides which of the processes in the ready queue is to be allocated to the CPU.
- The algorithm used by the scheduler to carry out the selection of a process for execution is known as scheduling algorithm.
- Scheduler algorithms are either preemptive or non-preemptive. Non-preemptive algorithms are designed so that once a process enters the running state, it cannot be preempted until it completes its allotted time.
- The preemptive scheduling is based on priority where a scheduler may preempt a low priority running process anytime when a high priority process enters into a ready state.
- A number of scheduling algorithms are available for CPU scheduling including First Come First Served (FCFS) scheduling algorithm, Shortest Job First (SJF) scheduling algorithm, Priority scheduling algorithm, Round Robin (RR) scheduling algorithm and so on.
- To find out performance of the scheduling algorithms, we can use Gantt charts. Gantt chart is nothing but a chart use to describe a schedule. It will represent the sequence in which the jobs will be scheduled by the CPU.

## 3.2.1 FCFS (First Come First Serve) Scheduling Algorithm

**[April 17, Oct. 17]**

- It is the simplest type of algorithm in which the process that requests the CPU first is allocated the CPU first.
- In FCFS scheduling algorithm processes are scheduled in the order they are received. The FCFS algorithm is easily implemented by using a queue data structure for the ready queue.
- Jobs are processed in the order of their arrival in the ready queue. It can be implemented with FIFO (First In First Out (FIFO)) queue.
- The FCFS scheduling algorithm is non-pre-emptive. Once the CPU has been allocated to a process, that process keeps the CPU until it wants to release the CPU, either by terminating or by requesting I/O.
- In time-sharing system it is not useful because process will hold the CPU until it finishes or changes a state to wait state.
- In FCFS once, the processes is given to the CPU it keeps it till the completion of execution, (See Fig. 3.6).

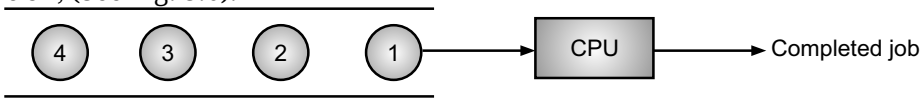


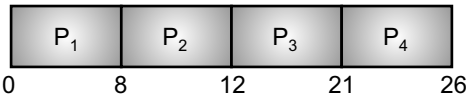

**Fig. 3.6: Concept of FCFS**

- Performance of FCFS is often very poor because a process with a long CPU burst will hold up other processes. Moreover, it can affect overall throughput since I/O on processes in the waiting state may complete; while the CPU bound process is still running.
- Average waiting time for FCFS algorithm is not minimal, and it also varies substantially if the process CPU burst time vary greatly.

**Example 1:** Consider the four jobs are scheduled for execution (all jobs arrived at same time). Find the average waiting time and turnaround time.

| Process | CPU Burst Time |
|---|---|
| $P_1$ | 8 |
| $P_2$ | 4 |
| $P_3$ | 9 |
| $P_4$ | 5 |

**Solution: Gantt Chart:**

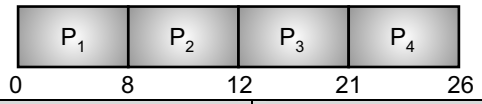



| Waiting Time | Turnaround Time |
|---|---|
| Waiting time of Process $P_1$ = 0<br>Waiting time of Process $P_2$ = 8<br>Waiting time of Process $P_3$ = 12<br>Waiting time of Process $P_4$ = 21 | TAT of process $P_1$ = 8<br>TAT of process $P_2$ = 12<br>TAT of process $P_3$ = 21<br>TAT of process $P_4$ = 26 |
| Average Waiting Time = (0 + 8 + 12 + 21)/4 = 41/4 = 10.25 msec. | Average TAT = (8 + 12 + 21 + 26)/4 = 67/4 = 16.75 msec. |

**Example 2:** Consider the four jobs are scheduled for execution. Find the average waiting time and turnaround time.

| Process | Arrival Time | CPU Burst Time |
|---|---|---|
| $P_1$ | 0 | 8 |
| $P_2$ | 1 | 4 |
| $P_3$ | 2 | 9 |
| $P_4$ | 3 | 5 |

**Solution: Gantt Chart:**




| Waiting Time | Turnaround Time |
|---|---|
| Waiting time = Starting time − Arrival time<br>Waiting time of Process $P_1$ = 0 − 0 = 0<br>Waiting time of Process $P_2$ = 8 − 1 =7<br>Waiting time of Process $P_3$ = 12 − 2 = 10<br>Waiting time of Process $P_4$ = 21 − 3 = 18 | TAT = Ending Time − Arrival Time<br>TAT of process $P_1$ = 8 − 0 = 8<br>TAT of process $P_2$ = 12 − 1 = 11<br>TAT of process $P_3$ = 21 − 2 = 19<br>TAT of process $P_4$ = 26 − 3 = 23 |
| Average Waiting Time = (0 + 7 + 10 + 18)/4 = 35/4 = 8.75 msec. | Average TAT = (8 + 11 + 19 + 23)/4 = 61/4 = 15.25 msec. |

**Advantages:**

1. FCFS is easier to understand and implement as processes are simply to be added at the end and removed from the front of queue. No process from in between the queue is required to be accessed.
2. FCFS is well suited for batch systems where the longer time periods for each process arc often acceptable.

**Disadvantages:**

1. Average waiting time is very large.
2. FCFS is not an attractive alternative on its own for a single processor system.
3. Another difficulty with FCFS tends to favor processor bound. Processes over I/O bound processes and may result in inefficient use of both the processor and the I/O devices.

## 3.2.2 SJF (Shortest Job First) Scheduling Algorithm [Oct. 17, April 18]

- The SJF scheduling algorithm is also knows as Shortest Process Next (SPN) scheduling algorithm or Shortest Request Next (SRN) scheduling algorithm that schedules the processes according to the length of the CPU burst they require.
- In SJF algorithm, the process with the shortest expected processing time is assigned to CPU. Hence, the name is shortest job first.
- Jobs or processes are processed in the ascending order of their CPU burst times. Every time the job with smallest CPU burst-time is selected from the ready queue.

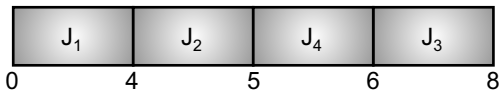
- If the two processes having same CPU burst Time then they will be scheduled according to FCFS algorithm. Performance of this algorithm is very good in comparison to FCFS.
- The SJF scheduling algorithm is probably optimal; it gives minimal average waiting time for a given set of processes.
- By moving short process before a long one, the waiting time of short process decreases. Consequently average waiting time reduces.
- In SJF scheduling algorithm the process in the ready queue with the shortest expected processing time is assigned to CPU next. SJF can be evaluated in two different manners:
  1. **Non pre-emptive SJF:** In this method, if CPU is executing one job, it is not stopped in between before completion.
  2. **Pre-emptive SJF:** In this method, while CPU is executing a job, if a new job arrives with smaller burst time, then the current job is pre-empted (sent back to ready queue) and the new job is executed. It is also called Shortest Remaining Time First (SRTF).

**Example:** Consider following table, find turnaround time and wait time.

| Jobs | Burst Time | Arrival Time |
|---|---|---|
| $J_1$ | 4 | 0 |
| $J_2$ | 1 | 1 |
| $J_3$ | 2 | 2 |
| $J_4$ | 1 | 3 |

1. **Non-preemptive SJF:**
   - At time unit only one job ($J_1$) is in the ready queue. So we must start with $J_1$.
   - $J_2$ and $J_4$ has same burst times, therefore by applying FCFS we will consider $J_2$ before $J_4$.
   - As it is non-pre-emptive, $J_1$ will not be pre-empted before completion.

**Gantt Chart:**




| Jobs | Turnaround Time | Wait Time |
|---|---|---|
| $J_1$ | (4 – 0) = 4 | (0 – 0) = 0 |
| $J_2$ | (5 – 1) = 4 | (4 – 1) = 3 |
| $J_3$ | (8 – 2) = 6 | (6 – 2) = 4 |
| $J_4$ | (6 – 3) = 3 | (5 – 3) = 2 |
| | 17 | 9 |

Average turnaround time = $\frac{17}{4}$ = 4.25 msec.

Average wait time = $\frac{9}{4}$ = 2.25 msec.

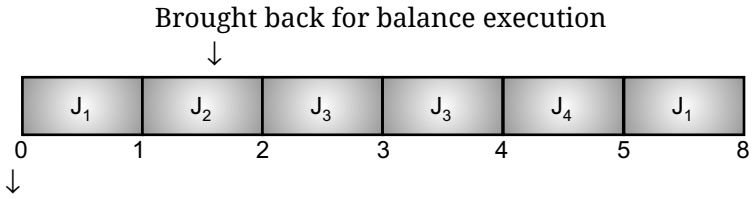
**2. Pre-emptive SJF:**

At zero time units there is only one job in the ready queue. Therefore we must start with $J_1$. At one time unit a new job $J_2$ arrives with smaller burst time therefore $J_1$ will be pre-empted and $J_2$ will complete its execution.

At two time units $J_3$ arrives and CPU starts executing it. $J_3$ has a burst time of two units. After one unit of it is over (i.e. at 3 time units) $J_4$ arrives with 1 as burst time.

Now if we compare balance burst time of $J_3$ (i.e. 2 – 1) and burst time of $J_4$ (i.e. 1) both is exactly same. Applying FCFS, $J_3$ will continue. At the end $J_4$ will be executed and then balance of $J_1$.

**Gantt Chart:**




| Jobs | Turnaround Time | Wait Time |
|---|---|---|
| $J_1$ | (8 – 0) = 8 | (0 – 0) + (5 – 1) = 4 |
| $J_2$ | (2 – 1) = 1 | (1 – 1) = 0 |
| $J_3$ | (4 – 2) = 2 | (2 – 2) = 0 |
| $J_4$ | (5 – 3) = 2 | (4 – 3) = 1 |
| | 13 | 5 |

Average Turnaround Time (TAT) = $\frac{13}{4}$ = 3.25 msec.

Average wait time = $\frac{5}{4}$ = 1.25 msec.

- In the above mentioned example $J_1$ started its execution at zero time units but as it was pre-empted, it's execution was over at 8 time units. Therefore, turnaround time for $J_1$ is 8 – 0 = 8.
- J1 is arrived at zero time units and immediately started executing. Therefore, wait time for J1 is (0 – 0) = 0.
- But when $J_1$ was pre-empted, it was again waiting in the ready queue for CPU. This time (5 – 1) if added to previous wait time.

  ∴ Total wait time of $J_1$ = (0 – 0) + (5 – 1) = 4.

**Example:** Consider the following processes, with the CPU burst time given in milliseconds. Calculate average waiting time and turnaround time. (Consider all jobs arrived at same time).

| Process | Burst Time |
|---|---|
| $P_1$ | 6 |
| $P_2$ | 8 |
| $P_3$ | 7 |
| $P_4$ | 3 |

**Solution: Gantt Chart:**


| Waiting Time | Turnaround Time |
|---|---|
| Waiting time of Process $P_1$ = 3<br>Waiting time of Process $P_2$ = 16<br>Waiting time of Process $P_3$ = 9<br>Waiting time of Process $P_4$ = 0 | TAT of process $P_1$ = 9<br>TAT of process $P_2$ = 24<br>TAT of process $P_3$ = 16<br>TAT of process $P_4$ = 3 |
| Average Waiting Time = (3 + 16 + 9 + 0)/4 = 28/4 = 7 msec. | Average TAT = (9 + 24 + 16 + 3)/4 = 52/4 = 13 msec. |

- The SJF algorithm may be either pre-emptive or non-preemptive. The choice arises when a new process arrives at ready queue while a previous process is executing.
- The new process may have a shorter next CPU burst than what if left of the currently executing process. A pre-emptive SJF algorithm will preempt the currently executing process and CPU is allocated to newly arrived process.

  Whereas, a non-preemptive SJF algorithm will allow the currently running process to finish its CPU burst. Pre-emptive SJF scheduling is sometimes called Shortest Remaining Time First (SRTF) scheduling.

**Advantages of SJF Scheduling Algorithm:**

1. Overall performance is significantly improve in terms of response time.
2. SJF algorithm eliminates the variance in waiting and turnaround times.

**Disadvantages of SJF Scheduling Algorithm:**

1. There is a risk of starvation of longer processes.
2. It is very difficult to know the length of next CPU burst.

**Shortest Remaining Time (SRT) Scheduling:**

- SRT is a pre-emptive version of Shortest Process Next (non-preemptive version) in which the scheduler always chooses the process that has the shortest expected remaining process time.
- In SRT, process with the smallest estimated run time to completion is run next, in SJF once a job begin executing, it runs to completion. In SRT a running process may be preempted by a user process with a shorter estimated run time.
- Consider an example, where three processes arrived in the order P1, P2, P3 at the time mentioned below, and then the average waiting time using SJF scheduling algorithm will be calculated as:

| Process | CPU Burst Time | Time of Arrival |
|---|---|---|
| $P_1$ | 10 | 0 |
| $P_2$ | 5 | 1 |
| $P_3$ | 2 | 2 |

**Gantt Chart:**


Waiting time for $P_1$ = 8 – 1 = 7 msec.

Waiting time for $P_2$ = 4 – 2 = 2 msec.

Waiting time for $P_3$ = 0 msec.

Average waiting time = (7 + 2 + 0) / 3 = 4.5 msec.

**Advantages of SRT Scheduling:**

1. SRT scheduling gives superior turnaround time performance to shortest process next because a short job is given immediate preference to a running longer job.
2. Throughput in SRT scheduling is high.

**Disadvantages of SRT Scheduling:**

1. In SRT scheduling elapsed time (i.e., execution-completed-time) must be recorded, it results an additional overhead on the processor.
2. In SRT scheduling starvation may be possible for the longer processes.

**Shortest Remaining Time First (SRTF):**

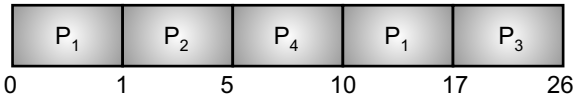
- Shortest Remaining Time (SRT) is a pre-emptive version of SJF. As the name suggests, it selects the process for execution which has the smallest amount of time remaining until completion.
- In SJF, once a process begins execution, it runs till completion. In SRTF a running process may be preempted by a user process with a smallest estimated run time.
- SRT also known as Shortest Remaining Time First (SRTF) in which a scheduler chooses the process that has the shortest remaining processing time.

**Example:** Consider four processes with the length of CPU burst time given in milliseconds.

| Process | Arrival Time | Burst Time |
|---|---|---|
| $P_1$ | 0 | 8 |
| $P_2$ | 1 | 4 |
| $P_3$ | 2 | 9 |
| $P_4$ | 3 | 5 |

**Solution:** Using Pre-emptive Shortest Job First Algorithm:

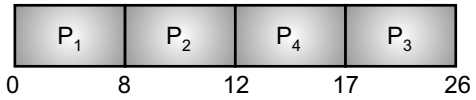
**Gantt Chart:**



| Waiting Time | Turnaround Time |
|---|---|
| Waiting time of Process $P_1$ = 10 – 1 = 9<br>Waiting time of Process $P_2$ = 0<br>Waiting time of Process $P_3$ = 17 – 2 = 15<br>Waiting time of Process $P_4$ = 5 – 3 = 2 | TAT of process $P_1$ = 17<br>TAT of process $P_2$ = 5 – 1 = 4<br>TAT of process $P_3$ = 26 – 2 = 24<br>TAT of process $P_4$ = 10 – 3 = 7 |
| Average Waiting Time = (9 + 0 + 15 + 2)/4 = 26/4 = 6.5 msec. | Average TAT = (17 + 4 + 24 + 7)/4 = 52/4 = 13 msec. |

**Using Non-Preemptive Shortest Job First Algorithm:**

**Gantt Chart:**




| Waiting Time | Turnaround Time |
|---|---|
| Waiting time of Process $P_1$ = 0<br>Waiting time of Process $P_2$ = 8 – 1 = 7<br>Waiting time of Process $P_3$ = 17 – 2 = 15<br>Waiting time of Process $P_4$ = 12 – 3 = 9 | TAT of process $P_1$ = 8<br>TAT of process $P_2$ = 12 – 1 = 11<br>TAT of process $P_3$ = 26 – 2 = 24<br>TAT of process $P_4$ = 17 – 3 = 14 |
| Average Waiting Time = (0 + 7+ 15 + 9)/4 = 31/4 = 7.75 msec. | Average TAT = (8 + 11 + 24 + 14)/4 = 57/4 = 14.25 msec. |

**Advantages of SRTF Scheduling:**

1. Many processes execute in less amount of time. So, throughput is increased.
2. Processes which have short burst time run fast.

**Disadvantages of SRTF Scheduling:**

1. Practically it is not possible to predict the burst time.
2. Processes which have long burst time will have to wait for long time for execution.

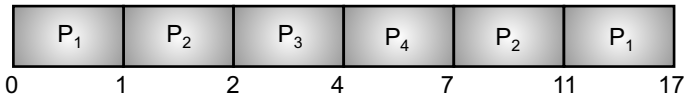
**Shortest Remaining Time Next (SRTN) Scheduling Algorithm:**

- SRTN scheduling algorithm is a pre-emptive form of Shortest Job First (SJF) scheduling algorithm. The shortest remaining time next also known as Shortest Time to Go (STG) scheduling algorithm.
- SRTN is a scheduling discipline in which the next scheduling entity, a job or a process, is selected on the basis of the shortest remaining execution time.
- SRTN scheduling may be implemented in either the non-preemptive or the pre-emptive variety.
- In this algorithm a job is chosen whose remaining run time is the shortest. For a new job, its run time is compared with remaining time of current job.
- If new job needs less time to complete than the current job, then, the current job is blocked and the new job is run. It is used for batch systems and provides advantages to new jobs.

- Consider following table, which contains four processes P1, P2, P3 and P4, with their arrival times and required CPU burst (in milliseconds):

| Processes | Arrival Time | CPU Burst |
|---|---|---|
| $P_1$ | 0 | 7 |
| $P_2$ | 1 | 5 |
| $P_3$ | 3 | 2 |
| $P_4$ | 4 | 3 |

**Gantt Chart:**




| Waiting Time | Turnaround Time |
|---|---|
| Waiting time of Process $P_1$ = 11 – 1 = 10<br>Waiting time of Process $P_2$ = 7 – 2 = 5<br>Waiting time of Process $P_3$ = 2 – 2 = 0<br>Waiting time of Process $P_4$ = 4 – 4 = 0 | TAT of process $P_1$ = 17<br>TAT of process $P_2$ = 11 – 1 = 10<br>TAT of process $P_3$ = 4 – 2 = 2<br>TAT of process $P_4$ = 7 – 3 = 4 |
| Average Waiting Time = (10 + 5 + 0 + 0)/4 = 15/4 = 3.75 msec. | Average TAT = (17 + 10 + 2 + 4)/4 = 33/4 = 8.25 msec. |

**Advantages of SRTN Scheduling Algorithm:**

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

**Disadvantages of SRTN Scheduling Algorithm:**

1. It favors only those long processes that are just about to complete and not those who have just started their operation. Thus, starvation of long processes still may occur.
2. Like SJF, SRTN also requires an estimate of the next CPU burst of a process in advance.
3. Favoring a long process nearing its completion over the several short processes entering the system may affect the turnaround limes of short processes.

## 3.2.3 Priority Scheduling Algorithm [April 17, 18, 19]

- A priority is associated with each process and the CPU is allocated to the process with the highest priority, hence it is called Priority Scheduling.
- In priority scheduling algorithm, a priority is associated with each process and the scheduler always picks up the highest priority process for execution from the ready queue. Equal priority processes are scheduled in FCFS order.

- The SJF algorithm is a special case of the priority scheduling algorithm, where the priority is the inverse of the next CPU burst, i.e. lower priority is assigned to the larger CPU burst.
- Priorities are roughly categorized into internal and external priorities as explained below:
  1. **Internal priorities** are based on burst time, memory requirements, number of open files etc. measurable quantities.
  2. **External priorities** are human created e.g. seniority, influence of any person, etc. In our syllabus only internal priorities are considered.
- The priority scheduling can be either pre-emptive or non-pre-emptive. When process enters into the ready queue, its priority is compared with the priority of the current running process.
- In a pre-emptive priority scheduling algorithm, at any time, the CPU is allocated one process and if the priority of the newly arrived process is higher than the priority of the currently running process, the process which was allocated is interrupted and return to the queue. The higher priority job is started for execution.
- A non-preemptive priority - scheduling algorithm will simply put the new process at the head of the ready queue.
- A problem with priority scheduling algorithm is indefinite blocking or starvation. Here, the low priority process may be indefinitely locked out by the higher priority processes.
- In general, completion of a process within finite time of its creation cannot be guaranteed with this scheduling policy. A solution to the problem of starvation of low priority processes is aging.
- Aging is process in which the priority of each process is gradually increased after the process spends a certain amount of time in the system.
- Even a process with an initial priority 0, get the higher priority in the system and it would be executed.

**Working of Priority Scheduling Algorithm:**

- Fig. 3.7 shows a system with four priority classes.

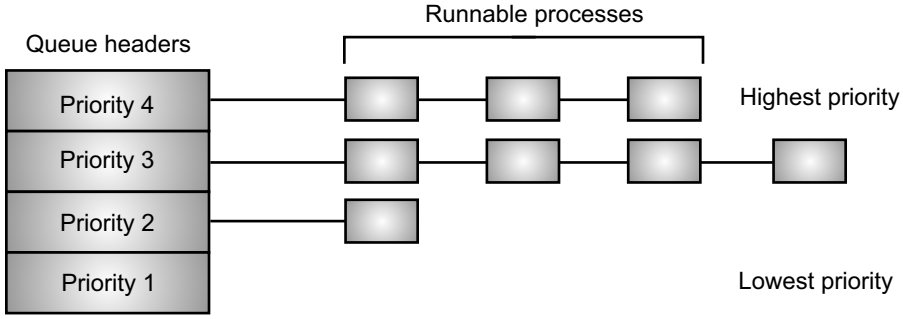



**Fig. 3.7: A Scheduling Algorithm with Four (4) Priority Classes**

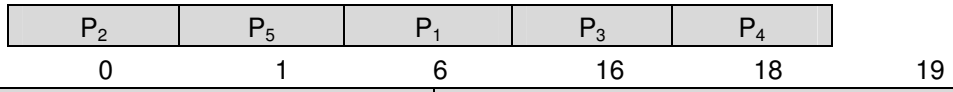
- The scheduling algorithm is as follows as long as there are runnable processes in priority class 4, just run each one for one quantum (i.e. round robin fashion) and never bothers with lower priority classes.
- If priority class 4 in empty, then run the class 3 processes in round robin fashion, and so on. If priorities are not adjusted from time to time, lower priority classes may all starve.

**Example 1:** Consider the following set of processes assumed to have arrived at time 0, in the order $P_1$, $P_2$, $P_3$, …… $P_5$, with the length of the CPU burst time given in milliseconds. Calculate the average TAT and Waiting Time.

| Process | Burst Time | Priority |
|---|---|---|
| $P_1$ | 10 | 3 |
| $P_2$ | 1 | 1 |
| $P_3$ | 2 | 3 |
| $P_4$ | 1 | 4 |
| $P_5$ | 5 | 2 |

**Solution:** Using priority scheduling, we would schedule these processes according to the following,

**Gantt Chart:**




| Waiting Time | Turnaround Time |
|---|---|
| Waiting time of Process $P_1$ = 6<br>Waiting time of Process $P_2$ = 0<br>Waiting time of Process $P_3$ = 16<br>Waiting time of Process $P_4$ = 18<br>Waiting time of Process $P_5$ = 1 | TAT of process $P_1$ = 16<br>TAT of process $P_2$ = 1<br>TAT of process $P_3$ = 18<br>TAT of process $P_4$ = 19<br>TAT of process $P_5$ = 6 |
| Average Waiting Time = (6 + 0 + 16 + 18 + 1)/5 = 41/5 = 8.2 | Average TAT = (16 + 1 + 18 + 19 + 6)/5 = 60/5 = 12 msec. |

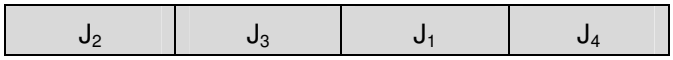
**Example 2:** Consider following table.

| Jobs | Burst Time | Priority |
|---|---|---|
| $J_1$ | 4 | 4 |
| $J_2$ | 6 | 1 (Highest) |
| $J_3$ | 2 | 3 |
| $J_4$ | 3 | 5 (Lowest) |

**Note:**
- If there are more than one jobs with same priority apply FCFS.
- If along with priorities highest and lowest is not specified, please do not forget to make your own assumption and specify it before you solve the problem.

**Solution:** Gantt chart for the above example would be:

Arrival of all jobs assumed to be zero.




| Jobs | Turnaround Time | Wait Time |
|---|---|---|
| $J_1$ | (12 – 0) = 12 | (8 – 0) = 8 |
| $J_2$ | (6 – 0) = 6 | (0 – 0) = 0 |
| $J_3$ | (8 – 0) = 8 | (6 – 0) = 6 |
| $J_4$ | (17 – 0) = 17 | (12 – 0) = 12 |
| | 43 | 26 |

Average turnaround time = $\frac{43}{4}$ = 10.75 msec.

Average wait time = $\frac{26}{4}$ = 6.5 msec.

**Advantages of Priority Scheduling Algorithm:**

1. It is simple in use.
2. Important processes are never made to wait because of the execution of less important processes.
3. Suitable for applications with varying time and resource requirements.

**Disadvantages of Priority Scheduling Algorithm::**

1. It suffers from the problem of starvation of lower priority processes, since the continuous arrival of higher priority processes will prevent lower priority processes indefinitely from acquiring the CPU.
2. Apriority scheduling can leave some low priority waiting processes indefinitely for CPU.

## 3.2.4 Round Robin (RR) Scheduling Algorithm [April 16, 19, Oct. 18]

- The Round Robin (RR) scheduling algorithm is designed especially for time sharing systems. RR is the pre-emptive process scheduling algorithm.
- Round robin scheduling is a preemptive version of First Come First Serve scheduling. Processes are (FCFS) dispatched in a First In First Out (FIFO) sequence but each process is allowed to run for only a limited amount of time.
- It is similar to FCFS scheduling but pre-emption is added to switch between processes. A small unit of time called a Time Quantum or Time Slice is defined.
- A time quantum is generally from 10 to 100 milliseconds. The ready queue is treated as a circular queue. The CPU scheduler goes around the ready queue, allocating CPU to each process for a time interval of up to 1 time quantum.
- In RR scheduling processes are dispatched FIFO but are given a limited amount of processor time called a time slice or a quantum.

- If a process does not complete before its quantum expires, the system preempts it and gives the processor to the next waiting process. The system then places the preempted process at the back of the ready queue.
- In Fig. 3.8, process P1 is dispatched to a processor, where it executes either until completion, in which case it exists the system, or until its time slice expires, at which point is preempted and placed at the tail of the ready queue. The scheduler then dispatches process P2.

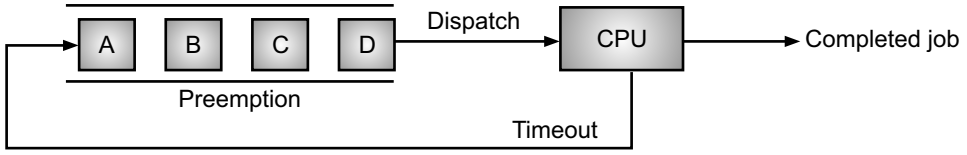



**Fig. 3.8: RR Scheduling**

- To implement RR scheduling we keep ready queue as FIFO queue of processes. New processes are added to the tail of the ready queue. The average waiting time under the RR policy however is often quite long.

**Example 1:** Consider the following set of processes that arrive at time 0, with the length of CPU burst time given in milliseconds. (time quantum=4 msec.)

| Process | Burst Time |
|---|---|
| $P_1$ | 24 |
| $P_2$ | 3 |
| $P_3$ | 3 |

If we use a time quantum of 4 milliseconds, then process $P_1$ gets the first 4 milliseconds, since it requires another 20 milliseconds, it is pre-empted after first time quantum and CPU is given to the next process in the queue, process $P_2$. Process $P_2$ does not need 4 milliseconds; it quits before its time quantum expires. The CPU is then given to next process, process $P_3$. Once each process has received 1 time quantum, the CPU is returned to process $P_1$ for an additional time quantum. The resulting RR schedule is:

**Solution: Gantt Chart:**

| $P_1$ | $P_2$ | $P_3$ | $P_1$ |
|---|---|---|---|

0 4 7 10 30

| Waiting Time | Turnaround Time |
|---|---|
| Waiting time of Process $P_1$ = 10 – 4 = 6<br>Waiting time of Process $P_2$ = 4<br>Waiting time of Process $P_3$ = 7 | TAT of process $P_1$ = 30<br>TAT of process $P_2$ = 7<br>TAT of process $P_3$ = 10 |
| Average Waiting Time = (6 + 4 + 7)/3 = 17/3 = 5.66 msec. | Average TAT = (30 + 7 + 10)/3 = 47/3 = 16.66 msec. |

**Example 2:** Consider the following process arrived at given time. Calculate TAT and Waiting time of each process, (time quantum=3).

| Process | Arrival Time | Service Time |
|---|---|---|
| $P_1$ | 0 | 5 |
| $P_2$ | 1 | 3 |
| $P_3$ | 2 | 8 |
| $P_4$ | 3 | 6 |

**Solution: Gantt Chart:**

| $P_1$ | $P_2$ | $P_3$ | $P_4$ | $P_1$ | $P_3$ | $P_4$ | $P_3$ |
|---|---|---|---|---|---|---|---|

0 3 6 9 12 14 17 20 22

| Waiting Time | Turnaround Time |
|---|---|
| Waiting time of Process $P_1$ = 12 – 3 = 9<br>Waiting time of Process $P_2$ = 3 – 1 = 2<br>Waiting time of Process $P_3$ = (6 – 2) + (14 – 9) + (20 – 17) = 4 + 5 + 3 = 12<br>Waiting time of Process $P_4$ = (9 – 3) + (17 – 12) = 6 + 5 = 11 | TAT of Process $P_1$ = 14<br>TAT of Process $P_2$ = 6 – 1 = 5<br>TAT of Process $P_3$ = 22 – 2 = 20<br>TAT of Process $P_4$ = 20 – 3 = 17 |
| Average Waiting Time = (9 + 2 + 12 + 11)/4 = 34/4 = 8.5 msec. | ATAT = (14 + 5 + 2 0 + 17)/4 = 56/4 = 14 msec. |

**Advantages of RR Scheduling Algorithm:**

1. Round robin is particularly effective in a general purpose time sharing system or transaction processing system.
2. It is efficient for time sharing systems where the CPU time is divided among the competing processes.
3. RR increases the fairness among the processes.
4. In RR scheduling algorithm overhead on processor is low.

**Disadvantages of RR Scheduling Algorithm:**

1. In RR the processes (even the short processes) may take long time to execute. This decreases the system throughput.
2. RR requires some extra hardware support, such as a timer, to cause interrupt after each time out.
3. In RR scheduling care must be taken in choosing quantum value.
4. Throughput in RR scheduling algorithm is low if time quantum is too small.

**Comparison of various Scheduling Algorithms:**

| Sr. No. | FCFS Scheduling Algorithm | SJF Scheduling Algorithm | Priority Scheduling Algorithm | RR Scheduling Algorithm |
|---|---|---|---|---|
| 1. | This scheduling algorithm is non-pre-emptive. | This scheduling algorithm is pre-emptive. | This scheduling algorithm is also pre-emptive. | This scheduling algorithm is also pre-emptive. |

*contd. …*

| | | | | |
|---|---|---|---|---|
| 2. | This is simplest scheduling algorithm. | This algorithm is difficult to understand and code. | This algorithm is also difficult to understand. | In this scheduling algorithm performance heavily depends upon the size of time quantum. |
| 3. | In this scheduling algorithm it allocates the CPU in the order in which the processes arrives. | In this scheduling algorithm CPU is allocated to the process with least CPU burst time. | This scheduling algorithm is based on priority the higher priority job can run first. | In this scheduling algorithm the pre-emption take place after a fixed interval of time. |
| 4. | This scheduling algorithm is good for non interactive system. | This scheduling algorithm is also good for non interactive system. | This scheduling algorithm is also good for non interactive system. | This scheduling algorithm is good for interactive system. |
| 5. | In this scheduling algorithm the Average waiting time is large. | In this scheduling algorithm the Average waiting time is small as compare to FCFS scheduling algorithm. | In this scheduling algorithm the Average waiting time is small as compare to FCFS scheduling algorithm. | In this scheduling algorithm the Average waiting time is large as compare to all the three scheduling algorithm. |
| 6. | This scheduling algorithm is never recommended whenever performance is a major issue. | In this scheduling algorithm the problem is to know the length of time for which the CPU is needed by the process. | This scheduling algorithm is the sometime becomes the biggest cause of starvation. | In this scheduling algorithm if the quantum size is large then this algorithm become same as FCFS algorithm and its performance degrade. |
| 7. | This scheduling algorithm is suitable for Batch system. | This scheduling algorithm is also suitable for Batch system. | This scheduling algorithm is based upon priority. | This scheduling algorithm is suitable for time sharing system. |

## 3.2.5 Multi-Level Queue (MLQ) Scheduling Algorithm

**[Oct. 17, April 18]**

- Multilevel Queue Scheduling based on response - time requirements. Some process required a quick response by the processor; some processes can wait.

- A multilevel queue scheduling algorithm partitions the ready queue into separate queues.
- In a multilevel queue scheduling processes are permanently assigned to one queue, depending upon their properties such as the size of the memory or the type of the process or priority of the process. So each queue follows a separate scheduling algorithm.
- In multilevel queue scheduling algorithm scheduling the processes are classified into different groups such as System processes, Interactive processes, Interactive editing processes, Batch processes, User processes etc., as shown in Fig. 3.9.
- The interactive processes are known as foreground processes and the batch processes are known as background processes. These two types of processes have different response-time requirements and so may have different scheduling needs.
- In addition, foreground processes may have priority (externally defined) over background processes.
- A multilevel queue scheduling algorithm partitions the ready queue in several separate queues.
- In Fig. 3.9 each queue has absolute priority over lower-priority queues. No process in the batch queue, for example, could run unless the queues for system processes, interactive processes, and interactive editing processes were all empty.

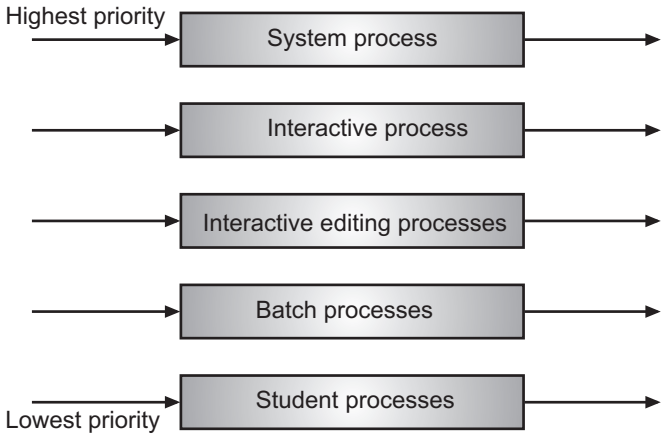



**Fig. 3.9**

- If an interactive editing process entered the ready queue while a batch process was running, the batch process would be preempted. Another possibility is to time-slice among the queues.
- Here, each queue gets a certain portion of the CPU time, which it can then schedule among its various processes.
- For instance, in the foreground-background queue example, the foreground queue can be given 80 percent of the CPU time for RR scheduling among its processes, whereas the background queue receives 20 percent of the CPU to give to its processes on an FCFS basis.

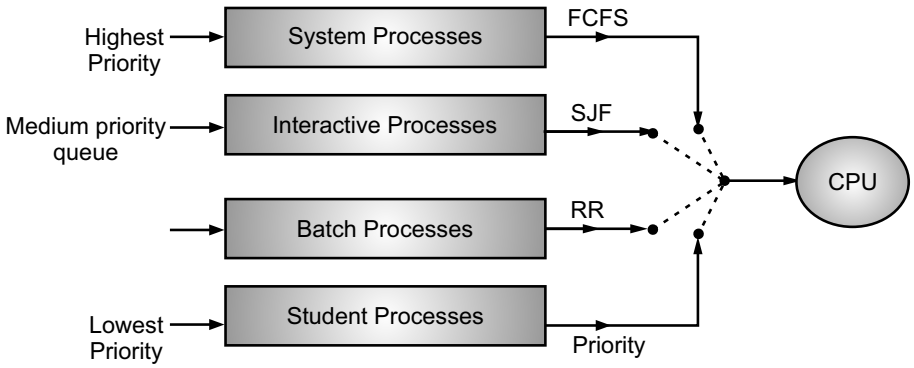



**Fig. 3.10: Multilevel Priority Queue Scheduling with Scheduling Algorithms**

**Advantages MLQ Scheduling Algorithm:**

1. In MLQ, the processes are permanently assigned to their respective queues and do not move between queues. This results in low scheduling overhead.
2. In MLQ one can apply different scheduling algorithms to different processes.
3. There are many processes which we are not able to put them in the one single queue which is solved by MLQ scheduling as we can now put them in different queues.

**Disadvantages MLQ Scheduling Algorithm:**

1. The processes in lower priority queues may have to starve for CPU in case processes are continuously arriving in higher priority queues.
2. In MLQ the process does not move from one queue to another queue.

## 3.2.6 Multi-Level Feedback Queue (MLFQ) Scheduling Algorithm

- Multi-level feedback queue scheduling is an enhancement of MLQ. The MLFQ scheduling also known as multilevel adaptive scheduling is an improved version of multilevel queue scheduling algorithm.
- In multilevel queue scheduling, processes cannot move from one queue to other because processes do not change their foreground or background nature.
- But in multilevel feedback queue scheduling, processes are not permanently assigned to a queue on entry to the system. **[Oct. 17]**
- Instead, they are allowed to move between queues. The idea is to separate processes with different CPU burst characteristics. If a process uses too much CPU time, it will be moved to a lower priority queue.
- Similarly, a process that waits too long in a low priority queue will be moved to a higher priority queue. This form of aging prevents starvation.
- A multilevel feedback queue scheduler is defined by the following parameters:
  1. The number of queues.
  2. The scheduling algorithm for each queue.
  3. The method used to determine when to upgrade a process to a higher priority queue.

