ਜਗਤ ਗੁਰੂ ਨਾਨਕ ਦੇਵ
ਪੰਜਾਬ ਸਟੇਟ ਓਪਨ ਯੂਨੀਵਰਸਿਟੀ
ਪਟਿਆਲਾ

The Motto of Our University
(SEWA)
SKILL ENHANCEMENT
EMPLOYABILITY
WISDOM
ACCESSIBILITY

SELF-INSTRUCTIONAL STUDY MATERIAL FOR JGND PSOU

JAGAT GURU NANAK DEV
PUNJAB STATE OPEN UNIVERSITY, PATIALA

(Established by Act No. 19 of 2019 of the Legislature of State of Punjab)

B.Sc.(Data Science)

Semester II

BSDB31201T

Operating Systems

Head Quarter: C/28, The Lower Mall, Patiala-147001
Website: www.psou.ac.in

The Study Material has been prepared exclusively under the guidance of Jagat Guru Nanak Dev Punjab State Open University, Patiala, as per the syllabi prepared by Committee of experts and approved by the Academic Council.

COURSE COORDINATOR AND EDITOR:

Dr. Amitoj Singh
Associate Professor
School of Sciences and Emerging Technologies
Jagat Guru Nanak Dev Punjab State Open University

LIST OF CONSULTANTS/ CONTRIBUTORS

Sr. No.	Name
1	Dr. Rohit Sachdeva
2	Dr. Amitoj Singh
3	Dr. Gurpreet Singh

JAGAT GURU NANAK DEV PUNJAB STATE OPEN UNIVERSITY, PATIALA
(Established by Act No. 19 of 2019 of the Legislature of State of Punjab)

PREFACE

Jagat Guru Nanak Dev Punjab State Open University, Patiala was established in December 2019 by Act 19 of the Legislature of State of Punjab. It is the first and only Open University of the State, entrusted with the responsibility of making higher education accessible to all, especially to those sections of society who do not have the means, time or opportunity to pursue regular education.

In keeping with the nature of an Open University, this University provides a flexible education system to suit every need. The time given to complete a programme is double the duration of a regular mode programme. Well-designed study material has been prepared in consultation with experts in their respective fields.

The University offers programmes which have been designed to provide relevant, skill-based and employability-enhancing education. The study material provided in this booklet is self-instructional, with self-assessment exercises, and recommendations for further readings. The syllabus has been divided in sections, and provided as units for simplification.

The University has a network of 10 Learner Support Centres/Study Centres, to enable students to make use of reading facilities, and for curriculum-based counselling and practicals. We, at the University, welcome you to be a part of this instituition of knowledge.

Prof. Anita Gill
Dean Academic Affairs

B.Sc. (Data Science)
Core Course(CC)
Semester II
BSDB31201T: Operating Systems

Total Marks: 100
External Marks: 70
Internal Marks: 30
Credits: 4
Pass Percentage: 35%

Objective

Understanding basics of operating system viz. system programs, system calls, user mode and kernel mode. Working with CPU scheduling algorithms for specific situation, and analyze the environment leading to deadlock and its rectification. Exploring memory management techniques viz. caching, paging, segmentation, virtual memory, and thrashing.

INSTRUCTIONS FOR THE PAPER SETTER/EXAMINER

1. The syllabus prescribed should be strictly adhered to.
2. The question paper will consist of three sections: A, B, and C. Sections A and B will have four questions from the respective sections of the syllabus and will carry 10 marks each. The candidates will attempt two questions from each section.
3. Section C will have fifteen short answer questions covering the entire syllabus. Each question will carry 3 marks. Candidates will attempt any ten questions from this section.
4. The examiner shall give a clear instruction to the candidates to attempt questions only at one place and only once. Second or subsequent attempts, unless the earlier ones have been crossed out, shall not be evaluated.
5. The duration of each paper will be three hours.

INSTRUCTIONS FOR THE CANDIDATES

Candidates are required to attempt any two questions each from the sections A and B of the question paper and any ten short questions from Section C. They have to attempt questions only at one place and only once. Second or subsequent attempts, unless the earlier ones have been crossed out, shall not be evaluated.

SECTION A

Unit- I: Introduction and System Structures: Computer-System Organization, Computer-System Architecture, Operating-System Structure, Operating-System Operations, Process Management, Memory Management, Storage Management, Protection and Security, Computing Environments, Operating-System Services, User and Operating-System Interface, System Calls, Types of System Calls, System Programs.

Unit II: Process Management: Process Concept, Process Scheduling, Operations on Processes, Multi-threaded programming: Multithreading Models, Process Scheduling: Basic Concepts, Scheduling Criteria, and Scheduling Algorithms.

Unit III: Deadlock: System Model, Deadlock Characterization, Methods for Handling Deadlocks, Deadlock Prevention, Deadlock Avoidance, Deadlock Detection, Recovery from Deadlock.

Unit IV: Memory Management: Basic Hardware, Address Binding, Logical and Physical Address, Dynamic linking and loading, Swapping, Contiguous Memory Allocation, Segmentation, Paging, Demand Paging, Page Replacement algorithms

SECTION B

Unit V: File Systems: File Concept, Access Methods, Directory and Disk Structure, File-System Structure, File-System Implementation, Directory Implementation, Allocation Methods, Free-Space Management.

Unit VI: **Introduction to Linux:** Linux's shell, Kernel, Features of Linux, Using file system: Filenames, Introduction to different types of directories: Parent, Subdirectory, Home directory; rules to name a directory, Important directories in Linux File System,

Unit VII: Linux Commands: cal, date, echo, bc, who, cd, mkdir, rmdir, ls, cat cp, rm, mv, more, gzip, tar, File ownership, file permissions, chmod, Directory permission, change file ownership,

Unit VIII: Shell Scripting: Creating and Executing Shell Programs, Using variables: Assigning a value to a variable, Accessing the value of a variable, Positional Parameters and other Built-In Shell Variables; Special Characters, Conditional Statements : if Statement, case Statement; Iteration Statements : for Statement, while Statement, until Statement

Suggested Readings

1. A Silberschatz, P.B. Galvin, G. Gagne, Operating Systems Concepts, 8th Edition, John Wiley Publications, 2009
2. A.S. Tanenbaum, Modern Operating Systems, 3rd Edition, Pearson Education, 2014
3. G. Nutt, Operating Systems: A Modern Perspective, 2nd Edition Pearson Education, 2000
4. S. Das, Unix Concepts and Applications, 4th edition, McGraw Hill Education, 2017

JAGAT GURU NANAK DEV PUNJAB STATE OPEN UNIVERSITY, PATIALA
(Established by Act No. 19 of 2019 of the Legislature of State of Punjab)

BSDB31201T: OPERATING SYSTEMS

COURSE COORDINATOR AND EDITOR: DR. AMITOJ SINGH

B.Sc.(DATA SCIENCE)

SEMESTER-II

OPERATING SYSTEM

UNIT I: INTRODUCTION AND SYSTEM STRUCTURES

STRUCTURE

1.0 OBJECTIVE

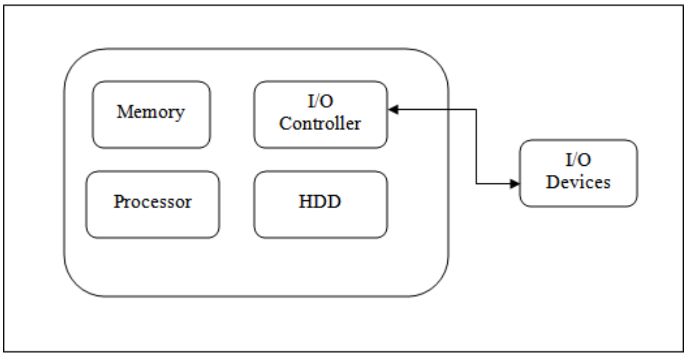
To understand the following

- Computer-System Architecture
- Operating-System Structure, Operating-System Operations
- Operating-System Services, User and Operating-System Interface
- System Calls, Types of System Calls, System Programs.

1.1 Operating System

- An operating system is a program which manages all the computer hardware.
- It provides the base for application program and acts as an intermediary between a user and thecomputer hardware.
- The operating system has two objectives such as:
  - Firstly, an operating system controls the computer's hardware.
  - The second objective is to provide an interactive interface to the user and interpretcommands so that it can communicate with the hardware.
- The operating system is very important part of almost every computer system.

Managing Hardware



- The prime objective of operating system is to manage & control the various hardware resourcesof a computer system.
- These hardware resources include processer, memory, and disk space and so on.
- The output result was display in monitor. In addition to communicating with the hardware the operating system provides on error handling procedure and display an error notification.
- If a device not functioning properly, the operating system cannot be communicate with thedevice.

Providing an Interface

- It provides a stable and consistent way for applications to deal with the hardware without theuser having known details of the hardware.

- If the program is not functioning properly, the operating system again takes control, stops theapplication and displays the appropriate error message.
- Computer system components are divided into 5 parts
  - Computer hardware
  - Operating system
  - Utilities
  - Application programs
  - End user
- The operating system controls and coordinate a user of hardware and various applicationprograms for various users.
- It is a program that directly interacts with the hardware.
- The operating system is the first encoded with the Computer and it remains on the memory alltime thereafter.

System Goals

- The purpose of an operating system is to be provided an environment in which an user canexecute programs.
- Its primary goals are to make the computer system convenience for the user.
- Its secondary goals are to use the computer hardware in efficient manner.

1.2 VIEW OF OPERATING SYSTEM

- **User view:**The user view of the computer varies by the interface being used. The examplesare -windows XP, vista, windows 7 etc. Most computer user sit in the in front of personal computer (pc) in this case the operating system is designed mostly for easy use with some attention paid to resource utilization. Some user sit at a terminal connected to amainframe/minicomputer. In this case other users are accessing the same computer through the other terminals. There user are share resources and may exchange the information. The operating system in this case is designed to maximize resources utilization to assume that all available CPU time, memory and I/O are used efficiently and no individual user takes more than his/her fair and share.The other users sit at workstations connected to network of other workstations and servers. These users have dedicated resources but they share resources such as networking and servers like file, compute and print server. Here the operating system is designed to compromise between individual usability and resource utilization.
- **System view:** From the computer point of view the operating system is the program which is most intermediate with the hardware. An operating system has resources as hardware and software which may be required to solve a problem like CPU time, memory space, file storage space and I/O devices and so on. That’s why the operating

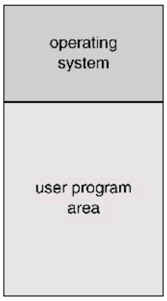
system acts as manager of these resources. Another view of the operating system is it is a control program. A control program manages the execution of user programs to present the errors in proper use of the computer. It is especially concerned of the user the operation and controls the I/O devices.

1.3 TYPES OF OPERATING SYSTEM

1. **Mainframe System:** It is the system where the first computer used to handle many commercialscientific applications. The growth of mainframe systems traced from simple batch system where the computer runs one and only one application to time shared systems which allowed for user interaction with the computer system

   a. **Batch /Early System:** Early computers were physically large machine. The common input devices were card readers, tape drivers. The common output devices were line printers, tape drivers and card punches. In these systems the user did not interact directlywith the computer system. Instead the user preparing a job which consists of programming data and some control information and then submitted it to the computer operator after some time the output is appeared.

   Memory Layout for a Simple Batch System

   The output in these early computer was fairly simple is main task was to transfer control automatically from one job to next. Theoperating system always resides in the memory. To speed up processing operators batched the jobs with similar needs and ran then together as a group. The disadvantages of batch system are that in this execution environment the CPU is often idle because thespeed up of I/O devices is much slower than the CPU.

   b. Multiprogrammed System: Multiprogramming concept increases CPU utilization by organization jobs so that the CPU always has one job to execute the idea behind multiprogramming concept. This set of job is subset of the jobs kept in the job pool. The operating system picks and beginning to execute one of the jobs in the memory. In this environment the operating system simply switches and executes another job. When a job needs to wait the CPU is simply switched to another job and so on. The multiprogramming operating system is sophisticated because the operating system makes decisions for the user. This is known asscheduling. If several jobs are ready to run at the same time the system choose one among them. This is known as CPU scheduling. The disadvantages of the multiprogrammedsystem are

- It does not provide user interaction with the computer system during the program execution.
- The introduction of disk technology solved these problems rather than reading the cards from card reader into disk. This form of processing is known as spooling.

SPOOL stands for simultaneous peripheral operations online. It uses the disk as a huge buffer for reading from input devices and for storing output data until the output devices accept them. It is also use for processing data at remote sides. The remote processing is done and its own speed with no CPU intervention. Spooling overlaps the input, output one job with computation of other jobs. Spooling has a beneficial effect onthe performance of the systems by keeping both CPU and I/O devices working at much higher time.

c. **Time Sharing System:** The time sharing system is also known as multi user systems. TheCPU executes multiple jobs by switching among them but the switches occurs so frequently that the user can interact with each program while it is running. An interactive computer system provides direct communication between a user and system. The user gives instruction to the operating systems or to a program directly using keyboard or mouse and wait for immediate results. So the response time will be short. The time sharing system allows many users to share the computer simultaneously. Since each action in this system is short, only a little CPU time is needed for each user. The system switchesrapidly from one user to the next so each user feels as if the entire computer system is dedicated to his use, even though it is being shared by many users. The disadvantages oftime sharing system are:

- It is more complex than multiprogrammed operating system
- The system must have memory management & protection, since several jobs are keptin memory at the same time.
- Time sharing system must also provide a file system, so disk management is required.
- It provides mechanism for concurrent execution which requires complex CPU scheduling schemes.

2. **Personal Computer System/Desktop System:** Personal computer appeared in 1970's. They are microcomputers that are smaller & less expensive than mainframe systems. Instead of maximizing CPU & peripheral utilization, the systems opt for maximizing user convenience & responsiveness. At first file protection was not necessary on a personal machine. But when other computers 2nd other users can access the files on a pc file protection becomes necessary.The lack of protection made if easy for malicious programs to destroy data on such systems. These programs may be self replicating& they spread via

worm or virus mechanisms. They can disrupt entire companies or even world wide networks. E.g : windows 98, windows 2000, Linux.

3. **Microprocessor Systems/ Parallel Systems/ Tightly coupled Systems**: These Systems have more than one processor in close communications which share the computer bus, clock, memory & peripheral devices. Ex: UNIX, LINUX. Multiprocessor Systems have 3 main advantages.
   a. **Increased throughput:** No. of processes computed per unit time. By increasing the no. of processors move work can be done in less time. The speed up ratio with N processorsis not N, but it is less than N. Because a certain amount of overhead is incurred in keeping all the parts working correctly.
   b. **Increased Reliability:** If functions can be properly distributed among several processors, then the failure of one processor will not halt the system, but slow it down. This ability tocontinue to operate in spite of failure makes the system fault tolerant.
   c. **Economic scale:** Multiprocessor systems can save money as they can share peripherals,storage & power supplies.

   The various types of multiprocessing systems are:

   - **Symmetric Multiprocessing (SMP):** Each processor runs an identical copy of the operating system & these copies communicate with one another as required. Ex: Encore'sversion of UNIX for multi max computer. Virtually, all modern operating systemincluding Windows NT, Solaris, Digital UNIX, OS/2 & LINUX now provide support forSMP.

   - **Asymmetric Multiprocessing (Master – Slave Processors):** Each processor is designed for a specific task. A master processor controls the system & schedules & allocates the work to the slave processors. Ex- Sun's Operating system SUNOS version 4provides asymmetric multiprocessing.

4. **Distributed System/Loosely Coupled Systems**: In contrast to tightly coupled systems, the processors do not share memory or a clock. Instead, each processor has its own local memory. The processors communicate with each other by various communication lines such as high speed buses or telephone lines. Distributed systems depend on networking for their functionalities. By being able to communicate distributed systems are able to share computational tasks and provide a rich set of features to the users. Networks vary by the protocols used, the distances between the nodes and transport media. TCP/IP is the most common network protocol. The processor is a distributed system varies in size and function. Itmay microprocessors, work stations, minicomputer, and large general purpose computers. Network types are based on the distance between the nodes such as LAN (within a room, flooror building) and WAN (between buildings, cities or countries). The advantages of distributed system are resource sharing, computation speed up, reliability, communication.
5. **Real time Systems:** Real time system is used when there are rigid time requirements on

the operation of a processor or flow of data. Sensors bring data to the computers. The computer analyzes data and adjusts controls to modify the sensors inputs. System that controls scientific experiments, medical imaging systems and some display systems are real time systems. The disadvantages of real time system are:

- **a.** A real time system is considered to function correctly only if it returns the correct result within the time constraints.
- **b.** Secondary storage is limited or missing instead data is usually stored in short term memory or ROM.
- **c.** Advanced OS features are absent.Real time system is of two types such as:
- **Hard real time systems:** It guarantees that the critical task has been completed on time.The sudden task is takes place at a sudden instant of time.
- **Soft real time systems:** It is a less restrictive type of real time system where a critical task gets priority over other tasks and retains that priority until it computes. These have more limited utility than hard real time systems. Missing an occasional deadline is acceptable
  e.g. QNX, VX works. Digital audio or multimedia is included in this category.

It is a special purpose OS in which there are rigid time requirements on the operation of a processor. A real time OS has well defined fixed time constraints. Processing must be done within the time constraint or the system will fail. A real time system is said to function correctly only if it returns the correct result within the time constraint. These systems are characterized by having time as a key parameter.

1.4 Functions of Operation System

The various functions of operating system are as follows:

1. **Process Management**

- A program does nothing unless their instructions are executed by a CPU.A process is a program in execution. A time shared user program such as a complier is a process. A word processing program being run by an individual user on a pc is a process.
- A system task such as sending output to a printer is also a process. A process needs certain resources including CPU time, memory files & I/O devices to accomplish its task.
- These resources are either given to the process when it is created or allocated to it while it is running. The OS is responsible for the following activities of process management.
- Creating & deleting both user & system processes.
- Suspending & resuming processes.
- Providing mechanism for process synchronization.
- Providing mechanism for process communication.
- Providing mechanism for deadlock handling.

2. **Main Memory Management**

The main memory is central to the operation of a modern computer system. Main memory is a large array of words or bytes ranging in size from hundreds of thousand to billions. Main memory stores the quickly accessible data shared by the CPU & I/O device. The central processor reads instruction from main memory during instruction fetch cycle & it both reads & writes data from main memory during the data fetch cycle. The main memory is generally theonly large storage device that the CPU is able to address & access directly. For example, for theCPU to process data from disk. Those data must first be transferred to main memory by CPU generated E/O calls. Instruction must be in memory for the CPU to execute them. The OS is responsible for the following activities in connection with memory management.

- Keeping track of which parts of memory are currently being used & by whom.
- Deciding which processes are to be loaded into memory when memory space becomesavailable.
- Allocating &deallocating memory space as needed.

3. **File Management**

File management is one of the most important components of an OS computer can store information on several different types of physical media magnetic tape, magnetic disk & optical disk are the most common media. Each medium is controlled by a device such as disk drive or tape drive those has unique characteristics. These characteristics include access speed, capacity, data transfer rate & access method (sequential or random).For convenient use of computer system the OS provides a uniform logical view of information storage. The OS abstracts from the physical properties of its storage devices to define a logical storage unit the file. A file is collection of related information defined by its creator. The OS is responsible for the followingactivities of file management.

- Creating & deleting files.
- Creating & deleting directories.
- Supporting primitives for manipulating files & directories.
- Mapping files into secondary storage.
- Backing up files on non-volatile media.

4. **I/O System Management**

One of the purposes of an OS is to hide the peculiarities of specific hardware devices from the user. For example, in UNIX the peculiarities of I/O devices are hidden from the bulk of the OSitself by the I/O subsystem. The I/O subsystem consists of:

- A memory management component that includes buffering, catching & spooling.
- A general device- driver interfaces drivers for specific hardware devices. Only the device driver knows the peculiarities of the specific device to which it is assigned.

5. **Secondary Storage Management**

The main purpose of computer system is to execute programs. These programs with the data they access must be in main memory during execution. As the main memory is too small to accommodate all data & programs & because the data that it holds are lost when power is lost. The computer system must provide secondary storage to back-up main memory. Most modern computer systems are disks as the storage medium to store data & program. The operating system is responsible for the following activities of disk management.

- Free space management.
- Storage allocation.
- Disk scheduling

Because secondary storage is used frequently it must be used efficiently.

1.5 NETWORKING

A distributed system is a collection of processors that don't share memory peripheral devices or a clock. Each processor has its own local memory & clock and the processor communicate with one another through various communication lines such as high speed buses or networks. The processors in the system are connected through communication networks which are configured in a number ofdifferent ways. The communication network design must consider message routing & connection strategies are the problems of connection & security.

1.6 PROTECTION OR SECURITY

If a computer system has multi users & allow the concurrent execution of multiple processes then the various processes must be protected from one another's activities. For that purpose, mechanisms ensure that files, memory segments, CPU & other resources can be operated on by onlythose processes that have gained proper authorization from the OS.

1.7 SYSTEM CALLS

System calls provide the interface between a process & the OS. These are usually available in the form of assembly language instruction. Some systems allow system calls to be made directly from ahigh level language program like C, BCPL and PERL etc. systems calls occur in different ways depending on the computer in use. System calls can be roughly grouped into 5 major categories.

1.7.1 Process Control

- **End, abort:** A running program needs to be able to has its execution either normally (end) orabnormally (abort).
- **Load, execute:** A process or job executing one program may want to load and executes another program.
- **Create Process, terminate process:** There is a system call specifying for the purpose

of creating a new process or job (create process or submit job). We may want to terminate a job or process that we created (terminates process, if we find that it is incorrect or no longer needed).

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

1.7.2 File Manipulation

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

1.7.3 Device Management

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

1.7.4 Information Maintenance

- **Get system data, set system data:** Other system calls may return information about thesystem like number of current users, version number of OS, amount of free memory etc.
- **Get process attributes, set process attributes:** The OS keeps information about all itsprocesses & there are system calls to access this information.

1.7.5 Communication

There are two modes of communication such as:

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

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

1.8 SYSTEM PROGRAMS

System programs provide a convenient environment for program development & execution. Theyare divided into the following categories.

a. **File manipulation:** These programs create, delete, copy, rename, print & manipulate filesand directories.

b. **Status information:** Some programs ask the system for date, time & amount of availablememory or disk space, no. of users or similar status information.

i. **File modification:**Several text editors are available to create and modify the contents of filestored on disk.

ii. **Programming language support:** compliers, assemblers & interpreters are provided to the user with the OS.

iii. **Programming loading and execution:** Once a program is assembled or compiled, it mustbe loaded into memory to be executed.

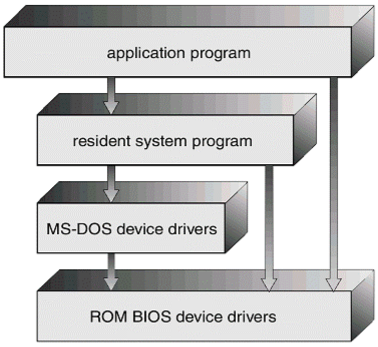
iv. **Communications:** These programs provide the mechanism for creating virtual connectionsamong processes users 2nd different computer systems.

v. **Application programs:** Most OS are supplied with programs that are useful to solve common problems or perform common operations. Ex: web browsers, word processors & text formatters etc.

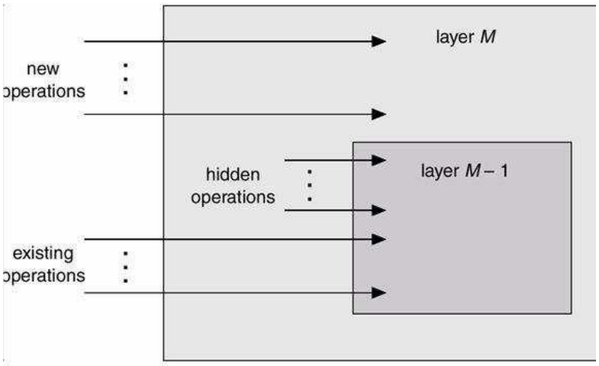
1.8 SYSTEM STRUCTURE

1. **Simple structure:** There are several commercial system that don’t have a well-defined structure such operating systems begins as small, simple & limited systems and then grow beyond their original scope. MS-DOS is an example of such system. It was

not divided intomodules carefully. Another example of limited structuring is the UNIX operating system.



2. **Layered approach:** In the layered approach, the OS is broken into a number of layers (levels) each built on top of lower layers. The bottom layer (layer o) is the hardware & top most layer (layer N) is the user interface. The main advantage of the layered approach is modularity.



- The layers are selected such that each users functions (or operations) & services of only lower layer.

This approach simplifies debugging & system verification, i.e. the first layer can be debugged without concerning the rest of the system. Once the first layer is debugged, its correct functioning is assumed while the 2nd layer is debugged & so on.

If an error is found during the debugging of a particular layer, the error must be on that layer because the layers below it are already debugged. Thus the design & implementation of the system are simplified when the system is broken down into layers.

Layers	Functions
5	User Program
4	I/O Management
3	Operator Process Communication
2	Memory Management
1	CPU Scheduling
0	Hardware

Each layer is implemented using only operations provided by lower layers. A layer doesn’t need to know how these operations are implemented; it only needs to know what these operations do.

The layer approach was first used in the operating system. It was defined in six layers.

The main disadvantage of the layered approach is:

- The main difficulty with this approach involves the careful definition of the layers, because a layer can use only those layers below it. For example, the device driver for thedisk space used by virtual memory algorithm must be at a level lower than that of the memory management routines, because memory management requires the ability to usethe disk space.
- It is less efficient than a non layered system (Each layer adds overhead to the system call & the net result is a system call that take longer time than on a non layered system).

1.10 Operating System Services

An operating system provides an environment for the execution of the program. It provides some services to the programs. The various services provided by an operating system are as follows:

- **Program Execution:** The system must be able to load a program into memory and to run that program. The program must be able to terminate this execution either normally or abnormally.
- **I/O Operation:** A running program may require I/O. This I/O may involve a file or a I/O device for specific device. Some special function can be desired. Therefore the operating system must provide a means to do I/O.
- **File System Manipulation:** The programs need to create and delete files by name and read and write files. Therefore the operating system must maintain each and every files correctly.
- **Communication:** The communication is implemented via shared memory or by the technique of message passing in which packets of information are moved between the processes by the operating system.
- **Error detection:** The operating system should take the appropriate actions for the occurrences of any type like arithmetic overflow, access to the illegal memory location andtoo large user CPU time.
- **Research Allocation:** When multiple users are logged on to the system the resources mustbe allocated to each of them. For current distribution of the resource among the various processes the operating system uses the CPU scheduling run times which determine which process will be allocated with the resource.
- **Accounting:** The operating system keep track of which users use how many and which kindof computer resources.
- **Protection:** The operating system is responsible for both hardware as well as software protection. The operating system protects the information stored in a multiuser computer system.

1.11 PRACTICE EXERCISE

a. Why is the operating system important?

b. What's the main purpose of an OS? What are the different types of OS?

c. What are the benefits of a multiprocessor system?

d. What is a bootstrap program in OS?

e. What is the main purpose of an operating system?

f. What are the different operating systems?

g. What is kernel?

h. What is monolithic kernel?

i. What do you mean by a process?

B.Sc.(DATA SCIENCE)

SEMESTER-II

OPERATING SYSTEM

UNIT II: PROCESS MANAGEMENT

STRUCTURE

2.0 OBJECTIVES

To understand the following

- Different types of Process Scheduling
- Operations on Processes
- Multi-threaded programming and Model
- CPU Scheduling
- Scheduling Concepts
- Scheduling Criteria
- Scheduling Algorithms

2.1 INTRODUCTION TO PROCESS

2.1.1 Process Basics

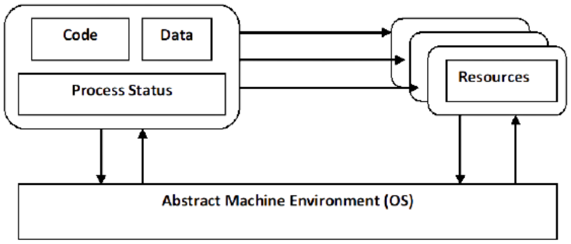
The fundamental activity of an operating system is the creation, management, and termination of processes. Now the question comes to mind, what is a process?

A process is a program under execution or the "animated" existence of a program or an identifiable entity executed on a processor by the operating system.

A process may be defined as an instance of a program in execution. It is also known as a task. An operating system manages each hardware resource attached to the computer by representing it as an abstraction. Abstraction hides unwanted details from the users and programmers allowing them to have a view of the resources in the form, which is convenient to them. A process is an abstract model of a sequential program in execution. The operating system can schedule a process as a unit of work. A process can be identifying in an operating system by its following components:

The object program (or code) to be executed.
The data on which the program will execute.
The status of the process execution.
The stack is associated with a process to store parameters and calling addresses.

A process may be represented schematically as in the figure.



Process Abstraction

2.1.2 Process Description

To control and manage the processes, the Operating system records the information about them in the primary process table. The primary process table is used to keep one entry per process in the operating system. Each entry contains at least one pointer to a process image. The Process Image contains **User Data**: It contains program data that can be modified etc., **Code:** The sequence of instructions (program) to be executed, **Stack:** Each process has one or more stacks associated with it. A stack is used to saved parameters of the process and calling addresses for process, system calls, and **Process Control Block (PCB)** of process in which data needed by the operating system to control the process (attributes and information about the process) is stored.

2.1.3 Process Control Block

The data structure that stores information about a process is called Process Control Block. When a process is initialized, the corresponding process control block of the process is created. Information in a process control block is updated during the transition of process states.

A process control block is a location in the main memory, where various information of a process regarding memory, process, and I/O management is stored. Each process has a single process control block. When a process is completed the process control block is unloaded from the memory. The information stored in a process control block is:

Pointer	**Process State**
Process Number	
Program Counter	
Registers	
Memory Limits **List of open files**	
. . . .	

Figure: Process Control Board (PCB)

Each process has a **priority**, implemented in terms of numbers. The higher priority processes have precedence over lower priority processes. The priority field is used for storing the priority of a process.

A new process can be created from the existing process. The existing process is called the parent of the newly created process. The field, **link** (pointer) to the parent process stores the address of the process control block of the parent process in the main memory. The field, link (pointer) to the child process stores the address of the process control block of the child process in the main memory.

Process State field stores the information about the recent state of the process. The state of the process may be new, waiting, ready, running, and so on.

Process Number filled to store the numeric value which is an identifier of the process.

The PCB stores information regarding the programming environment of a process. The programming environment information includes the value of the registers, stack, and program counter.

A **program counter** is a special register that saves the address of the next instruction to be executed for this process.

A PCB also stores information regarding memory management, such as the number of memory units allocated to the process and the addresses of the memory chunks allocated.

Registers: It includes a general-purpose register, stack pointers, index register, and accumulators, etc. Many register and accumulators etc. Some register and type of register depend upon the computer architecture.

The PCB control block stores file management information. An example of the file management information stored in the process control block is:

- The number of files opens.
- List of the open files.
- The access right of the files opens, such as read-only or read-write.

2.1.4 Role of PCB

The process control block is the most important data structure in an OS. Each Process Control Block contains all of the data about a process that is required by the Operating System. The blocks are read as well as modified by virtually every unit in the Operating System, including those which involved scheduling, interrupt processing, resource allocation, and performance monitoring and analysis of the process.

One can say that the set of process control blocks defines the state of the OS. This brings up an important design issue. Many routines within the OS will need access to information in process control blocks. The provision of direct access to these tables is not difficult. Each process is equipped with a unique ID, and this can be used as an index into a table of pointers to the process control blocks.

The difficulty is not access but rather protection. There are two problems :

1. A bug in a single routine, such as an interrupt handler, could damage process control blocks, which could destroy the system’s ability to manage the affected.
2. A design change in the structure or semantics of the process control block could affect many modules in the OS.

2.2 PROCESS SCHEDULERS

Several types of schedulers can be used in an OS. Schedulers are classified according to the type and duration of processes. Schedulers are classified as:

□ Long-term Scheduler/High-Level Scheduler
□ Medium-term Scheduler/ Intermediator Scheduler
□ Short-term Scheduler/Low-Level Schedular

2.2.1 Long-term Scheduler

Long-term scheduling identifying which programs is to be admitted to the system as new processes. Once a new process is accepted, it may enter the scheduling queues in one of two places:

- If all resources are initially fully available to the new process, they may be admitted to the tail of the ready queue.
- If all resources are not immediately available, the new process may be entered in the blocked-suspended queue until those resources are provided.