4. The method used to determine when to demote a process to a lower priority queue.
5. The method used to determine which queue a process will enter when that process needs service.

- Fig. 3.11 shows multilevel feedback queue. For example, consider a multilevel feedback queue scheduler with three queues, numbered from $Q_0$, $Q_1$ to $Q_2$.
- The scheduler first executes all processes in Queue 0. Only when Queue 0 is empty will it execute processes in Queue 1.
- Similarly, processes in Queue 2 will only be executed if Queues 0 and 1 are empty. A process that arrives for Queue 1 will preempt a process in Queue 2. A process in Queue 1 will in turn be preempted by a process arriving for queue 0.
- A process entering the ready queue is put in Queue 0. A process in Queue 0 is given a time quantum of 4 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 8 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 but are run only when Queues 0 and 1 are empty. This scheduling algorithm gives highest priority to any process with a CPU burst of 4 milliseconds or less. Such a process will quickly get the CPU, finish its CPU burst, and go off to its next I/0 burst.

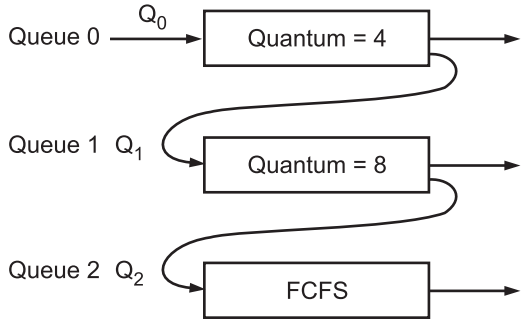


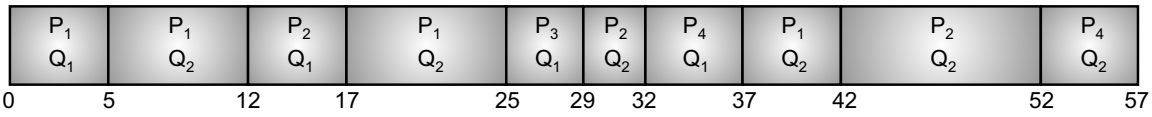

**Fig. 3.11: Multilevel Feedback Queue**

- Solaris 2.6 Time-Sharing (TS) scheduler implements multilevel feedback queue is a scheduling algorithm. The MacOS and Microsoft Windows schedulers can both be regarded as examples of the broader class of multilevel feedback queue schedulers.

**Example:** Consider following table, which contains four processes $P_1$, $P_2$, $P_3$ and $P_4$, with their arrival times and required CPU burst (in milliseconds):

| Process | Arrival Time | CPU Burst |
|---|---|---|
| $P_1$ | 0 | 25 |
| $P_2$ | 12 | 18 |
| $P_3$ | 25 | 4 |
| $P_4$ | 32 | 10 |

**Gantt Chart:**




| Waiting Time | Turnaround Time |
|---|---|
| Waiting time of Process $P_1$ = 5 + 12 = 17<br>Waiting time of Process $P_2$ = 12 + 10 = 22<br>Waiting time of Process $P_3$ = 0<br>Waiting time of Process $P_4$ = 52 – 37 = 15 | TAT of process $P_1$ = 42 – 0 = 42<br>TAT of process $P_2$ = 52 – 12 = 40<br>TAT of process $P_3$ = 29 – 25 = 4<br>TAT of process $P_4$ = 57 – 32 = 25 |
| Average Waiting Time = (17 + 22 + 0 + 15)/4 = 54/4 = 13.5 msec. | Average TAT = (42 + 40 + 4 + 25)/4 = 111/4 = 27.75 msec. |

**Advantages of Multilevel Queue Scheduling Algorithm:**

1. The multilevel feedback queue scheduling speeds up the flow of task execution.
2. It prevents starvation by moving a lower priority process to a higher priority queue if it has been waiting for too long.
3. It is fair to I/O bound (short) processes as these processes need not wait too long and are executed quickly.
4. It improves the overall performance of the system.

**Disadvantages of Multilevel Queue Scheduling Algorithm:**

1. The turnaround time for long processes may increase significantly.
2. It is the most complex scheduling algorithm.
3. In this algorithm, moving the processes between queues causes a number of context switches which results in an increased overhead.

## SOLVED PROBLEMS [April 17, Oct. 17, 18]

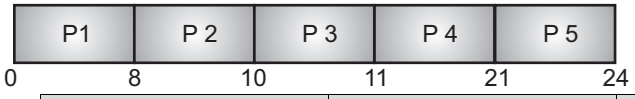
**Problem 1:** Consider the following set of processes that arrive in the ready queue at the same time.

| Processes | CPU Time |
|---|---|
| P1 | 8 |
| P2 | 2 |
| P3 | 1 |
| P4 | 10 |
| P5 | 3 |

Find out turnaround time and wait time for each of the process using FCFS (First Come First Served), SJF (Shortest Job First) and Round Robin (Quantum = 1).

**Solution:** Arrival time of all jobs assumed to be 0 FCFS.

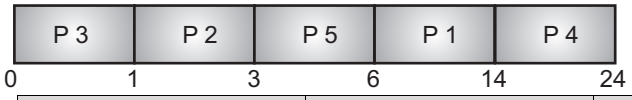
**Gantt Chart:**




| Process | Turnaround Time | Wait Time |
|---|---|---|
| P1 | (8 – 0) = 8 | (0 – 0) = 0 |
| P2 | (10 – 0) = 10 | (8 – 0) = 8 |
| P3 | (11 – 0) = 11 | (10 – 0) = 10 |
| P4 | (21 – 0) = 21 | (11 – 0) = 11 |
| P5 | (24 – 0) = 24 | (21 – 0) = 21 |

**SJF:** Pre-emptive or not pre-emptive is not specified. Therefore assumed to be non-preemptive.

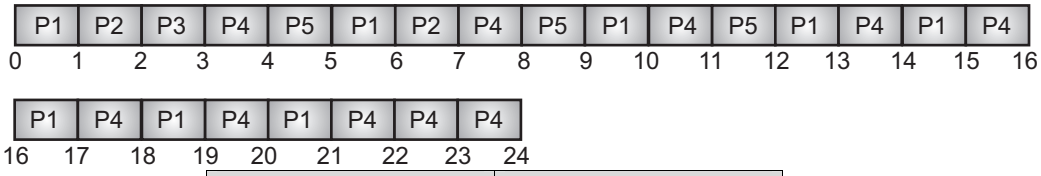
**Gantt chart:**




| Processes | Turnaround Time | Wait Time |
|---|---|---|
| P1 | (14 – 0) = 14 | (6 – 0) = 6 |
| P2 | (3 – 0) = 3 | (1 – 0) = 1 |
| P3 | (1 – 0) = 1 | (0 – 0) = 0 |
| P4 | (24 – 0) = 24 | (14 – 0) = 14 |
| P5 | (6 – 0) = 6 | (3 – 0) = 3 |

Round Robin with quantum 1.

**Gantt Chart:**




| Processes | Turnaround Time |
|---|---|
| P1 | (21 – 0) = 21 |
| P2 | (7 – 0) = 7 |
| P3 | (3 – 0) = 3 |
| P4 | (24 – 0) = 24 |
| P5 | (12 – 0) = 12 |

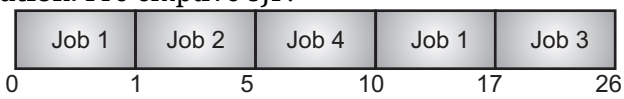
| Processes | Wait Time |
|---|---|
| P1 | (0 – 0) + (5 – 1) + (9 – 6) + (12 – 10) + (14 – 13) + (16 – 15) + (18 – 17) + (20 – 19) = 13 |
| P2 | (1 – 0) + (6 – 2) = 5 |
| P3 | (2 – 0) = 2 |
| P4 | (3 – 0) + (7 – 4) + (10 – 8) + (13 – 11) + (15 – 14) + (17 – 16) + (19 – 18) + (21 – 20) = 14 |
| P5 | (4 – 0) + (8 – 5) + (11 – 9) = 9 |

**Problem 2:** Consider snapshot of the system:

| Job | Arrival time | Burst time |
|---|---|---|
| 1 | 0 | 8 |
| 2 | 1 | 4 |
| 3 | 2 | 9 |
| 4 | 3 | 5 |

Compute average turn around time using pre-emptive SJF and non-pre-emptive SJF.

**Solution:** Pre-emptive SJF.



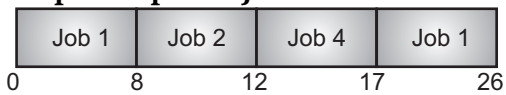

Average turnaround time:

| Job | Completion time - Arrival time |
|---|---|
| 1 | (17 – 0) = 17 |
| 2 | (5 – 1) = 4 |
| 3 | (26 – 2) = 24 |
| 4 | (10 – 3) = 7 |

$$\text{Average turn around time} = \frac{17 + 4 + 24 + 7}{4}$$

$$= \frac{52}{4} = 13$$

**Non-pre-emptive SJF:**




$$\text{Average turn around time} = \frac{(8 - 0) + (12 - 1) + (17 - 2) + (26 - 3)}{4}$$

$$= \frac{8 + 11 + 15 + 23}{4}$$

$$= \frac{57}{4}$$

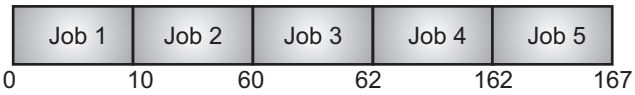
$$= \mathbf{14.25}$$

**Problem 3:** The following series of processes with the given estimated run times arrive in the ready queue in the order shown. For the FCFS and SJF scheduling policies, calculate the waiting time and wait time/run time ratio of each process. Comment on the results.

| Job | Estimated runtime |
|---|---|
| 1 | 10 |
| 2 | 50 |
| 3 | 2 |
| 4 | 100 |
| 5 | 5 |

**Solution:**

**(i) FCFS:** If the jobs arrive shortly one after the other in the order job1, job2, job3, job4, job5 the result shown in the following Gantt chart:




The waiting time is 0 time units for Job1, 10 time units for Job2, 60 time units for Job3, 62 time units for Job4, and 162 time units for Job5.

Thus, the wait time/runtime ratio for job1 = 0,

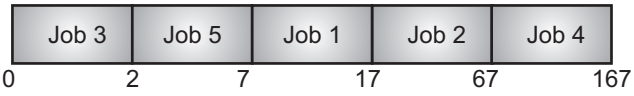
The wait time/runtime ratio for Job2 = $\frac{10}{50}$ = 0.2,

The wait time/runtime ratio for Job3 = $\frac{60}{2}$ = 30,

The wait time/runtime ratio for Job4 = $\frac{62}{100}$ = 0.62,

The wait time/runtime ratio for Job5 = $\frac{162}{5}$ = 32.4

**(ii) SJF:** The result of the above example, using SJF scheduling is shown in the following Gantt chart:




The waiting time is 7 for Job1, 17 time unit for Job2, 0 time unit for Job3, 67 time unit for Job4, and 2 time unit for Job 5.

Thus, wait time/runtime ratio,

for Job1 = $\frac{7}{10}$ = 0.7

for Job2 = $\frac{17}{50}$ = 0.34

for Job3 = $\frac{0}{2}$ = 0

for Job4 = $\frac{67}{100}$ = 0.67

for Job5 = $\frac{2}{5}$ = 0.4

From the above result, it is found that, SJF is a optional scheduling algorithm in terms of minimizing the average waiting time of a given work load.

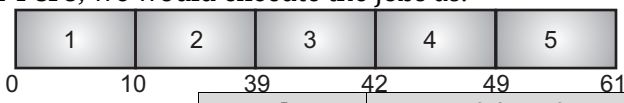
**Problem 4:** Five jobs arrive at time 0, in the order given.

| Jobs | Burst Time |
|---|---|
| 1 | 10 |
| 2 | 29 |
| 3 | 3 |
| 4 | 7 |
| 5 | 12 |

Considering FCFS, SJF and RR (quantum = 10) scheduling algorithms for this set of jobs, which algorithms would give the minimum average waiting time ?

**Solution:**

(a) For FCFS, we would execute the jobs as:

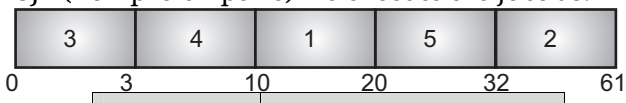



| Jobs | Waiting time |
|---|---|
| 1 | 0 |
| 2 | 10 |
| 3 | 39 |
| 4 | 42 |
| 5 | 49 |
| | 140 |

The average waiting time is then,

$$\frac{140}{5} = 28$$

(b) For SJF (non-pre-emptive) we execute the jobs as:

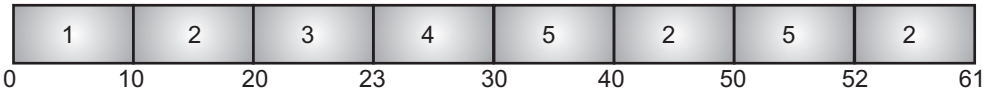



| Jobs | Waiting time |
|---|---|
| 1 | 10 |
| 2 | 32 |
| 3 | 0 |
| 4 | 3 |
| 5 | 20 |
| | 65 |

The average waiting time is,

$$\frac{65}{5} = 13$$

(c) With RR (quantum = 10), we start job 2, but pre-empt it after 10 time units, putting it in the back of the queue.




| Jobs | Waiting time |
|---|---|
| 1 | 0 |
| 2 | 32 |
| 3 | 20 |
| 4 | 23 |
| 5 | 40 |
| | 115 |

The average waiting time = $\frac{115}{5} = 23$

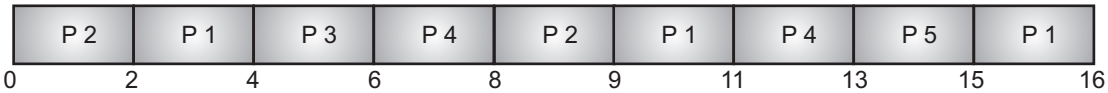
---

**Problem 5:** Consider the following snapshot.

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

Compute average turnaround time and average waiting time with RR algorithm with time slice = 2.

**Solution: Gantt chart:**




$$\text{Average turn around time} = \frac{(16-1)+(9-0)+(6-2)+(13-3)+(15-13)}{5}$$

$$= \frac{15+9+4+10+2}{5} = \frac{40}{5} = 8$$

$$\text{Average waiting time} = \frac{(15-4-1)+(8-2-0)+(4-2)+(11-2-3)+(13-13)}{5}$$

$$= \frac{10+6+2+6+0}{5}$$

$$= \frac{24}{5} = 4.8$$

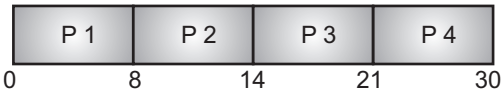
**Problem 6:** Consider the following set of processes with CPU burst time given in milliseconds.

| Process | Burst time | Arrival time | Priority |
|---|---|---|---|
| P1 | 8 | 0 | 4 |
| P2 | 6 | 1 | 6 |
| P3 | 7 | 3 | 3 |
| P4 | 9 | 3 | 1 (highest) |

Illustrate the evaluation of these process using non-preemptive SJF and priority preemptive CPU scheduling algorithm. Also calculate average waiting time.

**Solution: 1. Non-preemptive SJF:**

**Gantt Chart:**




$$\text{Average turn around time} = \frac{(8-0)+(14-1)+(21-3)+(30-3)}{4}$$

$$= \frac{8+13+18+27}{4}$$

$$= \frac{66}{4} = 16.5$$

$$\text{Average waiting time} = \frac{(0-0)+(8-1)+(14-3)+(21-3)}{4}$$

$$= \frac{0+7+11+18}{4} = \frac{36}{4} = 9$$

**2. Priority preemptive algorithm:**

**Gantt chart:**


$$\text{Average turn around time} = \frac{(24-0)+(30-1)+(19-3)+(12-3)}{4}$$

$$= \frac{24+29+16+9}{4}$$

$$= \frac{78}{4} = 19.5$$

$$\text{Average waiting time} = \frac{(19-3-0)+(24-1)+(12-3)+(3-3)}{4}$$

$$= \frac{16+23+9+0}{4}$$

$$= \frac{48}{4}$$

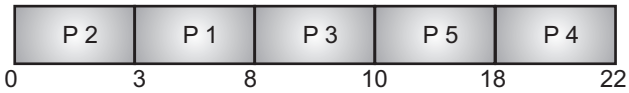
$$= 12$$

**Problem 7:** Consider the following set of process with CPU burst time given in milliseconds.

| Process | Burst time | Arrival time |
|---|---|---|
| P1 | 5 | 1 |
| P2 | 3 | 0 |
| P3 | 2 | 2 |
| P4 | 4 | 3 |
| P5 | 8 | 2 |

Illustrate the execution of these process using FCFS and preemptive SJF. Calculate average turn around time and average waiting time.

**Solution: 1. FCFS:**




$$\text{Average turn around time} = \frac{(8-1)+(3-0)+(10-2)+(22+3)+(18-2)}{5}$$

$$= \frac{7+3+8+19+16}{5}$$

$$= \frac{53}{5} = 10.6$$

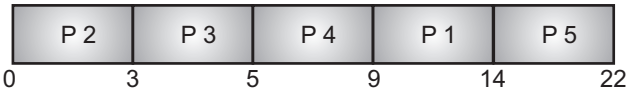
$$\text{Average waiting time} = \frac{(3-1)+(0-0)+(8-2)+(18-3)+(10-2)}{5}$$

$$= \frac{2+0+6+15+8}{5}$$

$$= \frac{31}{5} = 6.2$$

**2. Preemptive SJF:**

**Gantt chart:**




$$\text{Average turn around time} = \frac{(14-1)+(3-0)+(5-2)+(9-3)+(22-2)}{5}$$

$$= \frac{45}{5} = 9$$

$$\text{Average waiting time} = \frac{(9-1)+(0-0)+(3-2)+(5-3)+(14-2)}{5}$$

$$= \frac{23}{5} = 4.6$$

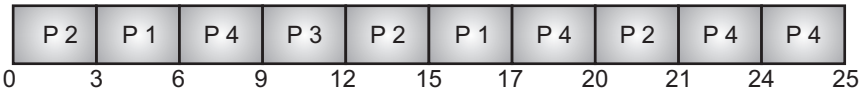
**Problem 8:** Consider the following set of process with CPU burst time given in milliseconds.

| Process | Burst time | Arrival time |
|---|---|---|
| P1 | 1 | 5 |
| P2 | 0 | 7 |
| P3 | 3 | 3 |
| P4 | 2 | 10 |

Illustrate the execution of these process using RR (with time quantum = 3 ms) and shortest remaining time first. Calculate average turn around time and average waiting time.

**Solution: 1. RR CPU Scheduling:**

**Gantt chart:**




$$\text{Average turn around time} = \frac{(17-1)+(21-0)+(12-3)+(25-2)}{4}$$

$$= \frac{16+21+9+23}{4}$$

$$= \frac{69}{4} = 17.25$$

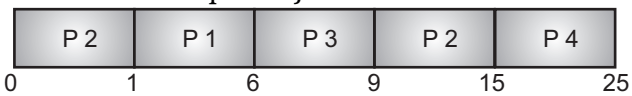
$$\text{Average waiting time} = \frac{(15-3-1)+(20-6-0)+(9-3)+(21-6-2)}{4}$$

$$= \frac{11+14+6+13}{4}$$

$$= \frac{44}{4} = 11$$

**2. SJF or SRTF: (Preemptive)**

**Gantt chart:** Preemptive SJF is also called shortest remaining time first.




$$\text{Average turn around time} = \frac{(6-1)+(15-0)+(9-3)+(25-2)}{4}$$

$$= \frac{5+15+6+23}{4} = \frac{49}{4} = 12.25$$

$$\text{Average waiting time} = \frac{(1-1)+(9-1-0)+(6-3)+(15-2)}{4}$$

$$= \frac{0+8+3+13}{4}$$

$$= \frac{24}{4} = 6$$

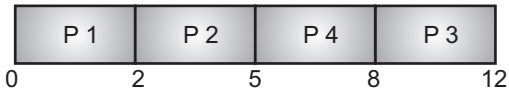
**Problem 9:** Consider the following set of process with CPU burst time given in milliseconds.

| Process | Burst time | Arrival time | Priority |
|---|---|---|---|
| P1 | 2 | 0 | 3 |
| P2 | 3 | 1 | 1 |
| P3 | 4 | 2 | 4 (Highest) |
| P4 | 3 | 3 | 2 |

Illustrate the execution of these process using non-preemptive SJF and non-preemptive priority. Calculate average turn around time and average waiting time.

**Solution: 1. Non-preemptive SJF:**

**Gantt Chart:**




$$\text{Average turn around time} = \frac{(2-0)+(5-1)+(12-2)+(8-3)}{4}$$

$$= \frac{2+4+10+5}{4} = \frac{21}{4} = 5.25$$

$$\text{Average waiting time} = \frac{(0-0)+(2-1)+(8-2)+(5-3)}{4}$$

$$= \frac{9}{4} = 2.25$$

**2. Non-preemptive priority:**

**Gantt chart:**


$$\text{Average turn around time} = \frac{(2-0)+(12-1)+(6-2)+(9-3)}{4}$$

$$= \frac{2+11+4+6}{4} = \frac{23}{4} = 5.75$$

$$\text{Average waiting time} = \frac{(0-0)+(9-1)+(2-2)+(6-3)}{4}$$

$$= \frac{8+0+3}{4}$$

$$= \frac{11}{4}$$

$$= 2.75$$

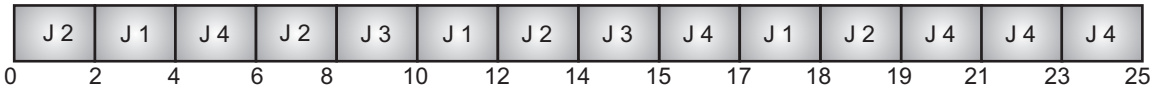
**Problem 10:** Consider the following set of process with CPU burst time given in milliseconds.

| Process | Burst time | Arrival time |
|---|---|---|
| J1 | 1 | 5 |
| J2 | 0 | 7 |
| J3 | 3 | 3 |
| J4 | 2 | 10 |

Compute average time around time using RR (time quantum = 2) and shortest remaining time first. **[Oct. 16]**

**Solution: 1. RR:**

**Gantt chart:**




$$\text{Average turn around time} = \frac{(18-1)+(19-0)+(15-3)+(25-2)}{4}$$

$$= \frac{17+19+12+23}{4} = \frac{71}{4} = 17.75$$

**2. Shortest remaining time first:**

**Gantt chart:**


$$\text{Average turn around time} = \frac{(6-1)+(15-0)+(9-3)+(25-2)}{4}$$

$$= \frac{5+15+6+23}{4}$$

$$= \frac{49}{4} = 12.25$$

---

**Problem 11:** Consider the following snapshot of a system:

| Process | CPU Burst time | Arrival time |
|---|---|---|
| P1 | 5 | 3 |
| P2 | 2 | 0 |
| P3 | 2 | 4 |
| P4 | 3 | 5 |

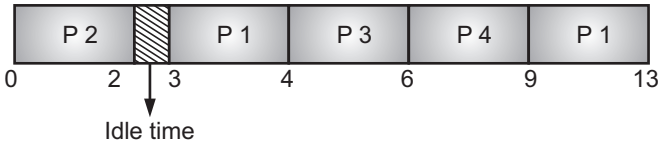
Draw the Gantt chart and find average waiting time for the following scheduling algorithms:

(i) Preemptive SJF.

(ii) Round Robin (time quantum = 2). **[5 M]**

**Solution:**

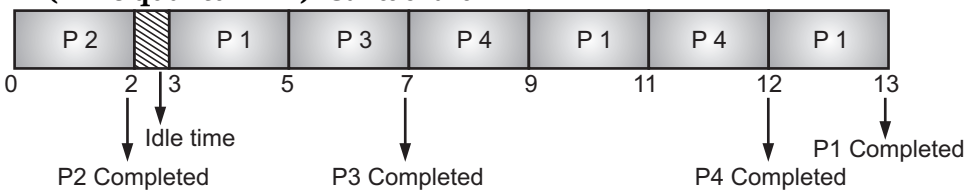
**(i) Preemptive SJF:** Gantt chart




**Waiting time:**

| Process | Waiting time |
|---|---|
| P1 | (3 – 3) + (9 – 4) = 5 + |
| P2 | (0 – 0) = 0 + |
| P3 | (4 – 4) = 0 + |
| P4 | (6 – 5) = 1 |
| Total waiting time | = 6 |
| Average waiting time | = 6/4 = 1.5 |

**(ii) RR (Time quantum = 2):** Gantt chart




**Waiting time:**

| Process | Waiting time |
|---|---|
| P1 | (3 – 3) + (9 – 5) + (12 –11) = 5 + |
| P2 | (0 – 0) = 0 + |
| P3 | (5 – 4) = 1 + |
| P4 | (7 – 5) + (11 – 9) = 4 |
| Total waiting time | = 10 |
| Average waiting time | = 10/14 = 2.5 |

**Problem 12:** Consider the following snapshot of a system:

| Process | Burst time | Arrival time |
|---|---|---|
| $P_1$ | 5 | 1 |
| $P_2$ | 3 | 0 |
| $P_3$ | 2 | 2 |
| $P_4$ | 4 | 3 |
| $P_5$ | 2 | 13 |

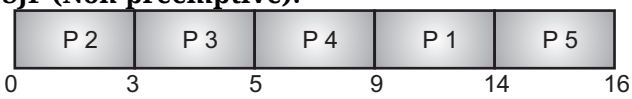
Compute the average turn around time and average waiting time using:

(i) SJF (Non-preemptive).

(ii) Round Robin (Time Quantum = 2). **[5 M]**

**Solution:**

**(i) SJF (Non-preemptive):**




$$\text{Turn around time} = (14 - 3) + (3 - 0) + (5 - 2) + (9 - 3) + (16 - 13)$$

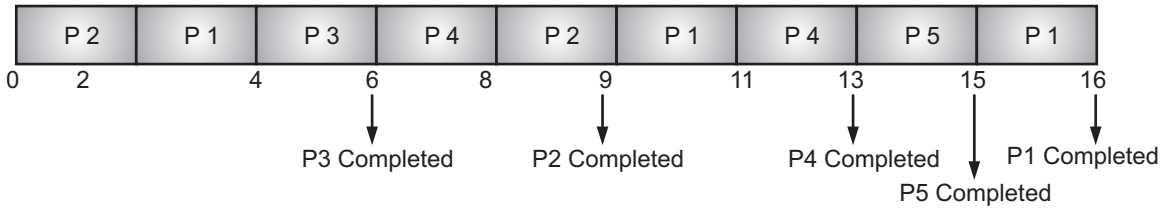
$$\text{Avg. TT} = \frac{(11) + (3) + (3) + (6) + (3)}{5} = \frac{26}{5} = 5.2$$

$$\text{Waiting time} = \frac{(9 - 1) + (0 - 0) + (3 - 2) + (5 - 3) + (14 - 13)}{5}$$

$$= \frac{8 + 1 + 2 + 1}{5}$$

$$\text{Avg. waiting time} = 2.4$$

**(ii) Round Robin (Time quantum = 2):**




$$\text{Average Turn around TT} = \frac{(16 - 1) + (9 - 0) + (6 - 2) + (13 - 3) + (15 - 13)}{5}$$

$$= \frac{15 + 9 + 4 + 10 + 2}{5}$$

$$\text{Average TT} = 8$$

$$\text{Average Waiting Time} = \frac{[(15 - 11) + (9 - 4) + (2 - 1)] + [(8 - 2) + 0] + [4 - 2] + [(11 - 6) + (6 - 3) + [13 - 13]}{5}$$

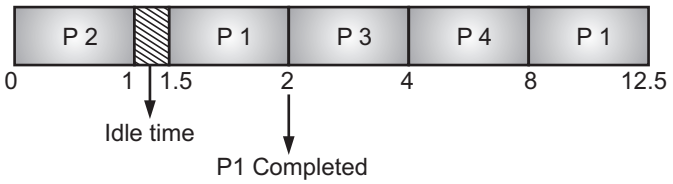
$$\text{Average Waiting Time} = \frac{10 + 6 + 2 + 8 + 0}{5} = 5.2$$

**Problem 13:** Consider the following set of processes, with the length of CPU burst time and arrival time in milliseconds.

| Process | Burst time | Arrival time |
|---|---|---|
| $P_1$ | 5 | 1.5 |
| $P_2$ | 1 | 0 |
| $P_3$ | 2 | 2 |
| $P_4$ | 4 | 3 |

Illustrate the execution of these processes using pre-emptive SJF CPU scheduling algorithm. Calculate average waiting time and average turn around time. Give the contents of Gantt chart.

**Solution: Preemptive SJF:**




$$\text{Average Turn around time} = \frac{(12.5 - 1.5) + (1 - 0) + (4 - 2) + (8 - 3)}{4}$$

$$= \frac{19}{4} = 4.75$$

$$\text{Average waiting time} = \frac{[(8 - 2) + (1.5 - 1.5)] + (0 - 0) + (2 - 2) + (4 - 3)}{4}$$

$$= \frac{7}{4} = 1.75$$

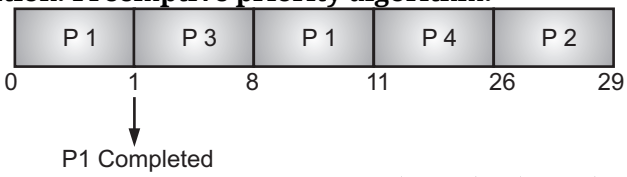
---

**Problem 14:** Consider the following set of processes, with the length of CPU burst time and arrival time in milliseconds.

| Process | Burst time | Arrival time | Priority |
|---|---|---|---|
| P1 | 4 | 0 | 3 |
| P2 | 3 | 2 | 1 (lowest) |
| P3 | 7 | 1 | 4 (highest) |
| P4 | 15 | 3 | 2 |

Illustrate the execution of these processes using pre-emptive priority algorithm. Draw Gantt chart and calculate average turn around time and waiting time.

**Solution: Preemptive priority algorithm:**




$$\text{Average Turnaround time} = \frac{(11 - 0) + (29 - 2) + (8 - 1) + (26 - 3)}{4}$$

$$= \frac{68}{4} = 17$$

$$\text{Average waiting time} = \frac{[(8 - 1) + (0 - 0)] + (26 - 0) + (1 - 1) + (11 - 3)}{4}$$

$$= \frac{41}{4} = 10.25$$

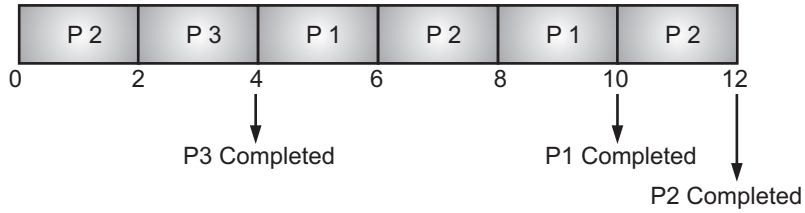
---

**Problem 15:** Consider the following set of processes, with the length of CPU burst time and arrival time in milliseconds.

| Process | Burst time | Arrival time |
|---|---|---|
| P1 | 4 | 2 |
| P2 | 6 | 0 |
| P3 | 2 | 1 |

Illustrate the execution of these processes using Round Robin (RR) CPU scheduling algorithm (quantum = 3 milliseconds). Calculate average waiting time and average turn around time. Give the contents of Gantt chart. **[5 M]**

**Solution: Round Robin Algorithm:**




$$\text{Average turnaround time} = \frac{(10-2)+(12-0)+(4-1)}{3}$$

$$= \frac{23}{3} = 7.6$$

$$\text{Average waiting time} = \frac{[(8-6)+(4-2)] + [(10-8)+(6-2)+0] + (2-1)}{3}$$

$$= \frac{4+6+1}{3} = \frac{11}{3} = 3.66$$

**Problem 16:** Consider the following snapshot of the system.

| Process | Burst time | Priority | Arrival time |
|---|---|---|---|
| P1 | 10 | 3 | 0 |
| P2 | 5 | 0 (high) | 4 |
| P3 | 2 | 1 | 3 |
| P4 | 16 | 2 | 5 |
| P5 | 8 | 4 (low) | 2 |

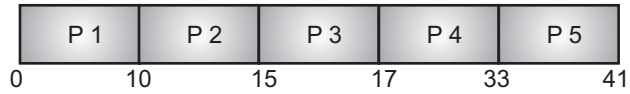
Schedule the above set of processes according to:

(i) Non-preemptive priority scheduling algorithm.

(ii) Preemptive priority scheduling algorithm.

Draw proper Gantt chart and find average turnaround time and waiting time.

**Solution: (i) Non-preemptive priority:**



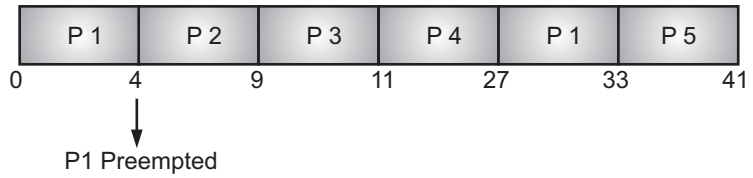

$$\text{Average Turn around time} = \frac{(10-0)+(15-4)+(17-3)+(33-5)+(41-2)}{5}$$

$$= \frac{102}{5} = 20.4$$

$$\text{Average waiting time} = \frac{[(0-0)]+(10-4)+(15-3)+(17-4)+(33-2)}{5}$$

$$= \frac{62}{5} = 12.4$$

**(ii) Preemptive priority scheduling:**




$$\text{Average turnaround time} = \frac{(33-0)+(9-4)+(11-3)+(27-5)+(41-2)}{5}$$

$$= \frac{107}{5} = 21.4$$

$$\text{Average waiting time} = \frac{[(27-4)+(0-0)+(4-4)+(9-3)+(11-5)+(33-2)]}{5}$$

$$= \frac{66}{5} = 13.2$$

**Problem 17:** Consider the following set of processes with the length of CPU burst time and arrival time in milliseconds:

| Process | Burst time | Arrival time | Priority |
|---|---|---|---|
| $P_1$ | 5 | 1 | 1 (h) |
| $P_2$ | 6 | 0 | 2 |
| $P_3$ | 2 | 1 | 1 |
| $P_4$ | 4 | 0 | 3 (*l*) |

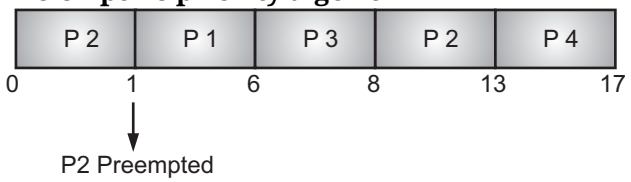
Illustrate the execution of these processes using pre-emptive priority and FCFS scheduling algorithm.

Calculate waiting time and turnaround time for each process and calculate average waiting time and average turnaround time.

Give the contents of Gantt chart.

**Solution:**

**(i) Pre-emptive priority algorithm:**



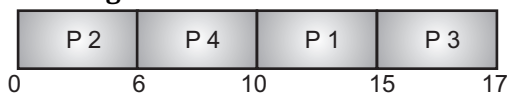

$$\text{Average turnaround time} = \frac{(6-1)+(13-0)+(8-1)+(17-0)}{4}$$

$$= \frac{42}{4} = 10.5$$

$$\text{Average waiting time} = \frac{(1-1)+[(8-1)+0]+(6-1)+(13-0)}{4}$$

$$= \frac{25}{4} = \frac{25}{4} = 6.25$$

**(ii) FCFS algorithm:**




$$\text{Average turnaround time} = \frac{(15-1)+(6-0)+(17-1)+(10-0)}{4}$$

$$= \frac{46}{4} = 11.5$$

$$\text{Average waiting time} = \frac{(10-1)+(0-0)+(15-1)+(6-0)}{4}$$

$$= \frac{29}{4} = 7.25$$

## PRACTICE QUESTIONS

**Q.I Multiple Choice Questions:**

1. Which technique allows OS to allocate a time interval of CPU execution for each process.
   (a) Process scheduling (b) Thread scheduling
   (c) Both (a) and (b) (d) None of the above
2. CPU scheduling is the basis of _____.
   (a) multiprocessor systems
   (b) multiprogramming operating systems
   (c) larger memory sized systems
   (d) none of the mentioned
3. With multiprogramming which is used productively.
   (a) time (b) space
   (c) money (d) all of the mentioned

4. What are the two steps of a process execution?
   (a) I/O and OS Burst (b) CPU and I/O Burst
   (c) Memory and I/O Burst (d) OS and Memory Burst
5. An I/O bound program will typically have,
   (a) a few very short CPU bursts (b) many very short I/O bursts
   (c) many very short CPU bursts (d) a few very short I/O bursts
6. A process is selected from the _____ queue by the _____ scheduler, to be executed.
   (a) blocked, short term (b) wait, long term
   (c) ready, short term (d) ready, long term
7. In the following cases non – preemptive scheduling occurs?
   (a) When a process switches from the running state to the ready state
   (b) When a process goes from the running state to the waiting state
   (c) When a process switches from the waiting state to the ready state
   (d) All of the mentioned
8. The switching of the CPU from one process or thread to another is called as,
   (a) process switch (b) task switch
   (c) context switch (d) all of the mentioned
9. What is dispatch latency?
   (a) the speed of dispatching a process from running to the ready state
   (b) the time of dispatching a process from running to ready state and keeping the CPU idle
   (c) the time to stop one process and start running another one
   (d) none of the mentioned
10. Scheduling is done so as to,
    (a) increase CPU utilization (b) decrease CPU utilization
    (c) keep the CPU more idle (d) none of the mentioned
11. Scheduling is done so as to,
    (a) increase the throughput
    (b) decrease the throughput
    (c) increase the duration of a specific amount of work
    (d) none of the mentioned
12. What is Turnaround time?
    (a) the total waiting time for a process to finish execution
    (b) the total time spent in the ready queue
    (c) the total time spent in the running queue
    (d) the total time from the completion till the submission of a process

13. Scheduling is done so as to,
    (a) increase the turnaround time
    (b) decrease the turnaround time
    (c) keep the turnaround time same
    (d) there is no relation between scheduling and turnaround time
14. What is Waiting time?
    (a) the total time in the blocked and waiting queues
    (b) the total time spent in the ready queue
    (c) the total time spent in the running queue
    (d) the total time from the completion till the submission of a process
15. Scheduling is done so as to,
    (a) increase the waiting time (b) keep the waiting time the same
    (c) decrease the waiting time (d) none of the mentioned
16. What is Response time?
    (a) the total time taken from the submission time till the completion time
    (b) the total time taken from the submission time till the first response is produced
    (c) the total time taken from submission time till the response is output
    (d) none of the mentioned
17. The processes that are residing in main memory and are ready and waiting to execute are kept on a list called as,
    (a) job queue (b) ready queue
    (c) execution queue (d) process queue
18. The interval from the time of submission of a process to the time of completion is termed as,
    (a) waiting time (b) turnaround time
    (c) response time (d) throughput
19. Which scheduling algorithm allocates the CPU first to the process that requests the CPU first?
    (a) first-come, first-served scheduling (b) shortest job scheduling
    (c) priority scheduling (d) none of the mentioned
20. In priority scheduling algorithm,
    (a) CPU is allocated to the process with highest priority
    (b) CPU is allocated to the process with lowest priority
    (c) Equal priority processes cannot be scheduled
    (d) None of the mentioned
21. In priority scheduling algorithm, when a process arrives at the ready queue, its priority is compared with the priority of,
    (a) all process (b) currently running process
    (c) parent process (d) init process

22. Which algorithm is defined in Time quantum?
    (a) shortest job scheduling algorithm
    (b) round robin scheduling algorithm
    (c) priority scheduling algorithm
    (d) multilevel queue scheduling algorithm
23. Process are classified into different groups in,
    (a) shortest job scheduling algorithm
    (b) round robin scheduling algorithm
    (c) priority scheduling algorithm
    (d) multilevel queue scheduling algorithm
24. In multilevel feedback scheduling algorithm,
    (a) a process can move to a different classified ready queue
    (b) classification of ready queue is permanent
    (c) processes are not classified into groups
    (d) none of the mentioned
25. Which one of the following cannot be scheduled by the kernel?
    (a) kernel level thread (b) user level thread
    (c) process (d) none of the mentioned
26. Round robin scheduling falls under the category of,
    (a) Non-preemptive scheduling (b) Preemptive scheduling
    (c) All of the mentioned (d) None of the mentioned
27. With round robin scheduling algorithm in a time shared system,
    (a) using very large time slices converts it into First come First served scheduling algorithm
    (b) using very small time slices converts it into First come First served scheduling algorithm
    (c) using extremely small time slices increases performance
    (d) using very small time slices converts it into Shortest Job First algorithm
28. The portion of the process scheduler in an operating system that dispatches processes is concerned with,
    (a) assigning ready processes to CPU
    (b) assigning ready processes to waiting queue
    (c) assigning running processes to blocked queue
    (d) all of the mentioned
29. What is FIFO algorithm?
    (a) first executes the job that came in last in the queue
    (b) first executes the job that came in first in the queue
    (c) first executes the job that needs minimal processor
    (d) first executes the job that has maximum processor needs

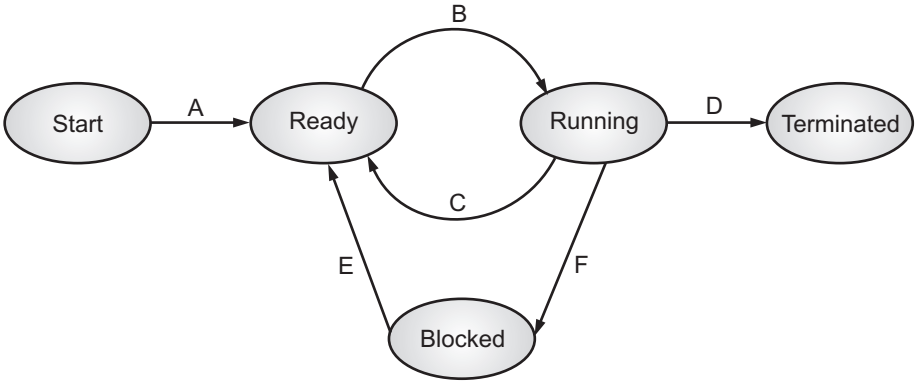
30. The strategy of making processes that are logically runnable to be temporarily suspended is called as,
    (a) Non preemptive scheduling (b) Preemptive scheduling
    (c) Shortest job first (d) First come First served
31. What is scheduling?
    (a) allowing a job to use the processor (b) making proper use of processor
    (c) all of the mentioned (d) none of the mentioned
32. There are 10 different processes running on a workstation. Idle processes are waiting for an input event in the input queue. Busy processes are scheduled with the Round-Robin time sharing method. Which out of the following quantum times is the best value for small response times, if the processes have a short runtime, e.g. less than 10ms?
    (a) tQ = 15ms (b) tQ = 40ms
    (c) tQ = 45ms (d) tQ = 50ms
33. Orders are processed in the sequence they arrive if which rule sequences the jobs.
    (a) earliest due date (b) slack time remaining
    (c) first come, first served (d) critical ratio
34. Which of the following algorithms tends to minimize the process flow time?
    (a) First come First served (b) Shortest Job First
    (c) Earliest Deadline First (d) Longest Job First
35. Under multiprogramming, turnaround time for short jobs is usually _____ and that for long jobs is slightly _____.
    (a) Lengthened; Shortened (b) Shortened; Lengthened
    (c) Shortened; Shortened (d) Shortened; Unchanged
36. Which of the following statements are true? (GATE 2010)
    (i) Shortest remaining time first scheduling may cause starvation
    (ii) Preemptive scheduling may cause starvation
    (iii) Round robin is better than FCFS in terms of response time
    (a) (i) only (b) (i) and (iii) only
    (c) (ii) and (iii) only (d) (i), (ii) and (iii)
37. Which is the most optimal scheduling algorithm?
    (a) FCFS – First come First served (b) SJF – Shortest Job First
    (c) RR – Round Robin (d) None of the mentioned
38. The real difficulty with SJF in short term scheduling is _____.
    (a) it is too good an algorithm
    (b) knowing the length of the next CPU request
    (c) it is too complex to understand
    (d) none of the mentioned

39. The FCFS algorithm is particularly troublesome for _____.
    (a) time sharing systems (b) multiprogramming systems
    (c) multiprocessor systems (d) operating systems
40. Consider the following set of processes, the length of the CPU burst time given in milliseconds.

| Process | Burst time |
|---|---|
| P1 | 6 |
| P2 | 8 |
| P3 | 7 |
| P4 | 3 |

    Assuming the above process being scheduled with the SJF scheduling algorithm.
    (a) The waiting time for process P1 is 3 ms
    (b) The waiting time for process P1 is 0 ms
    (c) The waiting time for process P1 is 16 ms
    (d) The waiting time for process P1 is 9 ms
41. Preemptive Shortest Job First scheduling is sometimes called _____.
    (a) Fast SJF scheduling
    (b) EDF scheduling – Earliest Deadline First
    (c) HRRN scheduling – Highest Response Ratio Next
    (d) SRTN scheduling – Shortest Remaining Time Next
42. An SJF algorithm is simply a priority algorithm where the priority is _____.
    (a) the predicted next CPU burst
    (b) the inverse of the predicted next CPU burst
    (c) the current CPU burst
    (d) anything the user wants
43. Choose one of the disadvantages of the priority scheduling algorithm?
    (a) it schedules in a very complex manner
    (b) its scheduling takes up a lot of time
    (c) it can lead to some low priority process waiting indefinitely for the CPU
    (d) none of the mentioned
44. What is 'Aging'?
    (a) keeping track of cache contents
    (b) keeping track of what pages are currently residing in memory
    (c) keeping track of how many times a given page is referenced
    (d) increasing the priority of jobs to ensure termination in a finite time
45. A solution to the problem of indefinite blockage of low – priority processes is,
    (a) Starvation (b) Wait queue
    (c) Ready queue (d) Aging

46. Which of the following statements are true? **(GATE 2010)**
    (i) Shortest remaining time first scheduling may cause starvation
    (ii) Preemptive scheduling may cause starvation
    (iii) Round robin is better than FCFS in terms of response time
    (a) (i) only (b) (i) and (iii) only
    (c) (ii) and (iii) only (d) (i), (ii) and (iii)
47. Which of the following scheduling algorithms gives minimum average waiting time?
    (a) FCFS (b) SJF
    (c) Round – robin (d) Priority
48. Which of the following process scheduling algorithm may lead to starvation,
    (a) FIFO (b) Round Robin
    (c) Shortest Job Next (d) None of the above
49. If the quantum time of round robin algorithm is very large, then it is equivalent to:
    (a) First in first out (b) Shortest Job Next
    (c) Lottery scheduling (d) None of the above
50. A scheduling algorithm assigns priority proportional to the waiting time of a process. Every process starts with priority zero (the lowest priority). The scheduler re-evaluates the process priorities every T time units and decides the next process to schedule. Which one of the following is TRUE if the processes have no I/O operations and all arrive at time zero?
    (a) This algorithm is equivalent to the first-come-first-serve algorithm
    (b) This algorithm is equivalent to the round-robin algorithm.
    (c) This algorithm is equivalent to the shortest-job-first algorithm..
    (d) This algorithm is equivalent to the shortest-remaining-time-first algorithm
51. Which of the following statements are true?
    (i) Shortest remaining time first scheduling may cause starvation
    (ii) Preemptive scheduling may cause starvation
    (iii) Round robin is better than FCFS in terms of response time
    (a) (i) only (b) (i) and (iii) only
    (c) (ii) and (iii) only (d) (i), (ii) and (iii)
52. In the following process state transition diagram for a uniprocessor system, assume that there are always some processes in the ready state: Now consider the following statements:



(i) If a process makes a transition D, it would result in another process making transition A immediately.

(ii) A process P2 in blocked state can make transition E while another process P1 is in running state.

(iii) The OS uses preemptive scheduling.

(iv) The OS uses non-preemptive scheduling.

Which of the above statements are TRUE?

(a) (i) and (ii) (b) (i) and (iii)

(c) (ii) and (iii) (d) (ii) and (iv)

53. Consider a set of n tasks with known runtimes $r_1$, $r_2$, .... $r_n$ to be run on auniprocessor machine. Which of the following processor scheduling algorithms will result in the maximum throughput?

(a) Round-Robin (b) Shortest-Job-First

(c) Highest-Response-Ratio-Next (d) First-Come-First-Served

54. The maximum number of processes that can be in Ready state for a computer system with n CPUs is,

(a) n (b) $n^2$

(c) $2^n$ (d) Independent of n

55. Consider an arbitrary set of CPU-bound processes with unequal CPU burst lengths submitted at the same time to a computer system. Which one of the following process scheduling algorithms would minimize the average waiting time in the ready queue?

(a) Shortest remaining time first

(b) Round-robin with time quantum less than the shortest CPU burst

(c) Uniform random

(d) Highest priority first with priority proportional to CPU burst length

56. Which module gives control of the CPU to the process selected by the short – term scheduler?

(a) Dispatcher (b) Interrupt

(c) Scheduler (d) Threading

57. Which of the following is not an optimization criterion in the design of a CPU scheduling algorithm?

(a) Minimum CPU utilization (b) Maximum throughput

(c) Minimum turnaround time (d) Minimum waiting time

58. A scheduling Algorithm assigns priority proportional to the waiting time of a process. Every process starts with priority zero (lowest priority). The scheduler reevaluates the process priority for every 'T' time units and decides next process to be scheduled. If the process have no I/O operations and all arrive at time zero, then the scheduler implements _____ criteria.

(a) Priority scheduling (b) Round Robin Scheduling

(c) Shortest Job First (d) FCFS

59. Which of the following statements is not true for Multi Level Feedback Queue processor scheduling algorithm?
    (a) Queues have different priorities
    (b) Each queue may have different scheduling algorithm
    (c) Processes are permanently assigned to a queue
    (d) This algorithm can be configured to match a specific system under design
60. In which of the following scheduling criteria, context switching will never take place?
    (a) Round Robin (b) Preemptive SJF
    (c) Non-preemptive SJF (d) Preemptive priority
61. In a multi-user operating system, 30 requests are made to use a particular resource per hour, on an average. The probability that no requests are made in 40 minutes, when arrival pattern is a Poisson distribution, is _____.
    (a) $e^{-15}$ (b) $1 - e^{-15}$
    (c) $1 - e^{-20}$ (d) $e^{-20}$
62. The algorithm uses by scheduler is called as,
    (a) dispatcher algorithm (b) scheduler algorithm
    (c) Both (a) & (b) (d) None of the Above
63. Which scheduling is the process of selecting a process and allocating the processor to the selected process for execution?
    (a) CPU (b) Process
    (c) Both (a) & (b) (d) None of the Above

## Answers

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1. (a) | 2. (b) | 3. (a) | 4. (b) | 5. (c) | 6. (c) | 7. (b) | 8. (d) | 9. (c) | 10. (a) |
| 11. (a) | 12. (d) | 13. (b) | 14. (b) | 15. (c) | 16. (b) | 17. (b) | 18. (b) | 19. (a) | 20. (a) |
| 21. (b) | 22. (b) | 23. (d) | 24. (a) | 25. (b) | 26. (b) | 27. (a) | 28. (a) | 29. (b) | 30. (b) |
| 31. (a) | 32. (a) | 33. (c) | 34. (b) | 35. (b) | 36. (d) | 37. (b) | 38. (b) | 39. (b) | 40. (a) |
| 41. (d) | 42. (a) | 43. (c) | 44. (d) | 45. (d) | 46. (d) | 47. (b) | 48. (c) | 49. (a) | 50. (b) |
| 51. (d) | 52. (c) | 53. (b) | 54. (d) | 55. (a) | 56. (a) | 57. (a) | 58. (b) | 59. (c) | 60. (c) |
| 61. (d) | 62. (b) | 63. (a) | | | | | | | |

**Q.II Fill in the Blanks:**

1. _____ is a set of policies and mechanisms which controls the order in which the work to be done is completed.
2. The scheduling program which is a system software concerned with scheduling is called the _____ and the algorithm it uses is called the _____ algorithm.
3. A scheduling algorithm should be designed so that CPU remains _____ as possible. It should make efficient use of CPU.
4. _____ is the amount of work completed in a unit of time.

5. _____ time is the time taken to start responding to the request.
6. In _____ scheduling the tasks are usually assigned with priorities.
7. Examples of preemptive scheduling include _____ rescheduling,
8. The process of allocating CPU time from one running process to another process is called _____ scheduling
9. Under _____ 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 waiting state.
10. Example of non-preemptive scheduling includes _____.
11. _____ is the module that gives control of the CPU to the process selected by the short-term scheduler.
12. The selecting of a process from among the _____ processes in the memory and allocating the CPU to the selected process for execution is called CPU scheduling.
13. Dispatch _____ is the time it takes for the dispatcher to stop one process and start another running.
14. In a _____ queue-scheduling algorithm, processes are permanently assigned to a queue on entry to the system.
15. Multilevel _____ queue scheduling, however, allows a process to move between queues. The idea is to separate processes with different CPU-burst characteristics.
16. The execution of process consists of a cycle of _____ burst and _____ burst.
17. In _____ jobs are executed on first come, first serve basis.
18. _____ scheduling is a method of scheduling processes that is based on priority.
19. _____ is also known as shortest job first, or SJF.
20. A process scheduler schedules different processes to be assigned to the CPU based on particular scheduling _____.
21. CPU burst is the amount of time, a process uses the CPU until it _____ waiting for some input or interrupted by some other process while I/O burst is the amount of time, a process _____ for input-output before needing CPU time.
22. In Round Robin algorithm each process is provided a fix time to execute, it is called a _____ quantum.
23. The _____ scheduling is based on priority where a scheduler may preempt a low priority running process anytime when a high priority process enters into a ready state.

## Answers

| | | | |
|---|---|---|---|
| 1. Scheduling | 2. scheduler, scheduling | 3. busy | 4. Throughput |
| 5. Response | 6. preemptive | 7. Round Robin | 8. CPU |
| 9. non-preemptive | 10. SJF | 11. Dispatcher | 12. ready |
| 13. latency | 14. multilevel | 15. feedback | 16. CPU, I/O |
| 17. FCFS | 18. Priority | 19. SJN | 20. algorithms |
| 21. starts, waits | 22. time | 23. preemptive | |

**Q.III State True or False:**

1. Process scheduling is a process of determining which process will own CPU for execution while another process is on hold.
2. The main task of CPU/process scheduling is to make sure that whenever the CPU remains idle, the OS at least select one of the processes available in the ready queue for execution.
3. The selection process in process scheduling will be carried out by the CPU scheduler who selects one of the processes in memory that are ready for execution.
4. Response time refers to the time between the moment of submission of a job/ process and the time of its completion.
5. Burst time/execution time is a time required by the process to complete execution, also called running time.
6. CPU/IO burst cycle characterizes process execution, which alternates between CPU and I/O activity.
7. Scheduler is a module that provides control of the CPU to the process.
8. In preemptive scheduling, whenever the high priority process wants to execute, then the running low priority process can be preempted and processor is assigned to that high priority process.
9. Multi-level feedback queue scheduling is an enhancement of MLQ.
10. Dispatcher is the module which gives the control of CPU to the process that has been selected by the short-term scheduler.
11. Dispatch latency is defined as the time taken by the dispatcher to stop one process and start running the another process.
12. Priority scheduling is a method of scheduling processes based on priority.
13. FCFS is a scheduling algorithm in which the process with the shortest execution time should be selected for execution next.
14. Multiple-level queues scheduling algorithm, processes are assigned to a queue based on a specific property of the process, like the process priority, size of the memory, etc.
15. Waiting time is the time a job waits for resource allocation when several jobs are competing in multiprogramming system..
16. Preemptive scheduling is used when a process switches from running state to ready state or from waiting state to ready state.
17. Preemptive Scheduling is used when a process terminates, or a process switches from running to waiting state. In this scheduling, once the resources (CPU cycles) is allocated to a process, the process holds the CPU till it gets terminated or it reaches a waiting state.

18. Dispatcher is a module that gives control of CPU to the process selected by short term scheduler.
19. The main function of the dispatcher is switching, it means switching the CPU from one process to another process.
20. In a multilevel queue-scheduling algorithm, processes are permanently assigned to a queue on entry to the system.
21. Multilevel feedback queue scheduling, however, allows a process to move between queues and the idea is to separate processes with different CPU-burst characteristics. First Come First Serve (FCFS) implementation is based on FIFO queue.
22. Priority scheduling is a non-preemptive algorithm and one of the most common scheduling algorithms in batch systems.
23. SRT is the preemptive version of the SJN algorithm.
24. The success of CPU scheduling depends on the process execution is cycle of CPU execution and I/O wait.
25. CPU scheduler or short-term scheduler selects the process from ready queue allocates to CPU for execution.
26. In FCFS scheduling, the process that requests the CPU first, is allocated the CPU first. Thus, the name First-Come-First-Served.
27. In SRT scheduling, processes are dispatched in FIFO but are given a small amount of CPU time known as time quantum or time slice.
28. In SJF, the process with the least estimated execution time is selected from the ready queue for execution. It associates with each process, the length of its next CPU burst. SJF algorithm can be preemptive or non-preemptive.
29. In priority scheduling, a priority is associated with all processes. The processes are executed in sequence according to their priority.
30. In RR scheduling, CPU is allocated to the process with highest priority. Priority scheduling can be preemptive or non-preemptive.

## Answers

| | | | | |
|---|---|---|---|---|
| 1. (T) | 2. (T) | 3. (T) | 4. (F) | 5. (T) |
| 6. (F) | 7. (T) | 8. (T) | 9. (T) | 10. (T) |
| 11. (T) | 12. (F) | 13. (T) | 14. (T) | 15. (T) |
| 16. (T) | 17. (F) | 18. (T) | 19. (T) | 20. (T) |
| 21. (T) | 22. (T) | 23. (T) | 24. (T) | 25. (T) |
| 26. (T) | 27. (F) | 28. (T) | 29. (F) | 30. (T) |

**Q.IV Answer the following Questions:**

**(A) Short Answer Questions:**

1. What is process scheduling?
2. What is process scheduler?
3. What is CPU-I/O burst cycle?

4. Define dispatcher.
5. Enlist scheduling criteria.
6. Define response time and turnaround time.
7. What is preemptive scheduling and non-preemptive scheduling?
8. What is the purpose of scheduling algorithm?
9. Enlist various scheduling algorithms.
10. What is FCFS?
11. What is SJF?
12. Define multiple queue scheduling.

**(A) Long Answer Questions:**

1. Define process scheduling. Which criteria followed by process scheduling?
2. With the help of diagram describe CPU-I/O burst cycle.
3. Explain the CPU scheduling algorithms with example:
   (i) RR, (ii) SJF, (iii) FCFS.
4. Describe the term dispatcher in detail.
5. What is meant by Multilevel Queue (MLQ) scheduling? Explain with diagrammatically.
6. Define the following terms:
   (i) CPU utilization,
   (ii) Response time,
   (iii) Turnaround time.
7. Compare FCFS and SJF.
8. Define CPU and I/O bound processes in detail.
9. Describe FCFS scheduling with example. Also state its advantages and disadvantages.
10. What is priority scheduling? Explain with example. State its advantages and disadvantages.
11. Write short note on: Multilevel feedback queue scheduling.
12. What is pre-emptive and non-preemptive scheduling? Compare them.
13. Consider the following snapshot of a system.

| Job | Arrival Time | CPU Burst Time |
|---|---|---|
| 1 | 0 | 7 |
| 2 | 1 | 2 |
| 3 | 2 | 5 |
| 4 | 3 | 4 |

Compute turnaround time using RR with quantum 3 and SJF (non-pre-emptive).

14. Consider following snapshot of the system.

| Job | Arrival Time | CPU Burst Time |
|---|---|---|
| 1 | 0.0 | 8 |
| 3 | 0.5 | 5 |
| 3 | 1.0 | 2 |

Compute average turn around time and average wait time using (i) FCFS, (ii) SJF (pre-emptive).

15. Assume that the following jobs are to be executed with one processor.

| Job | Burst time | Priority | Arrival time |
|---|---|---|---|
| 1 | 5 | 2 | 0 |
| 2 | 2 | 4 (lowest) | 3 |
| 3 | 1 | 3 | 3 |
| 4 | 3 | 1 (highest) | 4 |

Give the turn around time for each job using

(i) Pre-emptive shortest job first algorithm.

(ii) Non-pre-emptive priority algorithm.

16. Assume that you have the following jobs to be executed with one processor.

| Job | Arrival Time | Burst Time |
|---|---|---|
| 1 | 0 | 12 |
| 2 | 2 | 3 |
| 3 | 5 | 8 |
| 4 | 5 | 5 |
| 5 | 7 | 6 |

Calculate the average turnaround fine and total wait time using pre-emptive SJF scheduling algorithm.

17. Consider the following set of processes, with the length of the CPU-burst time given in milliseconds:

| Process | Burst Time | Priority |
|---|---|---|
| $P_1$ | 10 | 3 |
| $P_2$ | 1 | 1 |
| $P_3$ | 2 | 3 |
| $P_4$ | 1 | 4 |
| $P_5$ | 5 | 2 |

The processes are assumed to have arrived in the order $P_1$, $P_2$, $P_3$, $P_4$, $P_5$ all at time 0.

(i) Draw four Gantt charts illustrating the execution of these processes using FCFS, SJF, a non-pre-emptive (a smaller priority number implies a higher priority), and RR (quantum = 1) scheduling.

(ii) What is the turnaround time of each process for each of the scheduling algorithms in part a?

(iii) What is the waiting time of each process for each of the scheduling algorithms in part a?

(iv) Which of the schedules in part a results in the minimal average waiting time (over all processes).

18. Consider the following set of process with CPU burst time given in milliseconds.

| Process | Burst time | Arrival time |
|---|---|---|
| P1 | 1 | 5 |
| P2 | 0 | 7 |
| P3 | 3 | 3 |
| P4 | 2 | 10 |

Illustrate average time around time using

(i) RR (time quantum = 3)

(ii) Shortest remaining time first.

Calculate Average Time Around Time.

## UNIVERSITY QUESTIONS AND ANSWERS

**April 2016**

**1.** What will happen if all process is CPU bound in system? **[1 M]**

**Ans.** Refer to Section 3.1.1.

**2.** Define dispatch latency. **[1 M]**

**Ans.** Refer to Section 3.1.5.

**3.** Define turnaround time. **[1 M]**

**Ans.** Refer to Section 3.1.2 Point (3).

**4.** Consider the following set of processes, with the length of CPU burst time and arrival time in milliseconds:

| Process | Burst Time | Arrival Time |
|---|---|---|
| P1 | 4 | 2 |
| P2 | 6 | 0 |
| P3 | 2 | 1 |

Illustrate the execution of these processes using Round Robin (RR) CPU scheduling algorithm (quantum = 3 milliseconds). Calculate average waiting time and average turn around time. Give the contents of Gantt chart. **[5 M]**

**Ans.** Refer to Section 3.2.4.

**5.** State and explain criteria for computing various scheduling algorithms. **[4 M]**

**Ans.** Refer to Section 3.1.2.

**October 2016**

**1.** State functions of dispatcher. **[1 M]**

**Ans.** Refer to Section 3.1.5.

**2.** Consider the following set of processing with CPU burst time given in milliseconds.

| Process | Burst Time | Arrival Time |
|---|---|---|
| $J_1$ | 1 | 5 |
| $J_2$ | 0 | 7 |
| $J_3$ | 3 | 3 |
| $J_4$ | 2 | 10 |

Compute average time around time using RR (time quantum = 2) and shortest remaining time first. **[5 M]**

**Ans.** Refer to Section 3.2.4.

**3.** Define pre-emptive and non-preemptive scheduling. State disadvantages of pre-emptive scheduling. **[4 M]**

**Ans.** Refer to Sections 3.1.4.1 and 3.1.4.2.

**April 2017**

**1.** Consider the following snapshot of the system:

| Process | Burst Time | Arrival Time | Priority |
|---|---|---|---|
| $P_1$ | 10 | 0 | 3 |
| $P_2$ | 5 | 4 | 0 (high) |
| $P_3$ | 2 | 3 | 1 |
| $P_4$ | 16 | 5 | 2 |
| $P_5$ | 8 | 2 | 4 (low) |

Schedule the above set of processes according to:

(i) Non-preemptive priority algorithm.

(ii) FCFS.

Draw proper Gantt chart and find average turn around time and average waiting time. **[5 M]**

**Ans.** Refer to Sections 3.2.3 and 3.2.1.

**2.** Explain multilevel feedback queue scheduling with diagram. **[4 M]**

**Ans.** Refer to Section 3.2.7.

**October 2017**

**1.** What will happen if all processes are I/O bound in system? **[1 M]**

**Ans.** Refer to Section 3.1.1.

**2.** Give any four criteria for computing various scheduling algorithms. **[1 M]**

**Ans.** Refer to Section 3.1.2.

**3.** Consider the following set of processes CPU time given in milliseconds:

| Process | Burst Time | Arrival Time |
|---|---|---|
| $P_1$ | 5 | 1 |
| $P_2$ | 3 | 0 |
| $P_3$ | 2 | 2 |
| $P_4$ | 4 | 3 |
| $P_5$ | 8 | 2 |

Illustrate the execution of these process using FCFS and pre-emptive SJF CPU scheduling algorithm. **[5 M]**

**Ans.** Refer to Sections 3.2.1 and 3.2.2.

**4.** Explain multilevel queue scheduling with diagram. **[4 M]**

**Ans.** Refer to Section 3.2.5.

**April 2018**

**1.** What is dispatch latency? **[1 M]**

**Ans.** Refer to Section 3.1.5.

**2.** Write any two disadvantages of priority scheduling. **[1 M]**

**Ans.** Refer to Section 3.2.3.

**3.** Define I/O bound process. **[1 M]**

**Ans.** Refer to Section 3.1.1.

**4.** Consider the following set of processes with the length of CPU burst time and arrival time in milliseconds:

| Process | Burst Time | Arrival Time |
|---|---|---|
| $P_1$ | 0 | 8 |
| $P_2$ | 1 | 4 |
| $P_3$ | 2 | 9 |
| $P_4$ | 3 | 5 |

Illustrate the execution of these processes using pre-emptive SJF (Shortest Job First) CPU scheduling algorithm. Calculate average waiting time and average turn around time. Give the contents of Gantt Chart. **[5 M]**

**Ans.** Refer to Section 3.2.2.

**5.** Write a note on multilevel queue scheduling. **[4 M]**

**Ans.** Refer to Section 3.2.5.

**October 2018**

**1.** Define turn around time. **[1 M]**

**Ans.** Refer to Section 3.1.2 Point (3).

**2.** Consider the following set of processes with the length of CPU burst time and arrival time in milliseconds:

| Process | Burst Time | Arrival Time |
|---|---|---|
| $P_1$ | 5 | 1 |
| $P_2$ | 6 | 0 |
| $P_3$ | 2 | 1 |
| $P_4$ | 4 | 0 |

Illustrate the execution of these processes using pre-emptive round robin CPU scheduling algorithm. Also calculate wait time, turn around time of each process and calculate average waiting time and average turn around time for above situation. Also draw Gantt Chart. Consider the time quantum 3. **[5 M]**

**Ans.** Refer to Section 3.2.4.

**April 2019**

**1.** "Priority scheduling suffers from starvation", True / False Justify. **[1 M]**

**Ans.** Refer to Section 3.2.3.

**2.** Consider the following set of processes with the length of CPU burst time and arrival time in milliseconds:

| Process | Burst Time | Arrival Time |
|---|---|---|
| $P_1$ | 4 | 1 |
| $P_2$ | 2 | 0 |
| $P_3$ | 1 | 2 |
| $P_4$ | 3 | 3 |
| $P_5$ | 3 | 10 |

Compute average turn around time and average wait time using Round Robin (RR) CPU scheduling algorithm with Time Quantum = 2, also draw the Gantt chart. **[5 M]**

**Ans.** Refer to Section 3.2.4.

**3.** What is dispatch latency time? Write the functions performed by dispatcher. **[4 M]**

**Ans.** Refer to Section 3.1.5.

■■■

CHAPTER 4

# Synchronization

## *Objectives ...*

- To learn Concept of Process Synchronization
- To understand Critical-Section Problem
- To study Classical Synchronization Problems

## 4.0 INTRODUCTION

- In the Operating System (OS), there are a number of processes present in a particular state. At the same time, we have a limited amount of resources present, so those resources need to be shared among various processes.
- Process synchronization is needed when multiple processes execute concurrently sharing same system resources which results inconsistency of data and to avoid the inconsistent results.
- Consider an example in Fig. 4.1, process A changing the data in a memory location while another process B is trying to read the data from the same memory location.
- In this situation the both processes A and B access to the same shared resource or data at the same time can be lead to the inconsistency of shared data.
- To avoid this type of inconsistency of data, the processes need to be synchronized with each other.

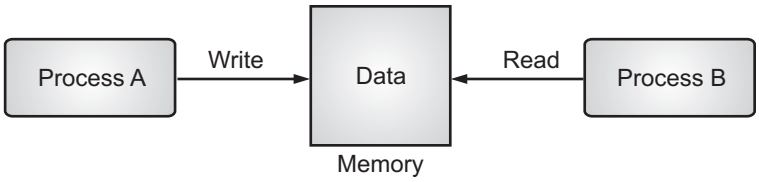



**Fig. 4.1**

- In this chapter we will discuss synchronization in operating system in detail.

## 4.1 BACKGROUND

- Concurrent execution of the process means simultaneous execution of multiple processes. Concurrent access often need access to shared data and shared resources.

- If there is no controlled access to shared data, it may result in data inconsistency. Maintaining data consistency requires synchronization mechanism to ensure the orderly execution of co-operating processes.
- Synchronization or process synchronization is the task of coordinating the execution of processes in a way that no two processes can have access to the same shared data and resources.
- The processes that coexist in the memory at same time are called concurrent processes. The concurrent processes may either be independent or cooperative.
- On the basis of synchronization, processes are categorized as one of the following two types:
  1. **Independent Process:** Execution of one process does not affect the execution of other processes.
  2. **Cooperative Process:** Execution of one process affects the execution of other processes.
- Process synchronization problem arises in the case of Cooperative process also because resources are shared in Cooperative processes.

**Race Condition:** **[April 16, 17, 18]**

- Race condition is a situation where multiple processes access and manipulate the same data concurrently and the outcome of the execution depends on the order in which the instructions execute.
- Consider a simple example as shown in Fig. 4.2. Assume that two processes are being run, each process occasionally requests that line be printed on the single printer.
- Depending on scheduling of Process 1 or Process 2, all the printout of Process 1 may precede or follow the printout of Process 2.
- But most likely the printout of each will be interspersed on the printer paper if they are interleaved.

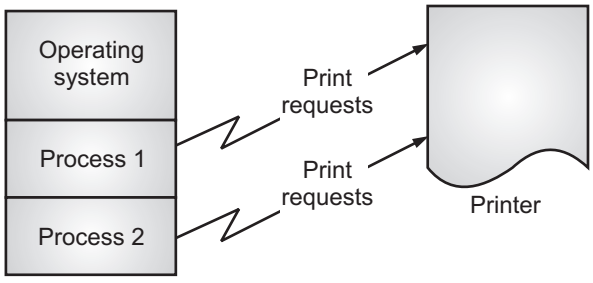



**Fig. 4.2: Simple Race Condition**

- In race condition several processes access and process the manipulations over the same data concurrently, and then the outcome depends on the particular order in which the access takes place.

- To guard against the race condition, we need to ensure that only one process at a time can be manipulating the common resource. To obtain this, we need process synchronization.
- The process which completes its task should release the resource and the process which requires the resources and resources are not available (currently in use) would automatically switch to wait state.

## 4.2 CRITICAL SECTION PROBLEM [April 16, 17, 18, Oct. 17, 18]

- To avoid race conditions and results, one must identify codes in critical sections in each process.
- Critical section is a code segment that can be accessed by only one process at a time. Critical section contains shared variables which need to be synchronized to maintain consistency of data variables.
- Consider a system consisting of n processes $\{P_0, P_1, \ldots, P_{n-1}\}$. 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.
- The important feature of the system is that, when one process is executing in its critical section, no other process is to be allowed to execute in its critical section. That is, no two processes are executing in their critical sections at the same time.
- The critical-section problem is to design a protocol that the processes can use to cooperate. Each process must request permission to enter its critical section. The section of code implementing this request is the entry section.
- The critical section may be followed by an exit section. The remaining code is the remainder section.
- The general structure of a typical process $P_i$ is shown in Fig. 4.3. The entry section and exit section are enclosed in boxes to highlight these important segments of code.

```
do
  {
  Entry Section
     critical section
  Exit Section
     remainder section
  } while (True);
```

**Fig. 4.3: General Structure of a Typical Process $P_i$**

- The critical section can be defined by following three sections:
  1. **Entry Section:** Each process must request permission to enter its critical section; processes may go to critical section through entry section.

2. **Exit Section:** Each critical section is terminated by exit section.
3. **Remainder Section:** The code after exit section is remainder section.

- A situation, where several processes execute concurrently sharing some data and the result of execution depends on the particular order in which the access to shared data takes place is called race condition.
- The shared data (resource) is called critical resource and the portion of the program, that uses this critical resource, is called critical section.
- In order to synchronize the cooperative processes, the main task is to solve the critical section problem. A solution to the critical-section problem must satisfy the following three requirements:
  1. **Mutual exclusion:** If a process is executing in its critical section, then no other process is allowed to execute in the critical section.
  2. **Progress:** If no process is executing in the critical section and other processes are waiting outside the critical section, then only those processes that are not executing in their remainder section can participate in deciding which will enter in the critical section next, and the selection cannot be postponed indefinitely.
  3. **Bounded waiting:** A bound must exist 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.
- In process synchronization, critical section plays the main role so that the problem must be solved. Following are some widely used methods to solve the critical section problem.

**Peterson Solution (For Two Processes):**

- Peterson's solution is widely used solution to critical section problems. This algorithm was developed by a computer scientist Peterson that's why it is named as a Peterson's solution.
- In Peterson solution, when a process is executing in a critical state, then the other process only executes the rest of the code, and the opposite can happen.
- This method also helps to make sure that only a single process runs in the critical section at a specific time.
- Peterson solution preserves all three conditions:
  1. **Mutual Exclusion** is comforted as at any time only one process can access the critical section. **[April 19]**
  2. **Progress** is also comforted, as a process that is outside the critical section is unable to block other processes from entering into the critical section.
  3. **Bounded Waiting** is assured as every process gets a fair chance to enter the Critical section.
- There are two processes $P_0$ and $P_1$. We can also assume that first process is $P_i$ and the other process is $P_j$.