Long-term Scheduler plans the CPU scheduling for batch jobs. Processes, which are resource-intensive and have a low priority, are called batch jobs are executed in a group or bunch. An example of a batch job is a user request for printing a bunch of files.

2.2.2 Medium-term Scheduler

Medium-term scheduling is part of the swapping role of an operating system. Medium-term Scheduler plans CPU scheduling for the processes that have been waiting for the completion of another process or an I/O task that requires a long time. A process is suspended or blocked marked as waiting if it is waiting for the completion of a long time I/O task. These processes are removed from the main memory and stored in the swapped-out queue in the secondary memory to create space in the main memory. The swapped-out queue is implemented in the secondary memory for storing the waiting processes that have been swapped out of the main memory. After completion of the I/O operation, the suspended or blocked processes are resumed and placed in the ready queue in the main memory. If the process is waiting for the completion of a short-term I/O task, the process is not swapped out of the main memory and is not handled by the medium-term scheduler. The success of the medium-term schedules is based on the degree of multiprogramming that it can maintain, by keeping as many processes “runnable” as possible.

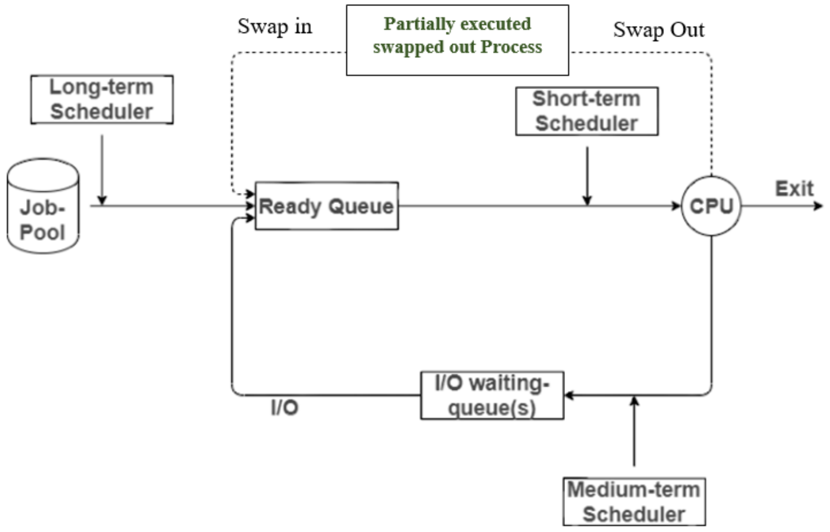


Figure: Working of medium-term scheduler

2.2.3 Short-term Scheduler

Short-term Scheduler plans the scheduling of the processes that are in a ready state. Short-term schedulers retrieve a process from the ready queue and allocate CPU time to it. The process state is changed from ready to run. If an interrupt or time-out occurs the scheduler places the running process back into the ready queue and marks the running process as ready. **The figure below shows how short, medium, long term schedulers work.**

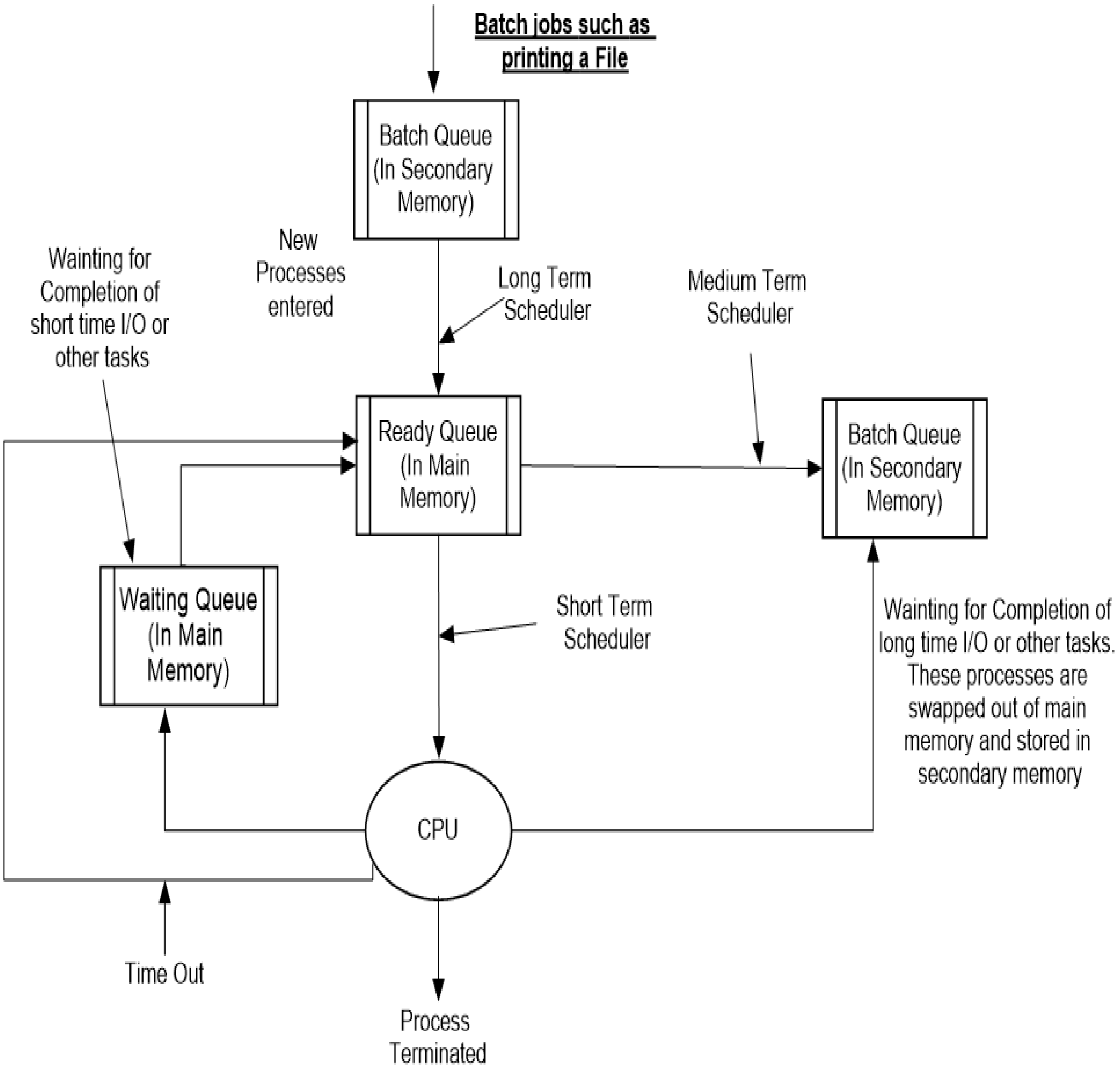

Batch jobs such as printing a File
Batch Queue (In Secondary Memory)
New Processes entered
Long Term Scheduler
Medium Term Scheduler
Wainting for Completion of short time I/O or other tasks
Ready Queue (In Main Memory)
Batch Queue (In Secondary Memory)
Waiting Queue (In Main Memory)
Short Term Scheduler
Wainting for Completion of long time I/O or other tasks. These processes are swapped out of main memory and stored in secondary memory
CPU
Time Out
Process Terminated

Figure: Role of various type schedulers

If a new process is a batch job, it is placed in the batch queue; otherwise, it is ready into the ready queue. The long-term scheduler selects a batch job on a first-come-first-served basis and sends it to the ready queue for execution.

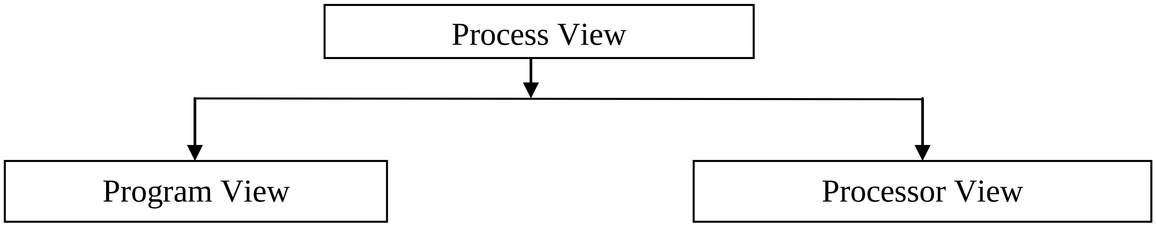
The short-term scheduler selects a process from the ready queue for execution. If a process waits for a short time for completion of an I/O task it is placed in the waiting queue, otherwise, the waiting process is swapped out of the main memory and placed in the swapped-out queue, which is implemented in the secondary memory.

Sr. No.	Long Term	Short Term	Medium Term
1.	It is a job scheduler.	It is a CPU scheduler.	It is swapping.
2.	Speed is less than short term schedular	Speed is very fast.	Speed is in between both.
3.	It controls the degree of Multiprogramming.	Less control over the degree of Multiprogramming.	Reduce the degree of Multiprogramming.
4.	Absent or minimal in time sharing system	Minimal in a time-sharing system	The Time-sharing system uses a medium-term scheduler.
5.	It selects processes from pool and load them into memory for execution.	It selects from among the processes that are ready to execute	The process can be reintroduced into memory and its execution can be continued.
6.	Process state is (new to Ready.)	Process state is (Ready to Running)	Process state is (waiting)
7.	Select a good process, a mix of I/O bound and CPU bound	Select a new process for a CPU quite frequently.	Select suspended Process

2.3 OPERATION ON PROCESSES

Process State

The principal function of a processor is to execute machine instructions residing in the main memory. We can view the process from two points of view: Program View and Processor View.



Different Process View

Program View: Its execution involves a sequence of instructions within that program. The behavior of individual process can be characterized by a list of the sequence of instructions – a ***trace*** of the process

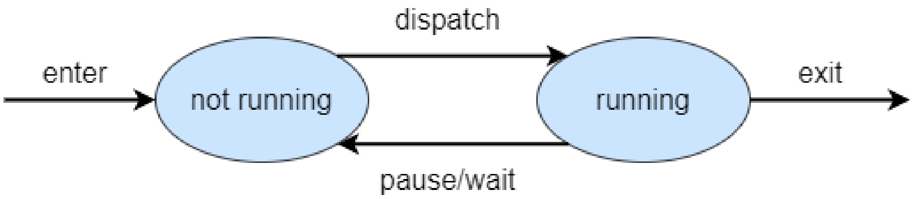
Processor View: It executes instructions from the main memory, as dictated by changing values in the program counter register. The behavior of the processor can be characterized by showing how the traces of various processes are interleaved.

The Simple Two-State Process Model

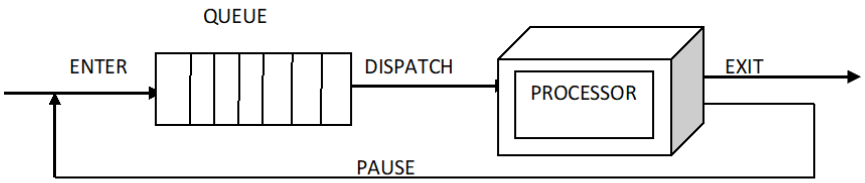
The operating system's principal responsibility is controlling the execution of processes. This includes determining the interleaving pattern for execution and allocating resources to processes. The first step in designing an OS to control processes is to describe the behavior that we would like the processes to exhibit.

We can construct the simplest possible model by observing that, at any time, a process is either being executed by a processor or not. In this model, a process may be in one of two states: **Running or Not Running**, as shown in the figure Two-State Process Model **State transition Diagram**. When the OS creates a new process, it creates a Process Control Block for the process and enters that process into the system in the Not Running state. The process exists, is known to the OS, and is waiting for an opportunity to execute. From time to time, the currently running process will be interrupted and the dispatcher portion of the OS will select some other process to run. The former process moves from the Running state to the Not Running state, and one of the other processes moves to the Running state.

From this simple model, we can already begin to appreciate some of the design elements of the OS. Each process must be represented in some way so that the OS can keep track of it. That is, there must be some information relating to each process, including the current state and location in memory; this is the Process Control Block. Processes that are not running must be kept in some sort of queue, waiting their turn to execute.



Two-State Process Model State transition Diagram



Two-State Process Model Queuing Diagram

Process Creation

When a new process is to be added to those currently being managed, the OS builds the data structures that are used to manage the process and allocates address space in the main memory to the process. These actions constitute the creation of a new process. Four common events leading to the creation of a process. In a batch environment, a process is created in response to the submission of a job. In an interactive environment, a process is created when a new user attempts to log on. In both cases, the OS is responsible for the creation of the new process. An OS may also create a process on behalf of an application. For example, if a user requests that a file be printed, the OS can create a process that will manage the printing. The requesting process can thus proceed independently of the time required to complete the printing task.

Traditionally, the OS created all processes in a way that was transparent to the user or application program, and this is still commonly found with many contemporary operating systems. However, it can be useful to allow one process to cause the creation of another. For example, an application process may generate another process to receive data that the application is generating and to organize those data into a form suitable for later analysis. The new process runs in parallel to the original process and is activated from time to time when new data are available. This arrangement can be very useful in structuring the application. As another example, a server process (which may be a print server or file server) may create a new process for each request that it handles. When the OS creates a new process at the explicit request by another process, this work is known as **process spawning**.

Parent Process

When one process spawns another, the former is referred to as the **parent process**.

Child Process

The spawned process is referred to as the **child process**.

Typically, the "related" processes need to communicate and cooperate. Achieving this cooperation is a difficult task for the programmer.

Process Termination

Any computer system must provide a means for a process to indicate its completion. A batch job should include a Halt instruction or an explicit OS service call for termination. In the former case, the Halt instruction will generate an interrupt to alert the OS that a process has been completed. For an interactive application, the action of the user will indicate when the process is completed. When the parent process terminates, the OS may automatically terminate all its children. For example, in a time-sharing system, the process for a particular user is to be terminated when the user logs off or turns off his or her terminal. On a personal computer or workstation, a user may quit an application (e.g., word processing or spreadsheet). All of these actions ultimately result in a service request to the OS to terminate the requesting process.

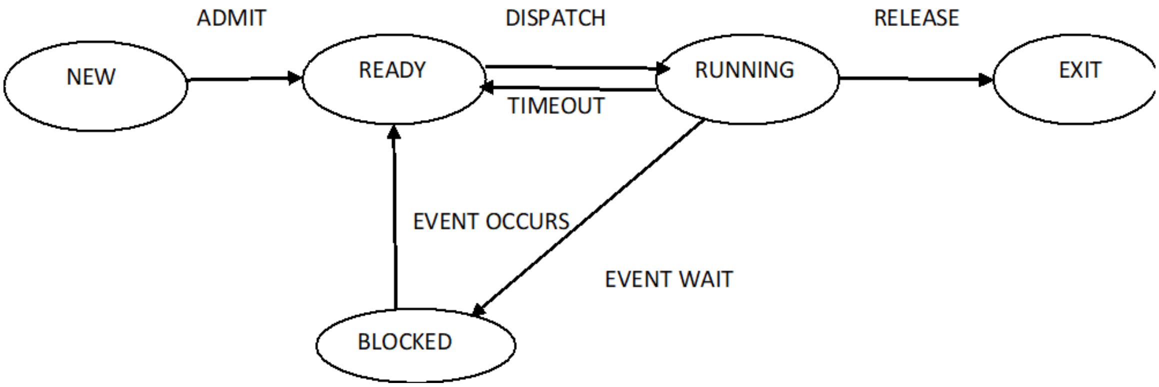
Five State Process Model

A process is a program in execution. In the five-state process model, there are five states of a process. A process may be in any one of the states during its lifetime. First of all, a process arrives into the system for its execution, then it becomes ready for execution and, then it gets the attention of the processor, and at last, the process is terminated after the completion of its execution. When the parent process terminates, the operating system may automatically terminate all its children.

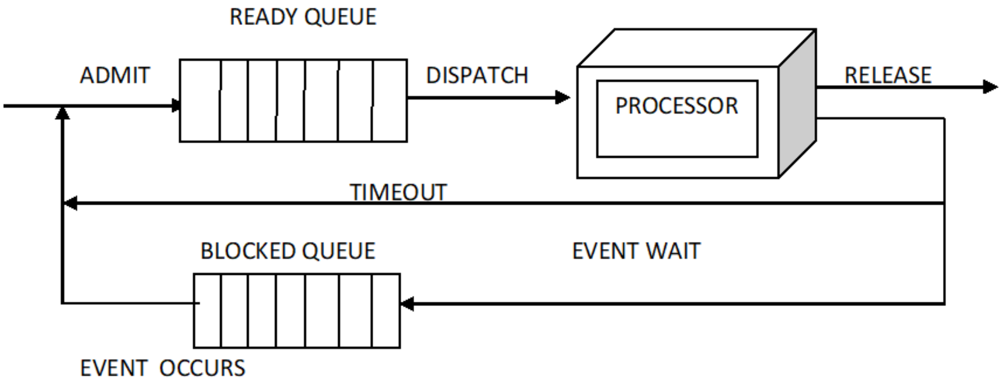
The queue is a first-in-first-out list and the processor operates in a **round-robin** fashion on the available processes (each process in the queue is given a certain amount of time, in turn, to execute and then returned to the queue, unless blocked). Some processes in the Not Running state are ready to execute, while others are blocked, waiting for an I/O operation to complete.

Thus, using a single queue, the dispatcher could not just select the process at the oldest end of the queue. Rather, the dispatcher would have to scan the list looking for the process that is not blocked and that has been in the queue the longest.

A more natural way to handle this situation is to split the Not Running state into two states: Ready and Blocked. We have added two additional states that will prove useful. The five states in this **new** diagram are as follows:



(i) **Five-State Process Model** State Transition Diagram



(ii) **Five-State Process Model** Queuing Diagram

So, the process either be in any of one of the below-mentioned states -:

1. **NEW**: The process is said to be in the NEW state when it arrives into the system for its execution. In other words, the process enters the JOB POOL of the system. A process that has been newly created but has not admitted yet into the pool of executable processes by the OS. Typically, a new process has not yet been loaded into the main memory, although its process control block has been created.

2. **READY**: The process is supposed to be in the READY state when it is ready to get the attention of the CPU In other words, the process enters the READY QUEUE of the system. A READY process is that, which is prepared to execute when given the opportunity.

3. **RUNNING**: A process is supposed to be running, when it gets the attention of the CPU for its execution. In other words, the process is being executed by the CPU in this stage.

4. **WAITING**: A process is said to be waiting when the process is blocked for some time due to some reason. In other words, the process switches (jumps) from the RUNNING stage to the BLOCKED stage for some time. A process that cannot execute until some event happens, the event may be the completion of an Input or Output operation. *Waiting* is a frequently used alternative term for *Blocked* as a process state. There are various reasons for the blocking of a process like I/O Required, Time Slice Elapses, Higher Priority Job Arrives, etc.

5. **TERMINATED**: A process is supposed to be in the TERMINATED state when the process completes its execution successfully. A process which released from the pool of executable processes by the Operating System, either because it paused or because it was abandoned for some reason.

There are two queues now: ready queue and blocked queue

When the process is admitted in the system, it is placed in the ready queue and when a process is removed from the processor, it is either placed in the ready queue or a blocked queue (depending on circumstances). If event time out occurs, then it moves to ready queue, and if event wait occurs then it moves to blocked queue. When an event occurs, all the processes waiting on that event are moved from the blocked queue onto the ready queue.

The New and Exit states are useful constructs for process management. The New state refers to a process when it has been just defined. For example, if a new user starts to log onto a system or a new batch job is submitted to the OS for execution, the OS can define a new process in two stages. First, the OS performs the necessary housekeeping chores. An identifier is associated with the process. Any tables that will be needed to manage the process are allocated and built. At this point, the process is in a New state. This means that the OS has performed the necessary actions to create the process but has not committed itself to the execution of the process. For example, the OS may limit the number of processes that may be in the system for reasons of performance or main memory limitation. While a process is in the new state, information concerning the process that is needed by the OS is maintained in control tables in the main memory. However, the process itself is not in the main memory. That is, the code of the program to be executed is not in the main memory, and no space has been allocated for the data associated with that program.

While the process is in the New state, the program remains in secondary storage, typically disk storage Similarly, a process exits a system in two stages. First, a process is terminated once it reaches its usual completion point, when it aborts due to an unrecoverable error, or when another process with the appropriate authority causes the process to abort. Termination moves the process to the exit state. At this point, the process is no longer eligible for execution. The tables and other information associated with the job are temporarily preserved by the OS, which provides time for auxiliary or support programs to extract any needed information. For example, an accounting program may need to record the processor time and other resources utilized by the process for billing purposes. A utility program may need to extract information about the history of the process for purposes related to performance or utilization analysis. Once these programs have extracted the needed information, the OS no longer needs to maintain any data relating to the process and the process is deleted from the system. Memory committed to existing processes. This limit assures that there are not so many active processes as to degrade performance.

2.4 MULTI-THREADED PROGRAMMING

A thread is a single sequential flow of control within a program. A process is defined sometimes as a ***heavyweight process*** and a thread is defined as a ***lightweight process***. A thread belongs only to one process. It is a unit of computation associated with a particular heavyweight process, using many of the associated process's resources. It has a minimum internal state and a minimum of allocated resources. Threads can share the same resources (files, memory space, etc) which are in that process. Thread can be created faster as compare to the process. Threads are widely used in real-time operating systems and modern operating systems. Each thread has its control block, with a state (Running/Blocked/etc.), saved registers, instruction pointer. Threads improve application performance through parallelism. This concept is useful in a server-client environment. A process can have a single thread or multiple threads. In multiple threads, each thread is associated precisely to one process and it always inside of the process, which means no thread can exist outside a process.

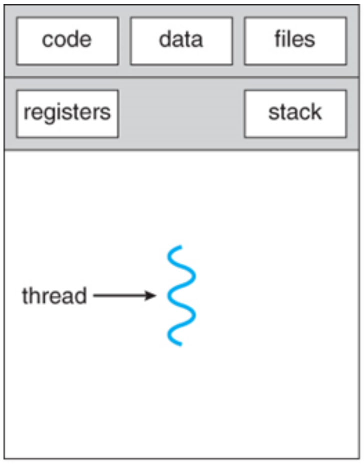


Figure: Representation of Single Thread Process

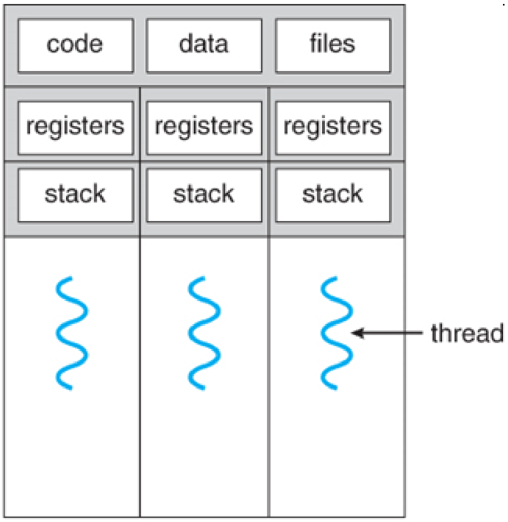


Figure: Representation of Multi-Thread Process

Advantages of Threads

- Thread minimizes context switching time.
- Threads help in the parallel execution of an application on shared-memory multiprocessors.
- The benefits of multi-threading can be greatly increased in a multiprocessor architecture.
- It is more cost-effective to create and context switch threads.
- Threads are dependent on each other so it has efficient communication.
- Threads improve application performance through parallelism.

Threads can be created at the user level and the kernel level so it is implemented by two ways:

(i) User Level (ii) Kernel Level

(i) User Level

In the user-level thread, all of the work of thread management is done by the application and the Kernel does not aware of its existence. In user space, the thread library contains code for creating and destroying threads, message passing, data, scheduling of thread execution, etc.

Characteristic of User Level Threads

- It is generally fast to create and easy to manage.
- User-type threads can run on any OS.
- A multithreaded application cannot take advantage of multiprocessing.
- Scheduling can be application-specific.

(ii) **Kernel Level**

In Kernel-level thread, all of the work of thread management is done by the Kernel. Kernel threads are directly supported by the operating system. The kernel maintains context information for the process as a whole and individual threads within the process. In Kernel space, thread library contains code for creating and destroying threads, message passing, data, scheduling of thread execution, etc.

Characteristic of Kernel Level threads

- In the kernel-level thread, scheduling is on a thread basis.
- It is generally slow to create.
- The kernel can schedule multiple threads from the same process on multiple processes.
- Kernel routines themselves can be multithreaded.
- Context switching between threads is time-consuming.

Comparison between process and thread

Process	Thread
The process is termed the heavyweight process	Thread is termed lightweight process
Processes are independent of one another	Threads are not independent of one another
In multiple process implementations, each the process executes the same code but has its memory and file resources	All threads can share the same set of open files, child processes.
It takes more time to create a new process.	It takes less time to create a new thread.
It takes more time to terminate a process	It takes less time to terminate a thread
While context switching needs the interface with the operating system.	In the thread, switching does not need to call an OS and cause an interrupt to the kernel.
It takes more time to switch between two processes	It takes less time to switch between two threads

Difference between User-level thread and Kernel level thread

User Level Thread	Kernel level thread
It is maintained at the user level.	It is maintained at the Kernel level.
User-level threads are quicker to create	Kernel level threads are not quickly created
It is easy and speedily managed	It is slowly managed.
It runs on any Operating System	These are specific to Operating System.
Implemented by a thread library at the user level	OS support directly to Kernel threads.

Daemon

A Daemon is a system process. It is created at boot time and keeps executing in the background. A daemon is created to perform specific tasks. It gets activated automatically when a request is received for performing a particular task. After completion of the task, it again goes back to the background.

2.5 MULTI-THREADED MODELS

Some operating system allows the facility of both level threads user-level thread and Kernel level thread. An example of this combined approach is Solaris. Multithreading permits the accomplishment of multiple parts of a program run in parallel. These parts of a program are called threads. It was also known as the lightweight process. By using multitasking and multithreading, CPU utilization increased.

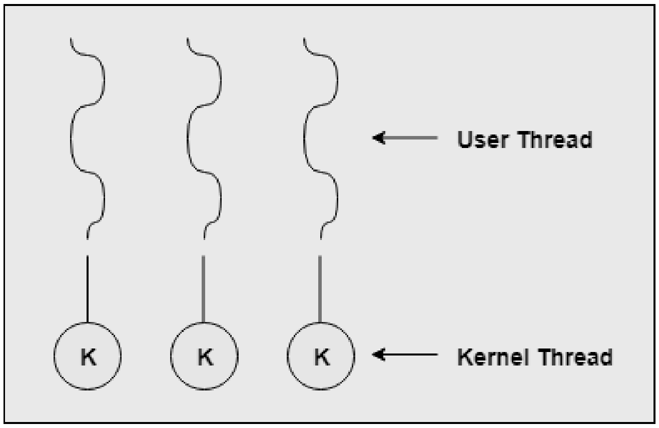
Multithreading models are 3 models namely:

one-one model, many to one model, and many to many models which are described below:

One to One Model

There is 1 to1 association between both the thread's user-level thread and kernel-level thread. This model offers more concurrency than the M:1 model. It also permits another thread to complete its work when a thread makes a blocking system call. A major disadvantage of this is when a user-level thread is created then the requirement of corresponding kernel thread. Therefore a lot of kernel-level threads are required which is a burden on the system, but there is a limit on the number of threads in the system.

A diagram that reveals the one-to-one model is given below –



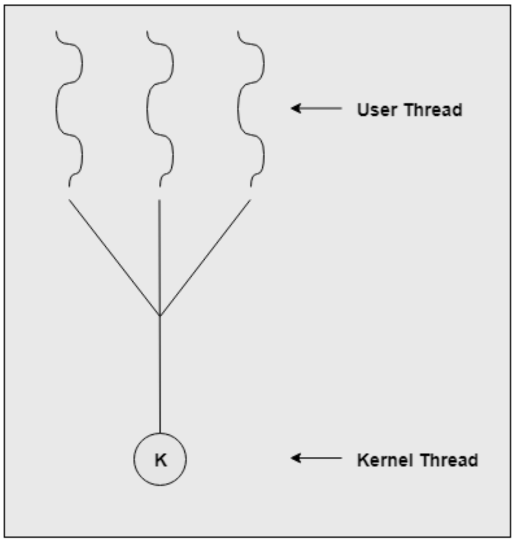
One to One Model

Many to One Model

The many to one model, in this model many of the user threads maps to a single kernel thread. This model is relatively efficient as compare to the 1:1 model as in this the user space was managed by the thread management.

A disadvantage of this model is that when a thread blocking system calls then it blocks the whole process. Another disadvantage is that multiple threads cannot run simultaneously because one thread can access the kernel at a single time.

A given below diagram that shows the many to one model:-



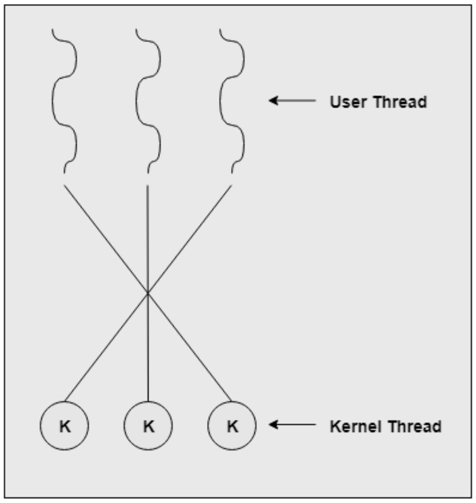
Many to One Model

Many to Many Model

In the many-to-many model, any number of user threads correspond with any number of kernel-level threads. There is no limit. So there is no disadvantage as compared to other models. These threads can execute simultaneously on a multiprocessor.

2.6 CPU SCHEDULING

Scheduling refers to the set of policies and mechanisms that an OS supports for determining the order of execution of pending jobs and processes. A scheduler is an OS Module that determines the next pending job to be admitted into the system for execution or the next ready process to be dispatched to RUN state.



Many to Many Model

Introduction

In the multiprogramming OS, the method or procedure for switching the CPU among multiple processes is called **CPU scheduling**. The CPU scheduler is a part of an OS, which is responsible for CPU scheduling. When a process performs an I/O-related task, it does not use certain resources, such as CPU and these resources remain idle.

CPU scheduling enables processes to utilize idle resources by assigning them to other processes. Whenever CPU becomes idle the CPU scheduler chooses a process among the processes which are in the ready queue and sends the process to the CPU for execution.

For example, process A is running and needs to perform an I/O-related task. Process A does not need the CPU while performing the I/O-related task. Process scheduler changes the state of process A from running to waiting and enables process B to use the CPU.

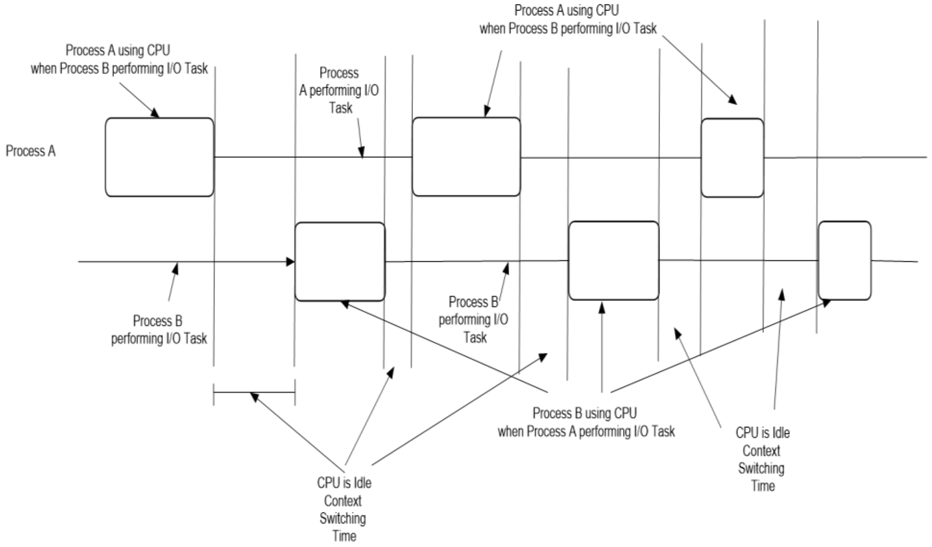

Process A using CPU when Process B performing I/O Task
Process A performing I/O Task
Process A using CPU when Process B performing I/O Task
Process A
Process B performing I/O Task
Process B performing I/O Task
Process B using CPU when Process A performing I/O Task
CPU is Idle Context Switching Time
CPU is Idle Context Switching Time

Figure: CPU Scheduling

For example, in the process of CPU scheduling, two processes, A and B, share the CPU times. When process A uses the CPU, process B performing I/O operations, and when the CPU is accessed by process B, Process A performs I/O operations.

Goals of Scheduling

(i) To optimize the utilization of system resources.
(ii) To ensure that more critical processes get priority over others processes.
(iii) To provide as fair a deal as possible to all jobs and processes which are pending.

When a process, in the ready state, is allocated the CPU in the place of a partially done process, the value of the various field of the partially done process, such as process state and I/O status is updated in the Process Control Block of the partially done process. The OS marks the partially done process as ready and sends it back to the ready queue. The OS reads the data from the process control block of the ready process and allocates CPU to it and marks the process state running. This is called **context switching**.

In other words, “Context switching” refer to the process of transferring control of the CPU from the currently running process to another process from the ready queue.

For example, a time-out has occurred for process A. Process A will be released. Process B is to be allocated the CPU. The steps performed by the **dispatcher** in context switching are:

- It retrieves and analyzes information about process A regarding program counter, memory, and registers.
- It updates the PCB of process A by writing the new values of the various fields of PCB. This includes changing the state of process A from running to ready or waiting.
- It moves process A to the appropriate queue. If the state of process A is changed into a waiting state, it is inserted in the waiting queue. If process A is transformed into the ready state, it is inserted into the ready queue.
- It retrieves and analyzes information about process B from the PCB of process B regarding program counter, memory, and registers.
- It allocates the CPU time to process B
- It restores the environment values, such as program counter, memory, and registers of process B.

The context switching is pure overhead. The extent of this overhead depends on the size of the process content. The larger the process content, the higher will be the context-switching overhead.

2.7 CPU SCHEDULING CRITERIA

Factors for measuring the performance of an operating system

There are certain factors on which the performance of an operating system depends. The efficiency and overall performance of the operating system can be measured in terms of the following factors:

(a) **CPU utilization**: CPU utilization refers to the usage of the processor during the execution of a process.CPU utilization may vary from 40% to 90%.CPU should remain as busy as possible. So, CPU utilization should be the maximum for the better efficiency of a system.

(b) **Throughput**: Throughput may be defined as the total number of processes completed per unit of time. It is the measure of work done by the CPU. It is expressed in terms of the number of jobs done in a given unit of time. It is important to know that the value of throughput does not depend only on the capability of the system but also on the nature of jobs, so, it may vary accordingly. If the jobs are CPU bound then the throughput will be less and if the jobs are I/O bound, the throughput will be high.

If the jobs are long and heavy, the throughput may be one or two processes per hour. On the other hand, if the jobs are short and light, the throughput may be 100 processes per hour. However, it should be as maximum as possible.

(c) **Turnaround time**: Turnaround time is the time which is the difference between the time of submission or entered and the time of ending of the job. It is a metric for batch systems. Turnaround time is the sum of the following components:

- The time spends in waiting for entry into the system.
- Total time spent in the ready queue.
- Total time spent in the device queue.
- Time spent in the execution of the process.
- Time spend in doing the Input/Output operation.

However, turnaround time should be as minimum as possible.

(d) **Response time:** Response time is defined as the difference between the time of submission of the job for processing and the time when it gets the first response of the system. It is considered the best metric for interactive systems.

(e) **Waiting time:** Waiting time may be defined as the sum of intervals for which a process has to wait in the ready queue. It should be minimum for the better efficiency of the system. The scheduling strategies try to minimize the waiting time for the processes.

Average Turnaround Time: The turnaround time of a process is the total time elapsed from the time the process is submitted to the time the process is completed. It is calculated as

Turnaround time (TAT) = Process finish time (T_1) – Process Arrwal time (T_0)

Average Turnaround Time =

The lower the average turnaround time, the better it is.

Average Waiting Time: Waiting time of a process is defined as the total time spent by the process while waiting in a ready state or suspended state.

Waiting time (WT) = Turnaround time (TAT) – Actual Execution time (□t)

Average Waiting Time =

The lower the average waiting time, the better it is.

For the better performance and efficiency of a system the CPU utilization should be ***maximum***, Response time should be ***minimum***, Waiting time should be ***minimum***, Throughput should be ***maximum*** and the Turnaround time should be ***minimum***.

DISPATCHER

The dispatcher is a module of an OS that gives control of the CPU to the process selected by the scheduler. The dispatcher is responsible for the context switching. The time taken by the dispatcher for halting the running process and starting the process selected by the scheduler for executions is called **dispatch latency**.

SCHEDULING STRATEGIES

Scheduling strategies refer to the various algorithms used to choose a particular process from among the various processes in the ready queue.

There may be several processes waiting for the attention of the CPU in the ready queue. The operating system chooses a particular process from the ready queue and allocates the processor to it. So, the operating system requires a mechanism to decide, which process should be chosen next for execution.

There are two types of jobs:

- **CPU Bound Jobs:** The jobs, that spend more time performing CPU operations and less I/O operations are called CPU-bound jobs/processes.
- **I/O Bound Jobs:** The jobs that spend more time performing I/O operations and less time doing CPU operations are called I/O bound jobs/processes.

CPU Scheduling can be **preemptive** or **non-preemptive**.

In **preemptive scheduling**, the scheduler removes the running process from the CPU before its completion so that another process can run. In other words, in preemptive scheduling, the running process only gives up the control of the CPU voluntarily.

In **non-preemptive scheduling,** nothing can remove a process from utilizing CPU time until it completes or a time-out occurs.

There are several strategies used for *CPU Scheduling*, called *Scheduling Strategies* like:

1. FIRST COMES FIRST SERVE (FCFS)
2. SHORTEST JOB FIRST (SJF) (Preemptive and Non-preemptive)
3. PRIORITY SCHEDULING (Preemptive and Non-preemptive)
4. ROUND ROBIN SCHEDULING (RR SCHEDULING)

The detailed description of the scheduling strategies is as follows:

First Come First Serve (FCFS)

It is the simplest of all the scheduling algorithms. It is purely non-preemptive scheduling. The key concept of this algorithm is:

"allocate the CPU to the processes in the order in which they arrive".

According to this algorithm, the process which arrives first will get the CPU before any other process. Same way, the process which arrives as a second, will get the CPU after the first process and so on. It assumes the Ready Queue as the FIFO QUEUE. When a process completes its execution, the CPU is allocated to that process which is the first process in queue i.e the FRONT of the queue. If a new process arrived then it enters in REAR of the queue.

When a process starts running then it is removed from the queue. This algorithm is NON-PREEMPTIVE by default. This means once the CPU is assigned to a process, that process keeps the CPU till the end of its execution.

Advantages:

The code for FCFS scheduling is simple to write and understand.

It is suitable for Batch systems.

It is considered to be a fair policy, as the job which arrives first will get the CPU first.

Disadvantages:

If shorter jobs arrive after the longer jobs, then the waiting time will be large.

It is not suitable for Time-sharing systems.

Low CPU utilization, because all the other processes wait for one long job to get off the CPU**.**

This algorithm is never recommended whenever performance is a major issue.

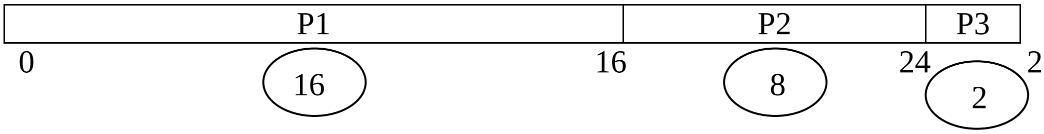
Example:

Consider the following snapshot of processes that arrive at a different time with the CPU-burst in milliseconds.

Note: Ignoring the arrival time in non-preemptive scheduling, the lowest number has the highest priority and time slice 2ms.

Process ID	**Arrival Time (T_0) ms**	**Priority**	**Next CPU Burst Time (Δt) ms**
P1	0	2	16
P2	1	3	8
P3	2	1	2

The Gantt chart of execution of the processes according to FCFS is the following:



Waiting time for P1 = 0 ms

Waiting time for P2 = 16 ms

Waiting time for P3 = 24 ms

Table depicting performance of FCFS

Process ID	**Arrival Time (T_0) ms**	**Next CPU Burst Time (Δt) ms**	**Finish Time (T_1) ms**	**Turnaround Time TAT=T_1-T_0 (ms)**	**Waiting Time= TAT-Δt (ms)**
P1	0	16	16	16	0
P2	0	8	24	24	16
P3	0	2	26	26	24
				66	40

As mentioned in question ignore the arrival time in non-preemptive scheduling so, here arrival time mentioned 0

Total waiting time = 0+16+24 = 40 milliseconds
Average waiting time = 40/3 = 13.33 milliseconds.
Total turnaround time = 16+24+26 = 66 milliseconds
Average turnaround time = 66/3 = 22 milliseconds.

However, if the jobs arrive in a different order, then the waiting time and turnaround time can be reduced. So, the waiting time and turnaround time depend upon the order of the jobs in which they arrive.

Shortest Job First (SJF)

The key concept of this algorithm is

“allocate the processor to the job which has the least CPU burst time”.

The ready queue has all the processes which require the processor for their execution. According to this algorithm, the processor is allocated to that job that has the smallest CPU burst time amongst all the processes in the ready queue.

If two processes have the same CPU burst, then the processor is allocated to the process which arrives first. This algorithm can either be PREEMPTIVE or NON-PREEMPTIVE.

Advantages

- This is considered to be an optimal algorithm as it helps to achieve the minimum waiting time.
- The shorter jobs have to wait for less time as compared to the longer jobs.

Disadvantages

- There is a need for the mechanism to know about the CPU burst of all the processes in advance.
- If the shorter jobs arrive again and again, the longer jobs may wait for a long period.

Example

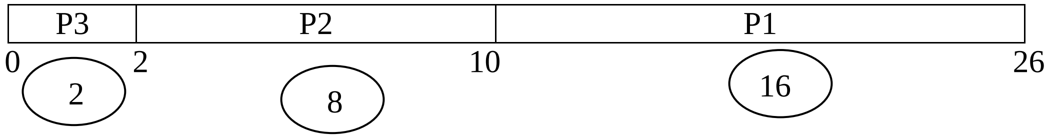
NON-PREEMPTIVE SJF

Consider the following snapshot of processes that arrive at a different time with the CPU-burst in milliseconds.

Note: Ignoring the arrival time in non-preemptive scheduling, the lowest number has the highest priority and time slice 2ms.

Process ID	Arrival Time (T_0) ms	Priority	Next CPU Burst Time (Δt) ms
P1	0	2	16
P2	1	3	8
P3	2	1	2

The Gantt chart of execution of the processes according to NON-PREEMPTIVE SJF is the following:



Waiting time for P1 = 10 ms
Waiting time for P2 = 2 ms
Waiting time for P3 = 0 ms

Table depicting performance of non-preemptive SJF

Process ID	Arrival Time (T_0) ms	Next CPU Burst Time (Δt) ms	Finish Time (T_1) ms	Turnaround Time TAT=T_1-T_0 (ms)	Waiting Time= TAT-Δt (ms)
P1	0	16	26	26	10
P2	0	8	10	10	2
P3	0	2	2	2	0
				38	12

Total waiting time	= 10+2+0	= 12 milliseconds
Average waiting time	= 12/3	=4 milliseconds.
Total turnaround time	= 26+10+2	= 38 milliseconds
Average turnaround time	=38/3	= 12.6 milliseconds.

PREEMPTIVE SJF: It is also termed as Shortest Remaining Time Next (SRTN) and Shortest Remaining Time First(SRTF).

Consider the set of the following processes with the CPU-burst in milliseconds.

Process ID	**Arrival Time (T_0) ms**	**Priority**	**Next CPU Burst Time (Δt) ms**
P1	0	2	16
P2	1	3	8
P3	2	1	2

Here, first of all, P1 arrives and the processor is allocated to it. After one second P2 arrives, since P2 has a smaller CPU burst as compare to P1, therefore the processor is preempted from P1 and is allocated to P2. Same way, after one second, P3 arrives whose CPU burst is less than the P2, so the processor is preempted from P2 and allocated to the P3. After the execution of P3, the processor is again allocated to P2 and so on.

The Gantt chart of execution of the processes according to PRE EMPTIVE SJF is the following:

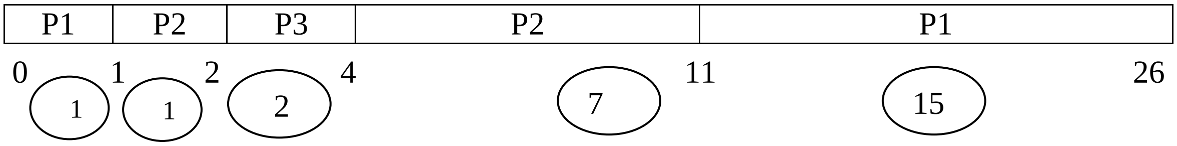


Table depicting performance of preemptive SJF

Process ID	**Arrival Time (T_0) ms**	**Next CPU Burst Time (Δt) ms**	**Finish Time (T_1) ms**	**Turnaround Time TAT=T_1-T_0 (ms)**	**Waiting Time= TAT-Δt (ms)**
P1	0	16	26	26	10
P2	1	8	11	10	2
P3	2	2	4	2	0
				38	12

Total waiting time	**= 10 + 2 + 0**	**= 12 milliseconds**
Average waiting time	=12/3	= 4 milliseconds.
Total turnaround time	= 26 + 10 + 2	= 38 milliseconds
Average turnaround time	=38/3	= 12.6 milliseconds.

In the case where it is not possible to know the CPU time for each process, this is estimated using predictors:

- $P_n = aO_{n-1} + (1-a)P_{n-1}$ where
  - O_{n-1} = previous service time
  - P_{n-1} = previous predictor
  - a is within [0,1] range
- If a = 1 then P_{n-1} is ignored
- P_n is dependent upon the *history* of the process evolution

Priority Scheduling

Priority scheduling is the scheduling mechanism in which each process in the system is assigned a priority. The processor is allocated to the processes according to their priority. The key concept of this algorithm is

"***Allocate the processor to the process which has higher priority"***.

The ready queue is assumed to be a priority queue in which each process is assigned a priority. First of all, the processor is allocated to the process having higher priority and then to the process having lower priority and so on. Priority scheduling can be of two types:

PREEMPTIVE or NON-PREEMPTIVE

Example:

NON-PREEMPTIVE PRIORITY SCHEDULING

Consider the set of the following processes with the CPU-burst in milliseconds.

Note: Ignoring the arrival time in non-preemptive scheduling, the lowest number has the highest priority and time slice 2ms.

Process ID	**Arrival Time (T_0) ms**	**Priority**	**Next CPU Burst Time (Δt) ms**
P1	0	2	16
P2	1	3	8
P3	2	1	2

The Gantt chart for the execution of the programs is as follows:

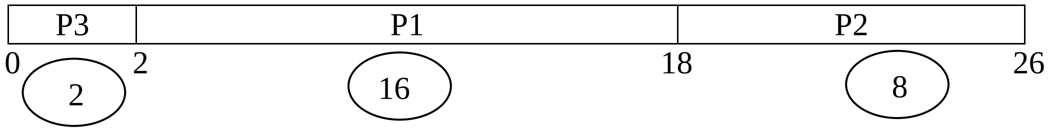


Table depicting performance of non-preemptive priority scheduling

Process ID	Arrival Time (T_0) ms	Next CPU Burst Time (Δt) ms	Finish Time (T_1) ms	Turnaround Time $TAT=T_1-T_0$ (ms)	Waiting Time= TAT-Δt (ms)
P1	0	16	18	18	2
P2	0	8	26	26	18
P3	0	2	2	2	0
				46	20

Waiting time for P1 = 2 ms
Waiting time for P2 = 18 ms
Waiting time for P3 = 0 ms

Total waiting time = 2 +18 + 0 = 20 milliseconds
Average waiting time =20/3 = 6.6 milliseconds.
Total turnaround time = 18 + 26 + 2 = 46 milliseconds
Average turnaround time =46/3 = 15.3 milliseconds.

PREEMPTIVE PRIORITY SCHEDULING

Consider the set of the following processes with the CPU-burst in milliseconds.

Process ID	Arrival Time (T_0) ms	Priority	Next CPU Burst Time (Δt) ms
P1	0	2	16
P2	1	3	8
P3	2	1	2

The Gantt chart for the execution of the programs is as follows:

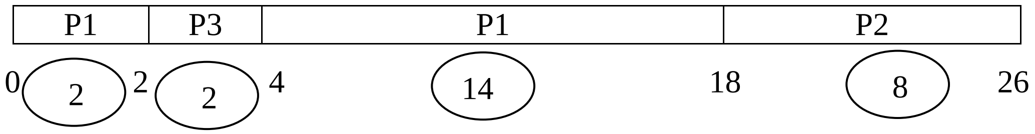


Table depicting performance of preemptive priority scheduling

Process ID	Arrival Time (T_0) ms	Next CPU Burst Time (Δt) ms	Finish Time (T_1) ms	Turnaround Time $TAT=T_1-T_0$ (ms)	Waiting Time= $TAT-\Delta t$ (ms)
P1	0	16	18	18	2
P2	1	8	26	25	17
P3	2	2	4	2	0
				46	19

Total waiting time = 2 + 17 + 0 = 19 milliseconds
Average waiting time =19/3 = 6.3 milliseconds.
Total turnaround time = 18 + 25 + 2 = 45 milliseconds
Average turnaround time =45/3 = 15 milliseconds.

The problem in Preemptive Priority Scheduling (Starvation or Blocking)

The main problem with priority scheduling is that low-priority processes can wait for a long time due to high-priority process arrivals. If an Operating system has not any precautions and just chooses the process with the highest priority, low priority processes wouldn’t get CPU, as long as high priority processes are runnable. This problem is known as **Starvation.**

Solution of Starvation (AGING)

Aging is a method of slowly increasing the priority of processes that are waiting in the system for a long time. The simplest solution is dynamic priorities. On one hand, the operating system can reduce the priority of a running process for each time quantum it used the CPU and on the other hand, it could increase the priority of other processes (which finally leads to the first situation because the boost should only be temporary) which didn’t get the CPU for a certain amount of time. Whether the operating system uses one or another solution, processes have a base priority that remains unchanged, and a real priority that is used for scheduling. Often the real priority is limited to a specific range so that important process still gets the processor when they need it.

Another way to avoid this problem is static priorities. The OS has to keep a record of how long a process has used the CPU. If it reaches a certain limit, the next highest priority process is allowed to run. This is not practically good.

Round Robin Algorithm (RR Algo)

The Round Robin scheduling algorithm is designed especially for the Time Sharing systems. It can be considered as FCFS scheduling along with the preemption. The processor is allocated to a process for a fixed amount of time called, **TIME SLOT** or **TIME QUANTUM**, or, **TIME SLICE**. It is a purely preemptive algorithm. Likewise FCFS, the processor is allocated to the processes in the order in which they arrive, but the processor is preempted from the process after the time slice is over, and the processor is allocated to the next process in the ready queue. A time quantum generally varies from 10 milliseconds to 100 milliseconds. The ready queue is assumed to be a circular queue. The short-term scheduler goes on allocating the processor to the processes in the ready queue for a fixed amount of time.

However, if a process has its CPU burst less than the time quantum, then the process releases the CPU voluntarily (itself). Otherwise, the processor is allocated to the process in the ready queue for a fixed amount of time (time quantum) and after that CONTEXT SWITCHING take place, and the processor is allocated to the next process in the ready queue, keeping the former process at the end of the ready queue. The RR algorithm is PREEMPTIVE by default.

The performance of the Round Robin algorithm depends on the value of time quantum or slice

- If the value of time quantum is large, then this algorithm becomes the same as FCFS.
- If the value of the time quantum is small, then the number of the context switching will be increased considerably, this is not desirable at all. It affects the system throughput adversely.
- Thus, the size of the time quantum should neither be very large nor too small for better efficiency.

Example:

Consider the set of the following processes with the CPU-burst in milliseconds.
Note: Ignoring the arrival time in non-preemptive scheduling, the lowest number has the highest priority and time slice 2ms.

Process ID	**Arrival Time (T_0) ms**	**Priority**	**Next CPU Burst Time (Δt) ms**
P1	0	2	16
P2	1	3	8
P3	2	1	2

The Gantt chart of execution of the processes according to the RR scheduling algorithm is the following. Let the time quantum be 2 milliseconds-:

P1	P2	P3	P1	P2	P1	P2	P1	P2	P1

0 2 4 6 8 10 12 14 16 18
26 2 2 2 2 2 2 2 2 2 8

Table depicting performance of Round Robin scheduling

Process ID	Arrival Time (T_0) ms	Next CPU Burst Time (Δt) ms	Finish Time (T_1) ms	Turnaround Time $TAT=T_1-T_0$ (ms)	Waiting Time= TAT-Δt (ms)
P1	0	16	26	26	10
P2	1	8	18	17	9
P3	2	2	6	4	2
				47	21

Waiting time for P1 = 10 ms
Waiting time for P2 = 9 ms
Waiting time for P3 = 2 ms

Total waiting time	= 10 + 9 + 2	= 21 milliseconds
Average waiting time	= 21/3	= 7 milliseconds.
Total turnaround time	= 26 + 17 + 4	= 47 milliseconds
Average turnaround time	= =47/3	= 15.6 milliseconds.

Comparison between FCFS and Round Robin Scheduling Algorithm

FCFS	Round Robin
FCFS is purely non-preemptive Scheduling	It is purely preemptive Scheduling
It has minimum overhead	It has a higher overhead as compare to FCFS as context switching occurs more.
Response time depends upon the size of the process	It offers a better response time
It is not designed for a time-sharing system	It is designed for a time-sharing system
The workload is simply processed in the order process arrived	It is similar to FCFS but it uses a time slice which means one process maximum uses CPU for time slice at once.

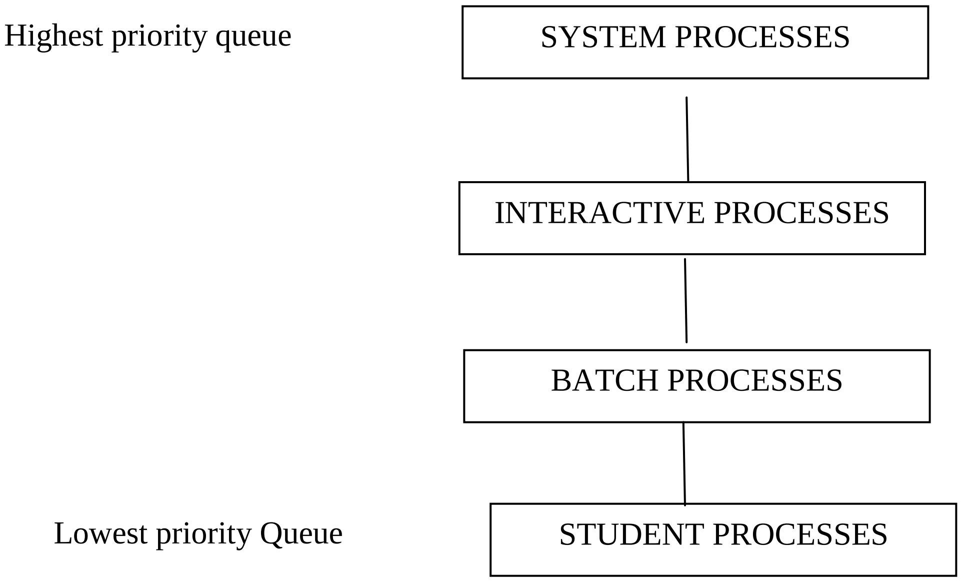
Multilevel Queue Scheduling

A multilevel queue scheduling algorithm is used when the processes are to be divided into different groups. Sometimes such a situation arises where two or more processes have different response time requirements, so there is a need for a different kind of schedule for both processes.

A multilevel queue scheduling division of the ready queue into numerous separate queues. The processes are permanently allocated to a particular queue, based on the properties of the process.

Sometimes there are two or more processes, among which some processes have higher priority and other having low priority. So, there is a need for a separate queue for the processes with higher priority and the processes with lower priority.

Consider an example of multilevel queue scheduling for different kinds of processes in the ready queue. There are four different queues for different processes which are made based on their priority in the system.



Each queue may have its scheduling criteria, based upon the nature and properties of the jobs.

Advantages

- Low scheduling overhead can be achieved by using a multilevel scheduling algorithm.

Disadvantages

- It is not considered as efficient in some of the cases, like if higher priority queues don't become empty for a long time, then the lower priority jobs may starve.

Multilevel Feedback Queue Scheduling

It is an enhancement of multilevel queue scheduling.

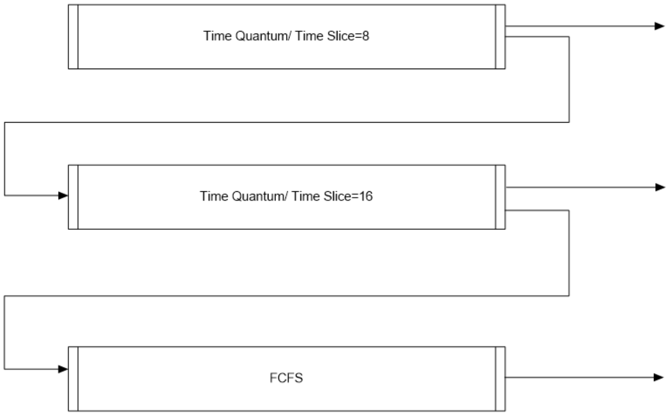
Actually, in multilevel queue scheduling, the processes are assigned a particular queue on entering the system. The processes do not move from a queue to any other queue. So, it is an inflexible approach.

Multilevel feedback queue scheduling permits a process to move between the queues. For example, if a process CPU-bound process which means it uses too much CPU then the process is moved (shifted) to a lower priority queue. Correspondingly, a lower priority job can also be shifted to a higher priority queue if it is waiting for a long time, etc.

Advantages

- It allows a process to move to any other queue.
- It is more flexible as compared to any other scheduling algorithm.

- A lower priority job, which is waiting for a long time, can be shifted to a higher priority queue.
- A higher priority job, affecting negatively the efficiency, can be moved to a lower priority queue.



Multilevel Queue Scheduling

Disadvantages

- However, it is considered to be a complex scheduling algorithm.
- While moving the processes from a queue to any other queue increases the CPU overhead.

Illustration

With reference to the following set of processes

Process ID	Arrival Time (T_0) ms	Next CPU Burst Time (Δt) ms	Priority
P1	0	29	1
P2	6	14	5
P3	8	10	3
P4	10	8	1
P5	13	6	2

Determine Average Waiting Time and Turn Around Time, for the following scheduling.

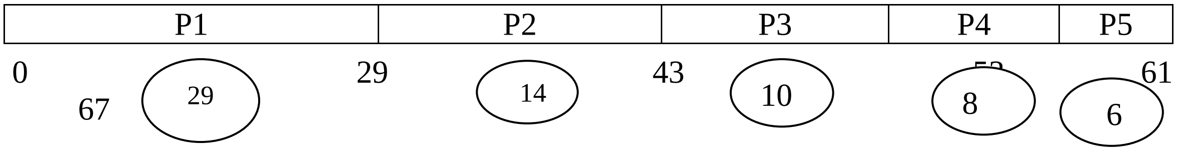
(a) First Come First Serve (FCFS)
(b) Shortest Job First (SJF) (Non-preemptive)
(c) Shortest Job First (Preemptive) or Shortest Remaining Time Next (SRTN)
(d) Priority Scheduling (Preemptive)
(e) Priority Scheduling (Non-preemptive)

(f) Round Robin (RR)

Note: Time Slice – 5 ms, the highest number has the highest priority and ignores the arrival time in non-preemptive scheduling.

(a) **First Come First Serve (FCFS)**

In this scheduling, jobs or processes are scheduled to run in the same order as those that have arrived in the system. It's a pure non-preemptive algorithm. Process P1. will execute first because its ID is 1, then P2, P3, P4, and P5. So its Gantt Chart is given below:



Calculating Waiting Time and Turnaround Time

In this problem, in a non-preemptive algorithm arrival time is ignored so, waiting time for all processes is when they start to run and turnaround time is when they complete the process.

Note Arrival time 0 in this problem because in non-preemptive algorithm arrival time is ignored.

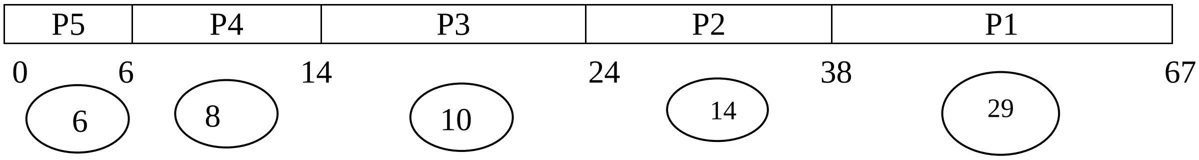
Table depicting performance of FCFS

Process ID	**Arrival Time (T_0) ms**	**Next CPU Burst Time (Δt) ms**	**Finish Time (T_1) ms**	**Turnaround Time** **TAT=T_1-T_0 (ms)**	**Waiting Time= TAT-Δt (ms)**
P1	0	29	29	29	0
P2	0	14	14	43	29
P3	0	10	10	53	43
P4	0	8	8	61	53
P5	0	6	6	67	61
				253	186

Average Waiting Time = (0+29+43+53+61)/5=37.2 ms
Average Turnaround Time = (29+43+53+61+67)/5=50.6ms

(b) **Shortest Job First (SJF)**

In shortest job first, job to be dispatch will be the one, which happens to be the shortest amongst the pending jobs. This is non-preemptive. So a job, once scheduled, is permitted to complete its next burst. In this problem, Arrival time is ignored for non-preemptive scheduling so, P5 has the shortest next burst time hence it will run first then P4, P3, P2 & P1 will run. So its Gantt chart is



Calculation of Waiting Time & Turnaround Time

In this problem, in a non-preemptive algorithm arrival time is ignored so, waiting time for all processes is when they start to run and turnaround time is when they complete the process.

Table depicting performance of non-preemptive SJF scheduling

Process ID	**Arrival Time (T_0) ms**	**Next CPU Burst Time (Δt) ms**	**Finish Time (T_1) ms**	**Turnaround Time TAT=T_1-T_0 (ms)**	**Waiting Time= TAT-Δt (ms)**
P1	0	29	67	67	38
P2	0	14	38	38	24
P3	0	10	24	24	14
P4	0	8	14	14	6
P5	0	6	6	6	0
				149	82

Note: Arrival time is 0 in this problem because non-preemptive algorithm arrival time is ignored.

Average Waiting Time = (38+24+14+6+0)/5=16.4 ms
Average Turnaround Time = (67+38+24+14+6)/5=29.8 ms

(c) **Shortest Remaining Time Next (SRTN)**

This is a preemptive algorithm, where the next job/process to be dispatched will be one that happens to be shortest amongst the pending jobs, at the time of making the decision. However, if a process/job arrives later, whose next burst time to be less than the remaining burst time of the currently running process, the currently running process will be preempted by the new process. The preempted process will be later re-dispatched when its remaining burst happens to be shortest amongst the pending processes.