**Algorithm 1:**

- In the first algorithm all processes share the common integer variable 'turn' initialize as 0 or 1. If 'turn=i', then process $P_i$ is allowed to execute in its critical section.
- This algorithm ensures that only one process at a time can be in critical section, but does not satisfy the progress condition.
- The structure of process $P_i$ is shown as in Fig. 4.4 (a).

```
do
{
   while(turn!=i)
   critical section
   turn=j;
   remainder section
} while (1);
```

**Fig. 4.4 (a)**

**Algorithm 2:**

- The problem of algorithm-1 is that it does not retain sufficient information about the state of each process, it remember only which process is allowed to enter its critical section.
- To remove this problem, we can replace the variable turn with the array as "Boolean flag". The elements of the array are initialized to false. If flag[i] is true, this value indicates that $P_i$ is ready to enter the critical section.
- This algorithm ensures mutual exclusion condition, but does not satisfy the progress condition. In this case process Pi is shown as in Fig. 4.4 (b).
- Process $P_i$ set flag[i] to be true, signaling that it is ready to enter its critical section, then $P_i$ checks to verify that $P_j$ is not also ready to enter its critical section.
- If $P_j$ is ready, then $P_i$ would wait until $P_j$ had indicated that it no longer needed to be in the critical section i.e. until flag[j] was false.
- At this point, $P_i$ would enter the critical section. On exiting critical section $P_i$ would set flag[i] to be false, allow the other process to enter its critical section.

```
do
{
   flag[i]=true;
   while(flag[j])
   critical section
   flag[i]=false;
   remainder section
} while(1);
```

**Fig. 4.4 (b)**

**Algorithm 3:**

- In this algorithm we combine the idea of both above algorithm where all critical section requirements are met. Each process shares two variables as:

  Boolean flag[2]; int turn;.
- Initially flag[0]=flag[1]=false and the value of turn is 0 or 1. This algorithm ensures all critical section requirements. In this case process $P_i$ is shown as in Fig. 4.4 (c). Process $P_i$ can enter to critical section if flag[i] is true and turn is equal to i.

```
do
{
    flag[i]=true;
    turn=j;
    while(flag[j]&&turn=j)
    critical section
    flag[i]=false;
    remainder section
} while(1);
```

**Fig. 4.4 (c)**

**Bakery Algorithm Solution (For Multiple Processes):**

- The Bakery algorithm is one of the simplest known solutions to the mutual exclusion problem for the general case of n (multiple) process.
- The Bakery algorithm is a critical section solution for n processes. The Bakery algorithm preserves the First Come First Serve (FCFS) property.
- This uses the concept which is used in bakeries to serve customers. When a customer enters in a bakery, he/she is given a token in ascending order of number.
- While providing the service, the customer with smallest token number is served first.
  1. It is generalization for n processes.
  2. Each process has an id. Process ids are ordered.
  3. Before entering its critical section, a process receives a number. The holder of the smallest number enters its critical section.
- The common data structures are used in Bakery algorithm:

  Boolean choosing [n];

  int number [n];
- Initially, above data structures are initialized to false and 0 respectively. For this we define the following notation:

  (a, b) < (c, d) if a < c or if a = c and b < d

  max($a_0$, . . ., $a_{n-1}$) is a number, k, such that k >= $a_i$ for i = 0, . . ., n - 1

- The structure for process $P_i$ is as follows:

```
repeat
   choosing[i] := true;
   number[i] := max(number[0], number[1], …, number[n - 1])+1;
   choosing[i] := false;
   for j := 0 to n - 1
      do begin
         while choosing[j] do no-op;
         while number[j] != 0
            and (number[j], j) < (number[i], i) do no-op;
      end;
      critical section

   number[i] := 0;
         remainder section
until false;
```

**Explanation:**

- Firstly the process sets its "choosing" variable to be TRUE indicating its intent to enter critical section. Then it gets assigned the highest ticket number corresponding to other processes.
- Then the "choosing" variable is set to FALSE indicating that it now has a new ticket number. This is in-fact the most important and confusing part of the algorithm.
- It is actually a small critical section in itself! The very purpose of the first three lines is that if a process is modifying its TICKET value then at that time some other process should not be allowed to check its old ticket value which is now obsolete.
- This is why inside the for loop before checking ticket value we first make sure that all other processes have the "choosing" variable as FALSE.
- After that we proceed to check the ticket values of processes where process with least ticket number/process id gets inside the critical section. The exit section just resets the ticket value to zero.

**Synchronization Hardware Solution:**

- Sometimes the problems of the critical section are also resolved by hardware. Some operating system offers lock functionality where a process acquires a lock when entering the critical section and releases the lock after leaving it.
- So when another process is trying to enter the critical section, it will not be able to enter as it is locked. It can only do so if it is free by acquiring the lock itself.
- In this topic we present some hardware instructions that are available on many systems and show how they can be used effectively in solving the critical section problem.

- TestAndSet is a hardware solution to the synchronization problem. In TestAndSet, we have a shared lock variable which can take either of the two values, 0 (unlock) or 1 (lock).
- Before entering into the critical section, a process inquires about the lock. If it is locked, it keeps on waiting till it become free and if it is not locked, it takes the lock and executes the critical section.
- In TestAndSet, mutual exclusion and progress are preserved but bounded waiting cannot be preserved. The TestAndSet instruction can be defined as shown in Fig. 4.5.
- The important characteristic is that the TestAndSet instruction is executed automatically.

```
boolean TestAndSet(boolean *target)
{
    boolean rv = target;
    target = true;
    return rv;
}
```

**Fig. 4.5: The definition of the TestAndSet instruction**

- If the machine supports the TestAndSet instruction, then we can implement mutual exclusion br declaring a Boolean variable lock, initialized to false. The structure of process $P_i$ is shown in Fig. 4.6.

```
do
{
    while (TestAndSet(lock));
        critical section
    lock = false;
        remainder section
} while(1);
```

**Fig. 4.6: Mutual-Exclusion Implementation with TestAndSet**

- The Swap instruction defined as shown in Fig. 4.7 which operates on the contents of two words like the TestAndSet instruction, it is executed automatically.

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

**Fig. 4.7: The Definition of the Swap Instruction**

- If the machine supports Swap instruction then mutual exclusion can be provided. The structure of process Pi is shown in Fig. 4.8. Each process has a local Boolean variable key.

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

        critical section

    lock = false;

        remainder section
} while(1);
```

**Fig. 4.8: Mutual-Exclusion Implementation with the Swap Instruction**

**Semaphore Solution:**

- Semaphore is simply a variable that is non-negative and shared between threads. It is another algorithm or solution to the critical section problem.
- It is a signaling mechanism and a thread that is waiting on a semaphore, which can be signaled by another thread.
- It uses two atomic operations, namely, Wait and Signal for the process synchronization.

**Example:**

```
Wait ( S ):
    while ( S <= 0 );
    S = S - 1;
Signal ( S ):
    S = S + 1;
```

## 4.3 SEMAPHORE [April 16, 18, Oct. 16, 17]

- The various hardware-based solutions to the critical region problem are complicated to use. This problem can be solved using synchronization tool semaphore.
- Semaphore is introduced by Edsger W. Dijkstra in 1965. It restricts the sequence or order of execution of the statements.
- A semaphore is a variable used to control access to a common resource by multiple processes and avoid critical section problem.
- A semaphore is an integer variable, which can be accessed only through two operations Wait() and Signal().

- The Wait operation decrements the value of its argument S, if it is positive. If S is negative or zero, then no operation is performed. The Signal operation increments the value of its argument S.
- The classical definition of Wait() operation is as follows:

```
Wait (S)
    {
    while (S <= 0) ;
    // no-operation
    S--;
    }
```

- The classical definition of Signal() operation is as follows:

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

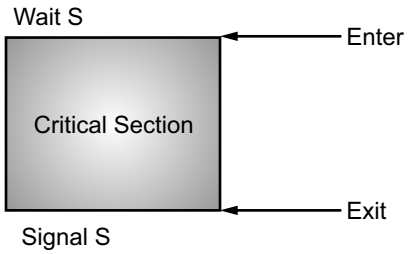



**Fig. 4.9: Concept of Semaphore**

**Advantages of Semaphores:**

1. Semaphores allow only one process into the critical section. They follow the mutual exclusion principle strictly and are much more efficient than some other methods of synchronization.
2. There is no resource wastage because of busy waiting in semaphores as processor time is not wasted unnecessarily to check if a condition is fulfilled to allow a process to access the critical section.

## 4.3.1 Usage [April 17, Oct. 18]

- We can use semaphore to deal with the n processes. The semaphore can be used in various synchronization problems.
- The two type's semaphores are binary semaphore and counting semaphore. A binary semaphore is restricted to values of zero (0) or one (1), while a counting semaphore can assume any non-negative integer value. The counting semaphores can range over an unrestricted domain.

**1. Binary Semaphore:**

- Binary semaphore is a semaphore with an integer value that can range only between 0 and 1. Binary semaphore is also known as mutex locks, as the locks can provide mutual exclusion.
- All the processes can share the same mutex semaphore that is initialized to 1. Then, a process has to wait until the lock becomes 0.
- Then, the process can make the mutex semaphore 1 and start its critical section. When it completes its critical section, it can reset the value of mutex semaphore to 0 and some other process can enter its critical section.
- We can use binary semaphores to deal with critical section problem for multiple processes.
- The n (multiple) processes share a semaphore mutex initialized to 1. Each process $P_i$ is organized as shown below:

```
do
{
    Wait (mutex)
    //Critical section
    Signal (mutex)
    //remainder section
}
while(TRUE);
```

- In OS a semaphore is used to solve number of synchronization problems. For example, consider process P1 and P2 are concurrently running and share a common semaphore synch initialized to zero.
- The Process P1 with a statement S1 and Process P2 with a statement S2. Suppose we requires that S2 can be executed only after S1 has completed then we can implement this scheme readily by letting P1 and P2 share a common semaphore synch, initialized to zero (0) and by inserting the statements in the following table:

| Process P1 | Process P2 |
|---|---|
| S1;<br>Signal(synch); | Wait(synch);<br>S2; |

- Because synch is initialized to 0, the Process P2 will execute statement S2 only after the Process P1 has invoked Signal(synch) which after statement S1.

**2. Counting Semaphore:** **[April 17]**

- The value of counting semaphore can range over an unrestricted domain. It can be used to control access to a given resource consisting of finite number of instances.
- The semaphore is initialized to number of resources available. Each process that wishes to use resource performs Wait() operation. When process releases resource, it performs Signal() operation.

- Counting semaphore S can be implemented in terms of binary semaphore. Counting semaphore can be used to control access a resource that has a limitation on the number of simultaneous accesses.
- The counting semaphore can be initialized to the number of instances of the resource. Whenever a process wants to use that resource, it checks if the number of remaining instances is more than zero, i.e., the process has an instance available.
- Then, the process can enter its critical section thereby decreasing the value of the counting semaphore by 1.
- After the process is over with the use of the instance of the resource, it can leave the critical section thereby adding 1 to the number of available instances of the resource.

## 4.3.2 Implementation

- The critical aspect of semaphore is that they be executed automatically. We must guarantee that no two processes can execute Wait() and Signal() operations on the semaphore at the same time. This is critical section problem.
- While one process is in its critical section, another process that tries to enter its critical section must loop continuously in the entry code. This continuous looping is problem in multiprogramming environment.
- In multiprogramming environment, only one CPU is shared among processes. A process which is waiting utilizes the CPU only to execute wait loop which is nothing but wastage of CPU cycles. This is called as problem of busy waiting.
- To overcome need of busy waiting; Wait() and Signal() operations are modified. With each semaphore, there is an associated waiting queue.
- Each entry in a waiting queue has two data items namely, value (of type integer) and pointer to next record in the list
- C structure of semaphore is as follows:

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

- Two operations are explained below:
  1. **block:** Place the process invoking the operation on the appropriate waiting queue.
  2. **wakeup:** Remove one of processes in the waiting queue and place it in the ready queue.
- A process which is waiting is blocked and added to list of blocked processes. The process which is blocked by operation Wait(), will be restarted by wakeup operation which is included in Signal() operation.

**Implementation of Wait Operation:**

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

**Implementation of Signal Operation:**

```
Signal (semaphore *s)
{
    s→value++;
    if (s → value < = 0)
    {
        remove a process P from the waiting queue
        wakeup(P);
    }
}
```

- In a uniprocessor environment (only one CPU), critical section problem can be solved by inhibiting interrupts during execution of Wait() and Signal() ( P or V) operation. So only currently running process executes until interrupts are re-enabled.
- In a multiprocessor environment, inhibiting interrupts will not work. Commands from different processes may be interleaved in some arbitrary way.
- If the hardware does not support any special instructions, we can provide any of the correct software solution for the critical section problem.

## 4.4 CLASSIC PROBLEMS OF SYNCHRONIZATION

- In this section, we will see number of classical problems of synchronization includes Bounded-buffer (or Producer-Consumer) Problem, Readers and Writers Problem and Dining-Philosphers Problem.

### 4.4.1 Bounded-Buffer Problem (Producer and Consumer Problem) [Oct. 16, 18]

- Producer consumer problem is a classical synchronization problem. We can solve this problem by using semaphores.
- Bounded buffer problem is also called Producer-Consumer problem. Solution to this problem is, creating two counting semaphores “full” and “empty” to keep track of the current number of full and empty buffers respectively.
- Producers produce a product and consumers consume the product, but both use of one of the containers each time.

**Producer:**

- Creates item and adds to the buffer.
- Do not want to overflow the buffer.

**Consumer:**

- Removes items from buffer (consumes it).
- Do not want to get ahead of producer.

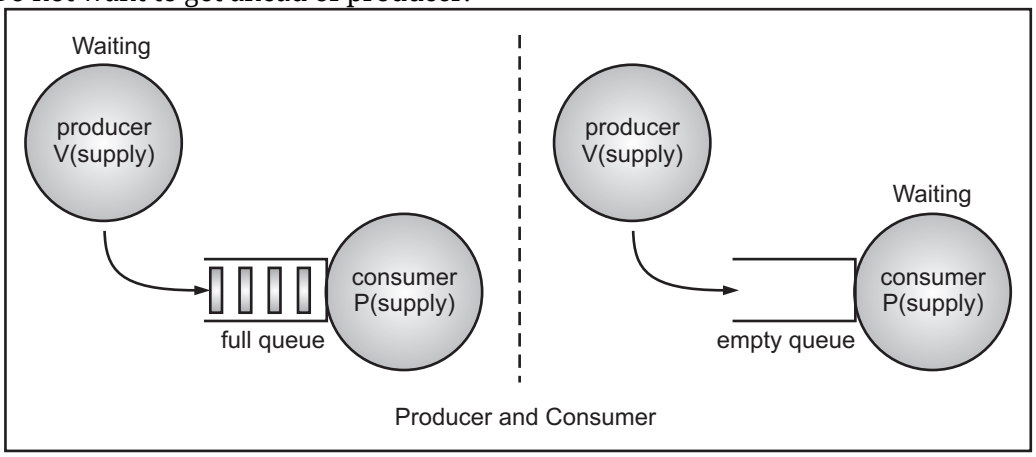



**Fig. 4.10: Producer and Consumer Problem**

- Bounded-buffer assumes that there is a fixed buffer size. Consumer waits for new item, producer waits if buffer is full. Producers and consumers are much like Unix pipes.
- The bounded buffer problem can be handled using semaphores, mutex semaphore provide mutual exclusion. Empty and full semaphores count number of empty and full buffers respectively.

Initialize Semaphore Full: = 0,
Semaphore empty: = n
Semaphore mutex: = 1

| Structure of Producer Process | Structure of Consumer Process |
|---|---|
| do<br>{<br>...<br>produce an item in nextp<br>...<br>Wait(empty);<br>Wait(mutex);<br>...<br>add nextp to buffer<br>...<br>Signal(mutex);<br>Signal(full);<br>}while(1); | do<br>{<br>Wait(full );<br>Wait(mutex);<br>...<br>remove an item from buffer to nextc<br>...<br>Signal(mutex);<br>Signal(empty);<br>...<br>consume the next item in nextc<br>...<br>}while(1); |

**Explanation:**

- While one process is producing or consuming, mutex value is made 0 and after producing and consuming is over; it is set back to 1. Therefore, mutual exclusion is preserved.
- For producer, if empty=0, then process cannot continue. It is made to wait and after producing is over; full is incremented.
- For consumer, exactly reverse is the case. If full =0; then process cannot continue. At the end, empty is incremented.

## 4.4.2 The Reader/Writer Problem

- The readers-writers problem is a classical problem of process synchronization, it relates to a data set such as a file that is shared between more than one process at a time.
- Among these various processes, some are Readers - which can only read the data set; they do not perform any updates, some are Writers - can both read and write in the data sets.

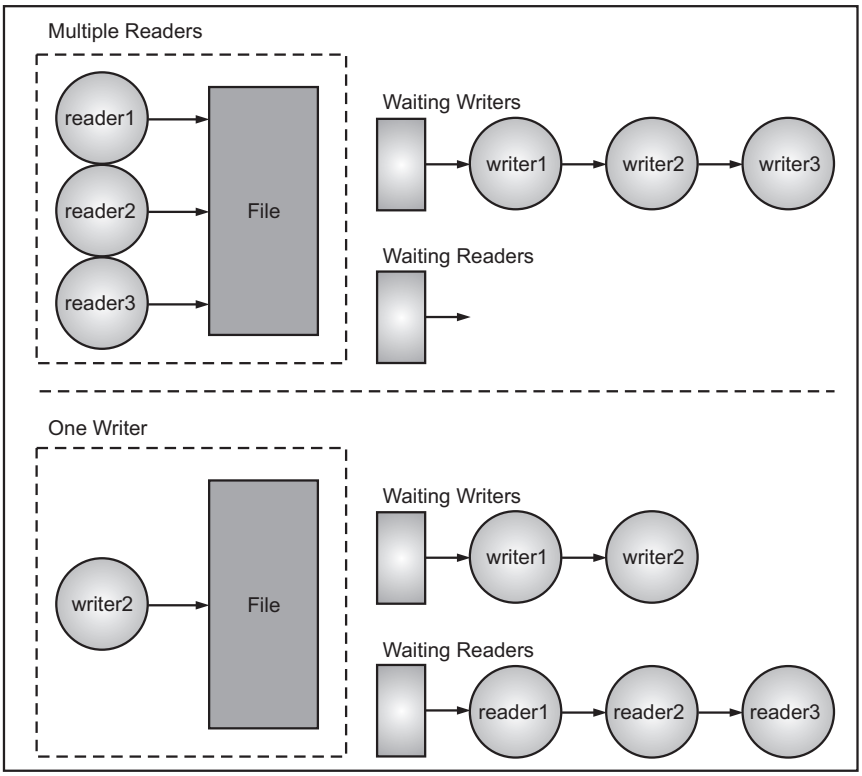



**Fig. 4.11: Reader –Writer Problem**

- The readers-writers problem is used for managing synchronization among various reader and writer process so that there are no problems with the data sets, i.e. no inconsistency is generated.
- A data set is shared among a number of concurrent processes. Some processes may want to read the data, whereas others may want to write the data.

- These two types of processes are referred as:
  1. **Readers** only read the data set; they do not perform any updates
  2. **Writers** can both read and write.
- The reader/writer problem is to allow multiple readers to read at the same time; but only one single writer can exclusively access the shared data at the same time. Such control can be achieved using semaphores.
- The mutex and wrt are common semaphores shared by all processes. readcount is a common integer which keeps count of concurrent reader processes.

```
Semaphore mutex = 1.
Semaphore wrt = 1.
integer readcount = 0
```

| Structure of Reader Process | Structure of Writer Process |
|---|---|
| while(true)<br>{<br>wait (mutex)<br>readcount ++;<br>if (readcount == 1)<br>wait (wrt);<br>signal (mutex)<br>// reading is performed<br>wait (mutex);<br>readcount - -;<br>if (readcount == 0)<br>signal (wrt);<br>signal (mutex);<br>} | while(true)<br>{<br>wait (wrt);<br>// writing is performed<br>signal (wrt);<br>} |

## 4.4.3 Dining Philosopher Problem [April 16, 19]

- The dining philosopher is a popular classic synchronization problem for concurrency control.
- The problem can be stated as follows:
  - Consider five philosophers spend their lives alternating between thinking and eating.
  - They are seated around a circular table. In the centre of table is a bowl of rice, and table is laid with five single chopsticks.
  - Each philosopher has access to the chopsticks at his/her left and right. In order to eat, a philosopher must be in possession of both chopsticks.
  - A philosopher may only pick up one chopstick at a time. Each philosopher attempts to pick up the left chopstick and then the right chopstick.

- When done eating, a philosopher puts both chopsticks back down on the table and begins thinking.
- Since the philosophers are sharing chopsticks, it is not possible for all of them to be eating at the same time.
- Now consider each philosopher as process and chopsticks are as resources. It is a simple presentation of the need to allocate several resources among several processes in a deadlock-free and starvation-free manner.

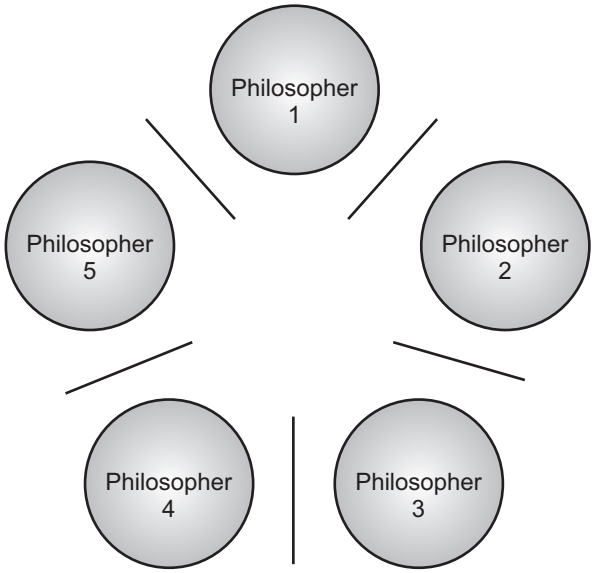



**Fig. 4.12: Dining Philosopher Problem**

- A solution of the dining philosopher's problem is to use a semaphore to represent a chopstick.
- A chopstick can be picked up by executing a Wait() operation on the semaphore and released by executing a Signal() operation semaphore.
- The structure of the chopstick is shown below:

  semaphore chopstick [5];
- Initially the elements of the chopstick are initialized to 1 as the chopsticks are on the table and not picked up by a philosopher.
- The structure of a random philosopher i is given as follows:

```
do {
  wait( chopstick[i] );
  wait( chopstick[ (i+1) % 5] );
  . .
  . eating the rice
  .
  signal( chopstick[i] );
```

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

- In the above structure, first wait operation is performed on chopstick[i] and chopstick[ (i+1) % 5]. This means that the philosopher i have picked up the chopsticks on his sides. Then the eating function is performed.
- After that, signal operation is performed on chopstick[i] and chopstick[ (i+1) % 5]. This means that the philosopher i has eaten and put down the chopsticks on his sides. Then the philosopher goes back to thinking.

**Practice Problems Based on Counting Semaphores:**

**Problem 1:** A counting semaphore S is initialized to 10. Then, 6 P operations and 4 V operations are performed on S. What is the final value of S?

**Solution:** We know, P operation also called as wait operation decrements the value of semaphore variable by 1. V operation also called as signal operation increments the value of semaphore variable by 1.

Thus, final value of semaphore variable S,

$= 10 - (6 \times 1) + (4 \times 1)$

$= 10 - 6 + 4$

$= 8$

**Problem 2:** A counting semaphore S is initialized to 7. Then, 20 P operations and 15 V operations are performed on S. What is the final value of S?

**Solution:** We know, P operation also called as wait operation decrements the value of semaphore variable by 1. V operation also called as signal operation increments the value of semaphore variable by 1.

Thus, final value of semaphore variable S,

$= 7 - (20 \times 1) + (15 \times 1)$

$= 7 - 20 + 15$

$= 2$

**Problem 3:** Consider three concurrent processes P1, P2 and P3 as shown below, which access a shared variable D that has been initialized to 100.

| P1 | P2 | P3 |
|---|---|---|
| :<br>:<br>D = D + 20<br>:<br>: | :<br>:<br>D = D − 50<br>:<br>: | :<br>:<br>D = D + 10<br>:<br>: |

The processes are executed on a uniprocessor system running a time-shared operating system. If the minimum and maximum possible values of D after the three processes have completed execution are X and Y respectively, then what will be the value of Y–X?

**Solution:**

**Case 1:**

Let initially P2 reads D = 100 and then got preempted.

Then P1 executes D = D + 20, so now D = 120.

Then P3 executes D = D + 10, so now D = 130.

Initially when P2 got preempted the value of D was = 100.

Now P2 executes D = D − 50 = 100 − 50 = 50

P2 writes D = 50 final value. This is minimum.

**Case 2:**

Let initially P1 reads D = 100 and then got preempted.

Then P3 reads D = 100 and then got preempted.

Then P2 reads D = 100 and executes D = D - 50 = 100 − 50 = 50 and writes it.

Then P1 executes D = D + 20 = 100 + 20 = 120

Then P3 reads D = 120 and executes D = D + 10 gives maximum value D = 130.

So, Y − X = 130 − 50 = 80.

---

**Problem 4:** Consider two processes P1 and P2 accessing the shared variables X and Y protected by two binary semaphores SX and SY respectively, both initialized to 1. P and V denote the usual semaphore operators, where P decrements the semaphore value, and V increments the semaphore value. The pseudo-code of P1 and P2 is as follows:

**Solution:**

**P1:**

```
while true do
{
    L1 : .....
    L2 : .....
    X = X + 1;
    Y = Y - 1;
    V(SX);
    V(SY);
}
```

**P2:**