In this problem, at 0 arrival time, only one process is there i.e. P1. So, P1 will execute first. After 6ms, a new process P2 arrives which has a 14ms burst time and P1 has 23ms (29-6) remaining burst time. So, P1 will be preempted by the P2 because P2 has the shortest burst time at 6ms. After 2ms i.e. 8ms a new process P3 arrives which has 10ms burst time and P2 has 12ms (14-2) remains burst time P3 has the shortest burst time at 8ms. So, P2 will be preemptive by P3. After 10ms a new process P4 arrives which has an 8ms burst time and P3 has 8ms (10-2) remains burst time. So there is a tie between P3 and P4. To break tie FCFS algorithm is adopted so, P3 will continue to execute. At time 13ms a new process, P5 arrived with 6ms burst time. At that time P3 has (10-(13-8)) i.e. 5ms remaining burst time which shortest than among all-time so P3 will continue. After completion of P3 at 18ms time. There is no new process arrived. Now remains burst time for a process are P1 is 23ms, P2 is 12ms, P4 is 8ms and P5 is 6ms. So amongst P5 is shortest than P4, P2 and P1 will execute. Gantt chart of this is given below:

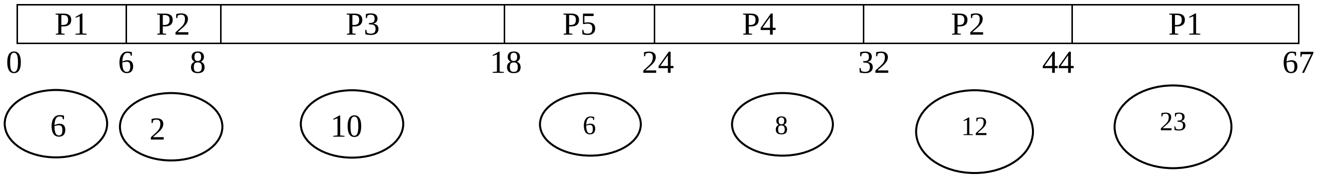


Table Depicting Performance of SRTN Algorithm

Process ID	Arrival Time (T_0) ms	Next CPU Burst Time (Δt) ms	Finish Time (T_1) ms	Turnaround Time TAT=T_1-T_0 (ms)	Waiting Time= TAT-Δt (ms)
P1	0	29	67	67	38
P2	6	14	44	38	24
P3	8	10	18	10	0
P4	10	8	32	22	14
P5	13	6	24	11	5
				148	81

Average Waiting Time =(38+24+0+14+5)/5=16.2 ms
Average Turnaround Time = (67+38+10+22+11)/5= 29.6 ms

(d) **Priority Scheduling (Preemptive)**

At the time of schedules, a dispatcher dispatched that process which has the highest priority amongst the processes which are waiting in the ready queue. When a process Pi is executing, if another process Pj arrived and has higher priority, then Pi will be preempted by Pj.

In this problem, at time 0ms only one process P1 is there so it is executed till a new process has arrived i.e. 6ms. After 6ms a new process P2 comes which has priority 5 which is greater than P1 priority i.e. 2 so, P1 will be preempted by P2. Now P2 will execute at 8ms a new process P3 arrive which have priority 3 but P2 have the highest priority at that time so, P2 will continue, after 10ms P4 arrive which have priority value 1 which lowest so, P2 will continue. After that at 13ms P5 comes whose priority is lower than P2. So P2 will continue till 20ms i.e. (6 + 14).

Now there are 4 processes at 20ms P1, P3, P4, P5 are ready and priorities are 1, 3, 1, 2 respectively. Among them, 3 is the highest priority. So, P3 will execute then P5. After that P1, P4 have the same priority so, there is a tie. To break the tie FCFS algorithm was adopted so P1 executes than P4. Gantt chart of this algorithm is:

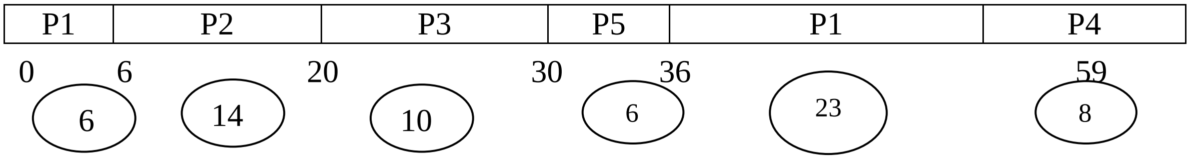


Table depicting performance of a priority-based preemptive algorithm.

Process ID	Arrival Time (T_0) ms	Next CPU Burst Time (Δt) ms	Finish Time (T_1) ms	Turnaround Time TAT=T_1-T_0 (ms)	Waiting Time= TAT-Δt (ms)
P1	0	29	59	59	30
P2	6	14	20	14	0
P3	8	10	30	22	12
P4	10	8	67	57	49
P5	13	6	36	23	17
				175	108

Average Waiting Time = (30+0+12+49+17)/5=21.6 ms
Average Turnaround Time = (59+14+22+57+23)/5=35 ms

(e) **Priority Scheduling (Non-preemptive)**

At the time of scheduling, a process that has the highest priority amongst all the processes which are waiting in the ready queue. Once dispatched, a process Pi is allowed to complete its burst time, even if Pj another process of having higher priority becomes ready while running Pi.

In this scheduling, arrival time is ignored so amongst all processes P2 has the highest priority which will execute first. Priority of above processes is:

P2 > P3 > P5 > P1 > P4

Gantt chart is:

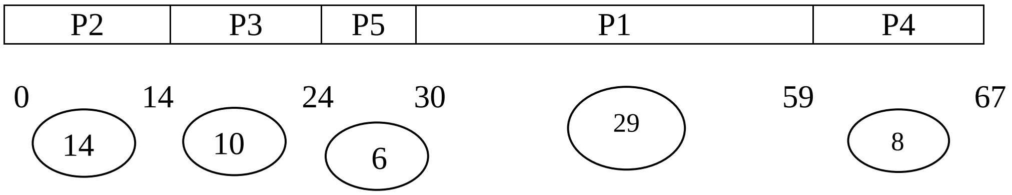


Table depicting performance of a priority-based non-preemptive algorithm.

Process ID	Arrival Time (T_0) ms	Next CPU Burst Time (Δt) ms	Finish Time (T_1) ms	Turnaround Time TAT=T_1-T_0 (ms)	Waiting Time= TAT-Δt (ms)
P1	0	29	59	59	30
P2	0	14	14	14	0
P3	0	10	24	24	14
P4	0	8	67	67	59
P5	0	6	30	30	24
				194	127

Note: Arrival time is 0 in this problem because non-preemptive algorithm arrival time is ignored.

Average Waiting Time = (30+0+14+59+24)/5=25.4 ms

Average Turnaround Time = (59+14+24+67+30)/5=38.8 ms

(f) **Round Robin (RR)**

It is purely a preemptive scheduling algorithm. A small unit of time is called a time slice or time quantum. It is the maximum time for which a process can execute at a time. New processes are added at the end of the ready queue and the head of the ready queue process Pi is dispatched for the time slice. If the process Pi has not finished its execution, then it is linked to the tail of the ready queue, and the next process in the ready queue is dispatched for the next time slice and so on.

In this problem, the time slice is 5ms so P1 will execute for 5ms but there is no more process at 5ms so, P1 again executes for the next 5ms. After 10ms P2 execute for 5ms and P3, P4, P5 so on. This process will continue till the process complete their execution.

Gantt Chart is:

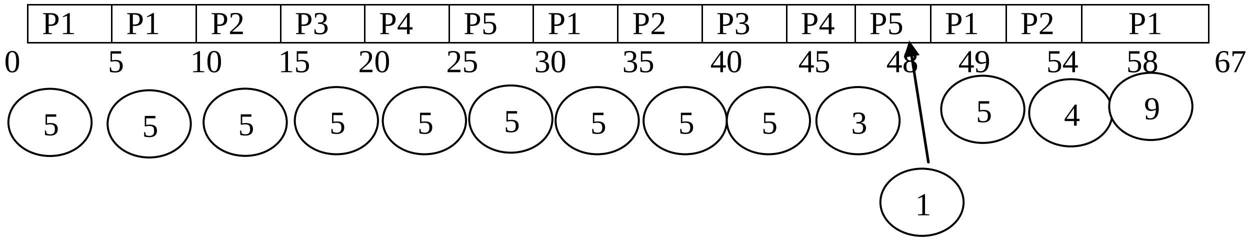


Table depicting performance of round robin algorithm

Process ID	**Arrival Time (T_0) ms**	**Next CPU Burst Time (Δt) ms**	**Finish Time (T_1) ms**	**Turnaround Time TAT=T_1-T_0 (ms)**	**Waiting Time= TAT-Δt (ms)**
P1	0	29	67	67	38
P2	6	14	58	52	38
P3	8	10	45	37	27
P4	10	8	48	38	30
P5	13	6	49	36	30
				230	163

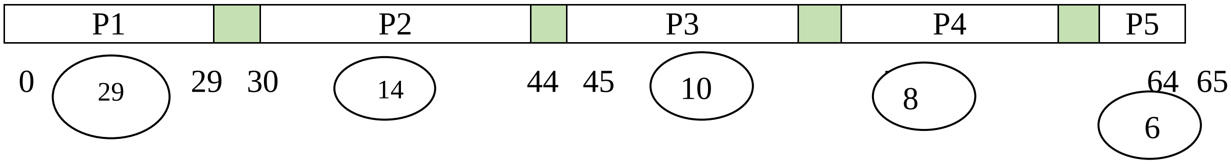
Average Waiting Time = (38+38+27+30+30)/5 =32.6 ms
Average Turnaround Time = (67+52+37+38+36)/5=46 ms

CPU Scheduling Algorithms analysis

Scheduling Algorithm	Turnaround Time	Waiting Time	Average Turnaround Time	Average Waiting Time
FCFS	253	186	50.6	37.2
SJF	149	82	29.8	16.4
SRTN	148	81	29.6	16.2
Priority(Preemptive)	175	108	35	21.6
Priority(non-Preemptive)	194	127	38.8	25.4
Round Robin	230	163	46	32.6

The above table depicts that SRTN scheduling gives better performance among the other scheduling.

Note: Some time context switching time is given in problem then that time is switching time and it is pure overhead. It will be increased the waiting time and turnaround time. If the context switch time is 1ms then in the above example the Gantt chart of FCFS as below:



Performance of all CPU Scheduling Algorithms

Scheduling Algorithm	CPU Utilization	Response Time	Average Waiting Time	Average Turnaround Time	Throughput
FCFS	Low	Low	High	High	Low
SJF	Medium	Medium	Medium	Medium	High
Priority	Medium	High	High	High	Low
Round Robin	High	High	Medium	Medium	Medium
Multi-Level	High	Medium	Medium	Medium	High

Points to Remember

- A program is a static entity.
- A process is an instance of a program under execution.
- The process is a dynamic entity.
- Process state is defined as the current activity of the process.
- In two states process model process has two states: Running or not running.
- In five states process model process has five states: New, Ready, Waiting Running, and Terminating.
- Each process has its Process Control Board (PCB).
- In-Process Control Board (PCB) attributes and information about the process is stored which is needed by OS to control the process.
- Schedulers are of three types: Long term Scheduler, Medium Term Scheduler, Short Term Scheduler.
- Long term scheduler is also known as a job scheduler. It selects the processes and loads them into memory. It changes process states from new to ready state.
- A short-term scheduler is also known as a CPU scheduler or dispatcher. It changes process states from ready to running state.
- The process is called the heavyweight process.
- A process can have a single thread as well as multiple threads.
- Thread is called a lightweight process.
- Threads are not independent of one another.
- Threads can be created at a user level and the kernel level.
- A context switch is switching the CPU to another process which requires saving the state of the old process and loaded the saved state for the new process.
- A Daemon is a system process that is created at boot time and keeps executing in the background.
- CPU scheduling is used to increase CPU utilization.
- In non-preemptive scheduling, once the process has been assigned to the CPU, the CPU cannot be taken away from that process until it terminates or is blocked.
- In preemptive scheduling, the CPU can be taken away from the process during execution.
- Context switching is required in preemptive scheduling.

- Throughput means how many processes the system can execute in a unit of time. Higher the number, the better it is.
- Response Time is the time from the entry of request until the first response is produced.
- Finish time means when the process finishes its execution (T1).
- Arrival time is the time when the process arrived (T0).
- Burst Time is the estimated time the process needed CPU for execution (**Δt**).
- Turnaround time is computed by subtracting the time the process entered the system from the time it terminated (TAT= T1-T0).
- Waiting Time is time spent by the process for CPU (WT=TAT- **Δt**).
- CPU-bound process spends most of its time in CPU.
- I/O bound process spends most of its time in I/O operations.
- First Come First Serve (FCFS) is a purely non-preemptive algorithm.
- In FCFS, the CPU is allocated to the process in order of arrival.
- Shortest Job First (SJF) scheduling algorithm can be either a non-preemptive or preemptive algorithm.
- In SJF, the CPU is allocated to the process which has the smallest burst time.
- Preemptive SJF is known as Shortest Remaining Time Next (SRTN).
- In Shortest Job First (SJF) scheduling (non-preemptive and preemptive) the next CPU burst time must be known in advance.
- A priority scheduling algorithm can be either a non-preemptive or preemptive algorithm.
- In priority scheduling, the CPU is allocated to the process which has the highest priority.
- In preemptive priority scheduling, there is a big problem known as **starvation.**
- In starvation, blocking of low priority processes due to high priority jobs keep arriving one after another.
- **Aging** is a solution to starvation, after some time makes the priority of a process go up the longer it stays run-able but isn't run.
- Round Robin is a purely preemptive algorithm.
- CPU is allocated to all processes in a queue for small-time, which is known as Time Slice, Time Quantum.
- Multilevel queue algorithms allow different algorithms to be used for various classes of processes.
- Multilevel feedback queues allow processes to move from one queue to another.

EXERCISES

1. What is a Process? Explain different states of a process.
2. Explain the Process State diagram in detail.
3. What do you mean by Process Creation and Termination?
4. What is Process Control Block?
5. Explain the following terms:
   (i) Throughput
   (ii) Waiting Time

(iii) Turn Around Time
(iv) Response Time
(v) CPU utilization

6. What are the various factors for measuring the performance of an operating system?
7. Explain the Operations on Processes.
8. What do you mean by Thread and explain different types of threads?
9. Difference between Thread and Process.
10. Difference between User Level Thread and Kernel Level Thread.
11. Explain how threads can improve system performance.
12. Discuss the following terms:

□ Process Spawning
□ Parent Process
□ Child Process
□ Halt
□ Dispatcher
□ Ready Queue
□ Blocked Queue
□ Daemon

13. What do you mean by CPU scheduling and why it is required?
14. What do you mean by I/O bound and CPU bound process?
15. Define the differences between preemptive and non-preemptive scheduling?
16. What are the scheduling criteria? Explain it.
17. Explain the different types of schedulers.
18. Consider the following set of processes

Process	**Arrival Time**	**Priority**	**Burst Time**
P0	0	4	20
P1	1	2	8
P2	3	1	10
P3	5	3	4
P4	8	1	2
P5	9	5	7

Calculate Average Turnaround Time, Average waiting time for the following algorithms. The time slice is 3 ms and the lowest number has the highest priority for the following scheduling.

(a) FCFS (b) SJF (Preemptive and non-preemptive)
(c) Priority (Preemptive and non-preemptive) (d) Round Robin

19. What is Multilevel queue scheduling. Why we use it?
20. Explain between long-term and short-term scheduler.
21. Explain with example the following:

(a) FCFS (b) SJF

22. Explain with example the following:

(a) Priority Scheduling (b) Round Robin Scheduling

23. Difference between FCFS and RR scheduling.

24. Consider the following set of processes

Process	Arrival Time	Priority	Burst Time
P1	0	3	5
P2	1	1	2
P3	1	2	8
P4	3	1	5
P5	7	4	7

Calculate Average Turnaround Time, Average waiting time for the following algorithms. The time slice is 2 ms and the highest number has the highest priority (ignore the arrival time in non-preemptive scheduling) for the following scheduling.

(a) FCFS (b) SJF (Preemptive and non-preemptive)

(c) Priority (Preemptive and non-preemptive) (d) Round Robin

25. Consider the following set of processes

Process	Arrival Time	Priority	Burst Time
P1	0	1	20
P2	0	5	12
P3	3	3	3
P4	4	4	18
P5	8	2	8

Calculate Average Turnaround Time, Average waiting time for the following algorithms. The time slice is 5 ms and the lowest number has the highest priority (1 ms is context switching time and ignore the arrival time in non-preemptive scheduling) for the following scheduling.

(a) FCFS (b) SJF (Preemptive and non-preemptive)

(c) Priority (Preemptive and non-preemptive) (d) Round Robin

B.Sc.(DATA SCIENCE)

SEMESTER-II

OPERATING SYSTEM

UNIT III:DEADLOCK

STRUCTURE

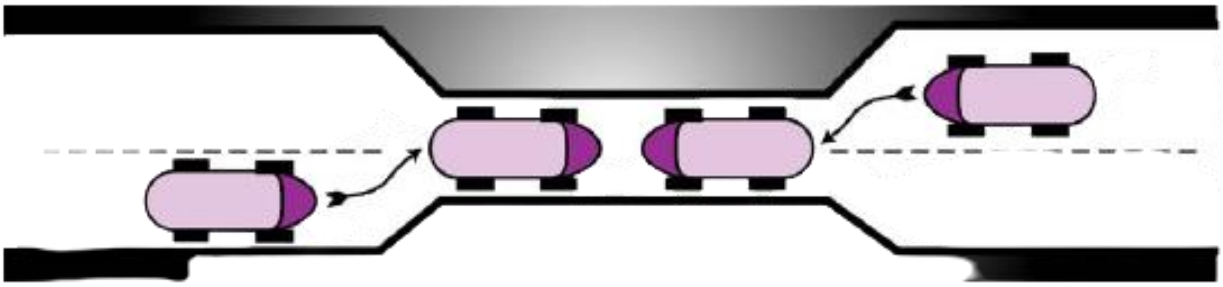
3.0 OBJECTIVES

- To understand the Deadlock Problem
- System Model
- Deadlock Characterization
- Methods for Handling Deadlocks
- Deadlock Prevention
- Deadlock Avoidance
- Deadlock Detection
- Recovery from Deadlock

3.1 INTRODUCTION TO DEADLOCK

There is a situation in which two processes wait for each other. The resources of a system may be limited or less as compared to the number of processes. In multiprogramming, numerous processes may require a predetermined number of resources. When a process requests for a resource, the Operating System checks whether the resource is available or not. If the resource is available, then it is allocated to the process. On the other hand, if the resource is not free (available), the process enters the waiting state. Sometimes it happens that the process remains to wait, for a long time because the requested resource is holding by other waiting processes. This situation leads to **DEADLOCK**. It is a condition, wherein a set of processes are waiting forever for the resources, held by each other. None of them can proceed with its execution.

In routine life, a traffic jam on the narrow bridge is an example of Deadlock, when cars came from both side, but only a single car crossed from the bridge at a time and both car drivers refuge to back the cars as shown in the figure below:

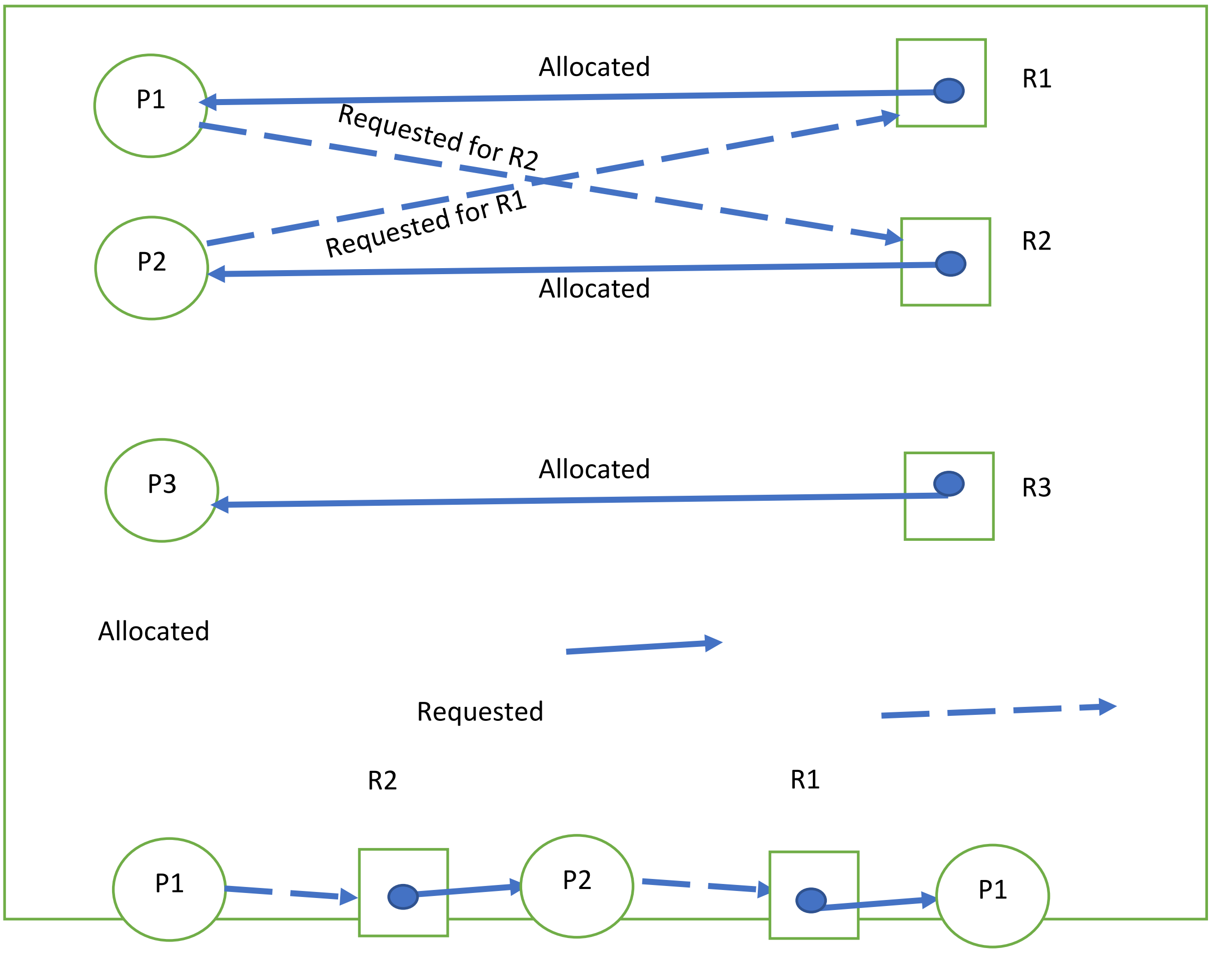


Example: Consider a scenario where there are three processes P1, P2, P3, and three resources R1, R2, R3. P1 requires two resources R1 and R2 for its execution. Also, P2 requires two resources R1 & R2 for its execution .P3 requires only R3. Initially, R1 is allocated to P1, and R2 is allocated to P2.

Here, (i) R1 is allocated to P1 and P1 is requesting R2.

(ii) R2 is allocated to P2 and P2 is requesting for R1.

(iii) R3 is allocated to P3.



3.2 SYSTEM MODEL

In the system, there are n number of processes namely p_1,p_2,p_3, …… $p_{n-1,}$ $p_{n,}$ and m number of resource types namely R_1, R_2,…..R_{m-1}, $R_{m.}$ In the System model, R resources are to be circulated among some processes P. The resources are then divided into many types, each consisting of some definite quantity of identical instances which is represented as w. The main example of resource types is *CPU cycles, memory space, Input-Output devices* such as keyboards, printers, and CD-DVD drives, directories, and files. Each resource type has instances. Instances mean the number of resource types that represent as w. Each resource type R_i has W_i instances When a system has 4 printers, then the resource type printer got four instances. Each process uses a resource as the following sequence:

- **Request:** Process request for the resources. If resources are available then the system allocates to process otherwise process waits for resources.
- **Use**: Then process used the resources when the system allocates them.
- **Release:** Process used the resources and then released them.

3.3 DEADLOCK CHARACTERIZATION

3.3.1 Necessary Conditions For Deadlock

The following four conditions must hold simultaneously, for a deadlock to occur:

1. **Mutual exclusion**

Mutual exclusion implies that a resource can be utilized exclusively by only one process at a time in a non-shareable mode. If any other process wants to use that resource, then it must have to wait, until it is released by the former process. For example, P1 holds R1 and P2 holds R2, both in mutually-exclusive mode.

2. **Hold and wait**

It implies that there exists a process that is holding at least one resource which is waiting for another resource or resources, which is being held by other processes. It implies that there exists a process that must be holding some resources (at least one) in a non-sharable mode and at the same time must be waiting for the other resources, which are held by other processes in a non-sharable mode. Like in the above example: P1 is holding R1 in a non-shareable mode and at the same time trying to acquire R2, which is currently held by P2 in a non-sharable mode. Similarly, P2 is holding R2 in a non-shareable mode and at the same time trying to acquire R1, which is currently held by P1 in a non-sharable mode. This situation is called hold and wait for conditions.

3. **No pre-emption**

It implies that the resources cannot be pre-empted forcefully from a process. After the completion of the process, a resource is released voluntarily by the process.

4. **Circular wait**

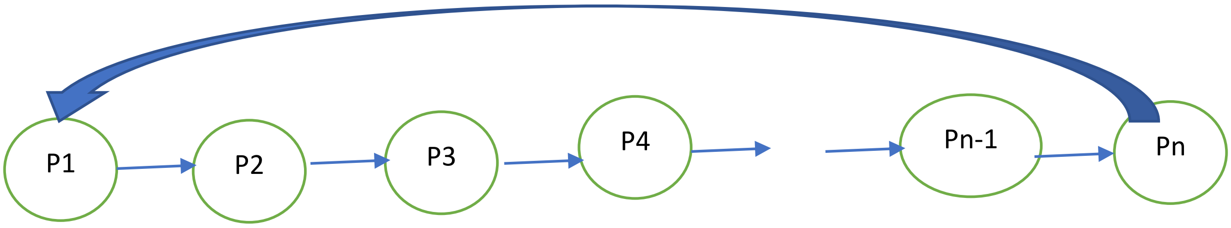
Circular wait implies that the processes are waiting for resources that are circularly held by another process. In other words, Pi is waiting for a resource held by Pi+1 and so on Pn-1 is waiting for a resource which is held by Pn, and at last, Pn is waiting for a resource held by Pi. For example, Let a set of processes, say (P1, P2, P3, ---, Pn). They must wait in a circular way for the resources held by each other i.e.

P1 is waiting for a resource which is held by P2

P2 is waiting for a resource which is held by P3

Pn-1 is waiting for a resource which is held by Pn

Pn is waiting for a resource which is held by P1



3.3.2 Deadlock Detection

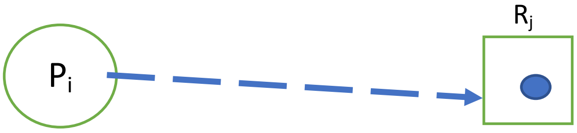
When all the four essential condition holds at the same time then deadlock occurs. Another way to detect deadlock is Resource Allocation Graph (RAG)

Resource Allocation Graph

The Resource Allocation Graph is a directed graph which is a set of vertices which is denoted by V and a set of edges which is denoted by E. Vertices (V) is divided into two types. Processes (P) and Resources (R). P = {P_1, P_2, - - -, $P_{n-1,}$ P_n} the set consisting of all the processes in the system. A process is represented by a circle, with the process name indicated with a label inside the circle. R = {R_1, R_2, - - - -, R_{n-1}, R_n} the set containing of all resources types in the system. A resource is represented by a square, with dots inside the square representing different instances of that resource.

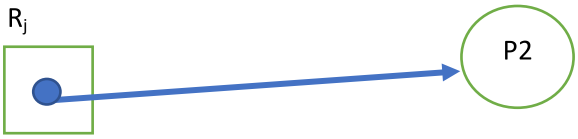
There are two types of edges: Request Edge & Assignment edge.

An edge from process P_1 to Resource R_j represents a request edge.



Request Edge

An edge from resource Rj to process Pi represents an assignment edge.



Assignment Edge

Note: The assignment edges originate from the instance within the resource symbol, which indicates which instance is assigned to the process.

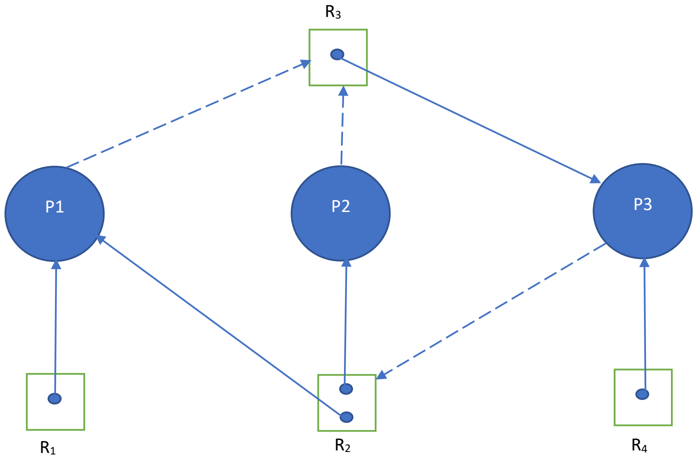
If there is no cycle in RAG, it indicates that there is no Deadlock occurred.

If there is a cycle detected in the graph and the resources involved in the cycle have only one instance per resource, then it indicates that a deadlock exists.

If there is a cycle detected in the graph and some of the resources involved in the cycle have multiple instances per resource, there may be a deadlock exists.

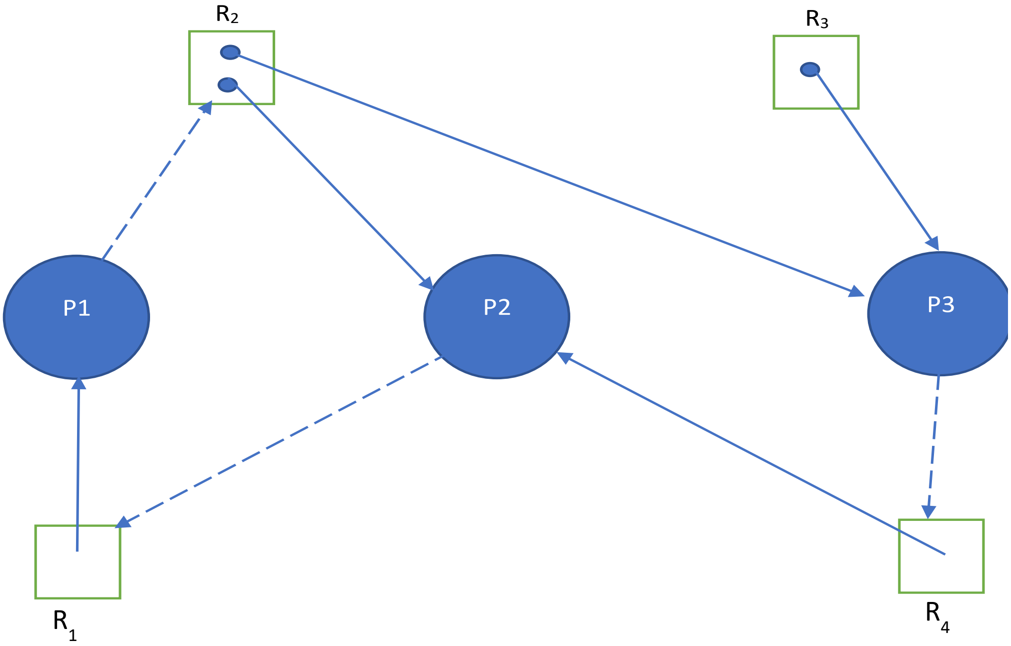
For example

Let there are 3 processes P1, P2, P3, and 4 resources R1, R2, R3, R4. R1 has one instance, R2 has 2 instances and R3 has one instance, R4 has one instance. R1, R2 is allocated to P1 and requested for R3. R2 is allocated to P2 and requested for R3. R3, R4 are allocated to P3 and requested for R2.



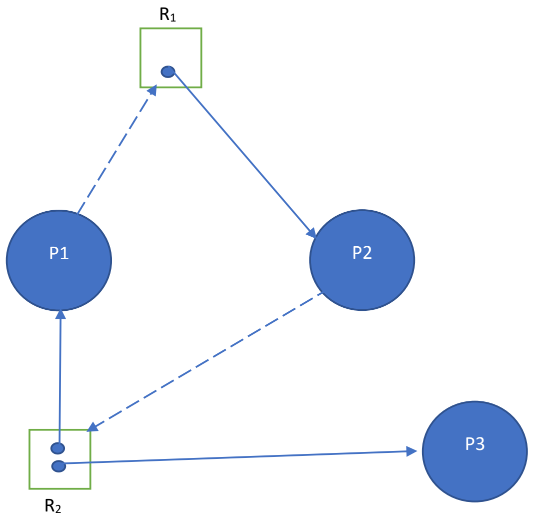
Resource Allocation Graph

Example of RAG with a cycle Deadlock



Resource Allocation Graph with Deadlock

Example of RAG with a cycle but no deadlock

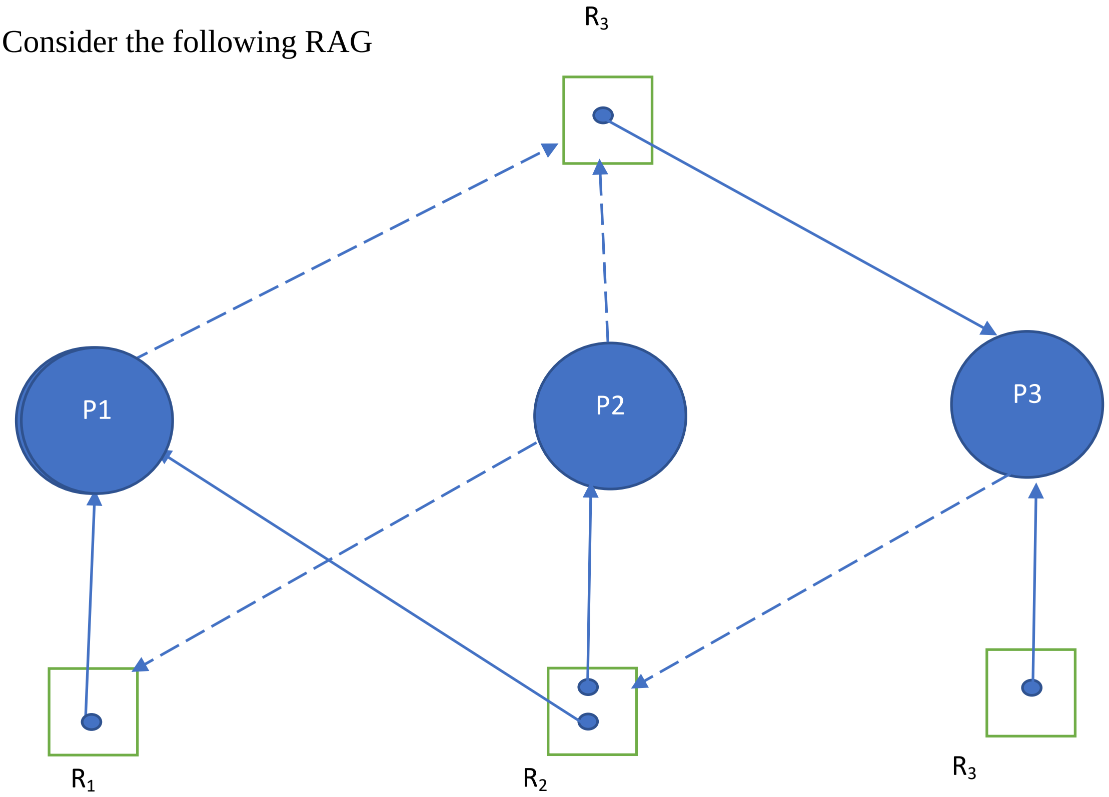


Resource Allocation Graph with Cycle but no Deadlock

Process Wait for Graph (PWFG)

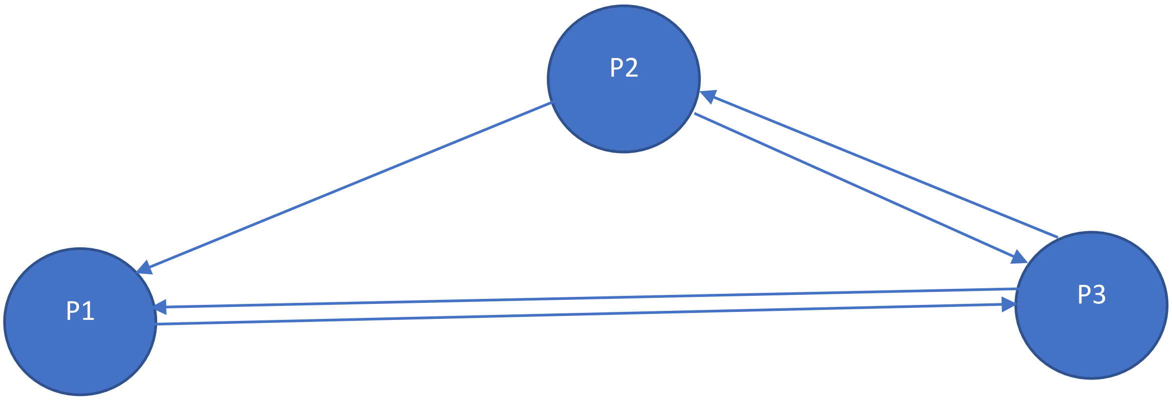
PWFG can be obtained by collapsing the resources symbol in RAG. An edge from process P1 to P2 indicates that P1 waiting for a resource that is currently held by P2.

Consider the following RAG



Example of Resource Allocation Graph

Equivalent PWFG for above RAG



Process Wait for Graph (PWFG)

The PWFG contains a cycle, thus indicated that deadlock exists.

3.3.3 Deadlocks Management

Different OS adopts different strategies to handle deadlocks. The policies to handle deadlocks can be classified into four major groups:

- Deadlock Prevention
- Deadlock Avoidance
- Deadlock Detection and Recovery
- Deadlock Ignorance

The deadlock prevention techniques prevent the occurrence of at least one of the four conditions that cause deadlocks. By preventing are of the necessary four conditions deadlock can be prevented. The deadlock avoidance techniques acquire information in advance about which resource a process will claim at what stage of execution. The OS allocates resources in such a manner that no deadlock occurs. Deadlocks can be avoided by maintaining the system always in a safe state. The system is said to be in a safe state if all pending processes can be successfully executed in some sequence. That sequence is called a safe sequence and processes are in a safe state. The third category of deadlock management strategies is known as detection and recovery. These types of techniques allow deadlocks to occur detect deadlock when it occurs and then apply certain methods to recover from deadlock.

Finally, an OS can assume that deadlock will never happen or rarely occur and fully ignore it. This approach might be referred to as the no-policy approach. The positive point of this approach is it saves CPU time and memory space for deadlock management, unlike those required for detection, prevention, or avoidance methods. This strategy has been adopted in UNIX.

3.3.4 Deadlock Prevention

This technique allocates resources in such a manner that at least one of the four necessary conditions of deadlock cannot occur. It denies at least one of the four conditions required for the occurrence of a deadlock. These techniques do not utilize a resource properly. For example, an OS adopting a deadlock prevention technique allows recourses to be preempted from a blocked process. No preemption condition is violated and deadlock does not occur.

We have already discussed that there are four conditions to be satisfied for the occurrence of a deadlock. Below is a brief discussion about how the conditions for a deadlock can be prevented.

1. **Mutual exclusion condition**

The mutual exclusion condition indicates the locking of resources in exclusive, also known as the non-shareable mode that leads to blocking of resources. An OS can avoid blocking resources by locking the resources in shareable mode, if feasible. For example, two processes need to read data from a file.

The OS can allow locking of that file in shareable mode by two processes simultaneously. Locking resources in shareable mode prevents waiting for a resource. When a process request to look at a resource in shareable mode, the OS allows the locking instantly and no time is wasted waiting to lock the resource exclusively.

Some resources cannot be shared by multiple processes. For example, two processes cannot simultaneously share a scanner. Again when a process writes data to a file, it needs to lock the file in exclusive mode.

2. **Hold and Wait**

An OS can avoid this by adopting a strategy that a process must request for all resources it requires at the same time. This strategy can be implemented in two ways.

Firstly, a process p requires all the resources it needs when the execution begins. On the other hand, a process can start execution with a minimum set of resources required and request the other resources when required at the time of execution. In the second approach, a process must release all the resources it presently holds before requesting any other required resources.

For example, a process p requires resources R1, R2, R3, and R4 in exclusive mode in the sequence: R3 and R1 simultaneously at the beginning of execution, then R1, R2, and R3 simultaneously, and lastly R1 and R4 simultaneously. When applying the first strategy, process P acquires all four simultaneously before it starts execution.

When applying the second strategy, the process p needs not to acquire all the four resources at the beginning. The process p1 acquires the resources R1 and R3 at the time of starting execution. Soon the process p requires R2. The process p releases all the acquired resources R1 and R3 and sends a request to the OS for acquiring the resources R1, R2, and R3. Later, the resources R2 and R3 are not required and R4 is needed. The process p releases all the acquired resources R1, R2, and R3 and sends a request to acquire R1 and R4.

3. **No Preemption**

An OS can adopt a policy to avoid this condition. If a process requests an unavailable resource, the process must release all the resources it has presently acquired and wait for the required resources. In other words, when a process, P requests a busy resource this policy allows other processes to preempt the resources currently held by P.

Certain resources are good candidates to be preempted. For example, CPU, and memory need a mention. While revoking the control of the resources from a process the OS updates the process control block of the process. Not all the resources can be preempted. For example, a partly updated file cannot be preempted since data will be lost.

4. **Circular Wait**

An OS can impose an ordering of resources for avoiding the circular wait. Resources in a system are grouped into certain categories. For example, all magnetic disks form a group and all scanners from another group. Each category is assigned a number. When a process acquires a resource belonging to a specific category, no other process belonging to a category having a lower number can be claimed.

Formally, if there are n categories of resources in a system, ranging from 0 to n-1, and if a process acquires a resource belonging to category c, then the process can only request a resource belonging to category c+1.

This helps to prevent a formation circle because a process holding the control of a resource of the category n-1 cannot claim a resource of the type 0. The disadvantage of this technique is each process needs to acquire all the required resources in a predetermined and specific order depending upon the arrangement of numbers assigned to various categories of resources

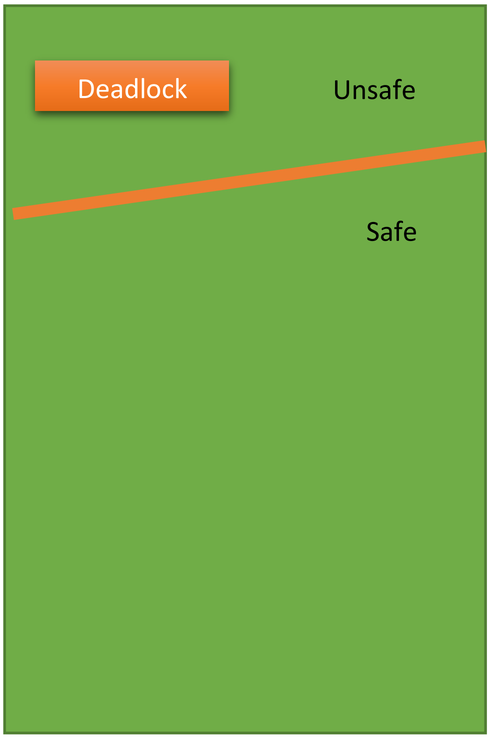
3.3.5 Deadlock Avoidance

In deadlock avoidance, Operating System requires additional information about which process requires which resource in advance to avoid a deadlock. The fundamental concept of deadlock avoidance is that OS only entertains those requests for resources by processes that will not lead to a deadlock. The deadlock-avoidance algorithm regularly inspects the resource-allocation state of the processes to confirm that there can never occur a circular-wait condition. Resource-allocation *state* is explained by the total number of available resources and number of resources allocated to the processes, and the maximum demands of resources by the processes.

Safe State

When processes request resources and it is available in the system and allocated immediately then the system in a **safe state**. The system is in a safe state if there exists an order of processes <P_1, P_2, ..., P_n> which satisfied their resources in that sequence. For each P_i, the resources that P_i can still request can be satisfied by currently available resources + resources held by all the P_j, with $j < i$. That is: If P_i resources wanted are not instantly available, then P_i can wait till all P_j have finished. When P_j is finished, P_i can get needed resources, execute, return allocated resources, and terminate. When P_i terminates, P_{i+1} can obtain its needed resources, and so on.

If a system is in a safe state then there is no Deadlock. If a system is in an unsafe state then there is the possibility of deadlock. To avoid deadlock, it is must confirm that a system never entered an unsafe state and it should be in a safe state.



Relationship between safe, unsafe, and deadlock state

Avoidance algorithms

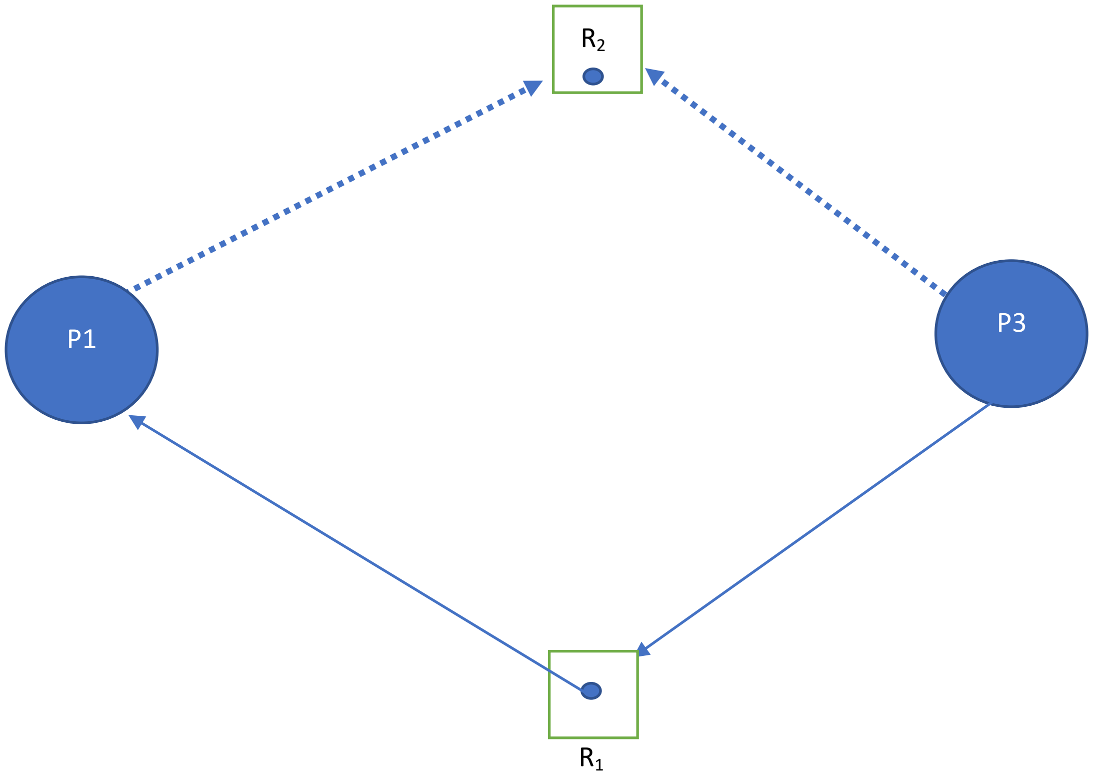
There are two algorithms to avoid deadlock

For the Single instance of a resource type, in this algorithm, we use a Resource-Allocation Graph (RAG) and the second algorithm which is deals with multiple instances of a resource type, then we use the Banker's algorithm.

Resource-Allocation Graph (RAG) Scheme

In Resource allocation Graph Scheme Claim edge $P_i \rightarrow R_j$ showed that process P_i may demand resource R_j; it is denoted by a dashed line. Claim edge changes to request edge when a process needs a resource. Request edge changed to an assignment edge when the resource is assigned to the process. When a resource is free by a process after execution, the assignment edge reconverts into a claim edge. Resources must be demanded in advance in the system.

Suppose that process P_i requests a resource $R_{j.}$ The request can be approved if and only if changing the request edge to an assignment edge does not result in the creation of a cycle in the Resource Allocation Graph(RAG).



(i) Unsafe state in RAG

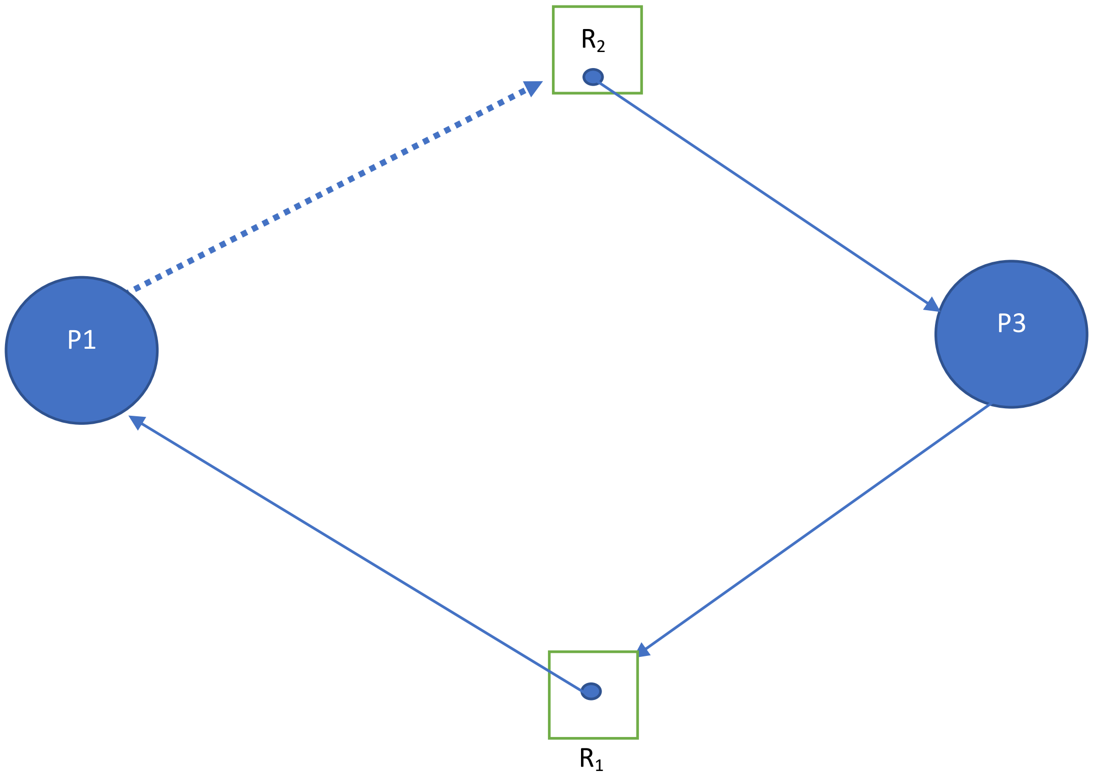


Figure showing (i) safe and (ii) unsafe state in RAG

BANKER'S ALGORITHM

The **banker's algorithm** adopts the deadlock avoidance strategy. The name, banker's algorithm, depicts the similarity of the concept in the field of banking. A banker never allots the cash available in such a manner that the banker cannot fulfill the needs of its clients.

A scheduling algorithm that can avoid deadlocks is BANKER'S ALGORITHM. It is modeled on the way that a single banker might deal with a group of customers to whom he has granted lines of credit. The banker's algorithm can be implemented for a single resource or multiple resources. The banker algorithm keeps track of all the resources currently available to the operating system and all the resources which are allocated to the processes. It maintains the various data structures to store the information about the number (instances) of resources. Whenever a process requests a resource, the banker's algorithm checks the availability of the resources and acts accordingly. Finally, it helps to check that the system is in a SAFE STATE or not.

Data Structures for the Banker's Algorithm

Suppose *n* be the number of processes, and *m* be the number of resource types.

Available: Array of length *m*. If available [*j*] = *k*, it means that *k* instances are available of resource type R_j.

Max: *n* x *m* matrix. where n is the number of rows and m is the number of columns. *Max* [*i,j*] = *k*, it means that k instances maximum needed by process P_i of the resource type R_j.

Allocation: *n* x *m* matrix. where n is the number of rows and m is the number of columns Allocation[*i,j*] = *k*, it means that k instances are currently allocated to process P_i of the resources R_j.

Need: *n* x *m* matrix. where n is the number of rows and m is the number of columns *Need*[*i,j*] = *k*, it means that P_i may need *k* more instances of R_j to complete its task. It can be calculated by matrix subtraction:

Need [*i,j] = Max*[*i,j*] – *Allocation* [*i,j*].

Safe State Algorithm

This algorithm is applied by an Operating System to determine whether a system is in a safe state or not. A system is safe if the system can allocate resources to every process in some order and still is in the safe state which means that its avoiding deadlock. In other words, a system is in a safe state only if there occurs, a safe sequence.

Let 'm' be the total number of resources and 'n' be the total number of processes in the system.

Let 'Work' and 'Finish' be the two arrays of length 'm' and 'n'.

(1) Set Work = Available and also Set value FALSE for all the members of array Finish

FINISH[i] = FALSE, for i = 0 to n - 1

In other words, it is assumed that all the processes are to be executed and neither of the processes is completed yet.

(2) Find a process “i” such that following both conditions should be satisfied

Finish[i] = FALSE and Need[i] < = Work

If no such “i” exists ,then go to step (4).

In other words, find a process that is not completed yet and whose need is less than available.

(3) Set Work = Work + Allocation$_i$ (Allocation of process i)

Set Finish[i] = TRUE.

Go to step (2).

In other words, after the completion of the execution of a process, the process will return all the resources to the system. So, the available resources of the system will increase.

(4) Check if Finish[i] = TRUE for all “i” at that time

System is in safe state

else

System is not in safe state

Consider an example

Let there are 3 resources and 5 processes such that

Processes	Allocation			Max			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P0	0	1	0	6	5	3	2	3	2
P1	2	0	0	3	3	2			
P2	3	0	2	9	0	2			
P3	2	1	1	2	2	2			
P4	0	0	2	5	3	2			

Find the Need matrix by using the formula:

Need = Max – Allocation

So, the need matrix is:

Processes	Need		
	R1	R2	R3
P0	6	4	3
P1	1	3	2
P2	6	0	0
P3	0	1	1
P4	5	3	0

Hereby considering the need of all the processes, we have to find such a process whose need is less than available. It is visibly that the need of the process P1 is less than Available, so the process P1 will be executed first. (As 1 3 2 < 2 3 2)

After the execution of process P1, the P1will return all the resources to the system. So, the available resources will increase as the following :

Available = Available + Allocation_i

In the same way, after the completion of a process, the Available will be updated, as the process returns the resources to the system after being executed.

So, the processes will be executed in the following sequence

< P1, P3, P4, P0, P2>

Explanation of Safety Algorithm

First, we set all processes in the false state which means that no processes are executed or completed yet, and set work is equal to available. Then we find repeatedly a process that is false state and whose need is less than equal to work. If we find such a process then we change its state to true and its resources are deallocated which is added in work. In this way, the number of available resources is increased. Then we check all the processes are true or not. If all the processes are true then the system is in a safe state. If even one process is in a false state then the system is in an unsafe state.

Resource Request Algorithm

The second part of the banker's algorithm, known as the resource-request algorithm determines whether granting a resource according to any claim by a process leads to an unsafe state. This algorithm is applied when a process sends a request to grant one or a set of resources.

Request_i = request vector for process

P_i = If request_i [j] = K,

it means that P_i requested the K instances of resource type R_j.

(1) If Request[i] <= Need [i] (Check request is genuine or not)

then

Goto step (2)

Else

"Raise an error condition"

(Because the process has exceeded its maximum claim)

(2) If Request[i] <= Available (Check weather system has available resources or not)

Then

Goto step (3)

Else

The process P_i has to wait because the resource is not available yet.

(3) Let the system pretend that the resources have been allocated by changing the following:

Available = Availablei – Requesti ;

Allocationi = Allocationi + Requesti ;

Needi = Needi – Requesti;

Then check with the safety algorithm, if the result of the safety algorithm is safe, then the transaction is completed and the process is allocated the requested resources.

However, if the system is in an unsafe state, then the process must wait and the old allocation state is restored.

Explanation of Resource Request Algorithm

In this algorithm, first, we check whether the request of the process is less than its need or not. If the request is more than need then it is an error because the process claims more resources from its maximum claim. Else we check current request is less than available or not. If it is less than or equal to available then we allocate the request and update data structure such as allocation of the process by adding a request in allocation, the request is subtracted from need as well as available. Then we check the system is in a safe state or not by applying a safe state algorithm. If available is less than need then the request can not be granted and the process will wait because the resource is not available yet.

Advantages

Advantages of banker's algorithm, it allows mutual exclusion, hold and wait, no preemption conditions. The system assures that a process will be allocated resources without deadlock.

Disadvantages

The disadvantage of a banker's algorithm is the overhead of simulation and calculation before each time a process requests for a resource. Deadlock avoidance techniques cannot be applied until an OS knows the resource requirement of a process in advance. This approach may lead a process to starvation while for a long time to get the requested resources allocated by the OS.

Illustration

Consider the following table with 5 processes P_0 to P_4 and 3 resource types namely X Y and Z; X has 9 instances, Y has 5 instances, Z has 8 instances.

	ALLOCATION	MAX
	X Y Z	**X Y Z**
P_0	2, 2, 3	6, 2, 3
P_1	2, 0, 1	6, 1, 2
P_2	0, 1, 1	3, 1, 2
P_3	1, 0, 0	1, 0, 1
P_4	1, 1, 2	6, 2, 3

Using Banker's algorithm check whether the system is in a safe state or not. If it is a safe state then if P_1 requests (1, 0, 1) then it is a safe state or not. If it is not in a safe state then, what changes are required in available resources to make the system safe?

Total instance of resources is

X = 9, Y = 5, Z = 8

Total allocated resources to P_0 to P_4 are

X = 6, Y = 4, Z = 7

Now available is

X = 3, Y = 1, Z = 1

Now calculate Need: i.e. $\text{Need}_i = \text{MAX}_i - \text{Allocation}_i$

Process	**Allocation**	**MAX**	**NEED**	**Available**
	X Y Z	**X Y Z**	**X Y Z**	**X Y Z**
P_0	2, 2, 3	6, 2, 3	4,0,0	3,1,1
P_1	2, 0, 1	6, 1, 2	4,1,1	
P_2	0, 1, 1	3, 1, 2	3,0,1	
P_3	1, 0, 0	1, 0, 1	0,0,1	
P_4	1, 1, 2	6, 2, 3	5,1,1	

Now apply safe state algorithm to check whether the system is safe state or not.

First step is set all process equal to false i.e. P_0, P_1, P_2, P_3, P_4 are FALSE.

Now find such process which false and whose $\text{need}_i \leq$ Available i.e. P_2.

Now update data structure and status of processes i.e. $finish_i$

Available = Available + $Allocation_2$

$Finish_2$ = True

Now new Available = 3, 1, 1 + 0,1, 1

= 3, 2, 2

All processes are not true so again go to step - 2 to find a process that is yet false and needs is less than equal to available.

$Need_3$ < = Available i.e. P_3

Now update data structure and status of processes i.e. $finish_i$

Now Available = Available + $Allocation_3$

New Available = 3, 2, 2 + 1, 0, 0

=> 4, 2, 2

$Finish_3$ = True

Now 3 processes are false i.e. P_0, P_1 & P_4 & Available = 4, 2, 2

All processes are not true so again go to step - 2 to find a process that is yet false and needs is less than equal to available.

$Need_0$ < = Available i.e. P_0

Now update data structure and status of processes i.e. $finish_i$

Now Available = Available + $Allocation_0$

=> 4, 2, 2 + 2,2,3

= 6, 4, 5

$Finish_0$ = True

Now 2 process are false i.e. P_1 & P_4 and Available = 6, 4, 5

All processes are not true so again go to step - 2 to find a process that is yet false and needs is less than equal to available.

$Need_1$ < = Available i.e. P_1

Now update data structure and status of processes i.e. $finish_i$

New Available = Available + $Allocation_1$

= 6,4,5 + 2, 0, 1

=> 8, 4, 6

Finish$_1$ = True

Now only one process i.e. P$_4$ is false if its need is greater than Available then system is not safe state but need$_4$ < Available

Update Available & Finish

New Available = Available + Allocation$_4$

= 8, 4, 6 + 1, 1, 2

=> 9, 5, 8

Now finish is true so there is no false in finish and all resources are available now i.e. 7, 3, 6.

□ System is safe state & sequence is

< P$_2$, P$_3$, P$_0$, P$_1$, P$_4$>

The system is in a safe state if the process runs in the above sequence.

Now check if P$_1$ requests (1, 0, 1) then if it is allocated to the process then whether it is in safe-state or not? To check that the resource request algorithm follows.

In this algorithm first request is to check whether it is less than Need$_i$ or not. If it is greater than Need$_i$ its mean P$_i$ request more resources as it claims earlier so it can’t be given & request directly terminated. If Request$_i$ is less than Need$_i$ then we check Request$_i$ < = Available if it is true then the request is granted & we check that now system is in a safe state or not by using the safe state algorithm used earlier. If it is false then at that time Request$_i$ can not be fulfilled and the process P$_i$ made to wait.

In this case Request P$_1$ is (1, 0, 1) which less than Need$_1$ and Available so we run resource request algorithm.

Set Available = Available - Request$_i$

= 3, 3, 1 - 1, 0, 1

Now Available = 2, 3, 0

Set Allocation$_1$ = Allocation1 + Request$_1$

= 2, 0, 1 + 1, 0, 1

Now Allocation$_1$ = 3, 0, 2

Set Need$_1$ = Need1 - Request$_1$

= 4, 1, 1 - 1, 0, 1

Now Need$_1$ = 3, 1, 0

Now Request is granted now we check after allocating the request whether the system is in a safe state or not. To check that we used a safe state algorithm. Now the situation of processes is given below:

Process	**Allocation**	**MAX**	**NEED**	**Available**
	X Y Z	**X Y Z**	**X Y Z**	**X Y Z**
P_0	2, 2, 3	6, 2, 3	4,0,0	2,3,0
P_1	3, 0, 2	6, 1, 2	3,1,0	
P_2	0, 1, 1	3, 1, 2	3,0,1	
P_3	1, 0, 0	1, 0, 1	0,0,1	
P_4	1, 1, 2	6, 2, 3	5,1,1	

Now by applying a safety algorithm there is no such process whose $need_i$ < = available. So if the request of $process_1$ is granted then the system comes in an unsafe state. So, we revoke the request of $Process_1$ so that system may run safely. To overcome this problem we must need at least one A-type and at least two C-type Resource. If it will available in the available list then the system may come in a safe state.

3.3.6 Deadlock Detection and Recovery

Another approach to deadlock management is detection and recovery. In this approach, the system is not prevented from occurring a deadlock. The operating system periodically searches or analyzes that whether a deadlock has occurred. If the system is in deadlock, OS recovers from deadlock.

This approach of deadlock management has two phases, detection, and recovery of deadlocks. The algorithm to detect deadlocks in an OS can use a data structure similar to that of the data structures used in the context of deadlock avoidance.

A variation of the safety algorithm discussed in deadlock avoidance has been discussed below. In this algorithm, the data structures ALLOCATION, CLAIMS (Request), and AVAILABLE are the same as the data structure used in the context of deadlock avoidance. Another data structure used in this algorithm is FLAG.

The FLAG is a vector of the length equal to the number of processes in a system. FLAG is used to mark and unmark a process. For example, FLAG[X]=TRUE means the process Px is marked, and FLAG[Y]=FALSE means that the process Py is unmarked.

An OS verifies after a certain interval whether a deadlock has occurred. This algorithm finds a sequence of execution of processes that do not lead to an unsafe state. For example, there are 3 processes P1, P2, P3. Executing these processes in the sequence P2□P1□P3 does not lead the system to deadlock. The steps in this algorithm are:

1. Initialize FLAG[i] to FALSE, where 0 <= i<=s-1, and s denotes the number of processes.
2. Find a process Px such that FLAG[X]=FALSE and $CLAIMS_X$<=AVAILABLE. Go to step 6, if no such process is found.