```
while true do
{
    L3 : .....
    L4 : .....
    Y = Y + 1;
    X = Y - 1;
    V(SY);
    V(SX);
}
```

In order to avoid deadlock, the correct operators at L1, L2, L3 and L4 are respectively

(A) P(SY), P(SX); P(SX), P(SY)

(B) P(SX), P(SY); P(SY), P(SX)

(C) P(SX), P(SX); P(SY), P(SY)

(D) P(SX), P(SY); P(SX), P(SY)

**Answer: (D)**

**Explanation:**

**Option A:** In line L1 ( p(Sy) ) i.e. process p1 wants lock on Sy that is held by process p2 and line L3 (p(Sx)) p2 wants lock on Sx which held by p1. So here circular and wait condition exist means deadlock.

**Option B:** In line L1 ( p(Sx) ) i.e. process p1 wants lock on Sx that is held by process p2 and line L3 (p(Sy)) p2 wants lock on Sx which held by p1. So here circular and wait condition exist means deadlock.

**Option C:** In line L1 ( p(Sx) ) i.e. process p1 wants lock on Sx and line L3 (p(Sy)) p2 wants lock on Sx . But Sx and Sy can't be released by its process.

**Problem 5:** The following program consists of 3 concurrent processes and 3 binary semaphores. The semaphores are initialized as S0 = 1, S1 = 0, S2 = 0.

| Process P0 | Process P1 | Process P2 |
|---|---|---|
| while (true)<br>{<br>wait (S0);<br>print '0';<br>release (S1);<br>release (S2);<br>} | wait (S1);<br>release(S0); | wait (S2);<br>release(S0); |

How many times will process P0 print '0'?

(A) At least twice
(B) Exactly twice
(C) Exactly thrice
(D) Exactly once

**Solution:**

**(A) Explanation:** Initially only P0 can go inside the while loop as S0 = 1, S1 = 0, S2 = 0. Minimum no. of time 0 printed is twice when execute in this order (P0 → P1 → P2 → P0). Maximum no. of time 0 printed is thrice when execute in this order (P0 → P1 → P0 → P2 → P0).

## PRACTICE QUESTIONS

**Q.I Multiple Choice Questions:**

1. Which means sharing system resources by processes in such a way that, concurrent access to shared data is handled thereby minimizing the chance of inconsistent data?
   (a) Process synchronization (b) Thread synchronization
   (c) Process virtualization (d) All of the above mentioned
2. When several processes access the same data concurrently and the outcome of the execution depends on the particular order in which the access takes place, is called as,
   (a) dynamic condition (b) race condition
   (c) essential condition (d) critical condition
3. If a process is executing in its critical section, then no other processes can be executing in their critical section. This condition is called as,
   (a) mutual exclusion (b) critical exclusion
   (c) synchronous exclusion (d) asynchronous exclusion
4. Which one of the following is a synchronization tool?
   (a) thread (b) socket
   (c) semaphore (d) None of the Above
5. Process synchronization can be done on,
   (a) hardware level (b) software level
   (c) both (a) and (b) (d) None of the Above
6. Which are used for signaling among processes and can be readily used to enforce a mutual exclusion discipline.
   (a) Semaphores (b) Messages
   (c) Monitors (d) None of the Above
7. Semaphores provide a primitive yet powerful and flexible tool for enforcing mutual exclusion and for coordinating processes called as,
   (a) binary semaphore (b) monitor
   (c) counting semaphore (d) None of the Above

8. Concurrent access to shared data may result in:
   (a) data consistency (b) data insecurity
   (c) data inconsistency (d) None of the Above
9. The shared data (resources) is called _____ and the portion of the program, that uses this critical resource, is called as _____.
   (a) critical resource, critical section (b) process resource, process section
   (c) thread resource, thread section (d) All of the Above
10. In the bakery algorithm to solve the critical section problem,
   (a) each process is put into a queue and picked up in an ordered manner
   (b) each process receives a number (may or may not be unique) and the one with the lowest number is served next
   (c) each process gets a unique number and the one with the highest number is served next
   (d) each process gets a unique number and the one with the lowest number is served next
11. Bounded waiting implies that there exists a bound on the number of times a process is allowed to enter its critical section _____.
   (a) after a process has made a request to enter its critical section and before the request is granted
   (b) when another process is in its critical section
   (c) before a process has made a request to enter its critical section
   (d) None of the Above
12. Mutual exclusion problem occurs,
   (a) Between two disjoint processes that do not interact
   (b) Among processes that share resources
   (c) Among processes that do not use the same resource
   (d) Between two processes that uses different resources of different machine
13. _____ is the ability of multiple processes to co-ordinate their activities by exchange of information.
   (a) Synchronization (b) Mutual exclusion
   (c) Dead lock (d) Starvation
14. _____ is a condition in which there is a set of concurrent processes, only one of which is able to access a given resource or perform a given function at any time.
   (a) mutual exclusion (b) busy waiting
   (c) deadlock (d) starvation
15. The segment of code in which the process may change common variables, update tables, write into files is known as,
   (a) program (b) critical section
   (c) non-critical section (d) synchronizing
16. Which process can be affected by other processes executing in the system?
   (a) cooperating process (b) child process
   (c) parent process (d) init process

17. A critical section is a program segment _____.
    (a) which should run in a certain specified amount of time
    (b) which avoids deadlocks
    (c) where shared resources are accessed
    (d) which must be enclosed by a pair of semaphore operations, P and V
18. When the result of a computation depends on the speed of the processes involved there is said to be,
    (a) cycle stealing (b) race condition
    (c) a time lock (d) a deadlock
19. A semaphore is a shared integer variable _____.
    (a) that can not drop below zero (b) that can not be more than zero
    (c) that can not drop below one (d) that can not be more than one
20. Process synchronization can be done on _____.
    (a) hardware level
    (b) software level
    (c) both hardware and software level
    (d) none of the mentioned
21. A solution to the Dining Philosophers Problem which avoids deadlock is _____.
    (a) ensure that all philosophers pick up the left fork before the right fork
    (b) ensure that all philosophers pick up the right fork before the left fork
    (c) ensure that one particular philosopher picks up the left fork before the right fork, and that all other philosophers pick up the right fork before the left fork
    (d) None of the above
22. Mutual exclusion can be provided by the _____.
    (a) mutex locks (b) binary semaphores
    (c) Both (a) and (b) (d) None of the mentioned
23. Requirement for mutual exclusion a process remains inside its critical section for a _____.
    (a) infinite time (b) finite time
    (c) time (d) none
24. Mutual exclusion implies that _____.
    (a) if a process is executing in its critical section, then no other process must be executing in their critical sections
    (b) if a process is executing in its critical section, then other processes must be executing in their critical sections
    (c) if a process is executing in its critical section, then all the resources of the system must be blocked until it finishes execution
    (d) None of the mentioned
25. A minimum of _____ variable(s) is/are required to be shared between processes to solve the critical section problem.
    (a) One (b) Two
    (c) Three (d) Four
26. The TestAndSet instruction is execution _____.
    (a) After a particular process (b) Periodically
    (c) Atomically (d) None of the mentioned

27. Semaphore is a/an _____ to solve the critical section problem.
    (a) Hardware for a system (b) Special program for a system
    (c) Integer variable (d) None of the mentioned
28. In semaphores, two or more processors can cooperate by means of simple _____.
    (a) Signals (b) Data
    (c) Registers (d) Buffers
29. The two atomic operations permissible on semaphores are _____.
    (a) Wait, Signal (b) Wait, Stop
    (c) Wait, Hold (d) Hold, Signal
30. The Wait operation of the semaphore basically works on the basic _____ system call.
    (a) Stop() (b) Block()
    (c) Hold() (d) Wait()
31. The signal operation of the semaphore basically works on the basic _____ system call.
    (a) Continue() (b) Wakeup()
    (c) Getup() (d) Start()
32. If the semaphore value is negative _____.
    (a) Its magnitude is the number of processes waiting on that semaphore
    (b) It is invalid
    (c) No operation can be further performed on it until the signal operation is performed on it
    (d) None of the mentioned
33. The code that changes the value of the semaphore is _____.
    (a) Remainder section code (b) Non-critical section code
    (c) Critical section code (d) None of the mentioned
34. The two kinds of semaphores are _____.
    (a) Mutex and counting (b) Binary and counting
    (c) Counting and decimal (d) Decimal and binary
35. A semaphore _____.
    (a) is a binary mutex
    (b) must be accesses from only one process
    (c) can be accesses from multiple processes
    (d) None of the mentioned
36. At a particular time of computation the value of a counting semaphore is 7. Then 20 P operations and 15 V operations were completed on this semaphore. The resulting value of the semaphore is _____.
    (a) 42 (b) 2
    (c) 7 (d) 12
37. A binary semaphore is a semaphore with integer values _____.
    (a) 1 (b) – 1
    (c) 0.8 (d) 0.5

38. Semaphores are mostly used to implement _____.
    (a) System calls (b) IPC mechanisms
    (c) System protection (d) None of the mentioned
39. The bounded buffer problem is also known as _____.
    (a) Readers - Writers problem (b) Dining - Philosophers problem
    (c) Producer - Consumer problem (d) None of the mentioned
40. In the bounded buffer problem, there are the empty and full semaphores that _____.
    (a) Count the number of empty and full buffers
    (b) Count the number of empty and full memory spaces
    (c) Count the number of empty and full queues
    (d) None of the mentioned
41. In the bounded buffer problem _____.
    (a) There is only one buffer
    (b) There are n buffers (n being greater than one but finite)
    (c) There are infinite buffers
    (d) The buffer size is bounded
42. To ensure difficulties do not arise in the readers - writers problem, _____ are given exclusive access to the shared object.
    (a) Readers (b) Writers
    (c) Readers and writers (d) None of the mentioned
43. The dining - philosophers problem will occur in case of _____.
    (a) 5 philosophers and 5 chopsticks (b) 4 philosophers and 5 chopsticks
    (c) 3 philosophers and 5 chopsticks (d) 6 philosophers and 5 chopsticks
44. The process invoking the wait operation is _____.
    (a) A suspended until another process invokes the signal operation
    (b) Waiting for another process to complete before it can itself call the signal operation
    (c) Stopped until the next process in the queue finishes execution
    (d) None of the mentioned
45. If no process is suspended, the signal operation _____.
    (a) Puts the system into a deadlock state
    (b) Suspends some default process execution
    (c) Nothing happens
    (d) The output is unpredictable
46. All processes share a semaphore variable mutex, initializes to 1. Each process must execute wait(mutex) before entering the critical section and signal(mutex) afterward. Suppose a process executes in the following manner : signal(mutex); …critical section … wait(mutex); In this situation _____.
    (a) A deadlock will occur
    (b) Processes will starve to enter critical section
    (c) Several processes may be executing in their critical section
    (d) All of the mentioned

47. The initial value of the semaphore that allows only one of the many processes to enter their critical sections is _____.
    (a) 8 (b) 1
    (c) 6 (d) 0
48. In implementation of semaphores, for a single processor system, it is possible to inhibited _____.
    (a) Deadlock (b) Interrupts
    (c) Lock Step (d) None
49. Using semaphores, each process has a critical section used to access the _____.
    (a) Computers (b) Processors
    (c) Resources (d) Users
50. In readers and writers problem, readers are processes that are not required to _____.
    (a) Include (b) Exclude
    (c) Describe (d) None

## Answers

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1. (a) | 2. (b) | 3. (a) | 4. (c) | 5. (c) | 6. (a) | 7. (b) | 8. (c) | 9. (a) | 10. (b) |
| 11. (a) | 12. (b) | 13. (a) | 14. (a) | 15. (b) | 16. (b) | 17. (c) | 18. (b) | 19. (a) | 20. (c) |
| 21. (c) | 22. (c) | 23. (b) | 24. (a) | 25. (b) | 26. (c) | 27. (c) | 28. (a) | 29. (a) | 30. (b) |
| 31. (b) | 32. (a) | 33. (c) | 34. (b) | 35. (c) | 36. (b) | 37. (a) | 38. (b) | 39. (c) | 40. (a) |
| 41. (b) | 42. (b) | 43. (a) | 44. (a) | 45. (c) | 46. (c) | 47. (b) | 48. (b) | 49. (c) | 50. (b) |

**Q.II Fill in the Blanks:**

1. Synchronization was introduced to handle problems that arose while _____ process executions.
2. Process Synchronization is the task of coordinating the execution of processes in a way that no two processes can have access to the _____ shared data and resources.
3. A _____ section is a segment of code which can be accessed by a signal process at a specific point of time.
4. In the critical section, only a _____ process can be executed. Other processes, waiting to execute their critical section, need to wait until the current process completes its execution.
5. _____ exclusion is a special type of binary semaphore which is used for controlling access to the shared resource.
6. A _____ condition typically occurs when two or more threads try to read, write and possibly make the decisions based on the memory that they are accessing concurrently.
7. The regions of a program that try to access shared resources and may cause race conditions are called critical_____.
8. To avoid race condition among the processes, we need to assure that only _____ process at a time can execute within the critical section.

9. In _____ process execution of one process does not affects the execution of other processes.
10. A race condition is a situation that may occur _____ a critical section.
11. A semaphore is a shared variable/integer variable to _____ the progress of interacting processes.
12. A semaphore uses two atomic operations, _____ and _____ for process synchronization.
13. _____ semaphores can only be either 0 or 1.
14. _____ synchronization is the task of coordinating the execution of processes in a way that no two processes can have access to the same shared data and resources.
15. _____ Semaphore value can range over an unrestricted domain. It is used to control access to a resource that has multiple instances.
16. Number of synchronization problems like bounded buffer problem , the reader writer problem, the dining philosopher problem etc. can be solved by using _____.

## Answers

| | | | |
|---|---|---|---|
| 1. multiple | 2. same | 3. critical | 4. single |
| 5. Mutual | 6. race | 7. section | 8. one |
| 9. independent | 10. inside | 11. synchronize | 12. wait, signal |
| 13. Binary | 14. Process | 15. Counting | 16. semaphores |

**Q.III State True or False:**

1. Synchronization was introduced to handle problems that arose while single process executions.
2. A cooperative process is the one which can affect the execution of other process or can be affected by the execution of other process.
3. Process synchronization is the task of coordinating the execution of processes in a way that no two processes can have access to the same shared data and resources.
4. A critical section is a segment of code which can be accessed by a signal process at a specific point of time. The section consists of shared data resources that required to be accessed by other processes.
5. A critical section is a segment of code that can be accessed by only one signal process at a certain instance in time.
6. A semaphore is simply a variable that stores an integer value.
7. The critical section code can be accessed by only multiple process at a time.
8. In the critical section, only a single process can be executed. Other processes, waiting to execute their critical section, need to wait until the current process completes its execution.
9. To avoid race condition among the processes, we need to assure that only one process at a time can execute within the critical section.
10. When more than one processes are executing the same code or accessing the same memory or any shared variable in that condition there is a possibility that the

output or the value of the shared variable is wrong so for that all the processes doing the race to say that my output is correct this condition known as a race condition.

11. The entry to the critical section is handled by the wait() function.
12. Race condition contains shared variables which need to be synchronized to maintain consistency of data variables.
13. A semaphore is basically a synchronizing tool and is accessed only through two standard atomic operations, wait and signal.
14. Counting semaphores takes more than one value.
15. Any solution to the critical section problem must satisfy three requirements namely, mutual exclusion, progress and bounded waiting.
16. Binary semaphores are also known as mutex locks, as the locks can provide mutual exclusion.
17. The exit from a critical section is controlled by the Wait() function.
18. Four elements of critical section are Entry section, Critical section, Exit section and Reminder section.

## Answers

| | | | | |
|---|---|---|---|---|
| 1. (F) | 2. (T) | 3. (T) | 4. (T) | 5. (T) |
| 6. (T) | 7. (F) | 8. (T) | 9. (T) | 10. (T) |
| 11. (T) | 12. (F) | 13. (T) | 14. (T) | 15. (T) |
| 16. (T) | 17. (F) | 18. (T) | | |

**Q.IV Answers the following Questions:**

**(A) Short Answer Questions:**

1. What is synchronization?
2. Define semaphore.
3. List types of semaphore.
4. What is critical section problem?
5. What is race condition?
6. List classical problem for synchronization.
7. What is dining philosopher problem?
8. Give reader writer problem in synchronization.
9. List signals for semaphore.

**(B) Long Answer Questions:**

1. Define synchronization. Why it is needed in OS? Explain in detail.
2. What is semaphore? Enlist its types.
3. Define critical section problem. Explain in detail.
4. Write note on: Race condition.
5. What are the types of semaphore? Explain its usage and implementation.

6. Discuss the following problems in concurrent programming and explain the use of semaphore in these problems:
   (i) The producer-consumer problem
   (ii) The reader-writer problem
   (iii) Dining Philosopher problem
7. Define synchronization. Explain in detail.
8. Compare binary and counting semaphores.
9. Give background for synchronization.
10. What is purpose of semaphores? Also explain usages of semaphore.
11. Define the terms:
    (i) Race condition.
    (ii) Bounded wait.
    (iii) Critical section.
    (iv) Binary semaphore.
    (v) Counting Semaphore.
12. Explain Bounded buffer problem? Give structure of producer and consumer.
13. How to implement an semaphore? Describe in detail.
14. Which two standard atomic operations can access semaphore value?
15. How semaphore can be used to solve the Dining Philosopher problem of concurrency control?
16. Which three requirements must be satisfied while designing a solution to the critical section problem? Explain each in detail.
17. Give Peterson's solution to solve critical section problem.
18. Explain the role of Wait() and Signal() operations used in semaphores.

## UNIVERSITY QUESTIONS AND ANSWERS

**April 2016**

**1.** What is race condition? **[1 M]**

**Ans.** Refer to Section 4.1.

**2.** State two general approaches that are used to handle critical section in operation system. **[1 M]**

**Ans.** Refer to Section 4.2.

**3.** What is a semaphore? Explain dining philosopher problem. **[5 M]**

**Ans.** Refer to Section 4.3.

**October 2016**

**1.** Define semaphore. **[1 M]**

**Ans.** Refer to Section 4.3.

**2.** Explain Bounded buffer problem. Give structure of producer and consumer. **[5 M]**

**Ans.** Refer to Section 4.4.1.

**April 2017**

**1.** "Counting semaphore can be implemented by using binary semaphore". True/Fal Justify. **[1 M]**

**Ans.** Refer to Section 4.3.1 Point (2).

**2.** What is race condition? **[1 M]**

**Ans.** Refer to Section 4.1.

**3.** What is critical section problem? Explain two general approaches to handle critical section in operation system. **[5 M]**

**Ans.** Refer to Section 4.2.

**October 2017**

**1.** Which two standard atomic operations can access semaphore value? **[1 M]**

**Ans.** Refer to Section 4.3.

**2.** What is critical section problem? How is it solved? **[5 M]**

**Ans.** Refer to Section 4.2.

**April 2018**

**1.** "A race condition exists when processes are running simultaneously". True/False. Justify. **[1 M]**

**Ans.** Refer to Section 4.1.

**2.** What is Semaphore? Explain bounded-buffer problems. **[5 M]**

**Ans.** Refer to Section 4.3.

**3.** What is critical section problem? Give Peterson's solution to solve critical section problem. **[4 M]**

**Ans.** Refer to Section 4.2.

**October 2018**

**1.** What are the types of semaphores? **[1 M]**

**Ans.** Refer to Section 4.3.1.

**2.** Explain critical section problem. **[5 M]**

**Ans.** Refer to Section 4.2.

**3.** Discuss the bounded buffer problem with its solution. **[4 M]**

**Ans.** Refer to Section 4.4.1.

**April 2019**

**1.** Write a short note on Dining philosopher problem. **[5 M]**

**Ans.** Refer to Section 4.4.3.

**2.** What is mutual exclusion? **[1 M]**

**Ans.** Refer to Page 4.4.

■■■

CHAPTER 5

# Memory Management

## *Objectives ...*

- To learn Basic Concepts in Memory Management such as Hardware, Physical and Logical Address Space, Basic Method for Memory Utilization like Dynamic Loading and Linking etc.
- To know various Memory Management Schemes like Contiguous Allocation, Paging and Segmentation etc.
- To understand Virtual Memory Management Technique

## 5.0 INTRODUCTION

- Computer memory is the storage space in the computer, where data is to be processed and instructions required for processing are stored.
- In a computer system, there may be multiple processes executing at the same time and every process that needs to execute, requires a certain amount of memory.
- To improve the utilization of the CPU and the speed of the computer's response to its users, we must keep several processes in memory i.e., we must share memory. Thus, multi-programming creates need of memory management.
- Memory management is important function of operating system which helps in allocating the main memory space to the processes and their data at the time of their execution. The main task of memory management is efficient memory utilization.
- Memory management keeps track of each and every memory location, regardless of either it is allocated to some process or it is free.
- There are various memory management schemes and techniques for managing memory.
- In selecting memory management scheme, various issues like hardware design, protection, binding of symbolic addresses to physical addresses, distinguishing logical and physical addresses, dynamic loading and linking etc. must be considered.

## 5.1 BACKGROUND

- The main purpose of a computer system is to execute programs (set of instructions). These programs together with the data they access must be in main memory during execution.

- As we know the important function of memory management is to improve utilization of CPU and the speed of its response to users, the computer system keeps several or multiple processes in memory.
- Main memory is usually too small to accommodate all the data and the programs permanently the computer system must provide secondary storage (disks) to backup main memory.
- The disks in computer system are the storage medium for information (both programs and data) and use a file system for access to data and programs residing on the disks.
- Memory is central to the operation of a modern computer system. To execute a program, it must be brought into memory. Memory consists of a large array of words or bytes, each with its own address.
- 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.
- Depending on the memory management scheme in use, the process may be moved between disk and memory during its execution.

## 5.1.1 Basic Hardware

- Memory management is the process of controlling and coordinating computer memory.
- The selection of a memory management method for a specific computer system depends on many factors especially on the hardware design of the computer system.
- Main memory and registers are only the storage that CPU can access directly. For any instruction in execution and any data being used by instruction must be in one of these storage devices.
- Registers are accessible generally in one clock of CPU. But main memory take more cycles of CPU to complete as CPU has to wait for data which is required for completing.
- To avoid this situation, a fast memory i.e. cache is added between main memory and CPU. In addition to speed of accessing memory, we must also ensure correct operation has to protect operating system from user access and user process from one another.
- This protection must be provided by the hardware. Two registers, base and limit register can be used to provide protection as shown in Fig. 5.1.
- The base register holds the starting base address or smallest legal physical memory address and limit register specifies size of the range.
- For example, if base register holds address 300040 and limit register contains range 120900 then process can access legal addresses from 300040 to 420940.

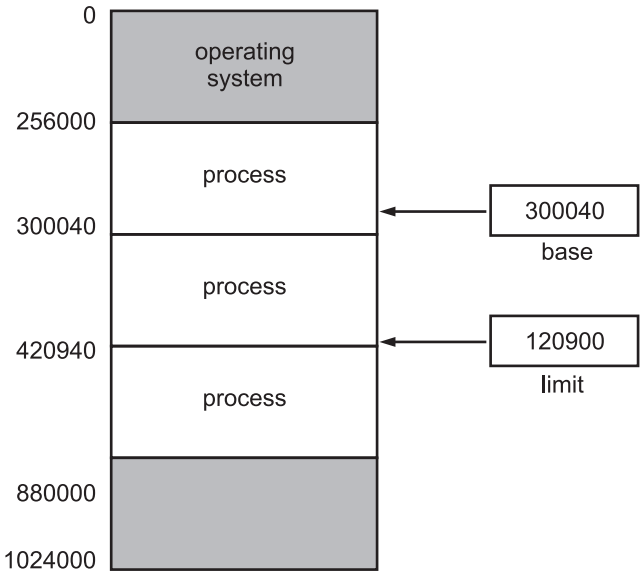



**Fig. 5.1: A Base Register and a Limit Register Define a Logical Address Space**

- Protection of memory space is accomplished by comparing every address generated by user's program with these registers.
- Any attempt made by user's program to access operating system's memory or other user's memory results in a trap to the operating system as shown in Fig. 5.2.
- Thus, this method prevents a user program from modifying code or data of operating system or other users.
- The base and limit registers can be loaded by operating system using special privileged instruction in kernel mode.

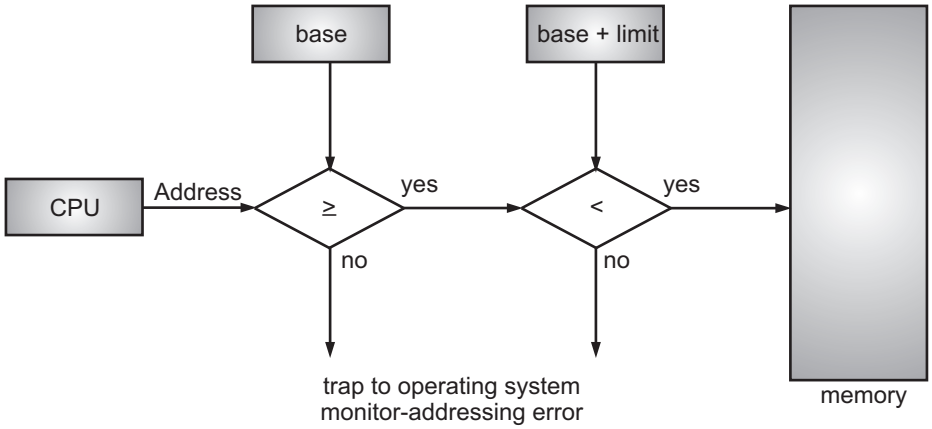



**Fig. 5.2: Hardware Address Protection with Base and Limit Registers**

## 5.1.2 Address Binding

- Generally, programs in computer system reside in executable form on the disk. For execution, program must be brought into memory and placed within process.

- So, to create active process in memory, symbolic addresses generated by program must be mapped into the process address space. This mapping is known as address binding.
- As process is executed, address binding accesses instructions and data from memory. Fig. 5.3 shows binding of instructions and data.
- Addresses in the source program are generally symbolic. A compiler will typically bind these symbol addresses to re-locatable addresses.
- The linkage editor or loader will bind these re-locatable addresses to absolute addresses. Each binding is a mapping from one address space to another. These absolute addresses loaded into memory to be executed.
- As program executes it accesses program instructions and data from memory by generating these absolute addresses.
- Eventually the program terminates, its memory space is declared available and the next program may be loaded and executed.

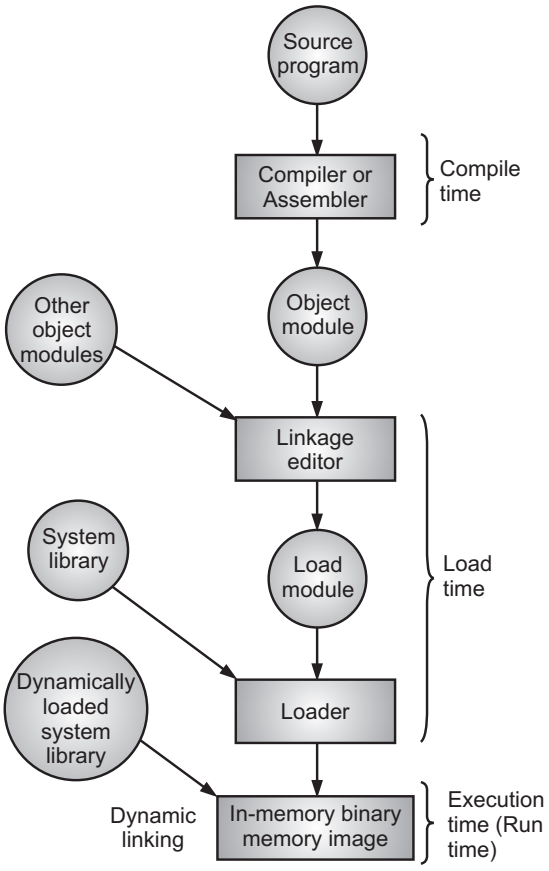



**Fig. 5.3: Steps in Address Binding**

- Address binding divided into following three types:
  1. **Compile Time Address Binding:** If the compiler is responsible for performing address binding then it is called compile-time address binding. If you know at

compile time where the process will reside in memory, then absolute code can be generated. If starting location changes, then it will be necessary to recompile this code. In MS-DOS, .COM files are bound at compile time. Compile time address binding will be done before loading the program into memory. The compiler requires interacts with an OS memory manager to perform compile-time address binding.

2. **Load Time Address Binding:** Load time address binding will be done after loading the program into memory and address binding will be done by loader. If it is not known at compile time where the process will reside memory, then compiler generates re-locatable code. Then binding is delayed until load time. If the starting address is changed, only reload the code to changed address.
3. **Execution Time or Dynamic Address Binding:** If the process can moved during its execution from one memory segment to another, then binding must be delayed until run time. Special hardware is needed to implement this scheme. Dynamic address binding will be postponed even after loading the program into memory. The program will be kept on changing the locations in memory until the time of program execution. The dynamic address binding done by the processor at the time of program execution.

## 5.1.3 Logical versus Physical Address Space [Oct. 16, April 19]

- Address uniquely identifies a location in the main memory. Memory locations determine how the computer's memory is organized in execution of program.
- Address space is the amount of memory allocated for all possible addresses for a computational task. An address space is a range of valid addresses in memory that are available for a program or process.
- Address space may refer to a range of either physical or virtual addresses accessible to a processor or reserved for a process.
- An address generated by the CPU is commonly known as logical address or virtual address.
- The address seen by the memory unit that is one loaded into the memory address register of the memory is commonly referred as the physical address.
- Logical address is generated by CPU while a program is running and used as a reference to access the physical memory location by CPU. Physical address identifies a physical location of required data in a memory.
- The compile time and load time address binding generates the identical logical and physical addresses.
- However, the execution time address binding scheme generates differing logical and physical addresses as process moves from one memory segment to another.
- The set of all logical addresses generated by a program is known as logical address space. The set of all physical addresses corresponding to these logical addresses is physical address space.

- The run time mapping from virtual address to physical address is done by a hardware device known as Memory Management Unit (MMU). The runtime mapping from virtual to physical address is done by the MMU.
- The operating system takes care of mapping the logical addresses to physical addresses at the time of memory allocation to the program. Different methods like contiguous allocation, paging, segmentation etc. can perform such mapping.
- The simplest memory mapping scheme is shown in the Fig. 5.4. Here, base register is known as relocation register.

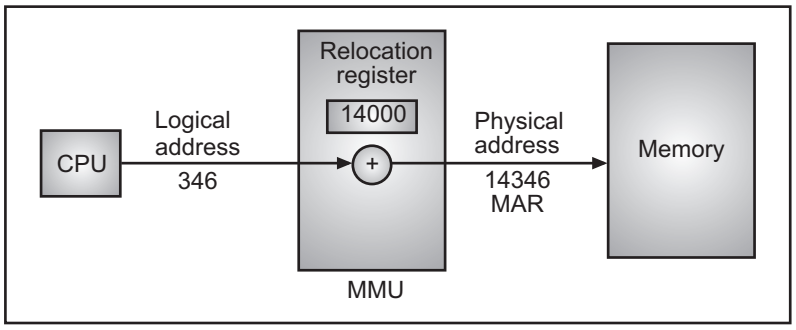



**Fig. 5.4: Dynamic Relocation using Relocation Register**

- The value in the relocation register is added to the address generated by a user process at the time it is sent to memory.
- Let's understand this situation with the help of example, if the base register contains the value 14000, then an attempt by the user to address location 346 is mapped to location 14346.
- The user program never sees the real physical address space, it always deals with the logical address.
- Logical address is in the range (0 to max) and Physical address is in the range(R to $R^{+max}$) where R is the value of relocation register.
- As it is clear that user program supplies only logical address, these logical address must be mapped to physical address before they are used.

## 5.1.4 Dynamic Loading

- In computer system all the programs are loaded in the main memory for execution. Sometimes complete program is loaded into the memory, but sometimes a certain part or routine of the program is loaded into the main memory only when it is called by the program, this mechanism is called dynamic loading.
- For process to execute, the entire program and data must be in physical memory. So the size of process is limited to size of physical memory.
- But all routines of program are not required at a time or program may contain some error routines which may be never called.
- To obtain better memory utilization, routines can be loaded in main memory as required. This is known as dynamic loading.

- In dynamic loading a routine is not loaded until it is called. All routines are kept on disk in re-locatable load format. Only main routine is loaded into memory and is executed.

**Advantages of Dynamic Loading:**

1. In dynamic loading unused routine is never loaded.
2. Dynamic loading also provides better memory space utilization.
3. No special support from the operating system is required, can be implemented through program design.

## 5.1.5 Dynamic Linking and Shared Libraries

- Linking is a method that helps operating system to collect and merge various modules of code and data into a single executable file. The file can be loaded into memory and executed.
- Static linking and dynamic linking are two processes of collecting and combining multiple object files in order to create a single executable.
- In static linking, system language libraries are treated as like any other object module and are combined by the loader into binary program image.
- Static linking is performed by the linker. The linker combines all modules needed by a program into a single executable program
- Static linking is done at the time of compilation of the program. In Dynamic linking method, libraries are linked at execution time.

**Dynamic Linking:**

- In dynamic linking, linking is postponed until execution time. This feature is usually useful in system libraries such as language subroutine libraries.
- Without this facility, each program on system must include one copy of language library. 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 stub is executed, it checks to see whether the needed routine is already in memory. If not, the program loads the routine into memory. Either way, the stub replaces itself with address of the routine and executes the routine.
- Thus, the next time that particular code segment is reached, the library routine is executed directly, incurring no cost for dynamic linking.

**Shared Libraries:**

- In 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 support, such programs would need to be re-linked to gain access to the new library.
- More than one version of a library may be loaded into memory, and each program uses its version information to decide which copy of library to use.
- Major changes increments the version number and minor changes retain the same version number.
- Thus, only programs that are compiled with new library version are affected by the incompatible changes incorporated in it.
- Other programs linked before the new library was installed will continue using the older library. This system is also known as shared libraries.
- Shared libraries are loaded into memory by programs when they start. When a shared library is loaded properly, all programs that start later automatically use the already loaded shared library.
- Shared libraries serve as sharing a single copy of library among multiple programs, hence they are called shared libraries, and the process of linking them with multiple programs is called dynamic linking.
- Advantages of shared libraries are as follows:
  1. Load time might be reduced because the shared library code might already be in memory.
  2. Run-time performance can be enhanced.

## 5.2 SWAPPING [Oct. 16]

- Swapping is a memory management technique in which any process can be temporarily swapped from main memory to secondary memory (disks) so that the main memory can be made available for other processes.
- At some later time, the system swaps back the process from the secondary storage such as disks to main memory for continue execution.
- The purpose of the swapping in operating system is to access the data present in the hard disk and bring it to RAM (main memory) so that the programs can use it.
- The process to be executed must be in main memory. A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution. This process is known as swapping.
- Swapping requires backing store. Backing store is a hard disk or some other secondary storage device that should be big enough in order to accommodate copies of all memory images for all users.
- Swapping is also capable of offering direct access to these memory images as shown in Fig. 5.5.
- Swapping involves performing two tasks called swap-in and swap-out. The task of placing the process from the hard disk to the main memory is called **swap-in**. The task of removing process from main memory to the hard disk is called **swap-out**.

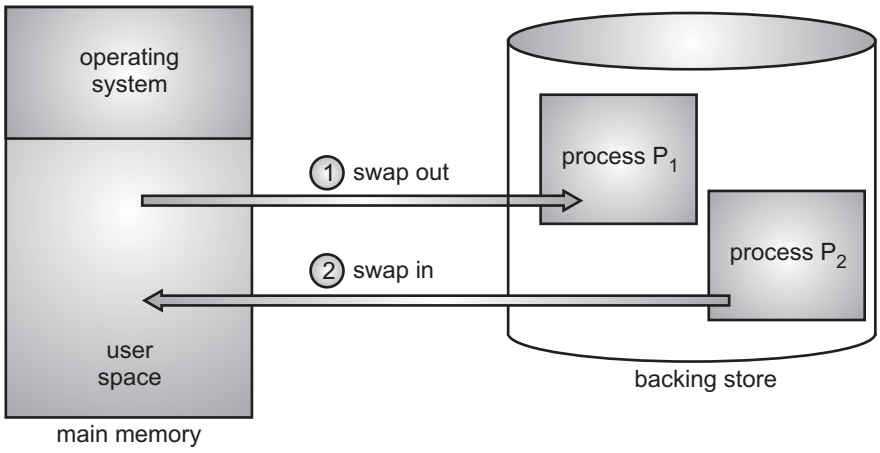



**Fig. 5.5: Swapping of Two Processes using a Disk as a Backing Store**

**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

$$\text{Time} = \frac{\text{Process size}}{\text{Transfer rate}}$$

$$= 2048 / 1024$$

$$= 2 \text{ seconds}$$

$$= 2000 \text{ milliseconds}$$

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. In the case of heavy swapping activity, if the computer system loses power, the user might lose all the information related to the program.
2. Inefficiency may arise in the case where a resources or a variable is commonly used by the processes which are participating in the swapping process.

## 5.3 CONTIGUOUS MEMORY ALLOCATION

- To get a process executed it must be first placed in the main memory. Assigning space to a process in the main memory is called memory allocation.

- Operating system allows accommodation of multiple processes in memory. Memory needs to be allocated efficiently in order to allow as many processes into memory as possible.
- The common method for main memory allocation is contiguous memory allocation. In contiguous memory allocation, each process is contained in a single contiguous section of memory.
- Main memory usually divided into two partitions (one for operating system and one for user processes) as shown in Fig. 5.6
- 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.
- For this the contiguous memory allocation technique is used in which each process is contained in a single contiguous section of memory.

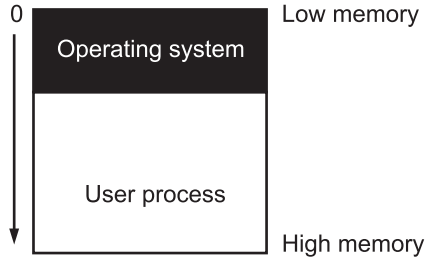



**Fig. 5.6: Memory Partition**

## 5.3.1 Memory Mapping and Protection

- The memory mapping is the process in which the operating system will map their corresponding virtual addresses to the physical addresses in main memory.
- The objectives of memory mapping are to translate from logical to physical address, to aid in memory protection and to enable better management or utilization of memory.
- The main purpose of memory protection is to prevent a process from accessing memory that has not been allocated to it.
- Memory protection means protecting the operating system from user process and protecting process from one another.
- Memory protection is provided by using a re-location register, with a limit register. The 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. This mapped address is sent to main memory as shown in Fig. 5.7.

- With help of relocation register, this code can be loaded during program execution. So size of operating system changes dynamically
- When the CPU scheduler selects a process for execution, the dispatcher loads the relocation 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 operating system and other users programs and data from being modified.

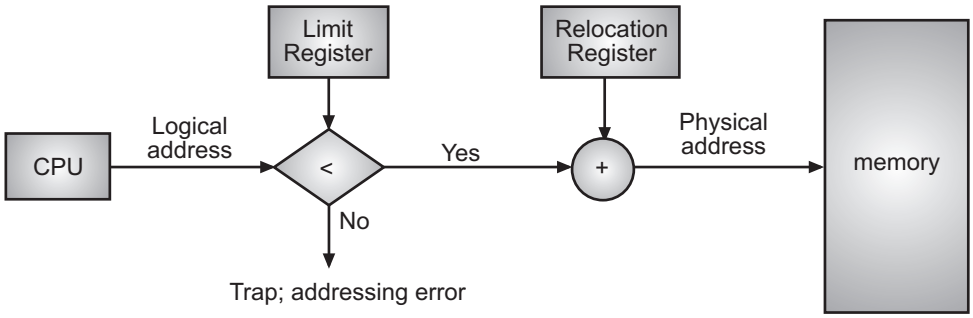



**Fig. 5.7: Hardware support for Relocation and Limit Registers**

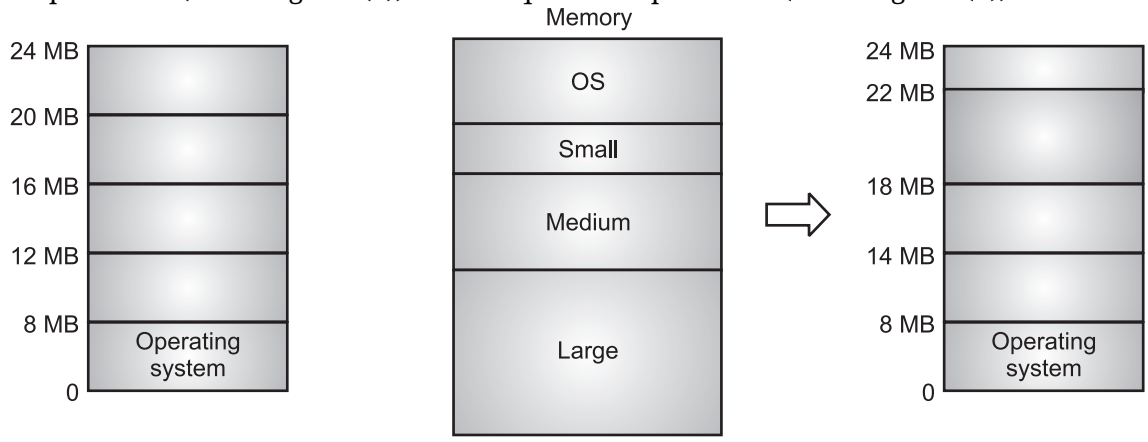
## 5.3.2 Memory Allocation [April 18]

- Memory allocation is a process by which computer programs are assigned memory or space to a process.
- In multiprogramming environment, several programs reside in main memory at a time. To support multiprogramming, main memory is divided into several partitions or regions each of which is allocated to a single process.
- The number of programs residing in memory will be bound by the number of partitions or regions.
- Depending upon how and when partitions are created, there may be two types of approaches namely, static (fixed sized or single) memory partitioning and dynamic (variable sized or multiple) memory partitioning.
- These two approaches divide memory into a number of regions or partitions. Static memory allocation method assigns the memory to a process, before its execution. The dynamic memory allocation method assigns the memory to a process, during its execution.

### 5.3.2.1 Static (Fixed Sized) Memory Partitioning

- In static (single) memory partitioning, the memory is divided into a number of fixed sized partitions or regions and do not change as the system runs.
- Each partition in static memory partitioning, contains exactly one process. So the number of programs to be executed (i.e. degree of multiprogramming) depends on the number of partitions.
- The static memory partitioning is originally used in the IBM OS/360 operating system. The static memory partitioning is also known as Multiprogramming with Fixed number of Tasks (MFT).

- There are two alternatives for fixed sized memory partitioning namely, equal sized partitions (Refer Fig. 5.8 (a)) and unequal sized partitions (Refer Fig. 5.8 (b)).




**(a) Equal Size Partitions** **(b) Unequal Size Partitions**

**Fig. 5.8**

**Job Scheduling in Fixed Sized Memory Partitions:**

- As jobs enter the system, they are put into a job queue. The job scheduler takes into account the memory requirement of each job and the available regions in determining which jobs are allocated memory.
- When a job is allocated space, it is loaded into a region. It can then complete for the CPU. When job terminates, it releases its memory region, which the job scheduler may then fill with another job from the job queue.
- A number of variations are possible in allocation of memory to jobs. One strategy is to classify all jobs on entry to system, according to its memory requirements.
- User specifies the maximum amount of memory required the system can attempt to determine memory requirements automatically.
- For example, a memory of 32K words might be divided into regions of the following sizes:

Resident Monitor = 12K
Very small jobs = 2K
Average jobs = 6K
Large jobs = 12K

- If we have three user memory regions of sizes 2K, 6K, and 12K we need three queues namely Q2, Q6 and Q12, (Refer Fig. 5.9 (a)).
- An incoming job requiring 4K of memory would be appended to Q6, a new job needing 8K would be put in Q12, and a job of 2K would go in Q2. Each queue is scheduled separately.

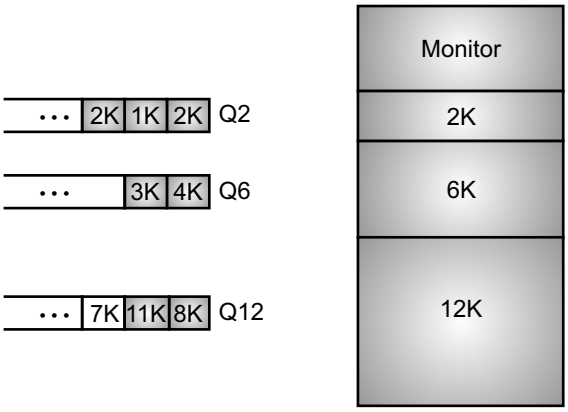



**Fig. 5.9 (a): MFT with Separate Queues for Each Region**

- Since each queue has its own memory region, there is no competition between queues for memory. Another approach is to throw all jobs into one queue.
- The job scheduler selects the next job to be run and waits until a memory region of that size is available.
- Suppose that we had FCFS job scheduler and regions of 2K, 6K, and 12K. We would first assign job1 (5K) to the 6K regions and job2 (2K) to the 2K regions. Since our next job requires 3K, we need the 6K regions.
- Since 6K regions are being used by job1, we must wait until job1 terminates, then job3 will be allocated the 6K regions. Job4 is then allocated to 12K regions and so on.

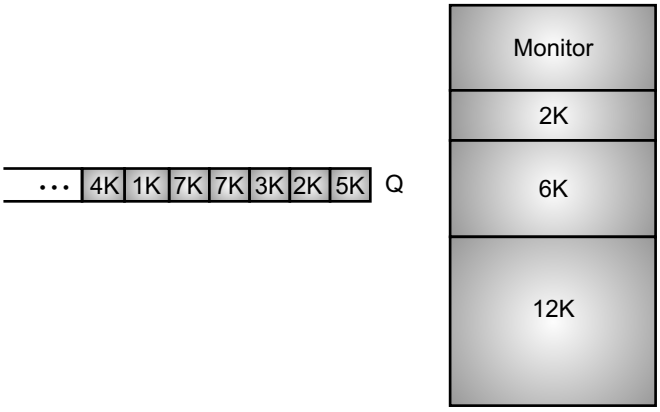



**Fig. 5.9 (b): MFT with a Unified Queue**

**Advantages:**

1. Implementation of fixed partitioning is simple and easy.
2. In fixed partitioning the overhead of processing is also low.

**Disadvantages:**

1. In fixed memory partitioning, as the size of the partition cannot change according to the size of the process. Thus, the degree of multiprogramming is very less and is fixed.

2. The main problem with the fixed partitioning method is how to determine the number of partitions, and how to determine their sizes.
3. The fixed memory partitioning suffers from problem of internal fragmentation.

## 5.3.2.2 Dynamic (Variable/Multiple) Memory Partitioning [Oct. 16]

- In dynamic memory partitioning, region or partition size is not fixed; it can vary dynamically.
- The dynamic memory partitioning creates partition according to requirements of process. In variable memory partitioning the amount of memory allocated is exactly the amount of memory a process requires.
- The dynamic memory partitioning is also known as Multiprogramming with Variable number of Tasks (MVT).
- Operating system maintains a table indicating which parts of the memory are free and which are allocated.
- Whenever a job requests for the memory, the table indicating the current status of the memory is referred.
- If memory is available, then the job is assigned to partition according to job scheduling policy and the table is updated to reflect the new status.
- When the job terminates, it releases the memory occupied by it. Therefore, at any instance of time, the snapshot of memory will show blocks of allocated memory and free holes distributed all over the memory.
- For example, assume 256K memory available and a resident monitor of 40K. This situation leaves 216K for user programs.

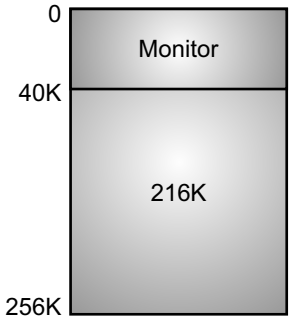



**Fig. 5.10**

| Job Queue | | |
|---|---|---|
| **Job** | **Memory** | **Time** |
| 1 | 60 K | 10 |
| 2 | 100 K | 5 |
| 3 | 30 K | 20 |
| 4 | 70 K | 8 |
| 5 | 50 K | 15 |

- Given the job queue and FCFS job scheduling, we can immediately allocate memory to job1, 2 and 3 creating the memory map of Fig. 5.11 (a).
- We have 26K of external fragmentation. Job2 will terminate at time 5 unit, releasing its memory allocation shown in Fig. 5.11 (b).
- We then return to our job queue and schedule the next job, job4 to produce the memory map of Fig. 5.11 (c). Job1 will terminate at time 10 unit to produce Fig. 5.11 (d) and job5 is then scheduled, producing Fig. 5.11 (e).

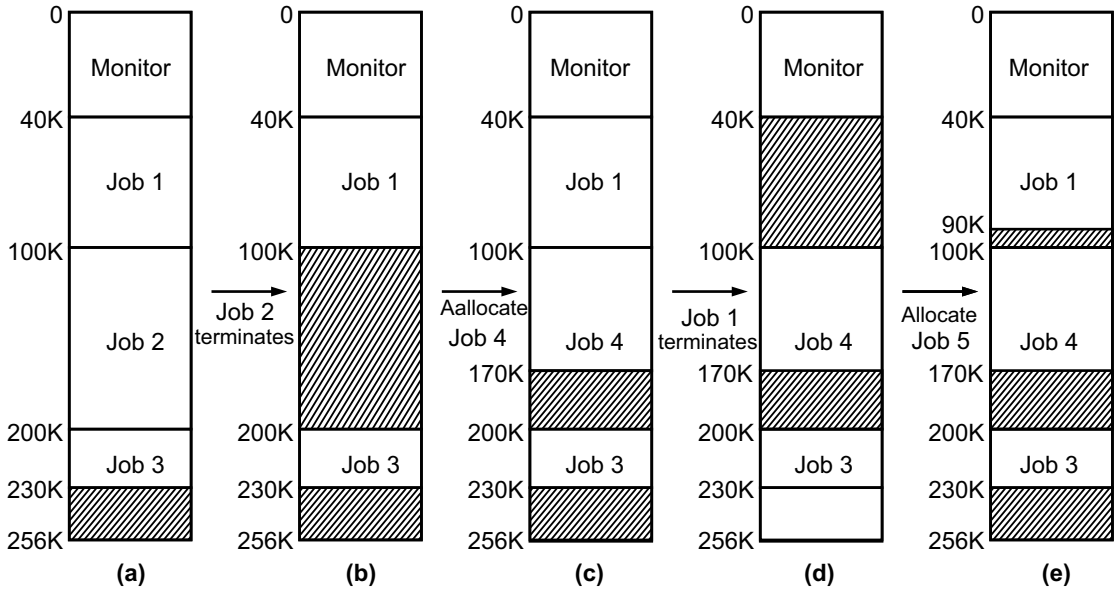



**Fig. 5.11: Example Memory Allocation and Job Scheduling for MVT**

**Advantages of MVT:**

1. It increases degree of multiprogramming due to which there is no unused space in the memory. Thus more processes can be loaded into the memory at the same time.
2. In MVT space in the main memory is allocated strictly according to the requirement of the process thus there is no chance of internal fragmentation.
3. In MVT the size of process can grow or shrink dynamically. In MVT the partition is allocated to the process dynamically thus the size of the process cannot be restricted because the partition size is decided according to the process size.

**Disadvantages:**

1. MVT suffers from external fragmentation.
2. Difficult to implement.

**Comparison of Static Memory Partitioning MFT (Multiprogramming with a Fixed number of Tasks) and MVT (Multiprogramming with a Variable number of Tasks):**

| Sr. No. | Factors | MFT | MVT |
|---|---|---|---|
| 1. | Region size | In this petitioning, memory is divided into several partitions of fixed size that size never changes. | In this petitioning, region or partition size is not fixed and can vary dynamically. |
| 2. | Process size | Cannot grow at run time. | Can grow or shrink at run time. |

***contd. ...***

| | | | |
|---|---|---|---|
| 3. | Degree of multi-programming | Fixed (i.e. number of partitions). | Dynamic. |
| 4. | Memory utilization | Poor. | Good. |
| 5. | Division of memory partitions | The division of memory into number of partitions and its size is made in the beginning prior to the execution of user programs and remains fixed thereafter. | The size and the number of partitions are decided during the run time by the operating system. |
| 6. | Degree of multi-programming | Limited and fixed. | Vary depending on program size. |
| 7. | Implementation | Difficult. | Easy. |
| 8. | Fragmentation | Suffers from internal fragmentation. | Suffers from external fragmentation. |
| 9. | Overhead | Eliminates the overhead of run-time allocation of partitions. | Runtime overhead of allocation of partitions. |
| 10. | Example | IBM 360, DOS etc. | IBM OS. |

**Dynamic Storage Allocation:** **[April 16, 18]**

- Disk space can be viewed as a large array of disk blocks. At any given time some of these blocks are allocated to files and others are free.
- Disk space seen as a collection of free and used segments, each segment is a contiguous set of disk blocks. An unallocated segment is called a Hole.
- The dynamic storage allocation problem is how to satisfy a request of size 'n' from a list of free holes. There are many solutions to this problem.
- The set of holes is searched to determine which hole is best to allocate. The most common strategies used to select a free hole from the set of available holes are first fit, best fit and worst fit.
  1. **First Fit:** Allocate the first hole (or free block) that is big enough for the new process. Searching can start either at the beginning of the set of holes or where the previous first fit search ended. We can stop searching as soon as we find a large enough free hole. First fit is generally faster.
  2. **Best Fit:** Allocate the smallest hole that is big enough. We search the entire list, unless the list is kept ordered by size. This strategy produces the smallest left over hole.
  3. **Worst Fit:** Allocate the largest hole. Again we must search the entire list, unless it is sorted by size.
- First fit and best fit are better than worst fit in both time and storage utilization. First fit is generally faster.

**Example:** Consider the following memory map and assume a new process P4 comes with a memory requirement of 3 KB. Locate this process.

(i) First fit algorithm allocates from the 10 KB block.

(ii) Best fit algorithm allocates from the 4 KB block.

(iii) Worst first algorithm allocates from the 16 KB block.

| OS | |
|---|---|
| P1 | |
| <free> | 10 KB |
| P2 | |
| <free> | 16 KB |
| P3 | |
| <free> | 4 KB |

**Solution:** New memory arrangements with respect to each algorithm will be as follows:

| OS | |
|---|---|
| P1 | |
| P4 | |
| <free> | 7 KB |
| P2 | |
| <free> | 16 KB |
| P3 | |
| <free> | 4 KB |

**First Fit**

| OS | |
|---|---|
| P1 | |
| <free> | 10 KB |
| P2 | |
| <free> | 16 KB |
| P3 | |
| P4 | |
| <free> | 1 KB |

**Best Fit**

| OS | |
|---|---|
| P1 | |
| <free> | 10 KB |
| P2 | |
| P4 | |
| <free> | 13 KB |
| P3 | |
| <free> | 4 KB |

**Worst Fit**

## 5.3.3 Fragmentation [April 16, 17, 18, Oct. 17]

- As processes are loaded and removed from memory, the free memory space is broken into little pieces. It happens after sometimes that processes cannot be allocated to memory blocks considering their small size and memory blocks remains unused. This problem is known as fragmentation.
- Fragmentation is of two types namely, external fragmentation and internal fragmentation.
- External fragmentation arises when free memory, areas existing in systems are too small to be allocated to processes. Internal fragmentation exists when memory allocated to a program is not fully utilized by it.

- In **internal fragmentation**, the process is allocated a memory block of size more than the size of that process. Due to this some part of the memory is left unused and this cause internal fragmentation.
- When internal fragmentation occurs, a process that needs 57 bytes of memory, for example, may be allocated a block that contains 60 bytes, or even 64.
- The extra bytes that the client does not need go to waste, and over time these tiny chunks of unused memory can build up and create large quantities of memory that cannot be put to use by the allocator.
- Because all of these useless bytes are inside larger memory blocks, the fragmentation is considered internal.

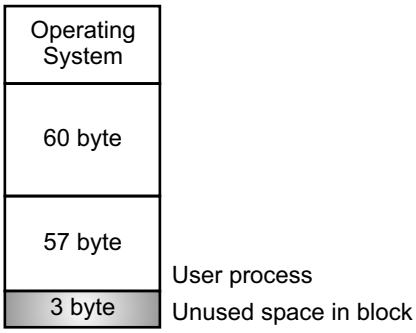



**Fig. 5.12: Internal Fragmentation**

- In **external fragmentation** the total memory space is enough to satisfy a request or to reside a process in it, but it is not contiguous, so it cannot be used.
- For example, there is a hole of 20K and 10K is available in multiple partition allocation schemes. The next process request for 30K of memory.
- Actually 30K of memory is free which satisfy the request but hole is not contiguous. To there is an external fragmentation of memory

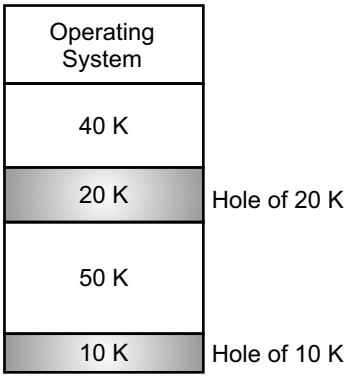



**Fig. 5.13: External Fragmentation**

- Compaction is a method used to overcome the external fragmentation problem. All free blocks are brought together as one large block of free space. **[April 19]**
- The collection of free space from multiple non-contiguous blocks into one large free block in a system's memory is called compaction.

- The goal compaction is to shuffle the memory contents to place all free memory together in one large block. For example the memory map of (e) can be compacted as shown in Fig. 5.14.
- The three holes of sizes 10K, 30K and 26K can be compacted into one hole of 66K. Compaction is not always possible.
- In Fig. 5.14 we moved job4 and job3 for these programs to be able to work in their new locations, all internal addresses must be relocated.
- Compaction is possible only if relocation is dynamic, at execution time, using base and limit registers.
- The simplest compaction algorithm is to simply move all jobs towards one end of memory, all holes move in the other direction, producing on large hole of available memory.
- Compaction can be quite expensive. Compaction changes the allocation of memory to make free space contiguous and hence useful. Compaction also consumes system resources.

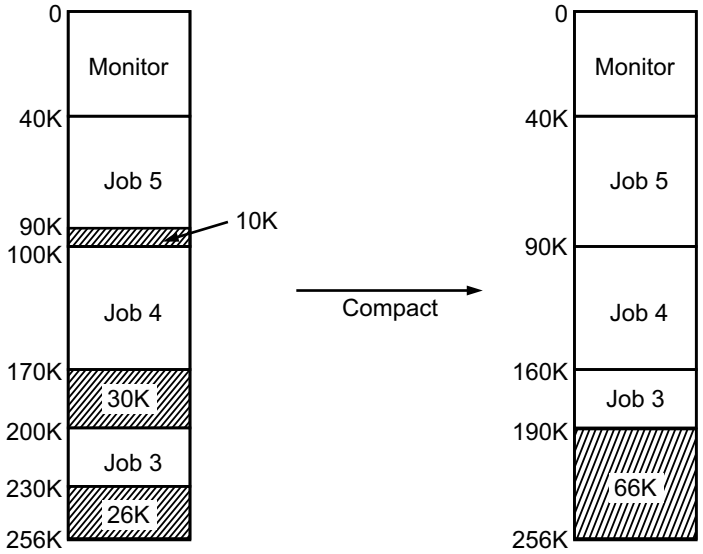



**Fig. 5.14: Compaction**

**Difference between Internal Fragmentation and External Fragmentation:**

| Sr. No | Internal Fragmentation | Internal Fragmentation |
|---|---|---|
| 1. | It occurs when fixed sized memory blocks are allocated to the processes. | It occurs when variable size memory space is allocated to the processes dynamically. |
| 2. | In this fragmentation the fixed size partition is assigned to a process whose size is less than the size of the partition due to which the rest of the space in the partition becomes unusable. | In this fragmentation the memory space in the system can easily satisfy the requirement of the processes, but this available memory space is non-contiguous, so it cannot be utilized further. |

***contd. ...***

| | | |
|---|---|---|
| 3. | It mainly refers to the unused space in the partition that resides in the allocated region | It mainly refers to the unused blocks of the memory that are not contiguous and hence are unable to satisfy the requirements of the process. |
| 4. | Internal fragmentation can be eliminated by allocating memory to processes dynamically. | External fragmentation can be eliminated by compaction technique. |

## 5.4 PAGING

- Paging is a memory management scheme that permits the physical address space of a process to be non-contiguous.
- Paging is a memory management technique by which a computer stores and retrieves data from secondary storage for use in main memory.

### 5.4.1 Basic Method [Oct. 18]

- The basic method for implementation of paging involves breaking physical memory into fixed-sized blocks called frames (f) and break logical memory into blocks of the same size called pages (p).
- When a process is to be executed its pages are loaded in to available frames from backing store. The backing store is also divided in to fixed-sized blocks of same size as the memory frames.
- The hardware support for paging is shown in Fig. 5.15.

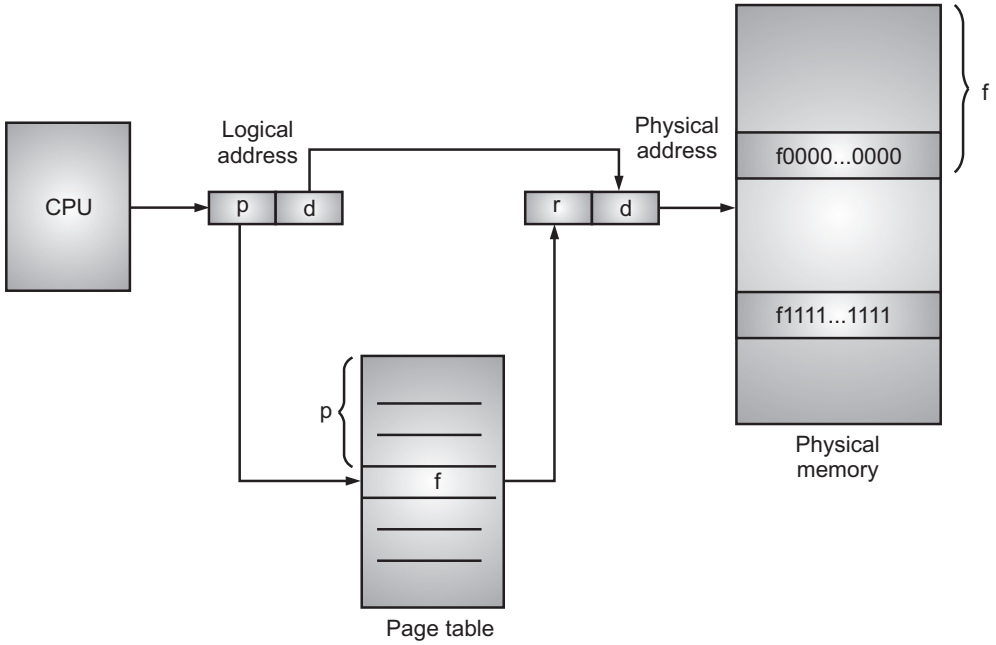



**Fig. 5.15: Paging Hardware**

- Every address generated by the CPU is divided into two parts namely Page number (p) and Page offset (d).
- The page number (p) is used as an index into a Page Table. A page table is used to map a page on a frame. Page table has an entry for each page.
- The page table has the base address of each page in physical memory; which is combined with page offset (d) to define the physical memory address which is sent to memory unit.
- The paging model of memory is shown in Fig. 5.16.

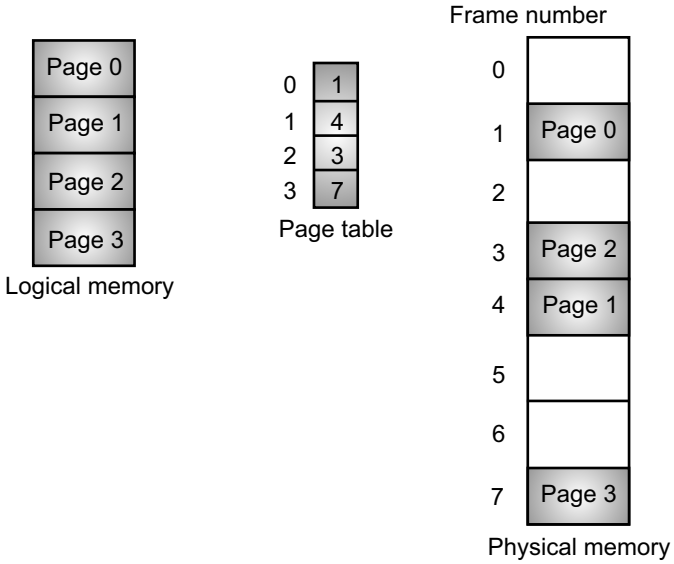
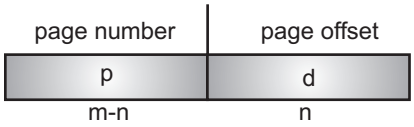



**Fig. 5.16: Paging Model of Logical and Physical Memory**

- Page size and frame is same and decided by hardware and is generally a power of 2 varying between 512 bytes and 16MB per page.
- If the size of logical address is $2^m$ and page size is $2^n$, then the high-order m-n bits of a logical address designate the page number and n low-order bits designate the page offset. Thus logical address is as follows:




Where, p is an index into page table and d is displacement within page.

- For example, consider the memory as shown in Fig. 5.14 using 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 i.e. $[(5 \times 4) + 0]$.
- Logical address 3 is page 0 and offset 3 maps to physical address 23 i.e. $[(5 \times 4) + 3]$.
- 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 i.e. $[(6 \times 4) + 0]$.

- 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 i.e. $[(2 \times 4) + 1]$.

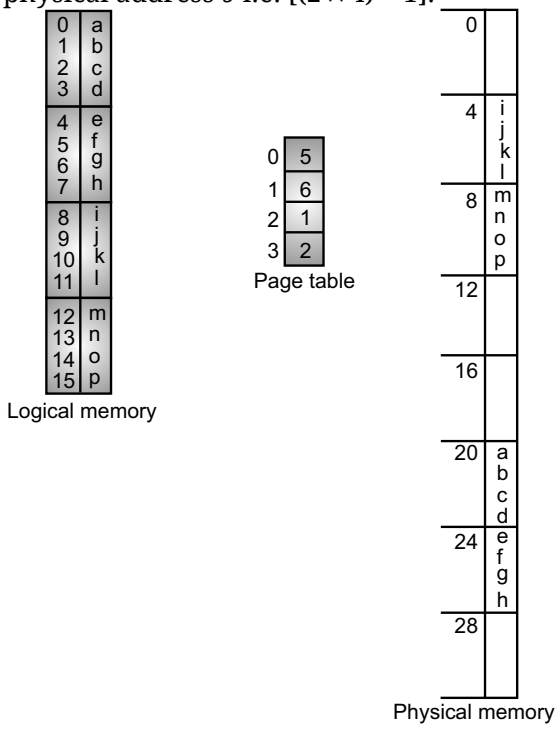



**Fig. 5.17: Paging Example for a 32-word Memory with 4-word Pages**

- Fig. 5.18 shows allocation of free frames.

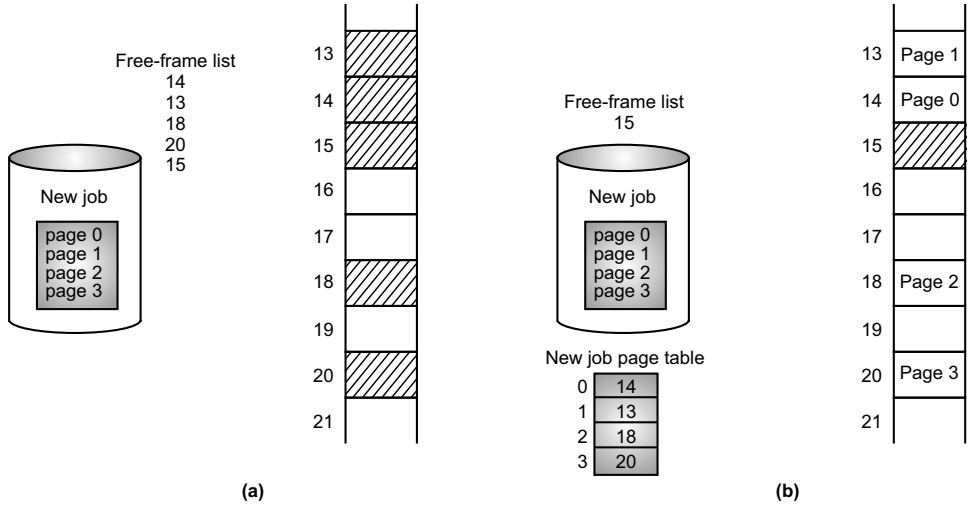



**Fig. 5.18: Allocation of Free Frames (a) Before and (b) After**

**Advantages of Paging:** **[Oct. 17]**

1. Paging allows jobs to be allocated in non-contiguous memory locations.
2. In paging memory used more efficiently.

3. Paging does not require any support for dynamic relocation because paging itself is a form of dynamic relocation.
4. Paging support higher degree of multiprogramming.

**Disadvantages of Paged Memory Allocation:**

1. Page address mapping hardware usually increases the cost of the operating system.
2. Internal fragmentation still exists, though in last page.
3. Paging requires the entire job to be stored in memory location.
4. In paging the size of page is crucial (not too small, not too large).

## 5.4.2 Hardware Support [April 13]

- Every access to any of page has to go through the page table. Therefore, implementation of the page table directly affects the performance of this kind of memory management.
- Though there are different ways on which page table can be implemented, generally the method depends upon the size of the page table.

1. **Page Table Implemented as Dedicated 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 efficient. In dedicated registers implementation 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.
2. **Page Table Implemented in the Main Memory:**
   - This method is used when page table is big or large. Page table is stored in the main memory.
   - PTBR (Pointer To Base Register) points to the base address of page table. The PTBR is mainly a processor register and is managed by the operating system.
   - In case, location of the page table is changed in the memory, only the contents of PTBR are changed.
   - This method gives reasonably slow access. The memory access is to be done twice, once for accessing page table for getting frame number and secondly for accessing actual frame.
3. **Page Table Implemented in Associative Registers:**
   - The problem with page table implementation in the main memory approach is the time required to access memory location.
   - This problem can be solved by using special, small, fast lookup hardware cache, called a Translation Look-aside Buffer (TLB) or associative registers as shown in Fig. 5.19.

- Each entry in the TLB consists of two parts namely Key and Value. When associative memory is presented with an item, the item is compared with all keys simultaneously. If the item is found, the corresponding value field is returned.

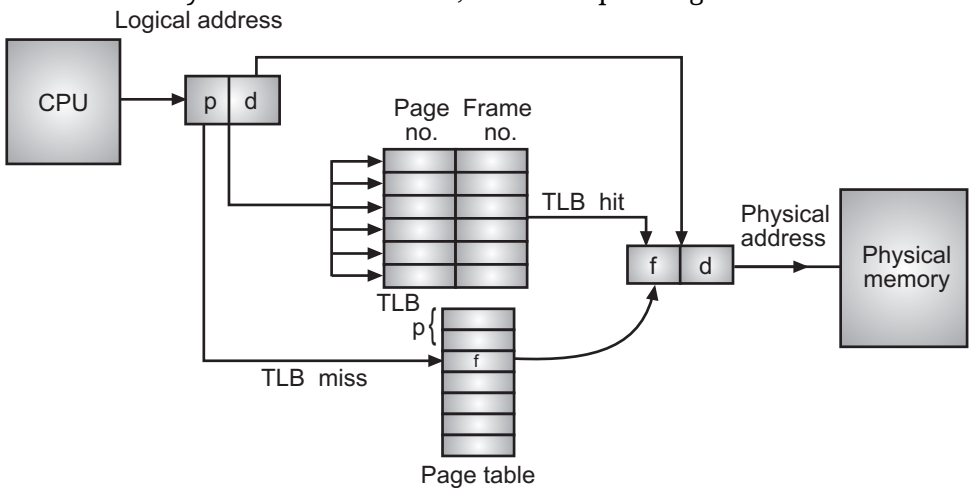



**Fig. 5.19: Paging hardware with TLB**

- The TLB is used with page tables in the following way:
  - The TLB contains only a few of the page-table entries. When a logical address is generated by the CPU, its page number is presented to the TLB. If page number is found, its frame number is immediately available and is used to access memory.
  - If the page number is not in the TLB (TLB miss) a memory reference to the page table must be made. In addition, we add the page number and frame number into TLB.
  - If the TLB already full, the OS have to must select one for replacement. Some TLBs allow entries to be wire down, meaning that they cannot be removed from the TLB, for example kernel code.
  - Some TLBs store address-space identifiers (ASIDs) in each TLB entry. An ASID uniquely identifies each process and is used to provide address-space protection for that process.
  - When the TLB attempts to resolve virtual page numbers, it ensures that the ASID for the currently running process matches the ASID associated with the virtual page. If the ASIDs do not match, the attempt is treated as a TLB miss.
  - In addition to providing address-space protection, an ASID allows the TLB to contain entries for several different processes simultaneously.
  - If the TLB does not support separate ASIDs, then every time a new page table is selected (for instance, with each context switch), the TLB must be flushed (or erased) to ensure that the next executing process does not use the wrong translation information.

- Otherwise, the TLB could include old entries that contain valid virtual addresses but have incorrect or invalid physical addresses left over from the previous process.
- The percentage of times that a particular page number is found in the TLB is called as hit ratio.
- An 80-percent hit ratio means that we find the desired page number in TLB 80 percent of time. If it takes 20 nanoseconds to search the TLB and 100 nanoseconds to access memory, then mapped memory access takes 120 nanoseconds when page is found in TLB.
- If TLB miss is there, then 220 nanoseconds will require for desired bytes. To find the effective memory access-time, weight each case by probability.
- Effective memory access time = $0.8 \times (100 + 20) + 0.2 \times (120 + 100) = 140$ nanoseconds.
- If our hit ratio is 98%, the effective memory access time = $0.98 \times (100 + 20) + 0.02 \times (120 + 100) = 122$ nanoseconds.

## 5.4.3 Protection

- Memory protection in a paged environment is supported by protection bits associated with each frame. Normally, these bits are kept in page table.
- One bit can define a page to be read-write or read only. Every reference to memory goes through page table to find correct frame number.
- At same time, protection bits can be checked to verify that no writes are being made to a read only page. An attempt to write to a read only page causes hardware trap to the operating system.
- We can easily expand this approach to provide a finer level of protection. We can create hardware to provide read-only, read-write, or execute-only protection or by providing separate protection bits for each kind of access.
- We can allow any combination of these accesses. Illegal attempts will be trapped to the operating system.
- One additional bit is attached to each entry in the page table i.e., a valid-invalid bit. Illegal addresses are trapped by using the valid-invalid bit.
  - When bit is set to “valid” indicates that the associated page is in the process’ logical address space, and is thus a legal or valid page.
  - When bit is set to “invalid” indicates that the page is not in the process’s logical address space.
- Consider an example as shown in Fig. 5.20.
  - Address space is 14 bits, (Address is from 0 to 16,383).
  - Page size is 2K (2048 words)
  - Program size is 0 to 10,468 (pages from 0 to 5).

- Total number of pages that can be addressed 16,383/2048 = 8.
- Any attempt to generate an address in pages 6 or 7, will find that valid-invalid bit is set to invalid, trap will be generated.

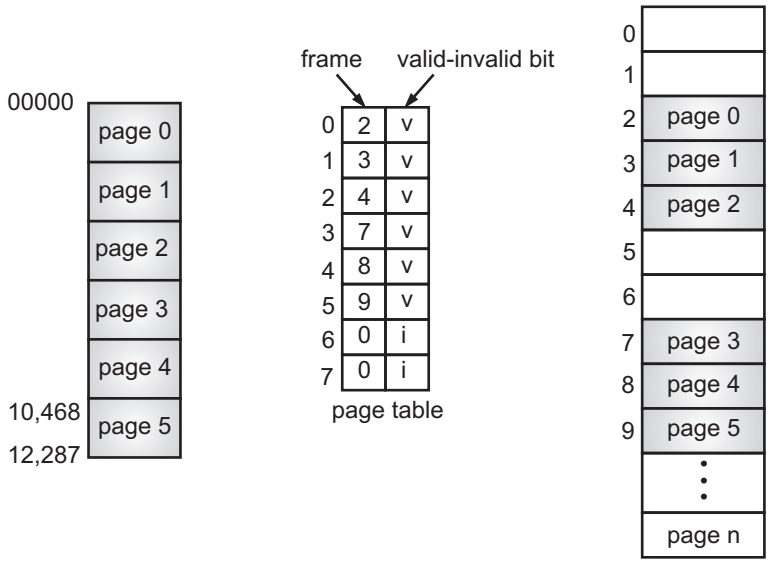



**Fig. 5.20: Valid (v) or Invalid Bit (i) in a Page Table**

## 5.4.4 Shared Pages [Oct. 17]

- An advantage of paging is the possible of sharing common code. This consideration is important for time-sharing environment.
- For example, consider a system that supports 40 users, each of whom executes a text editor. If the text editor consists of 150 KB of code and 50 KB of data space, we need (150 KB + 50 KB) × 40 = 8000 KB memory space to support the 40 users.
- If the code is reentrant code (or pure code), it can be shared, as shown in Fig. 5.21. Reentrant code or pure code is non-self modifying code; it never changes during execution.
- Thus, two or more processes can execute the same code at the same time. Each process has its own copy of registers and data storage to hold data for process executions. The data for two different processes will vary for each process.
- Only one copy of the editor need be kept in physical memory. Each user's page table maps onto the same physical copy of the editor, but data pages are mapped onto different frames.
- Thus, to support 40 users, we need only one copy of the editor (150 KB), plus 40 copies of the 50 KB of data space per user.
- To support 40 users we need only 150 KB + 40 × 50 KB = 2150 KB memory space. The total space required is now 2,150 KB instead of 8,000 KB - a significant savings.

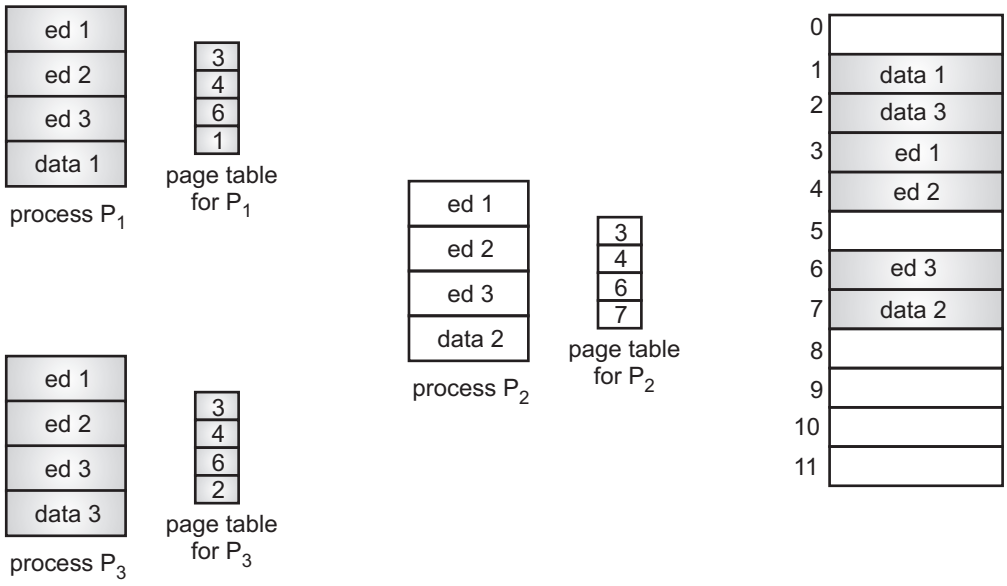



**Fig. 5.21: Sharing of Code in a Paging Environment**

- Other heavily used programs can also be shared like compilers, window systems, run-time libraries, database systems, and so on.
- To be sharable, the code must be reentrant. The read-only nature of shared code should not be left to the correctness of the code; the operating system should enforce this property.

## 5.5 SEGMENTATION [April 18]

- An important aspect of memory management approach in operating system that became unavoidable with paging is the separation of the user's view of memory and the actual physical memory.
- The user's view of memory is not same as actual physical memory in paging memory management scheme.
- The user's view is mapped onto physical memory which allows differentiation between physical and logical memory.
- Like paging, segmentation is also a memory management scheme that supports user's view of memory.

### 5.5.1 Basic Method

- Segmentation is a non-contiguous memory allocation memory allocation technique which supports the user's view of memory.
- Segmentation divides the user program and the data associated with the program into the number of segments.
- A segment is defined as, a logical grouping of instructions. Each segment is actually a different logical address space of the program.

- A segment is a logical unit such as main program, procedure, function, method, object, local variables, global variables, common block, stack, symbol table, arrays etc., as shown in Fig. 5.22.
- Thus, user prefers to view memory as collection of variable-sized segments, with no necessary ordering among segments.
- A process in segmentation is divided into segments. In segmentation, the entire logical address space is considered as a collection of segments with each segment having a number and a length.
- The length of a segment may range from 0 to some maximum value as specified by the hardware and may also change during the execution.
- For simplicity, segments are numbered and referred by segment number rather than segment name. Thus, Logical address is contains, <segment number, offset>.
- The segment number tells the specific segment of the process from which the CPU wants to read the data. The segment offset tells the exact word in that segment which the CPU wants to read.

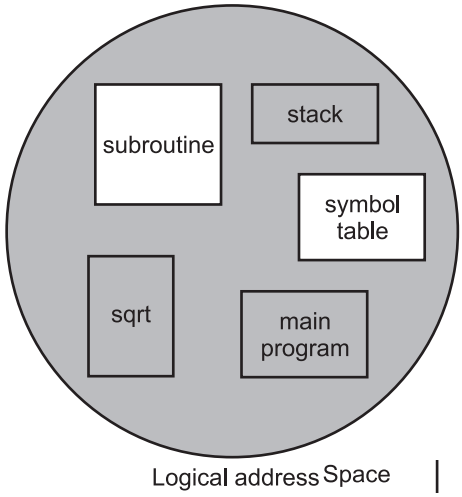



**Fig. 5.22: User's View of Program**

## 5.5.2 Segmentation Hardware [April 18, Oct. 18]

- Fig. 5.23 shows segmentation hardware. Each segment in segmentation has a name and a length.
- The addresses specify both the segment name and the offset within the segment. The user therefore specifies each address by two quantities namely, a segment name and an offset.
- A logical address consists of two parts a segment number 's' and an offset into that segment 'd'.
- Each entry of segment table has a segment base and a segment limit. The segment base contains the starting physical address where segment resides in memory while segment limit specifies the length of segment.

- A segment table stores the base address of each segment in the main memory. The use of segment table is shown in Fig. 5.23.
- A register called Segment Table Base Register (STBR) which holds the base value of the segment table. The segment table is also stored in the main memory itself.

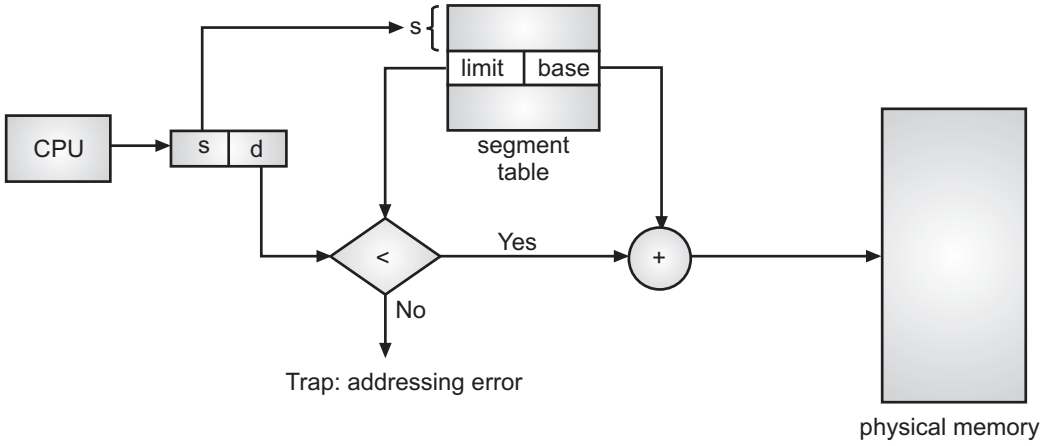



**Fig. 5.23: Segmentation Hardware**

- Example, consider the situation shown in Fig. 5.24. We have five segments numbered from 0 through 4. The segments are actually stored in physical memory as shown in Fig. 5.24.
- The segment table has a separate entry for each segment, giving the beginning address of the segment in physical memory (the base) and the length of that segment (the limit) for example segment 2 is 400 words long, beginning of location 4300.

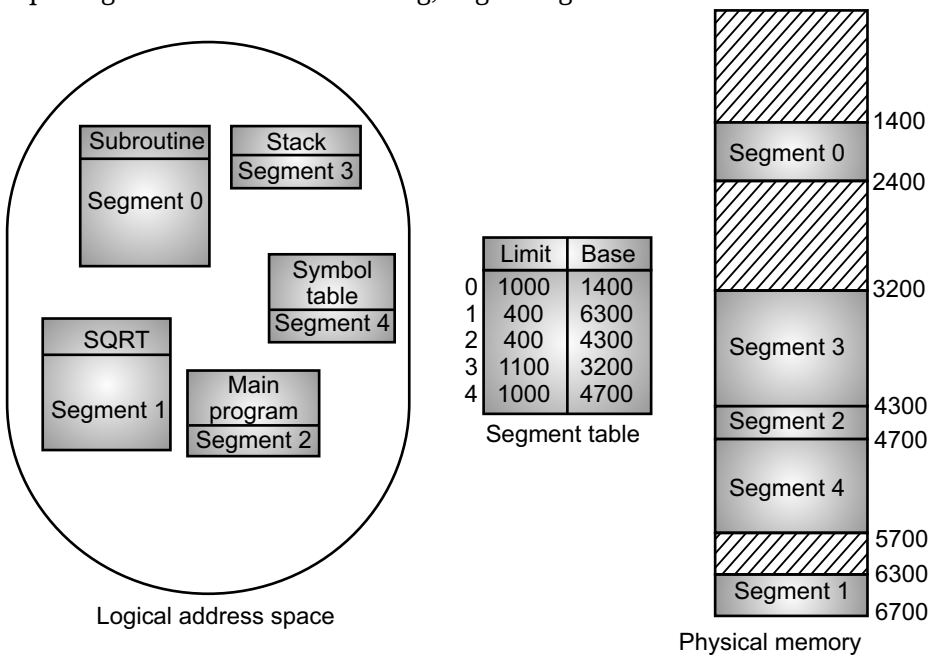



**Fig. 5.24: Example of Segmentation**

- Thus reference to word 53 of segment 2 is mapped on to location 4300 + 53 = 4353. A reference to segment 3, word 852 is mapped to 3200 (the base of segment 3) + 852 = 4052.
- A reference to word 1222 of segment 0 would result in a trap to the operating system, since this segment is only 1000 words long.

**Advantages of Segmentation:**

1. Segmentation eliminates fragmentation problem.
2. Segmentation allows dynamically growing segments.
3. It provides dynamic linking and loading.
4. Segmentation facilitates shared segments, (data area and procedures).
5. It enforced control access (i.e. Read, Write and Executed).
6. It provides a convenient way of organizing programs and data.

**Disadvantages of Segmentation:**

1. In segmentation the address translation i.e. conversion from logical address to physical address is not a simple function, as regards paging.
2. Considerable compaction overhead is incurred in order to support dynamic segment growth and eliminate fragmentation.
3. Increased complexity in the operating system.
4. Increased hardware cost processor overhead for address mapping.
5. There is a difficulty in managing variable size segments on the secondary storage.
6. In segmentation the maximum size of segment is limited by the size of main memory.

**Difference between Paging and Segmentation:**

| Sr. No. | Paging | Segmentation |
|---|---|---|
| 1. | In paging, the main memory is partitioned into page frames (or blocks). | In segmentation, the main memory is partitioned into segments. |
| 2. | In paging, the logical address space is divided into pages by the compiler or MMU (Memory Management Unit). | In segmentation, the logical address space is divided into segments as specified by the user/programmer. |
| 3. | The OS maintains a page map table for mapping between frames and pages. | The OS maintains a segment map table for mapping purpose. |
| 4. | Paging suffers from internal frag-mentation or page breaks. | Segmentation suffers from external fragmentation. |
| 5. | Paging does not support the user's view of memory. | Segmentation supports user's view of memory. |

*contd. ...*

| | | |
|---|---|---|