3. Update the AVAILABLE vector because, after the termination of the process, Px, OS revokes the control of all the resources held by Px. Perform AVAILABLE = AVAILABLE + ALLOCATION$_X$.
4. Mark the process Px as FLAG[x]=TRUE.
5. Go to step 2.
6. If FLAG[i]=TRUE for all i, where 0<=i<=s-1, then all processes can be completed properly and there exists no deadlock, else the system is deadlocked with the processes which are unmarked that is FLAG[i] is set to FALSE.

The algorithm needs an order of $O(m \times n^2)$ processes to detect that the system is in a deadlock state or not.

The frequency of verifyin5r4ag the system whether a deadlock has occurred is an important issue. An OS can adopt any of the two policies. First, an OS can verify the system state whenever a process sends a request for resources. Alternatively, an OS can verify the system state after a certain interval of time.

An OS can adopt two strategies to recover from deadlock. Firstly, all the processes involved in the deadlock can be **terminated forcefully**. Alternatively, the OS can apply a **preemption technique** from a deadlock period.

Aborting deadlocked processes can be done in two ways.

An OS can abort all the deadlocked processes when a deadlock is detected.

On the other hand, an OS can apply the trial and error method. This means, one of the deadlocked processes is aborted and it is verified that whether the deadlock is over. This procedure is continued until the system reaches a safe state.

Which process is to be aborted first depends on the cost of terminating the process. If a process is 95% done, OS tries to avoid aborting that process. The number of processes to be aborted is also important in this context. For example, four processes are deadlocked. The OS can recover itself from deadlock by aborting either all of the processes A, B, and C or a single process D. In this case, an OS will sacrifice a single process to recover from deadlock.

The second approach of recovery from deadlock is preempting a resource from a process. In this case, a victim is select, the process from which a resource is revoked by an OS is rolled back to a safe previous state and continued later on. In this approach, the starvation problem may occur by selecting the same process as a victim again and again.

3.3.7 Deadlock Ignorance

In deadlock ignorance, we assume that our system never enters in the deadlock condition. It is only a theoretical concept. Practically it is not possible because if the system is not in a safe state then the system may be in a deadlock state or we can say that if the system satisfied all the necessary four conditions simultaneously then the system must be deadlock state which can be ignored.

Comparison Between Various Deadlock Management

Deadlock Prevention	**Deadlock Avoidance**	**Deadlock Detection and Recovery**
It is easy method	It is complex method	It is also a complex method
It adversely affects Utilization	Utilization improved at a cost	Depends on the frequency of detection
Involves violating at least one of the necessary conditions for deadlock	RAG Algorithm Bankers Algorithm (Multiple instances)	Maintain a wait-for graph and periodically invoke a "cycle-search" algorithm
Used in critical systems	Used in applications that have deterministic needs	Used in Database Applications

Points to Remember

- Deadlock: Process is said to be in a deadlock state if the process is waiting for a specific event that will not occur.
- The reusable resource can be safely used by only one process at a time.
- Four necessary conditions for deadlock are:
- (i) Mutual exclusion (ii) Hold and wait
- (iii) Circular waiting (iv) No preemption
- Resource Allocation Graph is used to detect the deadlock.
- Resource Allocation Graph is also known as RAG.
- If RAG contains a cycle then it may or may not deadlock the situation.
- In RAG, the process is represented by a circle and the resource is represented by a square.
- Process Wait For Graph is obtained from Resource Allocation Graph.
- Process Wait For Graph is known as PWFG.
- The policies to handle deadlock can be classified into four major groups
- Deadlock Prevention
    - Deadlock Avoidance
    - Deadlock Detection and Recovery
    - Deadlock Ignorance
- Deadlock prevention is a set of methods for ensuring that at least one of the necessary conditions cannot hold.
- Deadlock avoidance needs extra information about how resources are to be demanded.
- Safe order is an order of the process in which there exists one order in which all the processes can be executed without resulting in a deadlock.
- The deadlock avoidance algorithm enthusiastically examines the resource-allocation state to confirm that a circular wait condition can never exist.
- A safe state is not a deadlock state.
- Not all unsafe states are deadlocks.

- Deadlocks occur only when some process makes a request that cannot be granted immediately.
- Banker's algorithm is a deadlock avoidance algorithm.
- A safety algorithm is used to find the safe state of the system.

3.4 PRACTICCE EXERCISES

1. What is deadlock?
2. What are the necessary conditions for the occurrence of a deadlock?
3. How deadlock can be avoided.
4. What do you mean by RAG?
5. Explain methods for deadlock detection.
6. Explain safe state.
7. What are the different methods of handling deadlock.
8. Explain the Banker's algorithm with an example.
9. Explain the safe state algorithm with an example.
10. Explain the resource algorithm with an example.
11. Consider the following snapshot of a system.

Process	**Allocation**	**MAX**	**Available**
	M N O P	**M N O P**	**M N O P**
P_0	2, 2, 1, 3	7, 5, 3, 3	1, 1, 2, 1
P_1	2, 2, 1, 1	6, 2, 2, 4	
P_2	1, 2, 1, 0	4, 2, 4, 0	
P_3	1, 1, 0, 1	2, 2, 2, 3	
P_4	2, 1, 2, 0	2, 4, 3, 3	

Whether the system is a safe state or not. If process P3 request 0, 0, 1, 2 can this request granted immediately or not and also check after granting above request system is in sate state or not. If it is not in a safe state then what changes are required in available resources to make the system safe.

B.Sc.(DATA SCIENCE)

SEMESTER-II

OPERATING SYSTEM

UNIT IV: MEMORY MANAGEMENT

STRUCTURE

4.0 OBJECTIVES

- Understanding Memory, Address Binding, Logical Vs Physical Memory
- Understanding Concept of Dynamic Loading and Linking, Swapping, Contiguous Memory Allocation, Segmentation, Paging, Demand Paging, Page Replacement algorithms

4.1 INTRODUCTION

The important task of allocating memory to processes, and efficiently, ensuring that processes have their instructions and data in main memory when needed, is termed as Memory Management.

Memory is one of the important resources of the computer system. The main memory is usually divided into two partitions, one for the resident operating system and the other for the user processes.

The operating system is responsible for memory management. It keeps track of the status of the memory, makes policy, allocates the memory, and then deallocates the memory from the process.

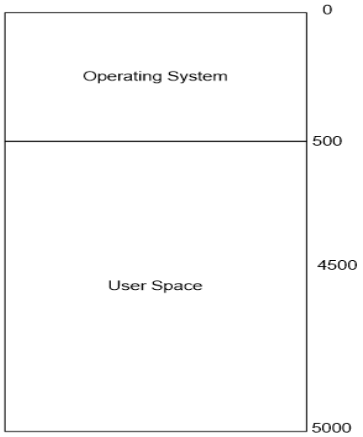


Figure: Main Memory

The functions of Memory Management

An operating system performs various activities for memory management:

1. It keeps track of the status of the memory, that whether it is free, allocated, available, or not available (full).
2. If the memory is free for the execution of the process, the Operating System chooses a policy to allocate the memory to the process.
3. According to the chosen policy (algorithm), it allocates the memory to the process.

4. After the execution of the process, it deallocates the memory.

4.2 ADDRESS BINDING

Address binding means map from one address space to another i.e. the logical addresses to real physical addresses in memory. To execute a program, it must be loaded into the main memory at a particular location. The instructions that use addresses in a program must be bound to proper address space in the main memory. Many instructions use "fixed" addresses these must be *bound* to "fixed" locations in the memory.

This binding of addresses of instruction and data to actual physical addresses can take place at compile-time, load time, and run time during the execution of a process.

To run a program there is a sequence of steps to execute a particular program which is as follow:

- A program is known as a source program and it is loaded into the main memory.
- A compiler or assembler or interpreter is required for compilation or assembly.
- After compilation or assembly, a code is generated by them which, is known as Object code.
- When a program is executed then, the program links with other object modules or system library which are loaded into linkage editors with the help of linker and loader.

At last, the process is stored in memory as a binary image, and system library files are dynamically loaded and dynamically linked.

4.3 DYNAMIC LOADING AND LINKING

In any program execution linking and loading plays a key role. Linking means creating an executable module of a program by linking the object codes which is created by the assembler or compiler. Other hand loaders, load these runnable modules to the main memory for execution.

Linking: It is the **procedure of linking all the modules or the sub-program or all the functions in a program for whole program execution**. It takes more than one object module and combines it into a single object file. The linker is also called a link editor. It takes object modules from the compiler or assembler and forms a runnable file for the loader.

Based on time Linking is classified into two categories – static linking and dynamic linking:

Linking of object modules is done at compile time and at load time. Compile-time linking is completed when the source code is converted into machine code is called Static linking. The load-time linking is done while the program is loaded into memory by the loader.

Loading

Loading is the **procedure of loading the program from auxiliary memory to the main memory** for execution.

Dynamic loading

It involves loading routines into memory only when required. This is done during execution. Dynamic loading reduces the memory requirements of large programs. This is especially the case if there is a large set of infrequently used routines.

Dynamic linking

It is often used for libraries. Only a “stub” of the library is kept in the image of the program. When a program calls one of these routines, the routine is loaded and linked into memory. All programs share one copy of the same library routine.

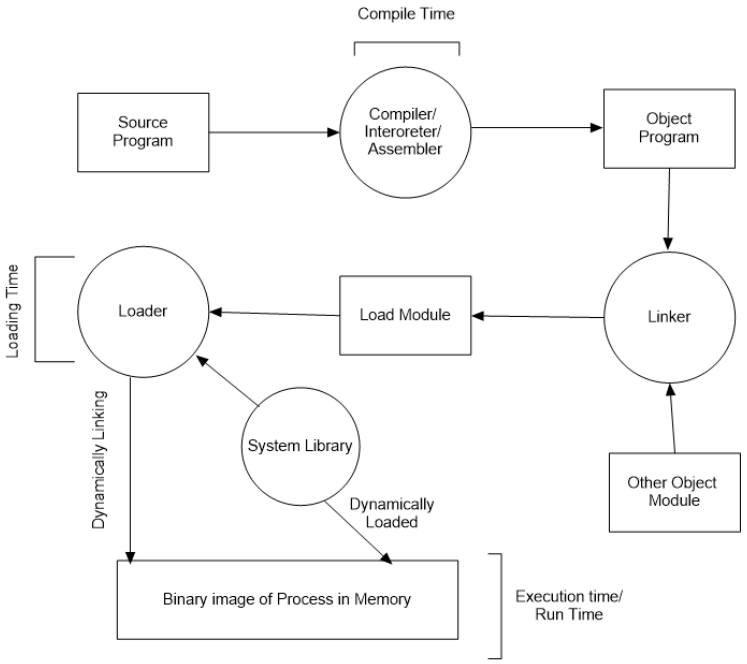


Figure Processing of a User Program

Address binding of instructions and data to memory addresses can happen at three different stages i.e. compile time, load time, execution time (as shown in figure Processing of User Program)

Compile-time: If the memory location is known a priori, **absolute code** can be generated. It must recompile code if starting location changes, after compiling the process.

Load time: It must generate **relocatable code** if memory location is not known at compile time. Final binding is delayed until load time.

Execution time: If the process can be moved during its execution from one memory segment to another, binding may be delayed. Then hardware support is needed for address maps. For example base and limit registers. Most general-purpose operating systems use the execution time-binding method.

4.4 LOGICAL VS. PHYSICAL ADDRESS SPACE

When a process is executing, the CPU would generate addresses, called **Logical Ad dresses**. There are two registers used to define logical address space. These are the Base register and Limit register. Base register stores the starting memory address of the process and the Limit register stores the size of a process. Let a process P whose size is 5000 i.e. value of the Limit register and it has the start address is 1000. i.e. value of Base Register. A pair of **base** and **limit** registers define the logical address space as the shown figure below

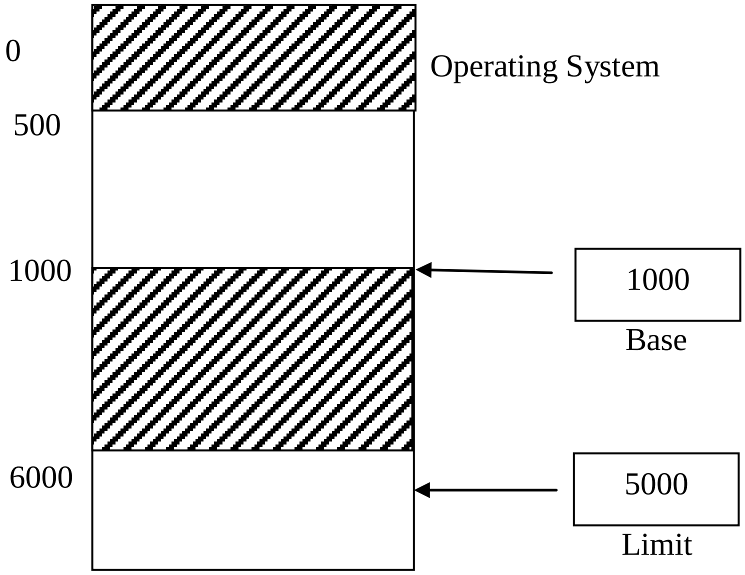


Figure Base and Limit Register

Executing processes that occupy corresponding physical addresses in the physical memory is known as **Physical Addresses.** Logical and physical addresses are the same in compile-time and load-time address-binding schemes but logical (virtual) and physical addresses differ in the execution-time address-binding scheme. A physical address is the effective memory address of instruction or data.

4.5 SWAPPING

Swapping is a mechanism in which a process can be swapped temporarily out of memory to a storage device (backing store) and another process brings in memory for execution. If a swapped-out process requires again then it is brought back into memory for continued execution.

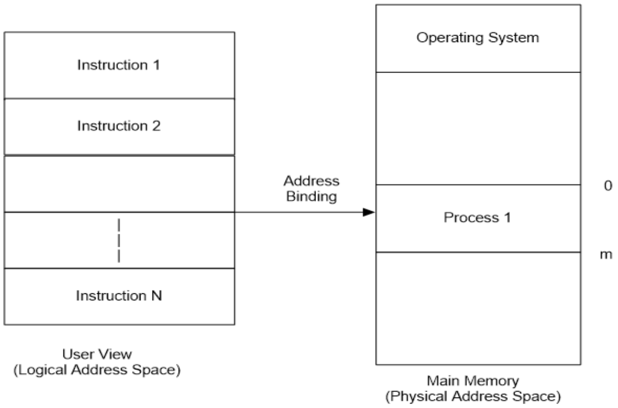


Figure Logical and Physical Address Space

The backing store is a fast disk large enough to accommodate copies of all memory images for all users. It must provide direct access to these memory images. When the process swapped out then it is called **Roll out** and when it brings in then it is called **Roll in.** The swapping variant is used for priority-based scheduling algorithms where the lower-priority process is swapped out due to the arrival of a higher-priority process. It can be loaded and executed. A major part of swap time is transfer time. The system maintains a ready queue of ready-to-run processes which have memory images on disk.

Example: Let there are n processes in the storage device and P4 in main memory which is not required. Now P2 is required so P4 is roll out and P2 is rolled in which is known as swapping as shown in the figure.

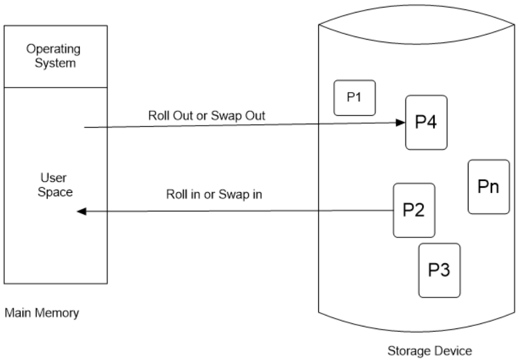


Figure: Swapping

4.6 MEMORY ALLOCATION

Memory management includes the various methods of allocating the memory to the different processes. The memory management techniques are basically of two types:

(i) Contiguous memory allocation.
(ii) Non-contiguous memory allocation.

Contiguous Memory Allocation

In contiguous memory allocation, the data and instructions of a program are sure to reside in a single contiguous memory area. It is a simple technique of allocating memory to the processes. There is no possibility of sharing data and instructions among the processes, which are loaded into entirely different portions of memory. The entire process is allocated a single contiguous memory area for its execution. Addressing is very simple as compare to the non-contiguous allocation methods.

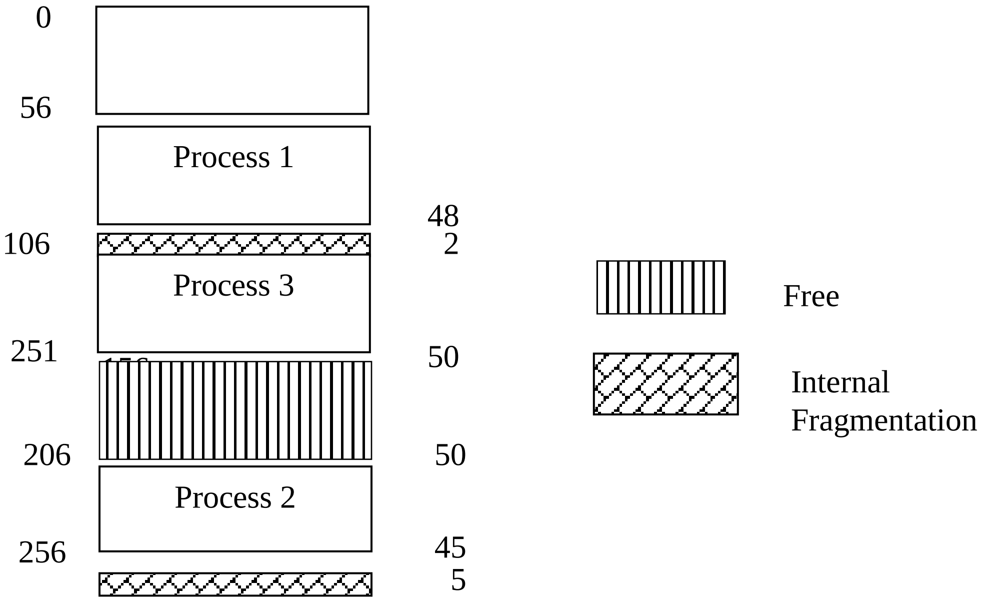
Fragmentation

In contiguous memory allocation, one problem occurs i.e. Fragmentation. It can be classified into two categories:

1. Internal Fragmentation
2. External Fragmentation

1. **Internal Fragmentation:** In fixed-size memory management, allocated memory to a process may be slightly larger than the requested memory by the process. The size difference between occupied memory and partition size, which is not used by another program is known as Internal Fragmentation.

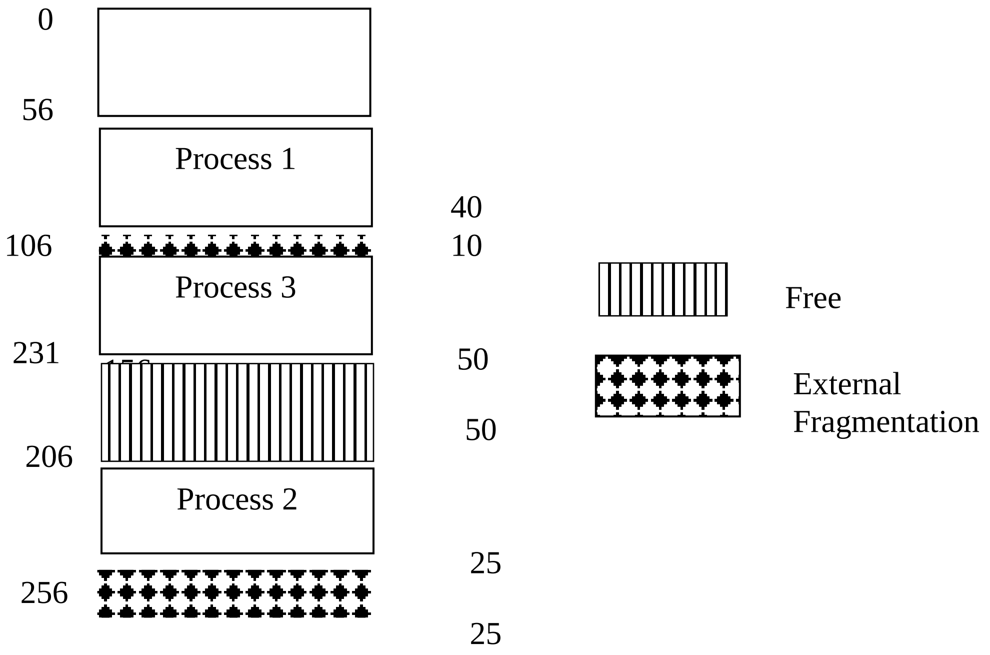
Example of Internal Fragmentation: Let, there are three processes P1, P2, P3 whose size is 48, 45, 50 respectively. The total memory size is 256 and OS occupies 56K and the rest of the memory is divided into fix partition of 50k.



Example of Internal Fragmentation

2. **External Fragmentation:** In variable size memory management, some small chunks are available in memory that cannot be effectively used. Total free memory is larger than the required memory by the requested process but it is not contiguous. It is known as External Fragmentation.

Example of External Fragmentation: Let, there are three processes P1, P2, P3 whose size is 40,25,50 respectively. The total memory size is 256 and OS occupies 56K and the rest of the memory is divided into fix partition of 50k.



Example of External Fragmentation

Non-contiguous Memory Allocation

In non-contiguous memory allocation, the data and instructions of a program may reside in non-contiguous memory locations. It is possible to allocate different memory locations to a process for its execution. In other words, the process can be divided into parts and can be allocated to different memory areas. It offers various advantages over, contiguous memory allocation like it permits sharing of code and data amongst processes. There is no external fragmentation of physical memory. It supports the vertical memory concept.

Contiguous Memory Allocation Methods

The memory is partitioned into different blocks of different sizes for accommodating the programs. The partitioning is of three types:

1. Single partitioning memory management
2. Multiple Fixed Partitioning
3. Multiple Variable Partitioning

Single Partition Memory Management

The main memory is divided into two partitions. One partition is for the resident operating system and, the other for the user processes. In the single partition memory management technique, the user area has a single partition. It is the simplest memory management technique.

Consider an example, where the total main memory is 256 MB. 56 MB is reserved for the operating system and 200 MB is for the user processes.

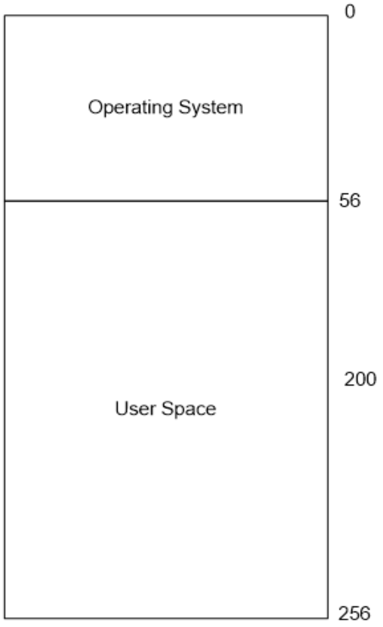


Figure: Single Partitioned Main Memory

Here, the user area has a single partition. At a time, only one program can be executed. The maximum length of the program, which the user can execute, is equal to the size of the partition of the user area. Here, the maximum size of a program can be 200 MB.

Internal fragmentation is much more. Since there is a single partition of memory, we can execute only one program at a time. Consider an example, where the size of a program is 50 MB. In this case, there will be an internal fragmentation of about 150.

Here, external fragmentation is also more. However, if the size of the program increases from the size of the partition, then it is not possible to execute the program.

Advantages

- It is a very simple memory management technique.
- Any program of size less than the size of the user area can be executed easily.

Disadvantages

- Multiprogramming is not possible.

- Internal fragmentation is more.
- It is not possible to execute a program of size greater than the size of the user area.
- The maximum size of a program to be executed is fixed.

Multiple Partitions with Equal Size

In this, whole memory is divided into a fixed number of partitions of different sizes, which may suit the program sizes. Each partition accommodates exactly one process. When a program needs, it is loaded into a partition that one big enough to accommodate the program. Sometimes, some space may be left unoccupied in a partition after loading a program, which is not used by another program that space is wasted and it is known as **Internal Fragmentation**. The degree of multiprogramming is fixed since the number of partitions is fixed is the drawback of this partitioning.

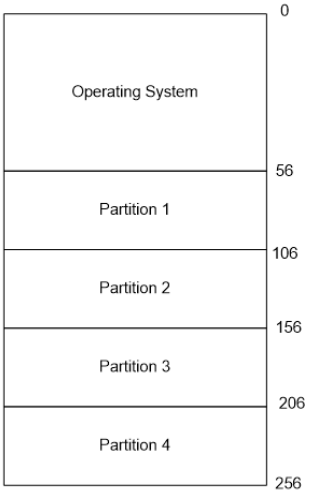


Figure: Multiple Fixed Partitioned Main Memory

In this technique, internal fragmentation is less as compared to the single partition technique. Consider an example, where a program of size 50 MB arrives. In this case, there will be no internal fragmentation. However, if a program of size 40 MB or 30MB arrives, then there will be an internal fragmentation of 10 MB or 20 MB respectively.

In this technique, external fragmentation may be more. Consider an example, where a process of size 55 MB arrives. Even if four of the partitions(200 MB) are free, but it is not possible to execute a program of 55 MB.

Advantages

- It is possible to execute more than one program at a time. In other words, multiprogramming is possible.
- Internal fragmentation is less as compared to a single partition.
- It is simple to maintain.

- The degree of multiprogramming depends upon the number of partitions.

Disadvantages

- It limits the size of the program, the user can execute.
- External fragmentation is high.
- It is not possible to execute a program having its size greater than the size of the partition.

Multiple Partitions with Variable Size

This scheme is free from the limitation encountered in the case of fixed partitioning. In this, the entire available memory is treated as a single partition. All programs, requesting memory are store in a waiting queue and loaded only when a free partition available which is big enough to occupy that program. When a program is allocated a space exactly equal to its size, the balance unoccupied space is treated as another free partition. When a free partition is too small to accommodate any program, it is called **External Fragmentation.**

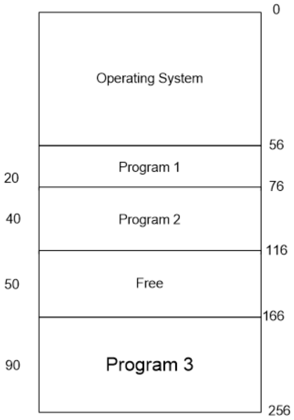


Figure: Multiple Variable Size Partitioned Main Memory

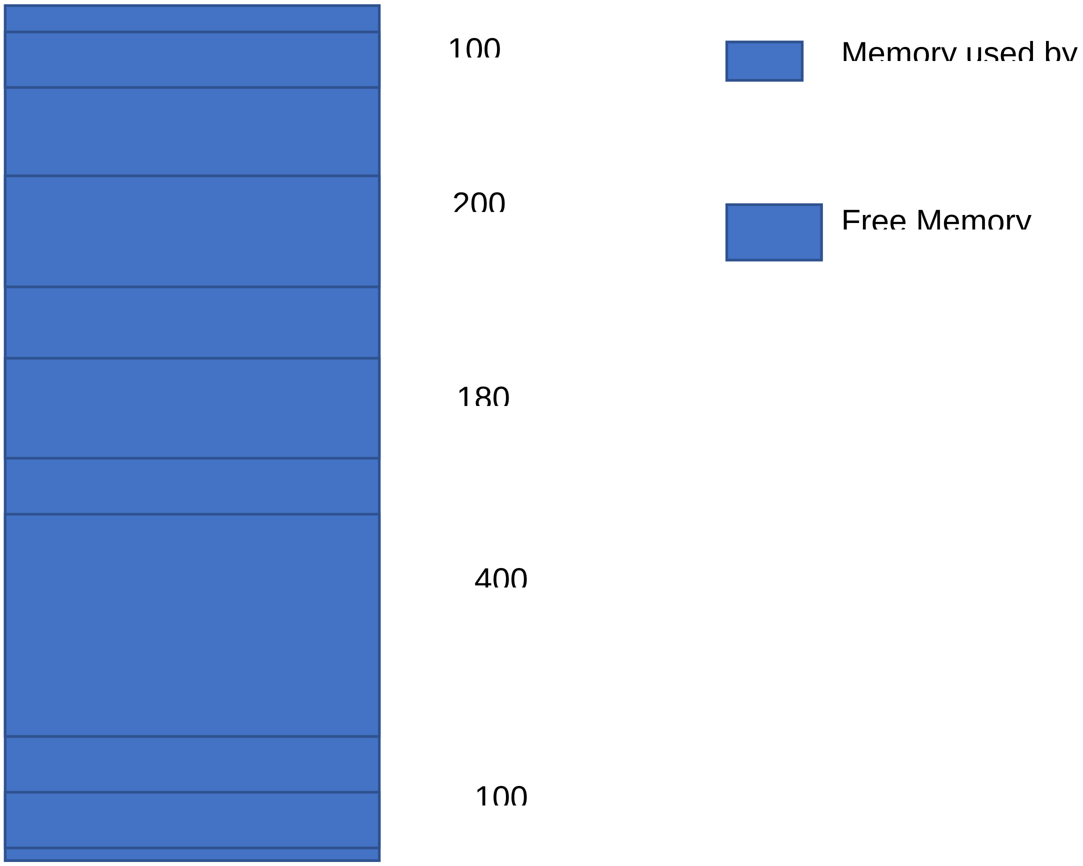
Three strategies can be used to allocate the memory to a process.

1. **First Fit:** According to this strategy, the O/S chooses the very first partition available, whose size is equal to or greater than the size of the process.
2. **Best Fit:** According to this strategy, the O/S chooses a partition that is smallest and whose size is greater than or equal to the size of the process. It may lead to the formation of small-sized unusable holes of memory.
3. **Worst Fit:** According to this strategy, the O/S chooses the largest partition and allocates it to the process. It may result in bigger-sized unusable holes of memory.

From the point of view of the size of the unusable holes, the first fit and best fit are better than the worst fit. From the point of view of time taken for searching the partition, the first fit is better than the other two i.e. Best fit and Worst fit

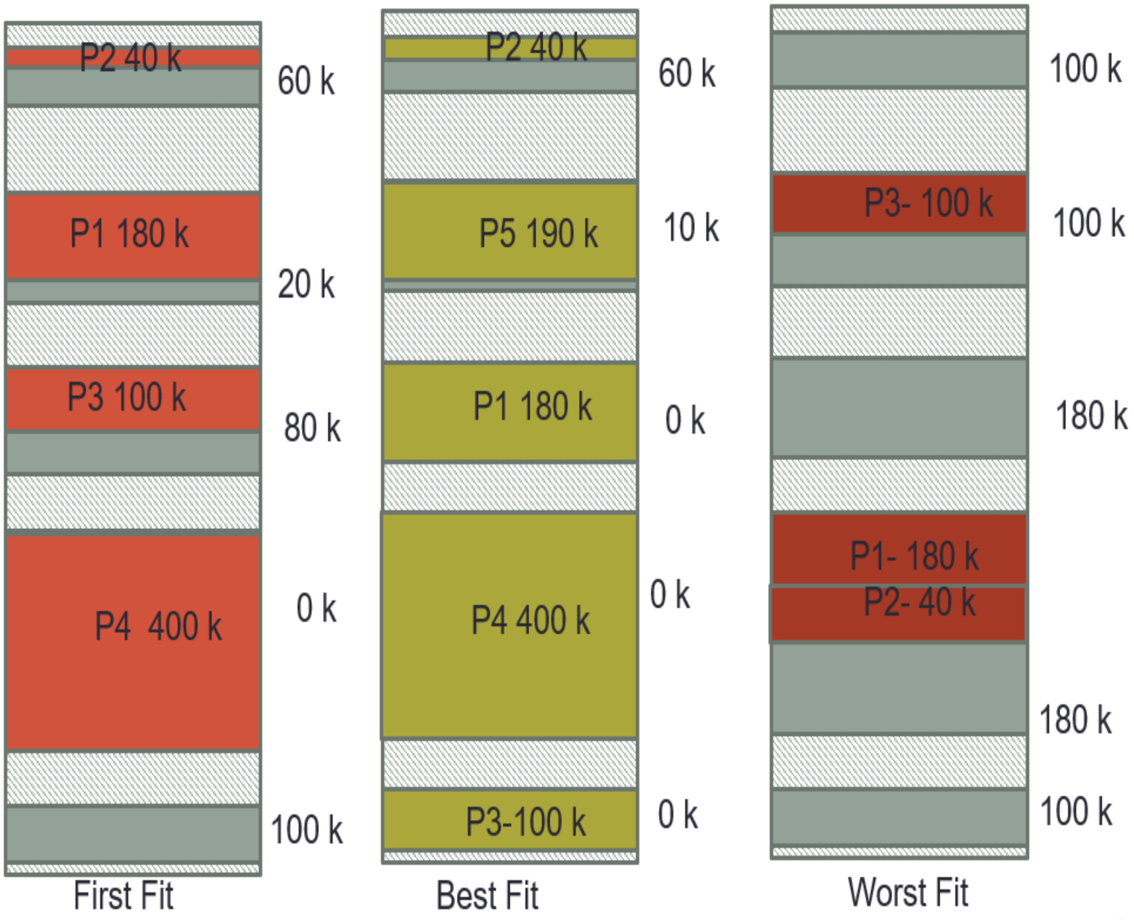
Example of Best Fit

Consider the requests from processes in given order 180K, 40K, 100K,400k, and 190K. Let there be four blocks of memory available of size 100K, 200k, 180K, 400k followed by a block size 100K.