| 6. | The hardware memory management unit maps pages to frames. | Segments are areas of memory that usually correspond to a logical grouping of information such as a code procedure or a data array. |
| 7. | In paging, processor uses page number, offset to calculate absolute address. | In segmentation, processor uses segment number, offset to calculate absolute address. |
| 8. | Paging is invisible to the user. | Segmentation is visible to the user. |
| 9. | Paging is faster than segmentation. | Segmentation is slower than paging. |

## 5.6 VIRTUAL MEMORY MANAGEMENT

- Virtual memory is a technique that allows the execution of process that is not completely in main memory.
- It abstracts main memory into an extremely large, uniform array of storage, separating logical memory as viewed by the user from physical memory.
- Virtual memory is a memory management technique that provides an idealized abstraction of the storage resources that are actually available on a given machine which creates the illusion to users of a very large (main) memory.
- Virtual memory allows processes to easily share files and address spaces and provides an efficient mechanism for process creation.
- Virtual address space of a process refers to a logical view of how a process is stored in memory.
- A virtual address space is the set of ranges of virtual addresses that an operating system makes available to a process.
- A process begins at a certain logical address-say address 0 and exists in contiguous memory as shown in Fig. 5.25.
- As shown in Fig. 5.25, heap is allowed to grow upward direction in memory as it is used for dynamic memory allocation and stack is allowed to grow in downward direction through function calls.
- The large blank space between heap and stack is virtual address space (Refer Fig. 5.25). Virtual address space includes holes that are known as sparse address spaces.

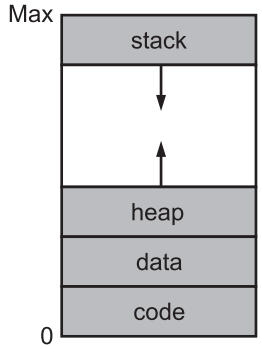



**Fig. 5.25: Virtual Address Space**

- Virtual memory technique offer following benefits:
  1. A program would no longer be constrained by the amount of physical memory. So programmer can concentrated on the problem to be implemented.
  2. More programs can reside in memory at same time, so degree of multi-programming is increased. CPU utilization and throughput also increased.
  3. Less I/O would be needed to load or swap each user program.
  4. In addition to separating logical memory from physical memory, virtual memory allows files and memory to be shared.
  5. System libraries can be shared by several processes through mapping of the shared object into virtual address space.
  6. Similarly virtual memory enables processes to share memory.
  7. It provides an efficient mechanism for process creation.

## 5.6.1 Background

- A computer can address more memory than the amount physically installed on the system. This extra memory is actually called virtual memory and it is a section of a hard disk that's set up to emulate the computer's RAM i.e. main memory.
- Virtual memory serves following two purposes:
  1. It allows us to extend the use of physical memory by using disk.
  2. It allows us to have memory protection, because each virtual address is translated to a physical address.
- Virtual memory is the separation of user logical memory from physical memory. This separation allows an extremely large virtual memory to be provided for programmers when only a smaller physical memory is available, (Refer Fig. 5.26).

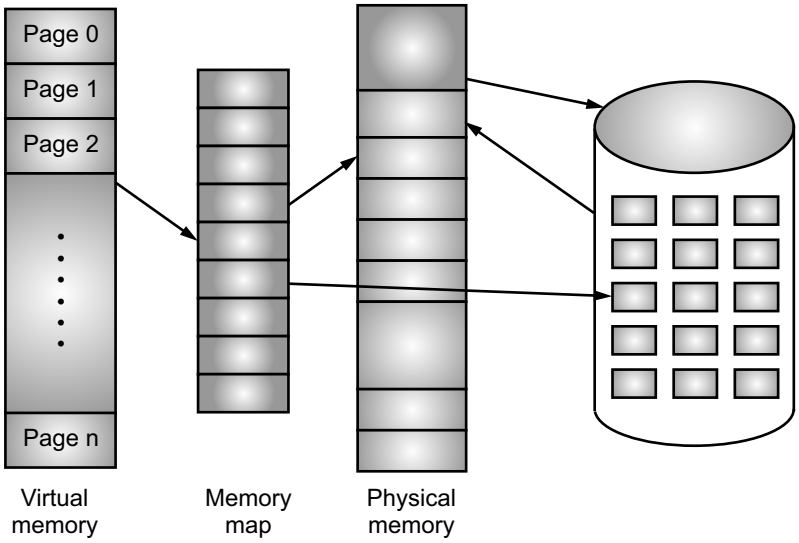



**Fig. 5.26: Virtual Memory that is larger than Physical Memory**

- Virtual memory makes the task of programming much easier because the programmer no longer needs to worry about the amount of physical memory available.
- Virtual memory is commonly implemented by demand paging. It can also be implemented by demand segmentation.

## 5.6.2 Demand Paging [April 17, 19]

- Demand paging is a method of virtual memory management.
- With demand-paged virtual memory, pages are only loaded when they are demanded during program execution; pages that are never accessed are thus never loaded into physical memory.
- A demand-paging system is similar to a paging system with swapping, where processes reside in secondary memory (usually a disk) as shown in Fig. 5.27. When we want to execute a process, we swap it into memory.
- Rather than swapping the entire process into memory, however, we use a swapper is also known as lazy swapper or pager. A lazy swapper never swaps a page into memory unless that page will be needed.
- When a process is to be swapped in, the pager guesses which pages will be used before the process is swapped out again. Instead of swapping in a whole process, the pager brings only those necessary pages into memory.
- Thus, it avoids reading into memory pages that will not be used in anyway, decreasing the swap time and the amount of physical memory needed.
- A swapper manipulates entire processes, whereas a pager is concerned with the individual pages of a process. We thus use pager, rather than swapper, in connection with demand paging.

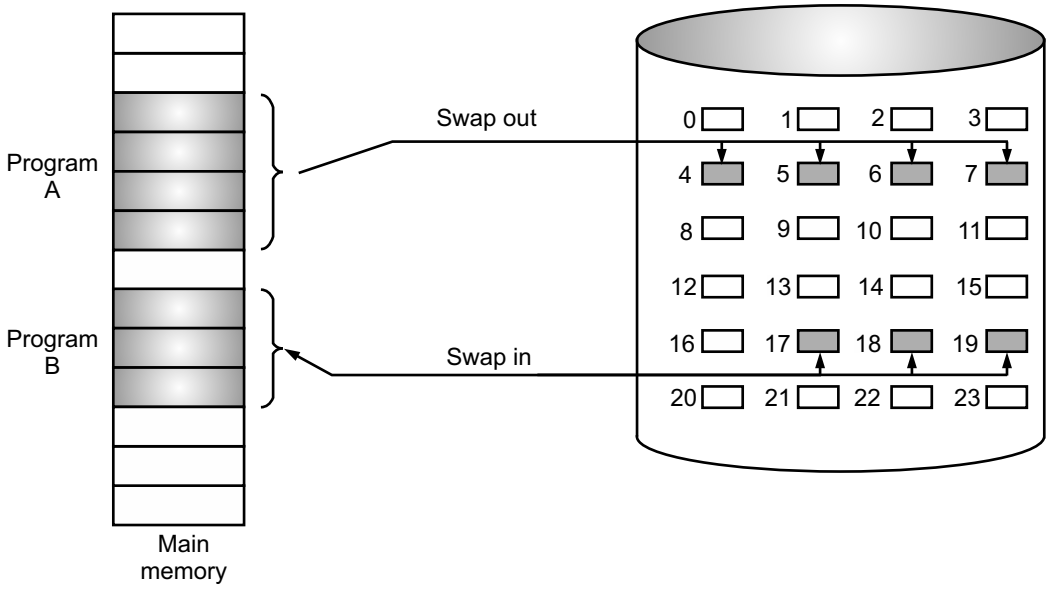



**Fig. 5.27: Transfer of a Paged Memory to Contiguous Disk Space**

- The hardware to support demand paging is same as hardware for paging and swapping:
  1. **Page Table:** This table has the ability to mark an entry invalid through a valid-invalid bit or special value of protection bits.
  2. **Secondary Memory:** It is usually high speed disk; also known as swap device. It holds those pages that are not in main memory.
- To distinguish between the pages that are in memory and the pages that are on the disk, valid-invalid bit is used.
- When this bit is set to "valid", the associated page is legal and in main memory. When this bit is set to "invalid", page may not be valid or page is valid but currently on the disk as shown in the Fig. 5.28.

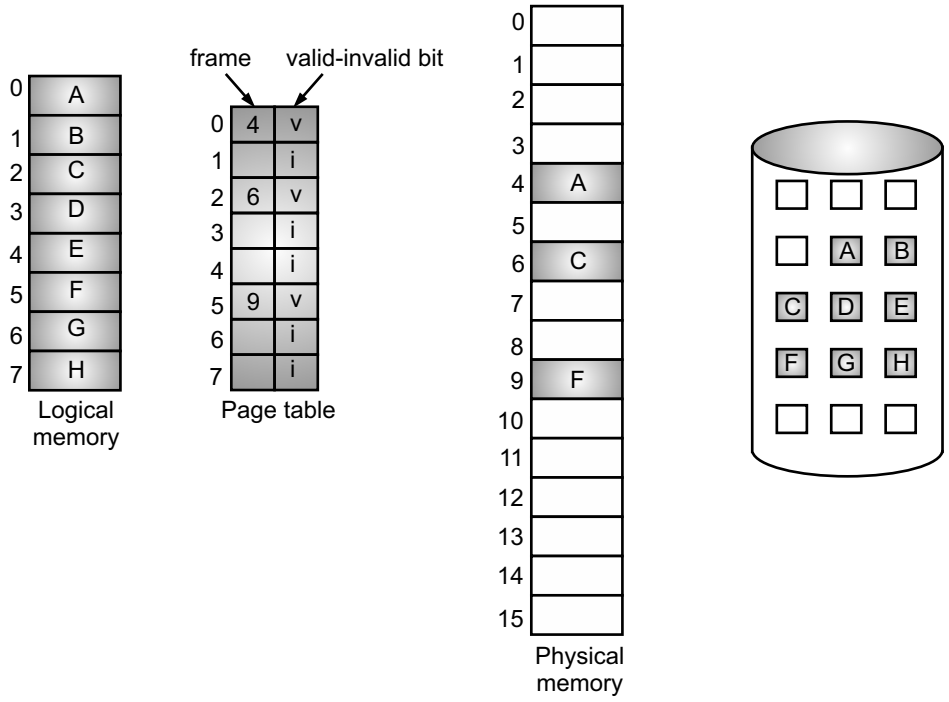



**Fig. 5.28: Page Table with Valid-Invalid Bit**

- Whenever, process tries to access a page which is not into memory, and then page fault occurs.
- The paging hardware, which translates the address through page table, will notice that invalid bit is set and cause trap to the operating system.
- A page fault occurs when a program accesses a page that has been mapped in address space, but has not been loaded in the physical memory.
- When the page (data) requested by a program is not available in the memory, it is called as a page fault.

- The steps for handling page fault are as follows and shown in Fig. 5.29.

  **Step 1:** Check an internal table for the process, to determine whether the reference was a valid or it was an invalid memory access.

  **Step 2:** If the reference was invalid, the process is terminated. If it was valid, but we not yet brought in memory, we now page it in.

  **Step 3:** Find a free frame, possibly from the free frame list.

  **Step 4:** Schedule a disk operation to read the desired page into the newly allocated frame.

  **Step 5:** When the disk read operation is complete, the process's page table is updated with the new frame number, and the invalid bit is changed to indicate that this is now a valid page reference. The page is now in memory.

  **Step 6:** Restart the instruction that was interrupted by the illegal address trap. The process can now access the page as though it had always been in memory.

- Therefore, the operating system reads the desired page into memory and restarts the process as though the page had always been in memory.

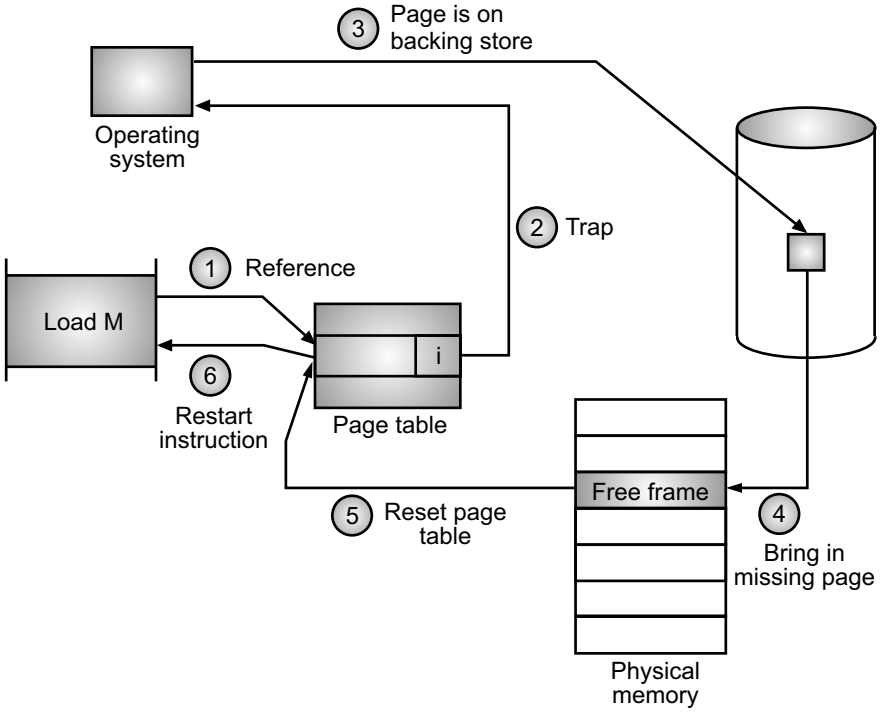



**Fig. 5.29: Steps in Handling a Page Fault**

**Pure Demand Paging:**

- In demand paging, when process starts execution with no pages in memory, when operating system sets the instruction pointer to the first instruction of the process, process immediately faults for the page.

- After this page is brought into memory, the process continues to execute, faulting as necessary until every page that it needs is in memory. Thus, it never brings a page into memory until it is required.

**Advantages of Demand Paging:**

1. Large virtual memory.
2. More efficient use of memory.
3. Unconstrained multiprogramming. There is no limit on degree of multi-programming.

**Disadvantages of Demand Paging:**

1. Number of tables and amount of processor overhead for handling page interrupts are greater than in the case of the simple paged management techniques.
2. Due to the lack of an explicit constraint on a job address space size.

## 5.6.3 Performance of Demand Paging

- Demand paging can significantly affect the performance of a system. For most computer systems, memory access time ranges from 10 to 200 nanoseconds.
- As long as we have no page faults, the effective access time is equal to memory access time. If page fault occurs, we must first read the relevant page from disk and access desired word.
- The performance of demand paging is often measured in terms of the effective access time.
- Let p be probability of page fault.

  If p = 0 then no page fault occurs.

  If p = 1, every reference is a fault.

  Effective Access Time (EAT) = $(1 - p) \times$ Memory access + p (Page Fault Overhead + Swap Page Out + Swap Page In + Restart Overhead)
- To compute the effective access time, we must know how much time is needed to service a page fault.
- Three major components affect the page fault service-time:
  1. Service the page-fault interrupt,
  2. Read in the page, and
  3. Restart the process.
- Let us assume that Memory access time = 200 nanoseconds.
- Average page-fault service time = 8 milliseconds

$$\begin{aligned} EAT &= (1 - p) \times 200 + p \text{ (8 milliseconds)} \\ &= (1 - p) \times 200 + p \times 8{,}000{,}000 \\ &= 200 + p \times 7{,}999{,}800 \end{aligned}$$

- If one access out of 1,000 causes a page fault, then EAT = 8.2 microseconds. This is a slowdown by a factor of 40 because of demand paging. Thus effective access time is directly proportional to page-fault rate.

- Effective access time can be reduced by:
  1. Keeping page-fault rate low,
  2. Using better page replacement algorithm, and
  3. Allocating more pages to a process.

## 5.6.4 Page Replacement

- Page replacement happens when a requested page is not present in the main memory and the available space is not sufficient for allocation to the requested page.
- In multiprogramming environment, it may happen that there is no free frame available on the free frame list and valid page fault occurs as shown in Fig. 5.30.
- A page fault occurs when a page referenced by the CPU is not found in the main memory. The operating system has following options at this point:
  - The operating system could terminate the user process.
  - Demand paging is an attempt to improve CPU utilization and throughput.
  - Users should not be aware that processes are running on a paged system.
- To bring user page into memory, operating system swaps out some process by freeing its frames. The process of freeing frame is known as page replacement.
- The frame which is selected for replacement is known as victim frame. The major issue is how to select victim frame, normally frame that is not currently being used and free it.

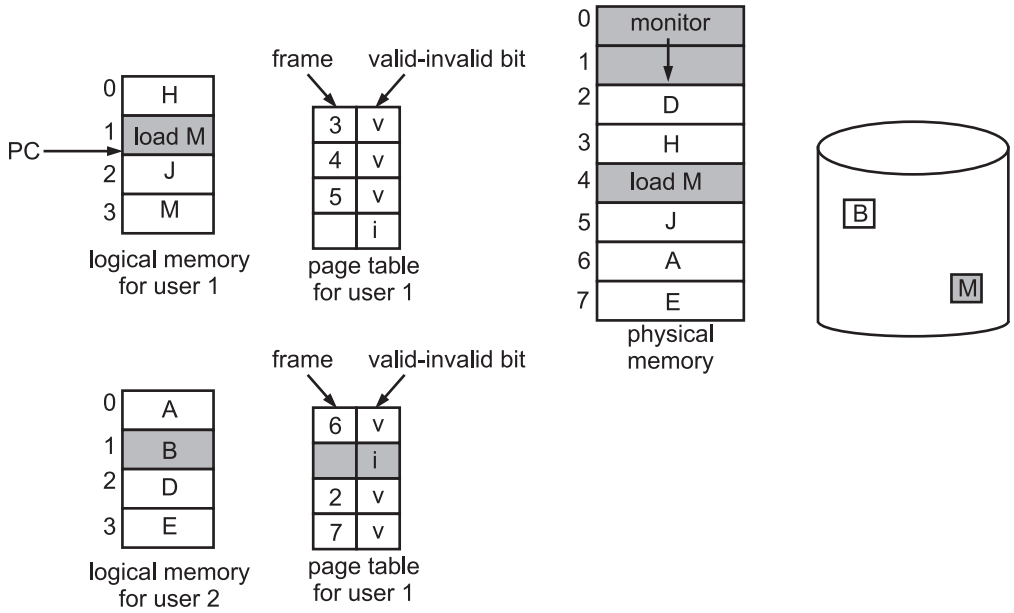



**Fig. 5.30: Need for Page Replacement**

- 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 approach of page replacement is, if no frame is free, we find one that is not currently being used and free it.
- We can free a frame by writing its contents to swap space and changing the page table (and all other tables) to indicate that the page is no longer in memory.
- We can now use the freed frame to hold the page for which the process faulted. We modify the page-fault service routine to include page replacement.
- Fig. 5.31 shows steps in page replacement.

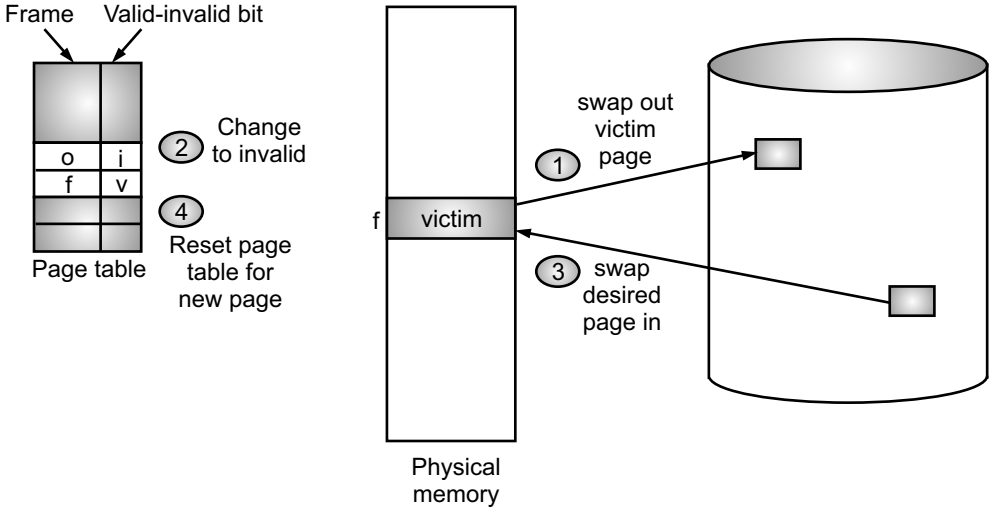



**Fig. 5.31**

- Steps in page replacement are listed below:
  1. Find the location of the desired page on the disk.
  2. Find a free frame:
     (i) If there is a free frame, use it.
     (ii) If there is no free frame, use a page-replacement algorithm to select a victim frame.
     (iii) Write the victim frame to the disk; change the page and frame tables accordingly.
  3. Read the desired page into the newly freed frame; change the page and frame tables.
  4. Restart the user process.
- In an operating system that uses paging for memory management, a page replacement algorithm is needed to decide which page needs to be replaced when new page comes in.
- Page replacement algorithms are the techniques using which an Operating System decides which memory pages to swap out, write to disk when a page of memory needs to be allocated.

## 5.6.4.1 Page Replacement Algorithms [April 17]

- Page replacement algorithm decides which page to remove, also called swap out when a new page needs to be loaded into the main memory.
- The different page replacement algorithms are FIFO (First In First Out) page replacement algorithm, Optimal page replacement algorithm, LRU (Least Recently Used) page replacement algorithm and so on.
- Page replacement algorithms works on the string of memory references, called as reference string.
- It also needs the available number of page frames. Then the number of page faults is determined for a particular reference string.
- Referenced string is a se0.quence of pages being referenced. For example, consider the sequence of addresses 123, 215, 600, 1234, 76, 96. If page size is 100, then the reference string is 1,2,6,12,0,0

**FIFO (First In First Out) Page Replacement Algorithm:** **[April 16, 17, Oct. 17]**

- FIFO is one of the simplest page replacement algorithms. In this algorithm, all the available frames are given to the pages from reference string serially i.e. first frame to first page, second frame to second page and so on.
- When all the available frames are over at that time, the first frame is selected as victim frame and at next time the second frame and so on.
- In FIFO algorithm, the operating system maintains a queue that keeps track of all the pages in memory.
- When a page must be replaced, the oldest page is chosen. FIFO queue is created to hold all pages in 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.

  **For example:** Consider the reference string 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5 and Number of frames are 3.

| | 1 | 2 | 3 | 1 | 2 | 3 | 5 | 1 | 2 |
|---|---|---|---|---|---|---|---|---|---|
| Page frame 0 | 1 | 1 | 1 | 4 | 4 | 4 | 5 | 5 | 5 |
| Page frame 1 | | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 1 |
| Page frame 2 | | | 3 | 3 | 3 | 2 | 2 | 2 | 2 |
| Page faulted | Y | Y | Y | Y | Y | Y | Y | N | N |

| | 3 | 4 | 5 |
|---|---|---|---|
| Page frame 0 | 5 | 5 | 5 |
| Page frame 1 | 3 | 3 | 3 |
| Page frame 2 | 2 | 4 | 4 |
| Page faulted | Y | Y | N |

Total no. of page faults = 9

**Fig. 5.32: Example of FIFO Page Replacement Algorithm**

**Explanation:**

- Number of page frames given are 3, numbered page frame 0, page frame 1 and page frame 2.
- Reference string is scanned serially.
- If the page is not present in one of the three page frames, then a page fault is counted and a page is brought into the page frame.
- In the above examples, first 3 pages 1, 2, 3 are brought in page frames 0, 1 and 2 respectively.
- When the page faulted for page 4, that time page frame containing page 1 is freed and 4 is brought in page frame 0.
- Page frame 0 was selected as victim frame because that was containing first of the first 3 pages.
- Next victim frame will be page frame 1 and so on. Total number of page faults in above example is 9.

**Advantages of FIFO Page Replacement Algorithm:**

1. It is simple to implement.
2. It is easiest algorithm
3. Easy to understand and execute.

**Disadvantages of FIFO Page Replacement Algorithm:**

1. It is not very effective.
2. System needs to track of each frame.
3. Its performance is not always good.
4. It suffers from Belady's anomaly.
5. Bad replacement choice increases the page fault rate and slow process execution.

**Belady's Anomaly:**

- Belady's anomaly is problem occurs in FIFO page replacement algorithm. Belady's anomaly is the phenomenon of increasing the number of page faults on increasing the number of frames in main memory.
- **For example:** We will again take the previous reference string 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5 and the number of available page frames be 4.

| | 1 | 2 | 3 | 4 | 1 | 2 | 5 | 1 |
|---|---|---|---|---|---|---|---|---|
| Page frame 0 | 1 | 1 | 1 | 1 | 1 | 1 | 5 | 5 |
| Page frame 1 | | 2 | 2 | 2 | 2 | 2 | 2 | 1 |
| Page frame 2 | | | 3 | 3 | 3 | 3 | 3 | 3 |
| Page frame 3 | | | | 4 | 4 | 4 | 4 | 4 |
| Page faulted | Y | Y | Y | Y | N | N | Y | Y |

| | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| Page frame 0 | 5 | 5 | 4 | 4 |
| Page frame 1 | 1 | 1 | 1 | 5 |
| Page frame 2 | 2 | 2 | 2 | 2 |
| Page frame 3 | 4 | 3 | 3 | 3 |
| Page faulted | Y | Y | Y | Y |

Total no. of page faults = 10

**Fig. 5.33: Example of Belady's Anomaly**

- In above example, page fault rate is increased though number of page frames is increased.
- For some page replacement algorithms, the page fault rate may increase as the number of allocated frames increases. Such exception is known as Belady's Anomaly.

**Optimal Page Replacement Algorithm:** **[Oct. 16, 17, April 19]**

- In optimal page replacement algorithm, the page that will not be required for longest period of time is replaced.
- Page fault rate in this case is less as compared to FIFO, hence the name optimal.
- An optimal page replacement algorithm has the lowest page fault rate of all algorithms and would never suffer from Belady's anomaly.
- An optimal page-replacement algorithm exists, and has been called OPT or MIN.
- With optimal page replacement algorithm, the page replacement decision requires complete reference string to be prepared before the execution of the algorithm starts because every page replacement needs to refer the reference string for the requirement of a particular page in future.
- Practically, this situation is difficult to achieve. Therefore, this algorithm is difficult to implement.

**For example:** Reference string is 1, 3, 3, 2, 5, 4, 5, 4, 1, 4, 2, 2, 5 and Number of page frames are 3.

| | 1 | 3 | 3 | 2 | 5 | 4 | 5 | 4 |
|---|---|---|---|---|---|---|---|---|
| Page frame 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Page frame 1 | | 3 | 3 | 3 | 5 | 5 | 5 | 5 |
| Page frame 2 | | | | 2 | 2 | 4 | 4 | 4 |
| Page faulted | Y | Y | N | Y | Y | Y | N | N |

| | 1 | 4 | 2 | 2 | 5 |
|---|---|---|---|---|---|
| Page frame 0 | 1 | 1 | 2 | 2 | 2 |
| Page frame 1 | 5 | 5 | 5 | 5 | 5 |
| Page frame 2 | 4 | 4 | 4 | 4 | 4 |
| Page faulted | N | N | Y | N | N |

Total no. of page faults = 6

**Fig. 5.34: Optimal Page Replacement Algorithm**

**Example:** Consider the reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5. Calculate the page fault using Optimal Page Replacement using 3 frames.

**Solution:**

| **Reference String → Frames ↓** | **1** | **2** | **3** | **4** | **1** | **2** | **5** | **1** | **2** | **3** | **4** | **5** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | ***1** | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | ***3** | 3 | 3 |
| 2 | | ***2** | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | ***4** | 4 |
| 3 | | | ***3** | ***4** | 4 | 4 | ***5** | 5 | 5 | 5 | 5 | 5 |
| Page Fault | Y | Y | Y | Y | N | N | Y | N | N | Y | Y | N |

Total Page faults = 7

**Fig. 5.35**

**Advantages of Optimal Page Replacement Algorithm:**

1. It is the best possible optimal algorithm.
2. It gives the smallest number of page faults.
3. It never suffers from Belady's anomaly.
4. Twice as good as FIFO page Replacement Algorithm

**Disadvantages of Optimal Page Replacement Algorithm:**

1. This algorithm is difficult to implement.
2. It is only use as a theoretical part of page replacement.
3. It requires future knowledge of reference string.

**LRU Page Replacement Algorithm**: **[April 16, 17, 19, Oct. 16, 18]**

- LRU stands for Least Recently Used. In this algorithm, the algorithm refers to the past references of the pages.
- LRU page 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 of time.
- **For example:** The reference string, 1, 3, 3, 2, 5, 4, 5, 4, 1, 4, 2, 2, 5 and number of page frames = 3.

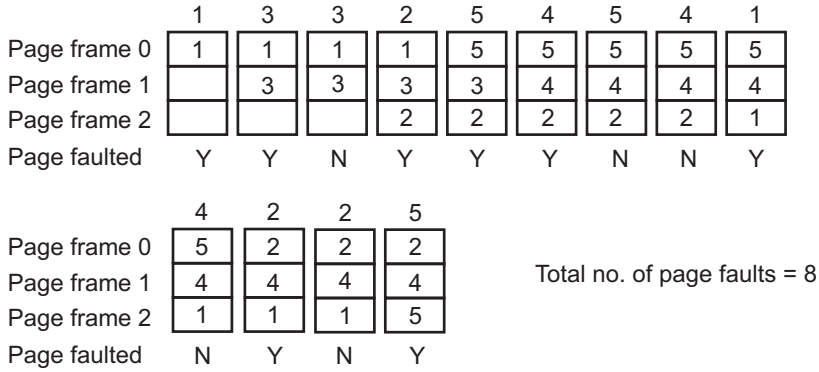



**Fig. 5.36: Least Recently Used (LRU) Page Replacement Algorithm**

**Explanation:**

- First 3 pages 1, 3 and 2 are brought into page frames 0, 1 and 2 respectively. When page fault for page 5 occurs, that time frame 0 is selected as victim frame.
- If you observe the reference string upto page 5, page number 2 is nearest to page 5 and page number 1 is farthest from page 5. The farthest page is the one which is least recently used.
- Therefore, page frame 0 is selected as victim frame (containing the farthest page) and $5^{th}$ page is brought into it.
- To bring page number 4 in the memory, frame 1 is selected as victim frame, because it contains the page 3 farthest from page number 4 (farthest in the past) and so on.

- Implementation of true LRU needs hardware assistance. Either of the following methods can be used to implement LRU:
  1. **Counters:** Each page table entry has a time of use register and adds to the CPU a logical clock or counter. The clock is incremented for every memory reference. Whenever a reference to a page is made, the contents of the clock register are copied to the time of use register in the page table for that page. Thus we always have the "time" of the last reference to each page. We replace the page with the smaller time value.
  2. **Stack:** This implementation keeps a stack of page numbers. Whenever a page is referenced, it is removed from the stack and put on the top. Thus the top of the stack is always the most recently used page and the bottom is the least recently used page.

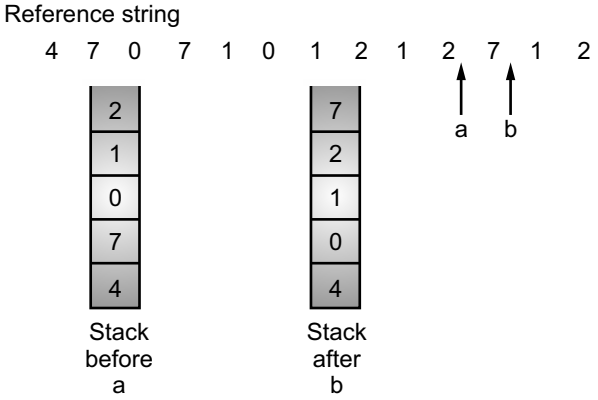



**Fig. 5.37: Use of a Stack to Record the Most Recent Page References**

**Advantages of LRU Page Replacement Algorithm:**

1. LRU is actually quite a good algorithm.
2. It never suffers from Belady's anomaly.
3. LRU algorithm is very feasible to implement.

**Disadvantages of LRU Page Replacement Algorithm:**

1. LRU algorithm is that it requires additional data structure and hardware support.
2. Its implementation is not very easy.

**Example:** Consider the reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5. Calculate the page fault using LRU Page Replacement using 3 frames.

| **Reference String →** **Frames ↓** | **1** | **2** | **3** | **4** | **1** | **2** | **5** | **1** | **2** | **3** | **4** | **5** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | ***1** | 1 | 1 | ***4** | 4 | 4 | ***5** | 5 | 5 | ***3** | 3 | 3 |
| 2 | | ***2** | 2 | 2 | ***1** | 1 | 1 | 1 | 1 | 1 | ***4** | 4 |
| 3 | | | ***3** | 3 | 3 | ***2** | 2 | 2 | 2 | 2 | 2 | ***5** |
| Page Fault | Y | Y | Y | Y | Y | Y | Y | N | N | Y | Y | Y |

Total Page Faults = 10.

## SOLVED PROBLEMS

**Problem 1:** Consider the page reference string 2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2. how many page faults occur for the following replacement algorithm, assuming 3 frames? (i) FIFO (ii) LRU (iii) Optimal.

**Solution: Using FIFO Page Replacement Algorithm:**

| **Reference String → Frames ↓** | **2** | **3** | **2** | **1** | **5** | **2** | **4** | **5** | **3** | **2** | **5** | **2** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | ***2** | 2 | 2 | 2 | ***5** | 5 | 5 | 5 | ***3** | 3 | 3 | 3 |
| 2 | | ***3** | 3 | 3 | 3 | ***2** | 2 | 2 | 2 | 2 | ***5** | 5 |
| 3 | | | | ***1** | 1 | 1 | ***4** | 4 | 4 | 4 | 4 | ***2** |
| Page Fault | Y | Y | N | Y | Y | Y | Y | N | Y | N | Y | Y |

Total Number of Page fault using three frames=9.

**Using LRU Page Replacement Algorithm:**

| **Reference String → Frames ↓** | **2** | **3** | **2** | **1** | **5** | **2** | **4** | **5** | **3** | **2** | **5** | **2** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | ***2** | 2 | 2 | 2 | 2 | 2 | 2 | 2 | ***3** | 3 | 3 | 3 |
| 2 | | ***3** | 3 | 3 | *5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| 3 | | | | ***1** | 1 | 1 | ***4** | 4 | 4 | ***2** | 2 | 2 |
| Page Fault | Y | Y | N | Y | Y | N | Y | N | Y | Y | N | N |

Total Number of Page fault using three frames=7.

**Using Optimal Page Replacement Algorithm:**

| **Reference String → Frames ↓** | **2** | **3** | **2** | **1** | **5** | **2** | **4** | **5** | **3** | **2** | **5** | **2** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | ***2** | 2 | 2 | 2 | 2 | 2 | ***4** | 4 | 4 | 4 | 4 | 4 |
| 2 | | ***3** | 3 | 3 | 3 | 3 | 3 | 3 | 3 | ***2** | 2 | 2 |
| 3 | | | | ***1** | ***5** | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| Page Fault | Y | Y | N | Y | Y | N | Y | N | N | Y | N | N |

Total Number of Page fault using three frames=6.

**Problem 2:** Consider the page reference string 1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 3. How many page faults will occur for FIFO and LRU page replacement algorithms? Assuming 3 frames, all are initially empty.

**Solution:** No. of frames = 3.

**1. FIFO Page Replacement Algorithm:**

| Reference String | 1 | 2 | 3 | 4 | 2 | 1 | 5 | 6 | 2 | 1 | 2 | 3 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Frame 1 | 1 | 1 | 1 | 4 | 4 | 4 | 4 | 6 | 6 | 6 | 6 | 3 |
| Frame 2 | | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 2 | 2 | 2 | 2 |
| Frame 3 | | | 3 | 3 | 3 | 3 | 5 | 5 | 5 | 1 | 1 | 1 |
| Page Fault | 1 | 2 | 3 | 4 | X | 1 | 5 | 6 | 2 | 1 | X | 3 |
| | Y | Y | Y | Y | N | Y | Y | Y | Y | Y | N | Y |

Total No. of page faults by FIFO = 10.

**2. LRU Page Replacement Algorithm:**

| Reference String | 1 | 2 | 3 | 4 | 2 | 1 | 5 | 6 | 2 | 1 | 2 | 3 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Frame 1 | 1 | 1 | 1 | 4 | 4 | 4 | 5 | 5 | 5 | 1 | 1 | 1 |
| Frame 2 | | 2 | 2 | 2 | 2 | 2 | 2 | 6 | 6 | 6 | 6 | 3 |
| Frame 3 | | | 3 | 3 | 3 | 1 | 1 | 1 | 2 | 2 | 2 | 2 |
| Page Fault | 1 | 2 | 3 | 4 | X | 1 | 5 | 6 | 2 | 1 | X | 3 |
| | Y | Y | Y | Y | N | Y | Y | Y | Y | Y | N | Y |

Total No. of page faults by LRU = 10.

**Problem 3:** Consider the page reference string 1, 2, 3, 4, 5, 3, 4, 1, 6, 7, 8, 7. How many page faults would occur for the LRU and Optimal replacement page replacement algorithms? Assuming 4r frames? All frames are initially empty.

**Solution:**

**1. LRU Page Replacement:**

| Reference String | 1 | 2 | 3 | 4 | 5 | 3 | 4 | 1 | 6 | 7 | 8 | 7 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Frame 1 | 1 | 1 | 1 | 1 | 5 | 5 | 5 | 5 | 6 | 6 | 6 | 6 |
| Frame 2 | | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 1 |
| Frame 3 | | | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 7 | 7 | 7 |
| Frame 4 | | | | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 8 | 8 |
| Page Fault | Y | Y | Y | Y | Y | N | N | Y | Y | Y | Y | N |

Total page faults by LRU = 9.

**2. Optimal Page Replacement:**

| Reference String | 1 | 2 | 3 | 4 | 5 | 3 | 4 | 1 | 6 | 7 | 8 | 7 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Frame 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 6 | 6 | 6 | 6 |
| Frame 2 | | 2 | 2 | 2 | 5 | 5 | 5 | 5 | 5 | 7 | 7 | 7 |
| Frame 3 | | | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 8 | 8 |
| Frame 4 | | | | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| Page Fault | Y | Y | Y | Y | Y | N | N | N | Y | Y | Y | N |

Total page faults by Optimal = 8.

**Problem 4:** Consider the following segment table:

| Segment | Base | Length |
|---|---|---|
| 0 | 600 | 120 |
| 1 | 1200 | 350 |
| 2 | 75 | 85 |
| 3 | 1760 | 90 |
| 4 | 2510 | 680 |

What are the physical addresses for the following logical addresses?

1. 0, 125
2. 1, 310
3. 3, 88
4. 2, 77
5. 4, 444

**Solution:**

1. Logical address = 0, 255
   Segment No. = 0,
   Offset = 255
   Base address of segment 0 = 600
   Since offset > Length (i.e. 255 > 120)
   Therefore invalid address and trap will be generated.
2. Logical address = 1, 310
   Segment No. = 1,
   Offset = 310
   Base address of segment 1 = 1200
   Physical address = Base address + Offset
   = 1200 + 310
   = 1510
3. Logical address = 3, 88
   Segment No. = 3,
   Offset = 88
   Since Offset < length, therefore address is valid
   Base Address of segment 3 = 1760
   Physical address = Base address + Offset
   = 1760 + 88
   = 1848

4. Logical address = 2, 77
Segment No. = 2
Offset = 77
Base address = 75
Physical address = 75 + 77
= 152

5. Logical address = 4,444
Segment No. = 4
Offset = 444
Base address = 2510
Physical address = 2510 + 444
= 2954

**Problem 5:** Consider the page reference string 1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3.

How many page faults would occur for the LRU and optimal page replacement algorithms, assuming 4 frames? All frames are initially empty.

**Solution:**

**1. LRU Replacement Method:**

| **Reference String** | **1** | **2** | **3** | **4** | **2** | **1** | **5** | **6** | **2** | **1** | **2** | **3** | **7** | **6** | **3** | **2** | **1** | **2** | **3** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Frame 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 6 | 6 | 6 | 6 | 6 | 6 |
| Frame 2 | | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| Frame 3 | | | 3 | 3 | 3 | 3 | 5 | 5 | 5 | 5 | 5 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| Frame 4 | | | | 4 | 4 | 4 | 4 | 6 | 6 | 6 | 6 | 6 | 7 | 7 | 7 | 7 | 1 | 7 | 7 |
| Page Fault | Y | Y | Y | Y | N | N | Y | Y | N | N | N | Y | Y | Y | N | N | Y | N | N |

Total page faults = 10.

**2. Optimal Replacement Method:**

| **Reference String** | **1** | **2** | **3** | **4** | **2** | **1** | **5** | **6** | **2** | **1** | **2** | **3** | **7** | **6** | **3** | **2** | **1** | **2** | **3** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Frame 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 7 | 7 | 7 | 7 | 1 | 1 | 1 |
| Frame 2 | | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| Frame 3 | | | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| Frame 4 | | | | 4 | 4 | 4 | 5 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| Page Fault | Y | Y | Y | Y | N | N | Y | Y | N | N | N | N | Y | N | N | N | Y | N | N |

Total page faults = 8.

**Problem 6:** Consider the page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 7, 0, 1.

How many page faults would occur for the optimal and FIFO page replacement algorithm, assuming 3 frames? All frames are initially empty.

**Solution:**

**1. Optimal Page Replacement:**

| **Reference String** | **7** | **0** | **1** | **2** | **0** | **3** | **0** | **4** | **2** | **3** | **0** | **3** | **2** | **1** | **2** | **0** | **7** | **0** | **1** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Frame 1 | 7 | 7 | 7 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 7 | 7 | 7 |
| Frame 2 | | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 4 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Frame 3 | | | 1 | 1 | 1 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 1 | 1 | 1 | 1 | 1 | 1 |
| Page Fault | Y | Y | Y | Y | N | Y | N | Y | N | N | Y | N | N | Y | N | N | Y | N | N |

Total page faults by optimal = 9.

**2. FIFO Page Replacement:**

| **Reference string** | **7** | **0** | **1** | **2** | **3** | **0** | **4** | **2** | **3** | **0** | **3** | **2** | **1** | **2** | **0** | **1** | **7** | **0** | **1** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Frame 1 | 7 | 7 | 7 | 2 | 2 | 2 | 4 | 4 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 7 | 7 |
| Frame 2 | | 0 | 0 | 0 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| Frame 3 | | | 1 | 1 | 1 | 0 | 0 | 0 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 1 |
| Page Fault | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | N | N | Y | Y | N | N | Y | Y | Y |

Total page faults = 15.

---

**Problem 7:** Consider the physical memory of 15 frames with each frame of size 8 bytes. Consider page table entries as:

| | |
|---|---|
| 0 | 2 |
| 1 | 7 |
| 2 | 1 |
| 3 | 13 |
| 4 | 10 |
| 5 | 5 |

Map following logical addresses to their physical addresses 5, 40, 23, 36 and 68.

**Solution:** Page size = 8 bytes

1. Logical address = 5

Page No. = $5/8 \Rightarrow 0$

Page offset = 5 % 8 = 5

Base address = Frame number $\times$ Frame size

$= 2 \times 8$

$= 16$

Physical address = Base + Offset address

$= 16 + 5$

Physical address = 21

2. Logical address = 40
   Page No. = 40/8 = 5
   Page offset = 40 % 8 = 0
   Frame No = 5
   Base address = $5 \times 8 = 40$
   Physical address = 40 + 0 = 40
3. Logical address = 23
   Page No. = $23/8 \Rightarrow 2$
   Page offset = $23 \% 8 \Rightarrow 7$
   Frame No. = 1
   Base address = $1 \times 8 = 8$
   Physical address = $8 + 7 \Rightarrow 15$
4. Logical address = 36
   Page No. = 36/8 = 4
   Page offset = 36 % 8 = 4
   Frame No. = 10
   Base address = $10 \times 4 = 40$
   Physical address = $40 + 4 \Rightarrow 44$
5. Logical address = 68
   Page No. = 68/8 = 8
   Page offset = 4

We can't find frame number since page 8 is not loaded in page table, so we can not find physical address.

**Problem 8:** Consider the page reference string 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5. How many page faults would occur for the LRU and FIFO page replacement algorithms? Assume three frames.

**Solution: LRU Page Replacement Algorithm:**

| **Reference String** | **1** | **2** | **3** | **4** | **1** | **2** | **5** | **1** | **2** | **3** | **4** | **5** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Frame 1 | 1 | 1 | 1 | 4 | 4 | 4 | 5 | 5 | 5 | 3 | 3 | 3 |
| Frame 2 | | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 4 | 4 |
| Frame 3 | | | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 5 |
| Page Fault | Y | Y | Y | Y | Y | Y | Y | N | N | Y | Y | Y |

Total pages faults by LRU = 10.

**FIFO Page Replacement Algorithm:**

| Reference String | 1 | 2 | 3 | 4 | 1 | 2 | 5 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Frame 1 | 1 | 1 | 1 | 4 | 4 | 4 | 5 | 5 | 5 | 5 | 5 | 5 |
| Frame 2 | | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 3 | 3 | 3 |
| Frame 3 | | | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 4 | 4 |
| Page Fault | Y | Y | Y | Y | Y | Y | Y | N | N | Y | Y | N |

Total pages faults by FIFO = 9.

**Problem 9:** Consider the page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2. How many page faults would occur for the LRU and optimal page replacement algorithm? Assume 3 frames.

**Solution: LRU Page Replacement Algorithm:**

| Reference String | 7 | 0 | 1 | 2 | 0 | 3 | 0 | 4 | 2 | 3 | 0 | 3 | 2 | 1 | 2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Frame 1 | 7 | 7 | 7 | 2 | 2 | 2 | 2 | 4 | 4 | 4 | 0 | 0 | 0 | 1 | 1 |
| Frame 2 | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 3 | 3 | 3 | 3 | 3 |
| Frame 3 | | | 1 | 1 | 1 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| Page Fault | Y | Y | Y | Y | N | Y | N | Y | Y | Y | Y | N | N | Y | N |

Total page faults = 10.

**Optimal Page Replacement Algorithm:**

| Reference String | 7 | 0 | 1 | 2 | 0 | 3 | 0 | 4 | 2 | 3 | 0 | 3 | 2 | 1 | 2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Frame 1 | 7 | 7 | 7 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| Frame 2 | | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 4 | 4 | 0 | 0 | 0 | 1 | 1 |
| Frame 3 | | | 1 | 1 | 1 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| Page Faults | Y | Y | Y | Y | N | Y | N | Y | N | N | Y | N | N | Y | N |

Total page faults by optimal = 8.

**Problem 10:** Consider a logical address space of 4 pages of 512 words each, mapped onto a physical memory of 16 frames.

(a) How many bits are there in logical address?

(b) How many bits are there in physical address?

**Solution:** Logical address space = 4 pages = $2^2$ (m = 2)

Page size = 512 = $2^9$ (n = 9)

n = offset = 9 bits

$\therefore$ Logical address = 9 + 2 = 11 bits

Physical memory = 16 Frames = $2^4$

Frame size = $2^9$ = 512 words

Physical address bits = $2^9 \times 2^4$

= $2^{13}$

$\therefore$ Physical address = 13 bits

**Problem 11:** Consider page reference string 7, 5, 6, 2, 9, 5, 7, 6, 2, 7, 6, 5, 2, 7, 2, 7, 8 and assume 3 frames. Find the number of page faults according to Optimal page replacement algorithm and Least Recently Used (LRU) page replacement algorithm. **[Oct. 16]**

**Solution: Optimal Page Replacement Algorithm:**

| **Reference String** | **7** | **5** | **6** | **2** | **9** | **5** | **7** | **6** | **2** | **7** | **6** | **5** | **2** | **7** | **2** | **7** | **8** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Frame 1 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 8 |
| Frame 2 | | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| Frame 3 | | | 6 | 2 | 9 | 9 | 9 | 6 | 6 | 6 | 6 | 5 | 5 | 5 | 5 | 5 | 5 |
| Page Fault | Y | Y | Y | Y | Y | N | N | Y | Y | N | N | Y | N | N | N | N | Y |

Total pages faults = 9.

**LRU Page Replacement Algorithm:**

| **Reference String** | **7** | **5** | **6** | **2** | **9** | **5** | **7** | **6** | **2** | **7** | **6** | **5** | **2** | **7** | **2** | **7** | **8** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Frame 1 | 7 | 7 | 7 | 2 | 2 | 2 | 7 | 7 | 7 | 7 | 7 | 7 | 2 | 2 | 2 | 2 | 2 |
| Frame 2 | | 5 | 5 | 5 | 9 | 9 | 9 | 6 | 6 | 6 | 6 | 6 | 6 | 7 | 7 | 7 | 7 |
| Frame 3 | | | 6 | 6 | 6 | 5 | 5 | 5 | 2 | 2 | 2 | 5 | 5 | 5 | 5 | 5 | 8 |
| Page Fault | Y | Y | Y | Y | Y | Y | Y | Y | Y | N | N | Y | Y | Y | N | N | Y |

Total pages faults = 13.

**Problem 12:** In a virtual memory system, size of virtual address is 32-bit, size of physical address is 30-bit, page size is 4 Kbyte and size of each page table entry is 32-bit. The main memory is byte addressable. What is the maximum number of bits that can be used for storing protection and other information in each page table entry?

**Solution:**

Virtual memory = 232 bytes

Physical memory = 230 bytes

Page size = Frame size = 4 * 103 bytes = 22 * 210 bytes = 212 bytes

Number of frames = Physical memory / Frame size = 230/212 = 218

Therefore, Numbers of bits for frame = 18 bits

Page Table Entry Size = Number of bits for frame + Other information

Other information = 32 – 18 = 14 bits

**Problem 13:** Consider a computer system with 40-bit virtual addressing and page size of sixteen kilobytes. If the computer system has a one-level page table per process and each page table entry requires 48 bits, then what is the size of the per-process page table?

**Solution:**

Size of memory = $2^{40}$

Page size = 16KB = $2^{14}$

No of pages= size of Memory/Page size = $2^{40} / 2^{14} = 2^{26}$

Size of page table = $2^{26} * 48/8$ bytes = $2^6 * 6$ MB = 384 MB

---

**Problem 14:** Let the page fault service time be 10ms in a computer with average memory access time being 20ns. If one page fault is generated for every $10^6$ memory accesses, what is the effective access time for the memory?

**Solution:**

Let P be the page fault rate

Effective Memory Access Time = p * (page fault service time) +
(1 - p) * (Memory access time)

= ( $1/(10^6)$ )* 10 * ($10^6$) ns + (1 - $1/(10^6)$) * 20 ns

= 30 ns (approx)

---

**Problem 15:** In a paged memory, the page hit ratio is 0.40. The time required to access a page in secondary memory is equal to 120 ns. The time required to access a page in primary memory is 15 ns.

**Solution:**

Average access time = hit ratio * primary memory access time + (1 – hit ratio) * secondary memory access time

Average access time = 0.4 * 15 + 0.6 * 120

Average access time = 6 + 72

Average access time = 78.

---

**Problem 16:** What is the size of the physical address space in a paging system which has a page table containing 64 entries of 11 bit each (including valid and invalid bit) and a page size of 512 bytes?

**Solution:**

Size of Physical Address = Paging bits + Offset bits

Paging bits = 11 – 1 = 10 (As 1 valid bit is also included)

Offset bits = $\log_2$ page size = 9

Size of Physical Address = 10 + 9 = 19 bits.

---

**Problem 17:** If there are 32 segments, each size 1 k bytes, then how many bits will be in the logical address?

**Solution:**

**Data:** Logical address size = L bytes

No of segments = n = 32

Size of each segment = s = k byte = $1 \times 2^{10}$ bytes

**Formula:**

Number of bits = $\log_2$ L

**Calculation:**

L = n × s = bytes

L = $32 \times 2^{10} = 2^{15}$ bytes

Number of bits = $\log_2 2^{15} = 15$

**Problem 18:** A memory management system has 64 pages with 512 bytes page size. Physical memory consists of 32 page frames. How many number of bits required in logical and physical address?

**Solution:**

We know that Number of pages = virtual memory space / page size.

Number of frames = physical memory space / frame size.

Page size is equal to frame size.

According to question and given data:

virtual memory space = Number of pages * page size

i.e. virtual memory space = $6^4 * 5^{12}$B

virtual memory space = $2^6 * 2^9$B

i.e., = $2^{15}$ B.

So, 15 bits are required for virtual memory space.

Physical memory space = Number of frames * frame size.

Physical memory space = $3^2 * 5^{12}$ B

Physical memory space = $2^5 * 2^9$B

i.e., = $2^{14}$

So, 14 bits are required for physical memory space.

## PRACTICE QUESTIONS

**Q.I Multiple Choice Questions:**

1. Which is an important function of operating system that helps in allocating the main memory space to the processes and their data at the time of their execution?
   (a) Memory management (b) Device management
   (c) CPU management (d) I/O management
2. A memory buffer used to accommodate a speed differential is called as,
   (a) stack pointer (b) cache
   (c) accumulator (d) disk buffer
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
11. In contiguous memory allocation,
    (a) each process is contained in a single contiguous section of memory
    (b) all processes are contained in a single contiguous section of memory
    (c) the memory space is contiguous
    (d) none of the mentioned
12. The relocation register helps in,
    (a) providing more address space to processes
    (b) a different address space to processes
    (c) to protect the address spaces of processes
    (d) none of the mentioned
13. With relocation and limit registers, each logical address must be _____ the limit register.
    (a) less than (b) equal to
    (c) greater than (d) none of the mentioned
14. The operating system and the other processes are protected from being modified by an already running process because,
    (a) they are in different memory spaces
    (b) they are in different logical addresses
    (c) they have a protection algorithm
    (d) every address generated by the CPU is being checked against the relocation and limit registers
15. When memory is divided into several fixed sized partitions, each partition may contain,
    (a) exactly one process (b) at least one process
    (c) multiple processes at once (d) none of the mentioned

16. In fixed size partition, the degree of multiprogramming is bounded by,
    (a) the number of partitions (b) the CPU utilization
    (c) the memory size (d) all of the mentioned
17. The first fit, best fit and worst fit are strategies to select a,
    (a) process from a queue to put in memory
    (b) processor to run the next process
    (c) free hole from a set of available holes
    (d) all of the mentioned
18. In internal fragmentation, memory is internal to a partition and _____.
    (a) is being used (b) is not being used
    (c) is always used (d) none of the mentioned
19. A solution to the problem of external fragmentation is,
    (a) compaction (b) larger memory space
    (c) smaller memory space (d) none of the mentioned
20. Another solution to the problem of external fragmentation problem is to,
    (a) permit the logical address space of a process to be noncontiguous
    (b) permit smaller processes to be allocated memory at last
    (c) permit larger processes to be allocated memory at last
    (d) All of the above mentioned
21. If relocation is static and is done at assembly or load time, compaction,
    (a) cannot be done (b) must be done
    (c) must not be done (d) can be done
22. The disadvantage of moving all process to one end of memory and all holes to the other direction, producing one large hole of available memory is,
    (a) the cost incurred (b) the memory used
    (c) the CPU used (d) all of the mentioned
23. _____ is generally faster than _____ and _____.
    (a) first fit, best fit, worst fit (b) best fit, first fit, worst fit
    (c) worst fit, best fit, first fit (d) None of the mentioned
24. External fragmentation exists when?
    (a) enough total memory exists to satisfy a request but it is not contiguous
    (b) the total memory is insufficient to satisfy a request
    (c) a request cannot be satisfied even when the total memory is free
    (d) None of the mentioned
25. External fragmentation will not occur when?
    (a) first fit is used
    (b) best fit is used
    (c) worst fit is used
    (d) no matter which algorithm is used, it will always occur

26. Sometimes the overhead of keeping track of a hole might be,
    (a) larger than the memory (b) larger than the hole itself
    (c) very small (d) all of the mentioned
27. When the memory allocated to a process is slightly larger than the process, then
    (a) internal fragmentation occurs
    (b) external fragmentation occurs
    (c) both internal and external fragmentation occurs
    (d) neither internal nor external fragmentation occurs
28. Physical memory is broken into fixed-sized blocks called as,
    (a) frames (b) pages
    (c) backing store (d) none of the mentioned
29. Logical memory is broken into blocks of the same size called as,
    (a) frames (b) pages
    (c) backing store (d) none of the mentioned
30. Every address generated by the CPU is divided into two parts namely,
    (a) frame bit and page number (b) page number and page offset
    (c) page offset and frame bit (d) frame offset and page offset
31. Which is used as an index into the page table?
    (a) frame bit (b) page number
    (c) page offset (d) frame offset
32. Which table contains the base address of each page in physical memory?
    (a) process (b) memory
    (c) page (d) frame
33. The size of a page is typically,
    (a) varied (b) power of 2
    (c) power of 4 (d) none of the mentioned
34. If the size of logical address space is 2 to the power of m, and a page size is 2 to the power of n addressing units, then the high order _____ bits of a logical address designate the page number, and the _____ low order bits designate the page offset.
    (a) m, n (b) n, m
    (c) m – n, m (d) m – n, n
35. With paging there is no _____ fragmentation.
    (a) internal (b) external
    (c) either type of (d) none of the mentioned
36. The operating system maintains a which table that keeps track of how many frames have been allocated, how many are there, and how many are available.
    (a) page (b) mapping
    (c) frame (d) memory

37. Paging increases the _____ time.
    (a) waiting (b) execution
    (c) context – switch (d) all of the mentioned
38. Smaller page tables are implemented as a set of,
    (a) queues (b) stacks
    (c) counters (d) registers
39. The page table registers should be built with,
    (a) very low speed logic (b) very high speed logic
    (c) a large memory space (d) none of the mentioned
40. For larger page tables, they are kept in main memory and a _____ points to the page table.
    (a) page table base register (b) page table base pointer
    (c) page table register pointer (d) page table base
41. For every process there is a,
    (a) page table (b) copy of page table
    (c) pointer to page table (d) all of the mentioned
42. Time taken in memory access through PTBR is,
    (a) extended by a factor of 3 (b) extended by a factor of 2
    (c) slowed by a factor of 3 (d) slowed by a factor of 2
43. Each entry in a translation look aside buffer (TLB) consists of,
    (a) key (b) value
    (c) bit value (d) constant
44. If a page number is not found in the TLB, then it is known as,
    (a) TLB miss (b) Buffer miss
    (c) TLB hit (d) All of the mentioned
45. Which is a uniquely identifies processes and is used to provide address space protection for that process?
    (a) address space locator (b) address space identifier
    (c) address process identifier (d) none of the mentioned
46. The percentage of times a page number is found in the TLB is known as,
    (a) miss ratio (b) hit ratio
    (c) miss percent (d) none of the mentioned
47. Memory protection in a paged environment is accomplished by,
    (a) protection algorithm with each page (b) restricted access rights to users
    (c) restriction on page visibility (d) protection bit with each page
48. When the valid – invalid bit is set to valid, it means that the associated page,
    (a) is in the TLB
    (b) has data in it
    (c) is in the process's logical address space
    (d) is the system's physical address space

49. Illegal addresses are trapped using the _____ bit.
    (a) error (b) protection
    (c) valid – invalid (d) access
50. When there is a large logical address space, the best way of paging would be,
    (a) not to page (b) a two level paging algorithm
    (c) the page table itself (d) all of the mentioned
51. In a paged memory, the page hit ratio is 0.35. The required to access a page in secondary memory is equal to 100 ns. The time required to access a page in primary memory is 10 ns. The average time required to access a page is?
    (a) 3.0 ns (b) 68.0 ns
    (c) 68.5 ns (d) 78.5 ns
52. To obtain better memory utilization, dynamic loading is used. With dynamic loading, a routine is not loaded until it is called. For implementing dynamic loading,
    (a) special support from hardware is required
    (b) special support from operating system is essential
    (c) special support from both hardware and operating system is essential
    (d) user programs can implement dynamic loading without any special support from hardware or operating system
53. In paged memory systems, if the page size is increased, then the internal fragmentation generally,
    (a) becomes less (b) becomes more
    (c) remains constant (d) none of the mentioned
54. In segmentation, each address is specified by,
    (a) segment number and offset (b) an offset and value
    (c) a value and segment number (d) a key and value
55. In paging the user provides only _____ which is partitioned by the hardware into _____ and _____.
    (a) one address, page number, offset (b) one offset, page number, address
    (c) page number, offset, address (d) none of the mentioned
56. Virtual memory is _____.
    (a) Large secondary memory (b) Large main memory
    (c) Illusion of large main memory (d) None of the above
57. Each entry in a segment table has a,
    (a) segment base (b) segment peak
    (c) segment value (d) none of the mentioned
58. The segment base contains the,
    (a) starting logical address of the process
    (b) starting physical address of the segment in memory
    (c) segment length
    (d) none of the mentioned

59. The segment limit contains the,
    (a) starting logical address of the process
    (b) starting physical address of the segment in memory
    (c) segment length
    (d) none of the mentioned
60. The offset 'd' of the logical address must be,
    (a) greater than segment limit (b) between 0 and segment limit
    (c) between 0 and the segment number (d) greater than the segment number
61. If the offset is legal,
    (a) it is used as a physical memory address itself
    (b) it is subtracted from the segment base to produce the physical memory address
    (c) it is added to the segment base to produce the physical memory address
    (d) none of the mentioned
62. When the entries in the segment tables of two different processes point to the same physical location,
    (a) the segments are invalid (b) the processes get blocked
    (c) segments are shared (d) all of the mentioned
63. The protection bit is 0/1 based on,
    (a) write only (b) read only
    (c) read – write (d) none of the mentioned
64. If there are 32 segments, each of size 1Kb, then the logical address should have,
    (a) 13 bits (b) 14 bits
    (c) 15 bits (d) 16 bits
65. Consider a computer with 8 Mbytes of main memory and a 128K cache. The cache block size is 4 K. It uses a direct mapping scheme for cache management. How many different main memory blocks can map onto a given physical cache block?
    (a) 2048 (b) 256
    (c) 64 (d) 8
66. A multilevel page table is preferred in comparison to a single level page table for translating virtual address to physical address because,
    (a) it reduces the memory access time to read or write a memory location
    (b) it helps to reduce the size of page table needed to implement the virtual address space of a process
    (c) it is required by the translation look aside buffer
    (d) it helps to reduce the number of page faults in page replacement algorithms
67. Which of the following page replacement algorithms suffers from Belady's anomaly?
    (a) FIFO (b) LRU
    (c) Optimal Page Replacement (d) Both LRU and FIFO

68. Increasing the RAM of a computer typically improves performance because:
    (a) Virtual memory increases (b) Larger RAMs are faster
    (c) Fewer page faults occur (d) Fewer segmentation faults occur
69. A computer system supports 32-bit virtual addresses as well as 32-bit physical addresses. Since the virtual address space is of the same size as the physical address space, the operating system designers decide to get rid of the virtual memory entirely. Which one of the following is true?
    (a) Efficient implementation of multi-user support is no longer possible
    (b) The processor cache organization can be made more efficient now
    (c) Hardware support for memory management is no longer needed
    (d) CPU scheduling can be made more efficient now
70. Page fault occurs when,
    (a) When a requested page is in memory
    (b) When a requested page is not in memory
    (c) When a page is corrupted
    (d) When an exception is thrown
71. Dynamic address translation,
    (a) is part of the operating system paging algorithm
    (b) is useless when swapping is used
    (c) is the hardware necessary to implement paging
    (d) storage pages at a specific location on disk
72. Dirty bit is used to indicate which of the following?
    (a) A page fault has occurred
    (b) A page has corrupted data
    (c) A page has been modified after being loaded into cache
    (d) An illegal access of page
73. Which of the following is incorrect for virtual memory?
    (a) Large programs can be written
    (b) More I/O is required
    (c) More addressable memory available
    (d) Faster and easy swapping of process
74. Match the following flag bits used in the context of virtual memory management on the left side with the different purposes on the right side of the table below.

| Name of the Bit | | Purpose |
|---|---|---|
| (i) | Dirty | (a) Page initialization |
| (ii) | R/W | (b) Write-back policy |
| (iii) | Reference | (c) Page protection |
| (iv) | Valid | (d) Page replacement policy |

    (a) (i)-d, (ii)-a, (iii)-b, IV-c (b) (i)-b, (ii)-c, (iii)-a, (iv)-d
    (c) (i)-c, (ii)-d, (iii)-a, IV-b (d) (i)-b, (ii)-c, (iii)-d, (iv)-a

75. In which one of the following page replacement algorithms it is possible for the page fault rate to increase even when the number of allocated frames increases?
    (a) LRU (Least Recently Used) (b) OPT (Optimal Page Replacement)
    (c) MRU (Most Recently Used) (d) FIFO (First In First Out)

## Answers

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1. (a) | 2. (b) | 3. (c) | 4. (a) | 5. (b) | 6. (d) | 7. (a) | 8. (a) | 9. (c) | 10. (a) |
| 11. (a) | 12. (c) | 13. (a) | 14. (d) | 15. (a) | 16. (a) | 17. (c) | 18. (b) | 19. (a) | 20. (a) |
| 21. (a) | 22. (a) | 23. (a) | 24. (a) | 25. (d) | 26. (b) | 27. (a) | 28. (a) | 29. (b) | 30. (b) |
| 31. (b) | 32. (c) | 33. (b) | 34. (d) | 35. (b) | 36. (c) | 37. (c) | 38. (d) | 39. (b) | 40. (a) |
| 41. (a) | 42. (d) | 43. (a) | 44. (a) | 45. (b) | 46. (b) | 47. (d) | 48. (c) | 49. (c) | 50. (b) |
| 51. (c) | 52. (d) | 53. (b) | 54. (a) | 55. (a) | 56. (c) | 57. (a) | 58. (b) | 59. (c) | 60. (b) |
| 61. (a) | 62. (c) | 63. (c) | 64. (a) | 65. (c) | 66. (b) | 67. (a) | 68. (c) | 69. (c) | 70. (b) |
| 71. (c) | 72. (c) | 73. (b) | 74. (d) | 75. (d) | | | | | |

**Q.II Fill in the Blanks:**

1. The task of subdividing memory between the O/S and processes is performed automatically by the O/S and is called _____ management.
2. The Memory Management task of moving the process image between different areas of memory as required to support swapping is referred to as _____.
3. The practice in which a program and data are organized in such a way that various modules can be assigned the same region of memory is called _____.
4. In almost all modern multiprogramming systems, memory is managed using a sophisticated technique known as _____ memory.
5. 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 _____ fragmentation.
6. The problem of internal fragmentation can be lessened in system employing a fixed-partition memory management scheme by using _____ size partitions.
7. In the Dynamic Partitioning technique of memory management, the process of shifting processes so they occupy a single contiguous block in memory is called _____.
8. In the Dynamic Partitioning technique of memory management, the placement algorithm that chooses the block that is closest in size to the request is called _____.
9. In the Dynamic Partitioning technique of memory management, the phenomenon that results in unused blocks of memory outside of existing partitions is called _____ fragmentation.
10. Programs that employ _____ addresses in memory are loaded using dynamic run-time loading.

11. In a system that employs a paging memory management scheme, the _____ table shows the frame location for each page of the process.
12. _____ refers to the technique of dividing the physical memory space into multiple blocks. Each block has specific length and is known as a segment and each segment has a starting address called the base address.
13. _____ is a technique in which the main memory of computer system is organized in the form of equal sized blocks called pages.
14. _____ is the technique involves performing two tasks called swapping in and swapping out.
15. The set of all logical addresses generated by a program is referred to as a _____ address space.
16. Dynamic _____ is the technique through which a computer program at runtime loads a library into memory.
17. Static _____ is done at the time of compilation of the program.
18. _____ memory is a memory management technique that provides an idealized abstraction of the storage resources that are actually available on a given machine which creates the illusion to users of a very large (main) memory.
19. Segment Table Base Register (STBR) is used to point the segment table's _____ in the memory.
20. A _____ paging mechanism is very much similar to a paging system with swapping where processes stored in the secondary memory and pages are loaded only on demand, not in advance.
21. Page _____ algorithms are the techniques using which an Operating System decides which memory pages to swap out, write to disk when a page of memory needs to be allocated.
22. The string of memory references is called _____ string.
23. An _____ page-replacement algorithm has the lowest page-fault rate of all algorithms. An optimal page-replacement algorithm exists, and has been called OPT or MIN.
24. Least Recently Used (LRU) page replacement algorithm keeps track of page usage over a _____ period of time.
25. In First In First Out (FIFO) algorithm the OS maintains a _____ that keeps track of all the pages in memory, with the oldest page at the front and the most recent page at the back.
26. Dynamic _____ defers much of the linking process until a program starts running.
27. The _____ mapping from virtual to physical address is done by the Memory Management Unit (MMU) which is a hardware device.

## Answers

| | | | |
|---|---|---|---|
| 1. memory | 2. relocation | 3. overlaying | 4. virtual |
| 5. internal | 6. unequal | 7. compacting | 8. best-fit |
| 9. external | 10. relative | 11. page | 12. Segmentation |
| 13. Paging | 14. Swapping | 15. logical | 16. loading |
| 17. linking | 18. Virtual | 19. location | 20. demand |
| 21. replacement | 22. reference | 23. optimal | 24. short |
| 25. queue | 26. linking | 27. runtime | |

**Q.III State True or False:**

1. The concept of virtual memory is based on one or both of two basic techniques namely, segmentation and paging.
2. A major problem with the largely obsolete Fixed Partitioning memory management technique is that of external fragmentation.
3. The problem of internal fragmentation can be lessened in a system employing a fixed-partition memory management scheme by using unequal size partitions.
4. In the Dynamic Partitioning technique of memory management, the best fit placement algorithm is usually the best performer of the available algorithms.
5. In the Dynamic Partitioning technique of memory management, compaction refers to shifting the processes into a contiguous block, resulting in all the free memory aggregated into in a single block.
6. In the Dynamic Partitioning technique of memory management, the first-fit placement algorithm scans memory from the location of the last placement and chooses the first available block it finds that satisfies the request.
7. The modify (M) bit is a control bit in a page table entry that indicates whether the contents of the corresponding page have been altered since the page was last loaded into memory.
8. A Page Fault occurs when the desired page table entry is not found in the Translation Look-aside Buffer (TLB).
9. One of the advantages to the programmer of virtual memory using segmentation is that it simplifies the handling of growing data structures by allowing the segment to grow or shrink as necessary.
10. In a combined paging/segmentation system, a user's address space is broken up into a number of fixed-size pages, which in turn are broken up into a number of segments.
11. The set of all physical addresses corresponding to these logical addresses is referred to as a physical address space.
12. In internal fragmentation the total memory space is enough to satisfy a request or to reside a process in it, but it is not contiguous, so it cannot be used.
13. Paging technique plays an important role in implementing virtual memory.
14. Segmentation is a memory management technique in which each job is divided into several segments of different sizes, one for each module that contains pieces that perform related functions.
15. A physical address is generated by CPU while a program is running.

16. The hardware device called Memory-Management Unit is used for mapping logical address to its corresponding physical address.
17. A logical address identifies the physical location of a specific data element in memory.
18. Paging is a memory management technique in which process address space is broken into blocks of the same size called pages.
19. A computer's address space is the total amount of memory that can be addressed by the computer.
20. Linking is a method that helps OS to collect and merge various modules of code and data into a single executable file. In dynamic linking method, libraries are linked at execution time.
21. In swapping any process must be in the memory for its execution, but can be swapped temporarily out of memory to a backing store and then again brought back into the memory to complete its execution.
22. Memory allocation is a process by which computer programs are assigned memory or space.
23. Segment Table Length Register (STLR) indicates the number of segments used by a CPU.
24. The segment table has two entries for each segment i.e. segment base address and segment limit.
25. A demand paging system is quite similar to a paging system with swapping where processes reside in secondary memory and pages are loaded only on demand, not in advance.
26. In the fixed-sized partition, the memory is divided into fixed-sized blocks and each block contains exactly two process.
27. The internal fragmentation occurs when the process does not fully utilize the memory allocated to it.
28. A program segment contains the program's main function, utility functions, data structures, and so on.
29. Page replacement algorithms are the techniques using which an Operating System decides which memory pages to swap out, write to disk when a page of memory needs to be allocated.
30. Reference strings are generated artificially or by tracing a given system and recording the address of each memory reference.
31. Page replacement algorithms help to decide which page must be swapped out from the main memory to create a room for the incoming page.
32. FIFO replaces the oldest page that has been present in the main memory for the longest time.
33. Optimal Page Replacement algorithm is the best page replacement algorithm as it gives the least number of page faults.
34. Least Recently Used (LRU) page replacement algorithm keeps track of page usage over a short period of time. It works on the idea that the pages that have been most heavily used in the past are most likely to be used heavily in the future too.

## Answers

| | | | | |
|---|---|---|---|---|
| 1. (T) | 2. (F) | 3. (T) | 4. (F) | 5. (T) |
| 6. (F) | 7. (T) | 8. (F) | 9. (T) | 10. (F) |
| 11. (T) | 12. (F) | 13. (T) | 14. (T) | 15. (F) |
| 16. (T) | 17. (F) | 18. (T) | 19. (T) | 20. (T) |
| 21. (T) | 22. (T) | 23. (F) | 34. (T) | 25. (T) |
| 26. (F) | 27. (T) | 28. (T) | 29. (T) | 30. (T) |
| 31. (T) | 32. (T) | 33. (T) | 34. (T) | |

**Q.IV Answers the following Questions:**

**(A) Short Answer Questions:**

1. What is memory management?
2. What is logical and physical address?
3. What is logical and physical address space?
4. What address binding?
5. What is dynamic loading?
6. Define dynamic linking.
7. What is meant by shared libraries?
8. Define swapping.
9. What is fragmentation?
10. What is paging?
11. What is meant by page replacement?
12. What is virtual memory?
13. What segmentation?
14. Define demand paging.
15. What is page and page table?
16. What is page replacement algorithm.
17. What is FIFO?
18. Compare LRU and MFU with any two points.

**(B) Long Answer Questions:**

1. What is memory management? List its functions.
2. Describe logical and physical address explain in detail. Also differentiate them.
3. Define address binding? Describe with diagrammatically.
4. Write short note on: Dynamic linking and Shared libraries.
5. With the help of diagram describe swapping.
6. Describe segmentation in detail.
7. What is fragmentation? Explain with its types.

8. Write a note on: Segmentation hardware.
9. Define the terms: (i) Paging, (ii) Swapping, (iii) Page fault, (iv) Frames, (v) Page table (vi) Segment.
10. Explain virtual memory diagrammatically.
11. What is page fault? Explain the different steps in handling a page fault.
12. With the help of example explain following page replacement algorithms:
    (i) FCFS,
    (ii) LRU,
    (iii) LFU,
    (iv) Optimal page replacement
13. With the help of diagram demand paging. State its advantages and disadvantages.
14. Describe paging diagrammatically.
15. Explain the terms logical address and physical address. How logical address is converted to physical address?
16. Explain protection offered by paging system in detail.
17. Consider the following string 1,2,3,4,2,1,6,51,2,1,3,7,6,3,2,1,2,3,6.how many page fault would occur for any one of the following page replacement algorithms, assuming 5 frames: (i) LRU, (ii) FIFO.
18. Consider the following string 4,3,2,1,6,9,1,2,4,2,3,1,6,4,2,.how many pages fault would occur for any one of the following page replacement algorithms, assuming 3 frames: (i) Optimal, (ii) LRU.
19. Consider a logical address space of 8 pages of 1024 words mapped onto a physical memory of 32 frames.
    (i) How many bits are there in the logical address?
    (ii) How many bits are there in the physical address?
20. Define page fault? Describe the steps for handling page fault with suitable diagram.

## UNIVERSITY QUESTIONS AND ANSWERS

**April 2016**

**1.** What are various dynamic allocation memory management methods? **[1 M]**

**Ans.** Refer to Page 5.16.

**2.** Consider the page reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5. How many page faults would occur for the following page replacement algorithm?
(i) LRU (ii) FIFO. **[5 M]**

**Ans.** Refer to Section 5.6.4.1.

**3.** Explain internal and external fragmentation. **[4 M]**

**Ans.** Refer to Section 5.3.3.

**October 2016**

**1.** Define physical address space. **[1 M]**

**Ans.** Refer to Section 5.1.3.

**2.** Consider page reference string: 7, 5, 6, 2, 9, 5, 7, 6, 2, 7, 6, 5, 2, 7, 2, 7, 8. Assume 3 frames. Find the number of page faults according to:
(i) Optimal page replacement algorithm.
(ii) LRU page replacement algorithm. **[5 M]**

**Ans.** Refer to Section 5.6.4.1.

**3.** Explain MVT with advantage and disadvantages. **[4 M]**

**Ans.** Refer to Section 5.3.2.2.

---

**April 2017**

**1.** What is the role of valid and invalid bit in demand paging? **[1 M]**

**Ans.** Refer to Section 5.6.2.

**2.** Consider page reference string: 1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 3. Assume 3 frames. Find the number of page faults according to:
(i) FIFO page replacement algorithm.
(ii) LRU page replacement algorithm. **[5 M]**

**Ans.** Refer to Section 5.6.4.1.

**3.** What is external fragmentation? What are various ways to avoid external fragmentation? **[4 M]**

**Ans.** Refer to Section 5.3.3.

---

**October 2017**

**1.** What is the advantage of paging with segmentation model? **[1 M]**

**Ans.** Refer to Section 5.4.

**2.** What is fragmentation? Explain internal and external fragmentation in detail. **[5 M]**

**Ans.** Refer to Section 5.3.3.

**3.** Consider the page reference string: 7, 5, 6, 2, 9, 5, 7, 6, 2, 7, 6, 5, 2, 7, 2, 7, 8. How many page faults will occur for the following page replacement algorithm? Assume 3 frames: (i) FIFO, (ii) Optimal Replacement. **[1 M]**

**Ans.** Refer to Section 5.6.4.1.

---

**April 2018**

**1.** Using segmentation, find the physical address from the logical address 2,280, having segment size 285 with base address 3000. **[1 M]**

**Ans.** Refer to Section 5.5.

**2.** List various dynamic allocation memory management methods. **[1 M]**

**Ans.** Refer to Page 5.16.

**3.** What is fragmentation? Explain types of fragmentation with suitable example. **[5 M]**

**Ans.** Refer to Section 5.3.3.

---

**4.** Consider the page replacement string: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2. How many page faults would occur for the LRU page replacement algorithm. (Assume three frames). **[4 M]**

**Ans.** Refer to Section 5.6.4.1.

**October 2018**

**1.** Write a note on segmentation hardware. **[5 M]**

**Ans.** Refer to Section 5.5.2.

**2.** Let the reference string is: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5. Use LRU algorithm to find the number of page faults:
(i) when there are 3 frames.
(ii) when there are 4 frames. **[4 M]**

**Ans.** Refer to Section 5.6.4.1.

**3.** What is page table? What are its contents? **[2 M]**

**Ans.** Refer to Section 5.4.1.

**April 2019**

**1.** Define logical address. **[1 M]**

**Ans.** Refer to Section 5.1.3.

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

**Ans.** Refer to Section 5.3.3.

**3.** Explain demand paging with example. Discuss hardware required for demand paging. **[5 M]**

**Ans.** Refer to Section 5.6.2.

**4.** Consider the page reference string: 7, 2, 8, 4, 5, 8, 4, 7, 6, 1, 3, 7. How many page faults would occur for the following page replacement algorithms assuming four frames.
(i) LRU (ii) Optimal replacement. **[4 M]**

**Ans.** Refer to Section 5.6.4.1.

■■■