Let there are three process P1, P2, P3, P4, and P5 of sizes 180, 40, 100, 400, and 190 respectively

By using an algorithm of First Fit, Best Fit & Worst Fit is shown by the figure below:



Total free 260 k but process 190k not loaded as memory is not in contiguous

Total free 70 k all process allocated in memory

Total free 660 k but process P4- 400 k and P5- 190k not loaded as memory is not in contiguous

Compaction

Compaction is a technique to deal with the problem of external fragmentation. Compaction means to move the processes in such a way that scattered pieces of unused (free) memory blocks can be placed together so that they can be used by any other process.

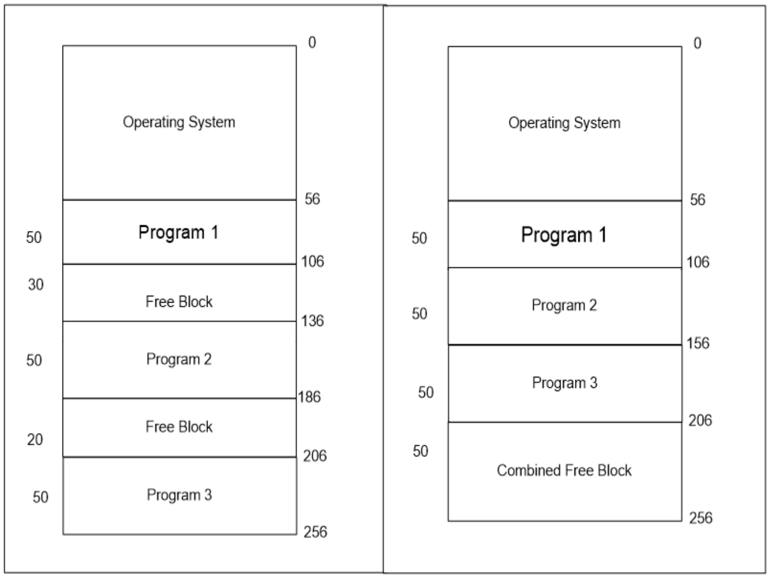


Figure: Before Compaction and After Compaction

Compaction involves the dynamic relocation of a program. This is done by using the relocation register.

However, the compaction algorithm is costly. The problem with the compaction algorithm is that it needs the dynamic relocation of addresses at the execution time of the process. So, the compaction is not possible if the relocation is done statically.

The cost of the compaction increases with the increase in the number of processes to be moved. Moreover, there is a need for a strategy to decide the direction (whether to move upwards or downwards) to move the processes.

Advantages

- Internal fragmentation is very less.
- The degree of multiprogramming can vary.

Disadvantages

- The O/S has to decide about the partition size, almost whenever a program arrives for execution.
- External fragmentation exists.

Memory Protection Contiguous Memory Allocation

When a logical address is generated by the CPU, then it is compared with the content of Limit Register (M). If the logical address is less than equal to M, then it is a valid address else it is an invalid address error, and the process is terminated. If the logical address is valid then the corresponding physical address is computed by adding the content of Relocation Register (B) i.e. base address to the logical address.

Physical Address = Logical Address L + Relocation value R

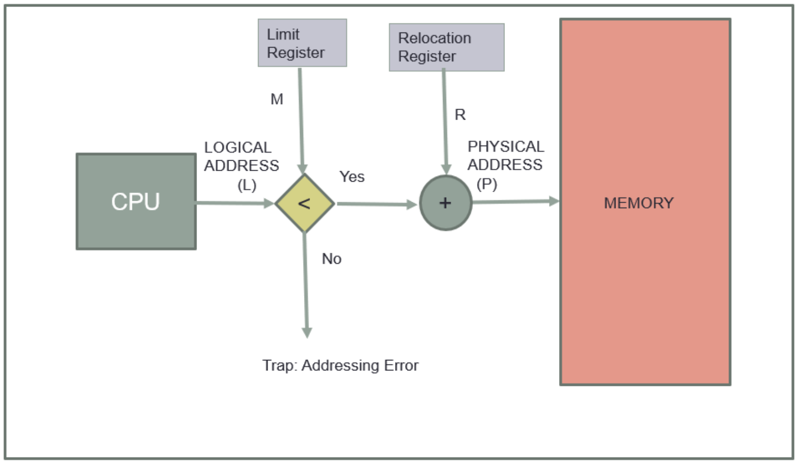


Figure: Memory Protection in Contiguous Memory Allocation

Non-Contiguous Memory Allocation Methods

Non-contiguous memory allocation involves a complex implementation and involves additional costs in terms of memory and processing.

The non-contiguous memory allocation method includes the various strategies like:

1. Paging
2. Segmentation
3. Paging with Segmentation

Paging

Many solutions have been suggested to handle the external fragmentation problem of the memory during the process execution. One scheme permits the logical address space of a process to be noncontiguous, thus allowing a process to be allocating physical memory wherever the latter is available. Memory paging implements this scheme. In the memory paging technique, the logical address space is divided into fixed-sized blocks known as **pages**. The physical address space is also divided into fixed-size blocks known as **frames**. A page is mapped into a frame. Individual pages and frames are recognized by a unique number known as page number and frame number respectively. The size of a page is equal to the size of a frame. A page number forms a part of the logical address and a frame number forms a part of a physical address.

Both page and frame size is usually a power of 2 and depends on the hardware. If a logical address consists of P number of digits, this means there are 2^p addresses in the logical address space. Similarly, if physical addresses consist of q number of digits, this means there are 2^q addresses in the physical address space. **It is always power by 2 because this is to divide logical address into a page number and page offset, easily.**

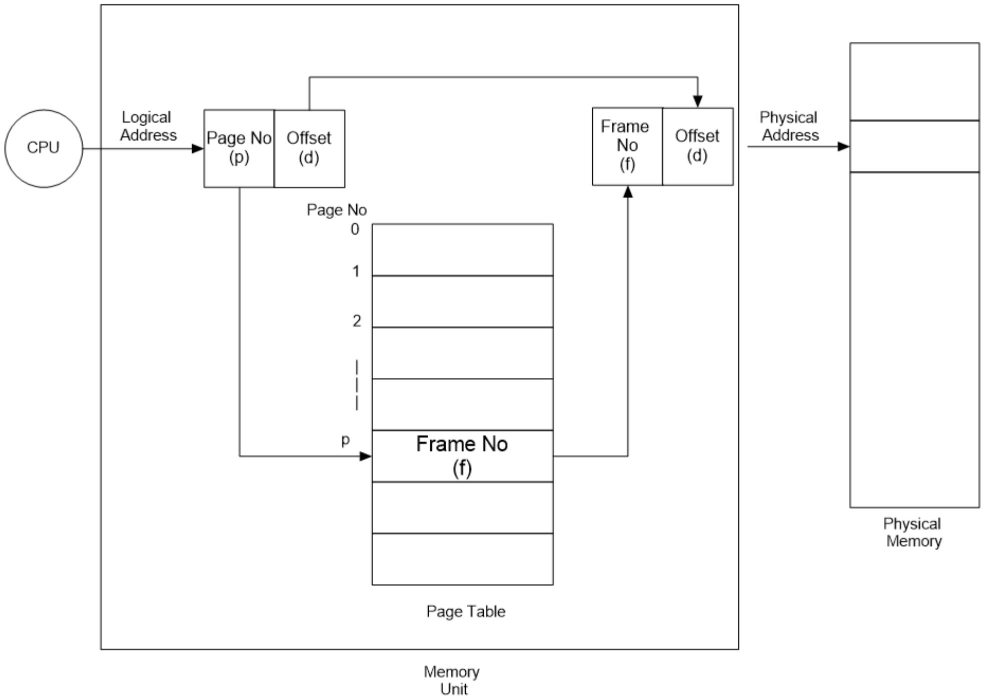


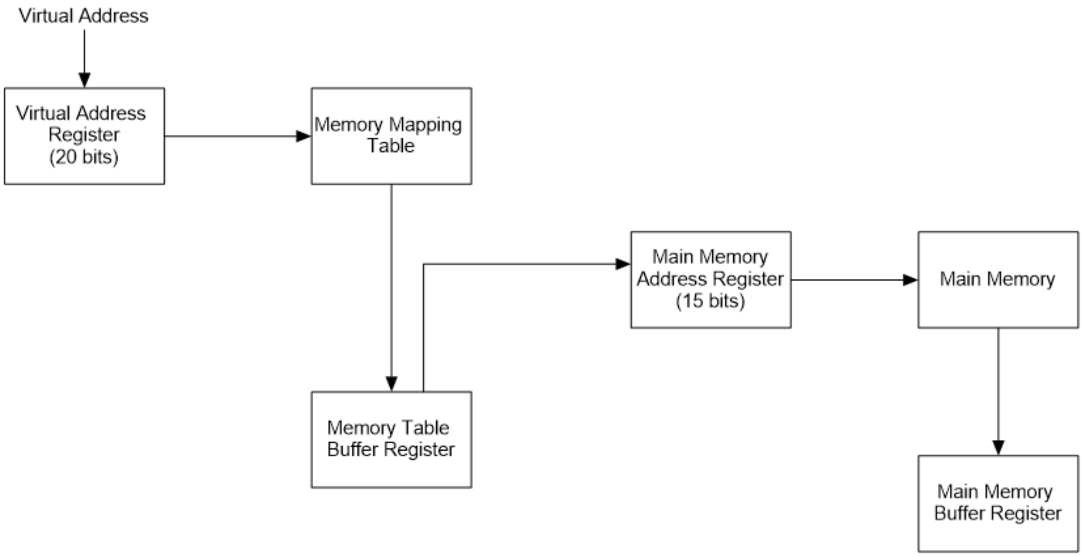
Figure: Paging Hardware

Mapping of Pages to Frame

A page table is maintained for mapping. It divides into two parts: Page Number & Page offset.

p	d
Page No	Offset

A logical address consists of two parts: Page number and Offset. If a logical address is P digits long and page size is 2^n digits long, then the higher or leftmost p-q digits in a logical address denote the page number, and the rest of the logical address denotes offset.


Virtual Address
Virtual Address Register (20 bits)
Memory Mapping Table
Memory Table Buffer Register
Main Memory Address Register (15 bits)
Main Memory
Main Memory Buffer Register

Memory Table for Mapping a Logical Address

For example, the logical address space can hold 2^{13} addresses. This means a logical address is 13 bits long. In other words, the logical address space is 8 KB. There are 8 pages in the logical address space. This means the size of each page is 2^{10} bits=1 KB. The size of the physical address space is 2^{12} bits= 4 KB. This means a physical address is 12 bit long. In the logical address of 13 bits the higher 13 bits (13-10) =3 bits denote the page number and the rest 10 bits denote the offsets. The logical address 011 1010101010 denotes page number 011 and the offset 1010101010.

A physical address consists of two parts: frame number and offset.

f	d
Frame No	Offset

The offset is the same as it is in the corresponding logical address. For example, in the previous example, the offset 1010101010 of the logical address 011 1010101010 is 10 bits long. The physical address corresponding to this logical address can be 01 1010101010, where 01 is the frame number and 1010101010 is the offset.

The operating system maintains a table to convert a logical address into its corresponding physical address. This table is called a **memory page table**. The method of converting a logical address into its corresponding physical address is known as **address translation**.

There are two fields in the page table. The first field stores the corresponding frame number of a page. For example, page 3 is stored in the 13th frame. In the first field of the page table value of the 4th cell will be 13. The second field stores a binary value either 0 or 1 and is known as the **presence bit**. If there exists a frame corresponding to a page, the value of the cell is marked as 1 otherwise the value is set to 0.

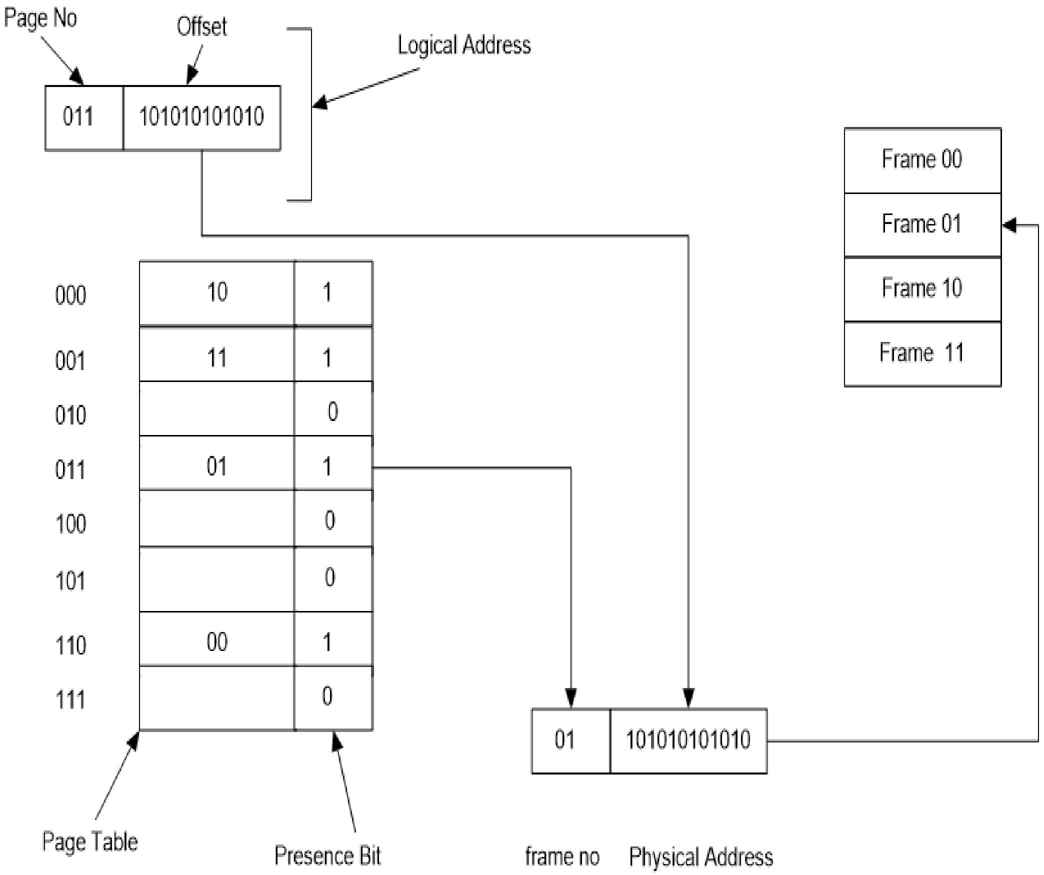
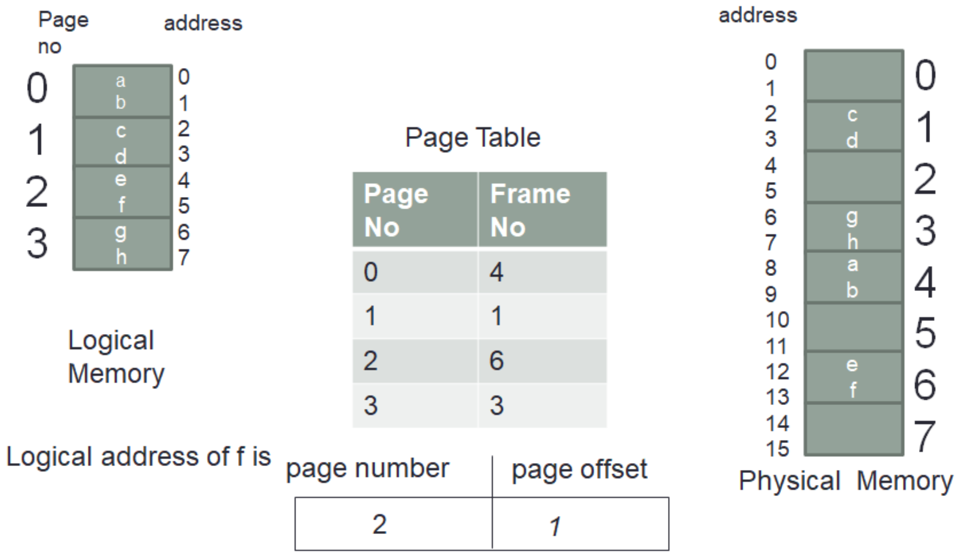


Figure: Mapping

Address Translation in a paging Scheme

The logical address 011 1010101010 is mapped into memory. From the page table, the operating system verifies the presence bit of cell 011, which is 1. This indicates that page 011 has been stored in a frame in the main memory.

Another example Using a page size of 2 bytes and a physical memory of 16 bytes (8 pages), we show how the user's view of memory can be mapped into physical memory.



Page Table

Page No	Frame No
0	4
1	1
2	6
3	3

page number	page offset
2	1

so physical address calculated (frame no x size)+ offset

The physical address of ‘f’ calculated: ‘f’ stores on page no 2 and offset is 1. page no 2 is loaded in frame 6.

So address is (6x2)+1 i.e. 13.

In a paging scheme, an operating system maintains a list of free frames. This is also known as **free frame pool, or chain or list**. Any frame that is not allocated to any process is known as a free frame. Whenever a page is released from memory after a process is terminated or aborted the page is added to the free list. When an operating system allocated pages to a process for storing instruction or data, it assigns a free page from the **free frames list**.

Structure of Page Table

Hierarchical Page Tables

With the increasing need for larger applications, there is also a need for larger memory space in a computer. As a result, it becomes difficult to work with long memory addresses. In paged memory system the page table itself becomes very lengthy. The solution is hierarchical paging.

In this, the page table itself is paged. There are two-page tables: outer and inner.

The inner page table is similar to that of a single-page table scenario. The outer page table stores the links for the inner page table. The logical address consists of three parts: outer page table entry, inner page table entry, and offset.

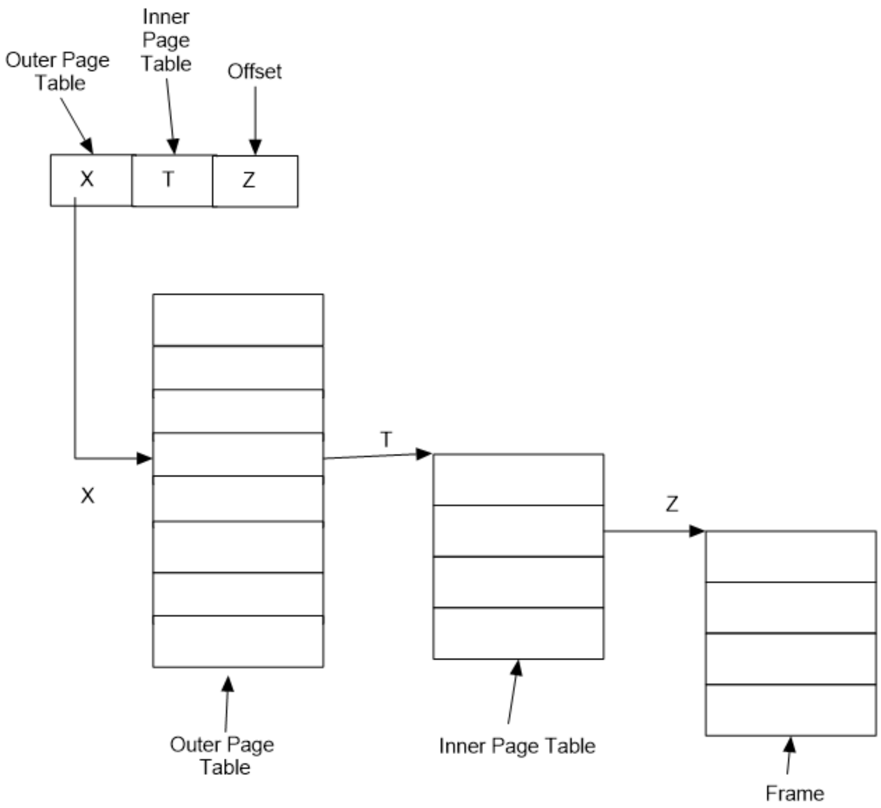


Figure: Hierarchical Page Table

In the process of hierarchical paging, a logical address has been divided into three parts: X, Y, and Z. the symbol X represents the entry in the outer page table, and Y represents the entry in the inner page table, Z represents the offset. The Xth position in the outer page table store the address of the inner page table associated with X. The Y position is searched in that inner page table and Y contains the address of the frame containing the searched page.

Hashed Page Table

This page table is created of length M. Whenever, logical address of generated, a hashing function is applied to the page Number p, to generate an index value i.

$$i = p \% M;$$

The index value i is used to indexing into the page table. Each entry in the page table is a pointer to a link list. It will provide a mapping between page number p and the corresponding frame number f. It accessed through the index value i will be traversed, till a match forms for page number p and the corresponding frame number f is obtained.

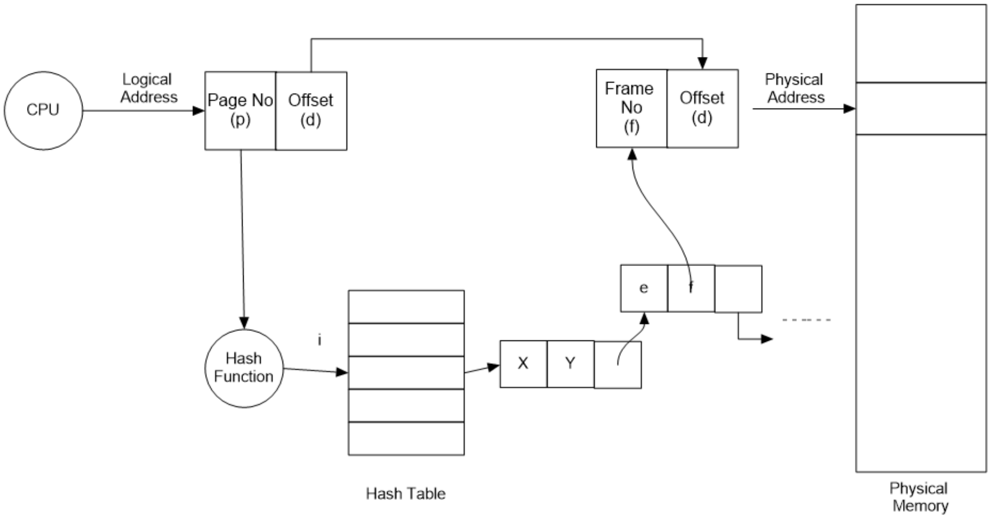


Figure Hashed Page Table

Inverted Page Table (IPT)

An inverted page table is similar to a simple page table along with another entry in the table i.e. process id along with page number. IPT is not process-specific and it does not need switching during content switching. A logical address generated by CPU contains:

Process id (P id),
Page number (p), and
Page offset (d).

Process id & page number is found and corresponding offset (d) which gives the frame number (f), where the desired page is residing. The frame number (f) combined with the offset d, gives the intended physical address.

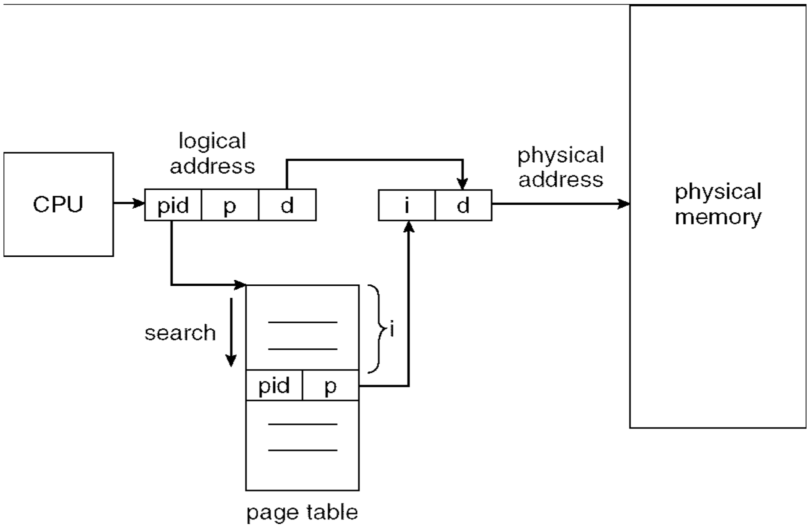


Figure: Inverted Page Table

Translation Look Aside Buffer (TLB)

The page table is implemented in the hardware can enhance the performance substantially. However, it does increase the cost proportionally. Though several schemes of a hardware implementation of the page table exist, in the simplest case, the page table is implemented as a set of dedicated registers. Care must be taken to employ very high-speed logic for these register otherwise the performance will not be up to the mark. The CPU dispatcher reloads these registers, just as it reloads the other registers. Instruction to load is modified the page-table register are, of course, privileged, so that only the operating system can change the memory map. This architecture is used in the DEC PDP-11 computers. It has a 16-bit address while the page size is 8K. As a result, the page consists of 8 entries that are kept in fast registers.

Having a dedicated set of registers for the implementation of the page table has a serious limitation. The register can be used for the purpose only when the page table is reasonably small. Modern computers are capable of having page tables containing as many as 1 million entries. Register implementation in such a system is grossly infeasible. The modern practice is to implement the page into the main memory rather than in the register. The page table is maintained in the main memory and is accessed by a pointer stored in the base register PTBR (Page Table Base Register). One advantage of this practice is that changing the page table requires only changing the register value thus avoiding time-consuming context switching.

This scheme is not without its downside. It suffers from memory access delay. With this approach, a memory location is accessed indirectly. First, the logical memory is indexed into the page table. The page table itself is accessed using the content of the PTBR register which provides the frame number, which is combined with the page offset to produce the actual address.

This approach requires two accesses to memory – one for the page table and one for the index into the page table thereby reducing the CPU time it may become unacceptable. The process can be accelerated using high-speed associative memory or a set of translation look-aside buffer (TLB_S). In associative memory, each register memory each register contains two entries – one for key and another for a value. During a search, the keys are matched with the search value and the corresponding value field is the result if the key was found. The use of associative memory enhances the search speed though at a price for the additional hardware support.

Even associative memory is not unlimited. Therefore, only a few entries of the page table are stored in the associative memory. When the CPU generates a logical address while executing a process the page number of the generated address is searched into the current associative memory for where the frame number is acquired. However, when the page number being searched does not exist in the associative memory the same is searched into the memory and the associative memory is updated with this page number so that it may be found in the next reference top the associative memory.

In this arrangement, a page being searched may not always be found in the associative registers. The extent to which a reference page is found in the associative memory is expressed in terms of hit ratio. Hit ratio is the percentage of times that a page number being searched is found in the memory. For instance, a hit ratio of 60% indicates that out of 100 times a page being searched it is found in the associative memory 60 times. Therefore, one of the performance goals of an operating system is to increase this hit ratio.

The time delay in memory access can be easily estimated in the following manner. Assuming that it takes 10 nanoseconds to search the associative registers, and 50 nanoseconds to access memory, the mapped memory access would take 10 + 50 = 60 ns provided the page number is found in the associative registers.

However, if the search ends in a miss in the associative register after spending 10 ns, then memory access is made for the page table and frame number taking additional 50 ns. Following this, the desired byte is accessed in the memory costing another 50 ns. The total time elapsed comes out to be 50 + 50 + 10=110 ns.

Example: Assuming a hit ratio of 60%, the effective or expected memory-access time can be calculated as shown here under:

Effective access time = (Probability of hit) * Access Time on hit +

(Probability of miss) * Access Time on miss

= 60% of 60 + 40% of 110

= 80 nanosecond

One way of increasing the hit ratio is to use more associative registers. However, as mentioned earlier it can be very costly. Therefore, usually, a trade-off is considered in most cases. One estimate indicates that a hit ratio of 80% to 90% can be achieved with 16 to 12 associative registers.

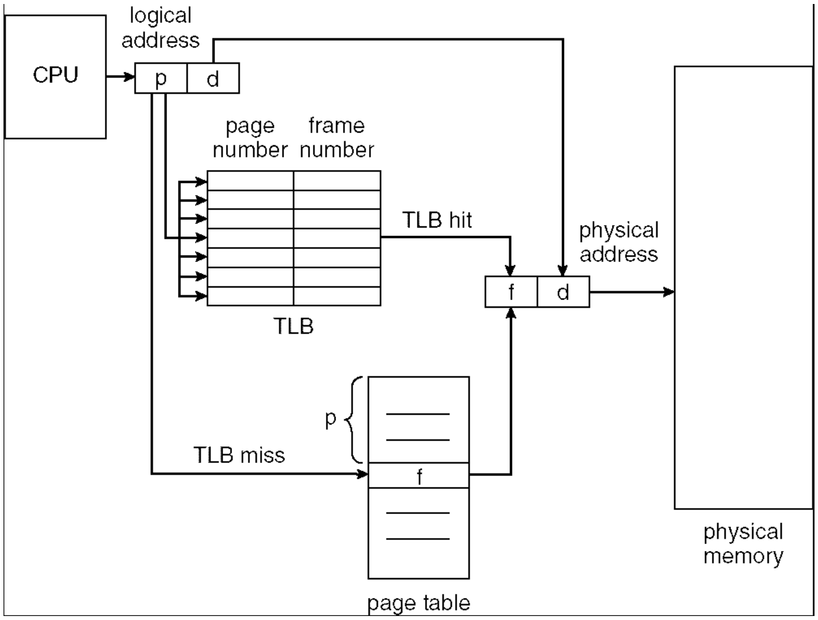


Figure: Paging with TLB

Protection

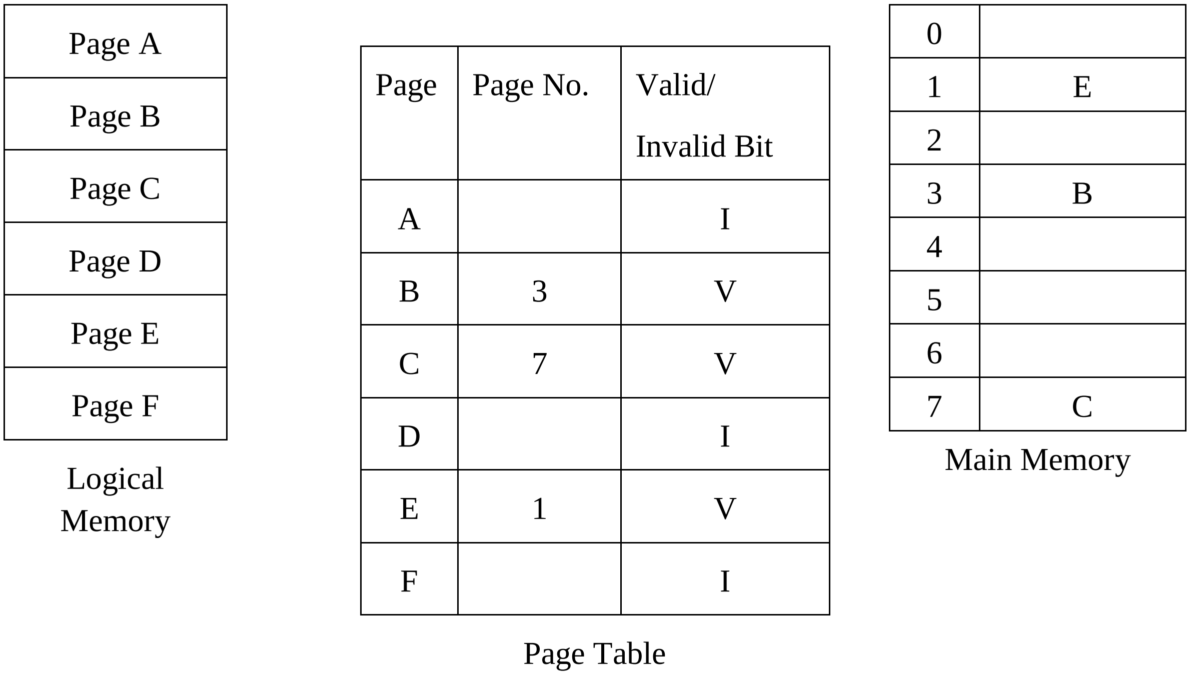
The issue of memory protection in a paging system is handled by attaching a few bits to each of the frames in the page table. The bit may indicate read-only or read-write or execute-only behaviors of the frame. At the time of accessing the frame number from the page table, the corresponding bits are also examined.

Here if a process attempts to write into a read-only frame a system interrupt is generated by the hardware. This interrupt is caught by the operating system, which in turn takes the memory protection violation action.

Notes

In addition to the protection bits, one more bit attached to each frame. This bit determines whether the frame/page is valid or invalid.

The “valid” status of the bit indicates that the associated page in the process’s logical address space, and is thus a legal page. Otherwise, the page is not in the process’s logical address space. Illegal addresses are trapped by the operating system using the valid-invalid bit. The operating system controls the accessibility of a frame by setting this bit.



Logical Memory
Page A
Page B
Page C
Page D
Page E
Page F

Page	Page No.	Valid/ Invalid Bit
A		I
B	3	V
C	7	V
D		I
E	1	V
F		I

Page Table

Main Memory	
0	
1	E
2	
3	B
4	
5	
6	
7	C

Figure: Paging **Protection**

Memory Segmentation

Despite all the obvious merits of a memory-paging scheme of memory management, it does not reflect the way a programmer would like to view the memory. In a paging memory system, there is a clear-cut distinction between the users and system views of memory. The user's view of memory is not the same as the actual physical memory. The user's view is translated onto physical memory by the operating system with the use of special hardware. It is due to this translation that the logical view is different from the physical view of the memory.

A programmer would not like to think that the memory is simply a larger array of bytes wherein at someplace data is stored and at another executable code is stored. The most preferred view of the majority of programmers is that a program is divided into different sections or segments. If you have programmed in COBOL you would appreciate that a COBOL program has different divisions each having one or more optional sections.

Similarly, in a C program, the code is divided between many functions. Your program may have any number of functions, subroutines, data structures, and several modules. Each of these modules or data elements is assigned a unique name by the programmer. Where and how these elements are loaded in the memory is not the concern of the programmer. The programmer does care as to whether the segment containing symbols table is stored in the memory before the main function or after it. Each of these segments must be of variable length. The element of the program should be addressable by the respective names assigned to them.

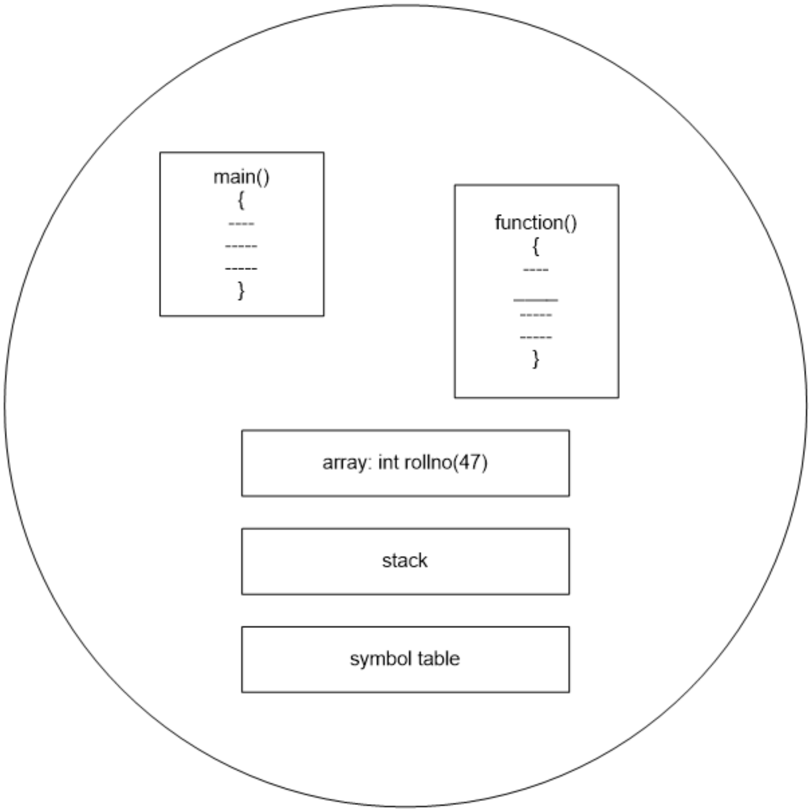


Figure: User's view of a program

An operating system may adopt a memory management scheme to simulate this view of memory.

Segmentation is a technique of non-contiguous memory management technique that employs this scheme. In this scheme, the logical memory space is divided into certain unequal size chunks known as **segments**. According to segmentation, an application is loaded into memory as a collection of modules. In other words, logically related instruction and data items are grouped and loaded into a segment.

For example, an application can consist of five modules: the main program, function A, function B, the stack for storing data when a function is called, and shared data for all modules such as a global variable. Each module is loaded in a distinct segment in the main memory.

Accessing Addresses in Segmentation:

In a segmented memory, a logical address consists of two parts: segment number and offset.

s	d
Segment No	Offset

Figure Address of Segment

An offset is a location within a segment, where any particular data is stored. For example, the logical address 3: 10000 indicates offset 10000 and segment number 3. A segment table is used for the validation of an address. A segment table has two fields: base address of the segment, and size of the segment. The base address of a segment is the location of a particular segment memory and its corresponding segment table:

Segment No	Base Address of the Segment	Size of the Segment
0	80000	6000
1	10000	10000
2	20000	15000
3	40000	35000
4	90000	5000

▨ Free Space

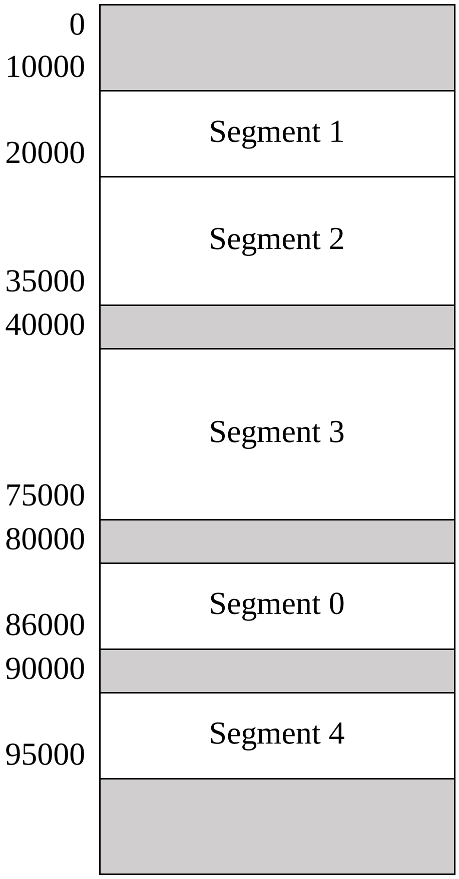


Figure: Segment Table

Segment and Segment Table

The above figure segment table elaborates various memory segments in the main memory and their corresponding entries in the segment table. Segments need not be loaded sequentially or contiguously. For example, segment 2 has been loaded in a lower position in the memory than segment 0.

The starting address and size of a segment are stored in the segment table. For example, the base or starting address of segment 2 is 20000, and is the size of segment 2 is 15000 bytes.

When accessing an address in a segmented memory, the segment table is used for address validation. The logical address 2: 12000 indicates the location 12000 starting from the base of segment 2. Segment 2 starts from location 20000. The size of segment 2 is 15000 bytes. In other words, range of the segment 2 is 20000 to 74FFF. The physical address corresponding to the logical address 2: 12000 is 20000 + 12000=32000. the address 32000 is within the range of segment 2 (i.e. 20000 to 74FFF as 32000<74FFF). Hence, the logical address 2: 12000 is a valid address.

Implementation of Segment Tables

As in the case of the paging system, segmentation can be implemented in a variety of ways. The segment table can reside either in the register or in fast memory. The advantage of register implementation is that both the limits checking and referencing can be done extremely rapidly and simultaneously.

The feasibility of register implementation depends on the number of segments a program has. The larger the number the more register are required making the proposition infeasible. In smaller segmentation register implementation may be recommended. However, modern programs tend to become very large causing a large number of segments. Therefore, in such a system it is more appropriate to maintain the segment table in the memory rather than in the registers. The data structure used for implementation is often a **Segment-Table Base Register or STBR**. This pointer points to the segment table located in the memory. Note that the number of segments used by a program varies widely. Therefore, a Segment-Table Length Register or STLR is used. The address translation for a given logical address (s, d) proceeds as follows:

1. Check that the segment number is legal (i.e. S< STLR)
2. Add the segment number to the STBR(i.e., STBR+S)
3. Read entry from memory at STBR + S
4. Check the offset against the segment length (d< limit)
   (a) if true then compute the physical address of the desired bytes as S+d
   (b) else raise trap to the OS

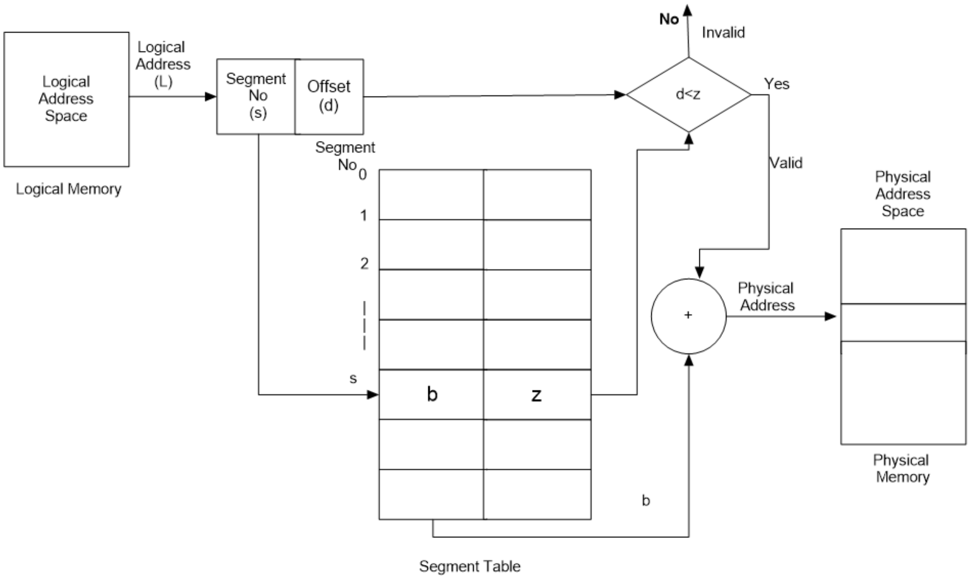


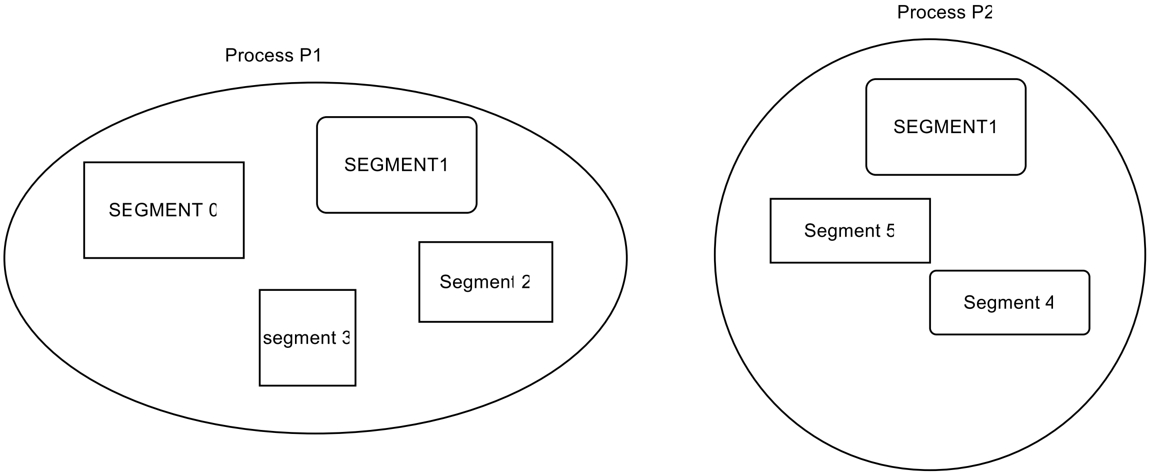
Figure: Segmentation Hardware

Both the schemes- paging and segmentation requires two memory references during address translation. This slows down the OS effectively by a factor of 2. As in the case of paging, a set of an associative registers can be used to speed up the operation. It has been found that a small set of associative registers generally reduces the time required for memory accesses.

Protection and Sharing

Segmentation enforces protection intuitively. Since each segment is a semantically cohesive unit all the corresponding entries are likely used similarly. Therefore, one needs to protect at the segment level using a few protection bits. For instance, since codes do not rewrite themselves the code segment can be assigned a read-only status whereas the data segment can be allowed to perform read and write operations. The memory mapping hardware will check the protection bit associated with each segment table entry to prevent illegal access to memory, such as attempts to write into a read-only segment or to use an execute-only segment as data. For example, by placing an array in its segment, the memory-management hardware will automatically check that array indexes are legal and do not stray outside the array boundaries. This enables the system to take corrective measures early on if a memory protection error does occur.

The main advantage of segmentation is that it can be shared between processes. This is true, particularly with code segments. For each process, a segment table is associated, which the dispatcher uses to define the hardware segment table when this process can share the same code segment as shown in the figure.

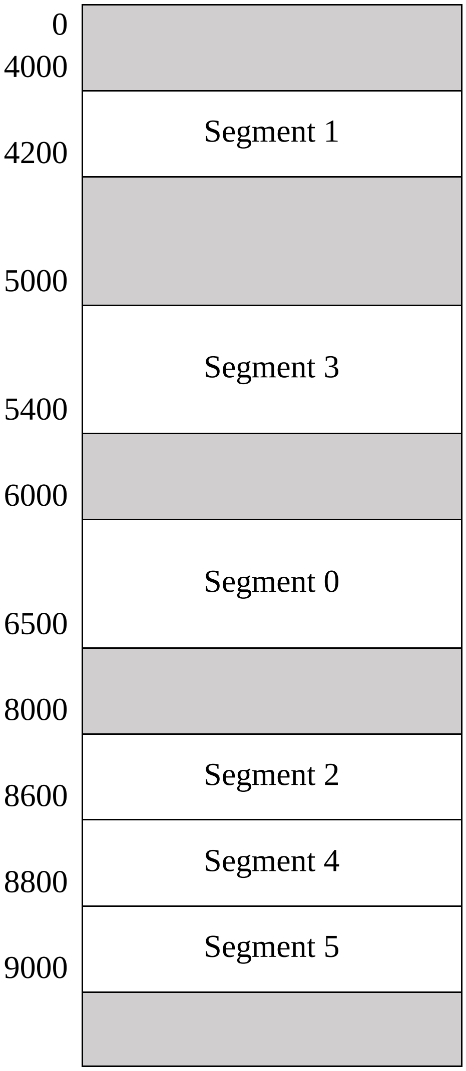


Segment Table P1

Segment No	Base Address of the Segment	Size of the Segment
0	6000	500
1	4000	200
2	8000	600
3	5000	400

Segment Table P2

Segment No	Base Address of the Segment	Size of the Segment
1	4000	200
4	8600	200
5	8800	200



▨ Free Space

Figure: Segmentation Sharing

Here Segment 1 is shared between P1 & P2

Sharing of the segment in a segment memory system

Here in this example, segment (1) is being shared between the two processes p1 and p2. Simply by managing entries in the segment table of processes a segment can be shared among them. The sharing occurs at the segment level. Thus, any information can be shared if it is defined to be segmented. Several segments can be shared in this way.

Segmentation with paging

Segmentation of memory can be implemented with a paging scheme. Unequal size segments can be represented by equal size pages. For example, in a logical address space of 64 KB, there are 32 pages of 2KB each. A program is divided into 3 segments of the size 15 KB, 18 KB, and 10 KB. These three segments require 8, 9, and 5 pages to be loaded into memory. The first segment occupies the memory space of 8 pages of 2 KB each or 16 KB, whereas its requirement is 15 KB memory space. Thus internal fragmentation of 1 KB occurs.

In segmentation with a paging scheme two tables are used: a segment table and a page table. The segment table has two fields: base address and size. The base address of a segment is the address of the first page in the segment. The size of a segment is the number of pages a segment occupies. The page table stores the addresses of the frame where a page is loaded into the main memory. For example, the value of the 15th cell of the page table is 38000. This means page 15 has been loaded into the frame located at the address 38000 in the main memory.

A logical address has three parts: segment number, page number, and offset. The logical address 4:3: 9E indicates the address 9E on page number 3 of segment number 4.

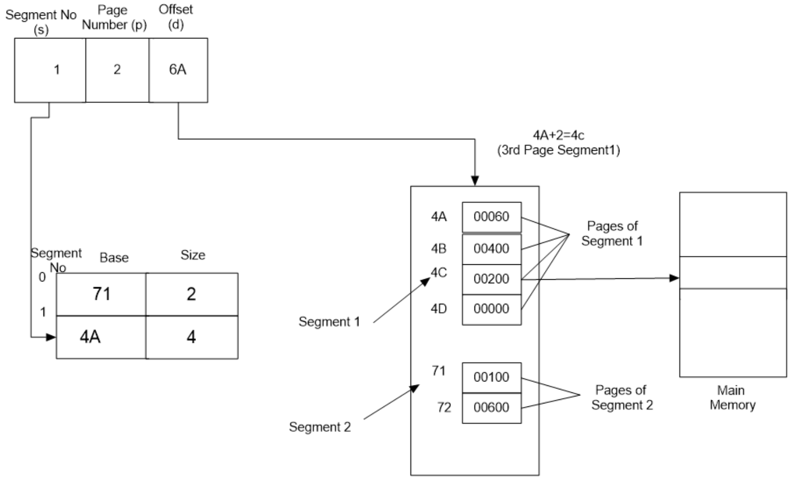


Figure: Segmentation with paging

The main memory is divided into certain equal size frames of 256 bytes (decimal 256 = 100 in hexadecimal) each. The address 6A of page number 2 of segment number 1 is being located. In the segment table, we found that the address of the first page in the page table of segment 1 is 4A. Segment 1 is consists of 4 pages.

From the page table we see that address of the frame corresponding to page 2 has been stored in the location 4C in the page table (because 4a + 2 = 4C). The cell number 4C of the page table is loaded in the frame located at the address 00200 in the main memory.

Shared Pages

In multiprogramming, more than one program can be loaded. It is common for many users to be executing the same program. If individual copies of these programs were given to each user, much of the main memory would be wasted. Its solution is to share those pages that can be shared. One copy of read-only code shared among processes (i.e., text editors, compilers, window systems). Shared code must appear in the same location in the logical address space of all processes. Each process keeps a separate copy of the code and data. The pages for the private code and data can appear anywhere in the logical address space.

Example

Let, there are four processes P1, P2, P3, and P4. P1 has four pages lib1, lib2, lib3, and data1. P2 has four pages lib1, lib4, lib5, and data2. P3 has four pages lib1, lib2, lib5, and data3. P4 has four pages lib1, lib2, lib4, and data4. So, to load all processes 16 frames are required. In all these processes some pages are shared. So, if we share those pages and loaded all shared pages once by giving the same frame number, then it needs only 8 pages as shown in the figure below.

lib1	2	lib1	2
lib 2	6	lib 4	12
lib 3	9	lib 5	0
data 1	4	data 2	7
Process P1	Page Table P1	Process P2	Page Table P2

lib1	2	lib1	2
lib 2	6	lib 2	6
lib 5	0	lib 4	12
data 3	10	data 4	14
Process P3	Page Table P3	Process P4	Page Table P4

0	lib 5
1	
2	lib1
3	
4	data 1
5	
6	lib 2
7	data 2
8	
9	lib 3
10	data 3
11	
12	lib 4
13	
14	data 4
15	

Shared Pages among Processes

Figure: Shared Pages

Demand Paging

The basic concept of virtual memory is storing instructions and data of a program in the secondary memory, and then they are loaded in the main memory. In other words, the part of a program that is required at any instant is loaded in the main memory while the rest of the program is in the secondary memory.

Virtual Memory

Virtual memory is useful especially in a multitasking operating system, where memory available in the main memory for user application is divided among several user programs and every program has to compete for its memory requirement.

Virtual memory is a technique that permits the execution of processes, with their code only partially loaded into physical memory. Virtual memory is used for the separation of logical address space available to the user and the actual physical memory. CPU-generated addresses are known as **logical addresses** or **virtual addresses**.

Programmers use virtual addresses in applications. The MMU converts the virtual addresses into the corresponding physical memory address.

Programmers are notified that they can fully utilize the logical address space. Since the logical address space, programmers have the illusion that they have a larger memory space at their disposal.

Virtual memory is implemented using a non-contiguous memory allocation technique known as demand paging. This technique is similar to paging except that in-demand paging, pages are swapped in and out of main memory.

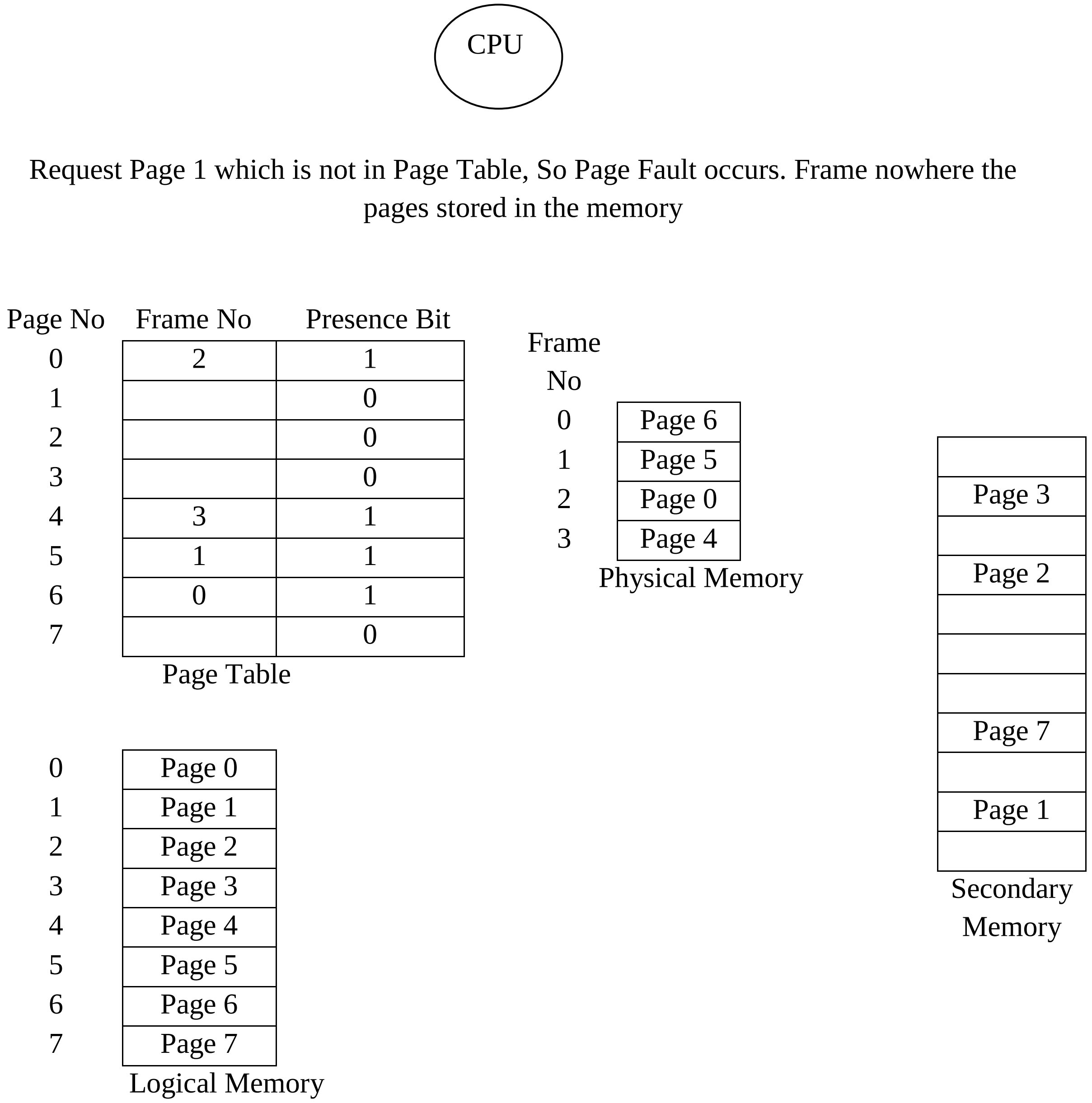
A program is initially stored in the secondary memory. When a page in the program is required, it is swapped into the main memory. This is called demand paging because until a page is not required, it is not loaded.

In demand paging, MMU also maintains an indexed table for address translation, known as a **page table**, similar to that of a simple paging scheme. The page table stores the information about, which page of a program is stored in which frame of the physical memory.

When a page is required, the operating system scans the page table to find out the physical address in the main memory that is in which frame the page is loaded. If the page is found, the operating system continues its processing. In case the page is not present in the main memory, a **page fault** occurs. A page fault is not a fault or an error; rather it indicates a situation that the page is requested by a program that is not currently loaded in the main memory.

When a page fault occurs, MMU swaps in the requested page from the secondary memory into the main memory. In case there is no free space in the main memory, MMU finds free space in the secondary memory, which is also known as the backing store in the context of virtual memory. MMU swaps out an unused page from the main memory and stores the unused page in the free space in the secondary memory for creating space in the main memory. The requested page is loaded into the main memory. Finally, the page table is updated.

Another way is a page table with a valid or invalid bit. The page table includes a valid or invalid bit for each entry. When a page is loaded in memory its frame number is entered and the page validity bit is set to valid. Thus if the bit is set to valid, it indicates that the page is in memory. If the page presence bit is invalid, it indicates that either page does not belong to the logical address space of the process, or it is still not loaded into memory.



Request Page 1 which is not in Page Table, So Page Fault occurs. Frame nowhere the pages stored in the memory

Page No	Frame No	Presence Bit
0	2	1
1		0
2		0
3		0
4	3	1
5	1	1
6	0	1
7		0

Page Table

Frame No	
0	Page 6
1	Page 5
2	Page 0
3	Page 4

Physical Memory

0	Page 0
1	Page 1
2	Page 2
3	Page 3
4	Page 4
5	Page 5
6	Page 6
7	Page 7

Logical Memory

Page 3
Page 2
Page 7
Page 1

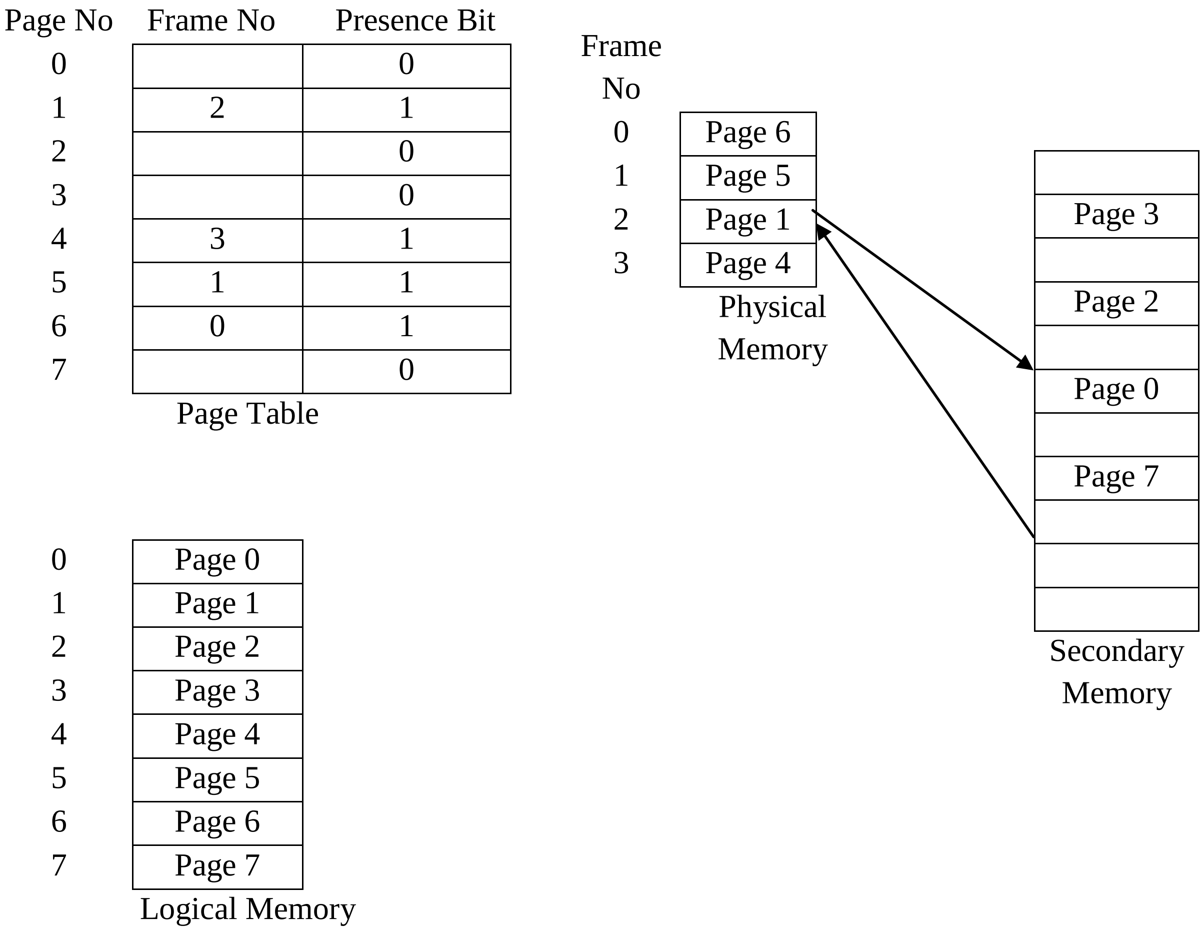
Secondary Memory

Figure: Before Page Replacement

Let, CPU request page 1. This page is not present in the page table because the presence bit of page1 in the page table is 0 that indicates the cell is empty. Thus page fault occurs.

When a page fault occurs, the operating system searches and loads that page in the memory replacing an existing page.

Page 1 caused a page fault. Page 0 is swapped out of the main memory and stored in the disk to create room for page 1 in the main memory. Then page 1 is swapped in the main memory.



Page No	Frame No	Presence Bit
0		0
1	2	1
2		0
3		0
4	3	1
5	1	1
6	0	1
7		0

Figure: After Page Replacement

In the above figure, an abstract and basic idea of the functionality of virtual memory has been shown. Actual implementation varies in different operating systems.

Advantages

- It uses memory more efficiently.

- There is no limit on the degree of multiprogramming.

Steps in Handling a Page Fault

Initially, memory is empty, if there is a reference of a page, the first reference to that page will trap to an operating system which is page faults. To handle the page fault there are the following steps:

1. Operating system looks at another table to decide whether the demand is valid or not in memory. If it is invalid demand then it is aborted, else it is not in memory.
2. After that it searches the empty frame if it is not available then the victim finds and swaps out.
3. Demanded page is swapped into that frame.
4. Page tables are reset which assign the frame number to the corresponding page number.
5. Set the valid-invalid bit to valid bit.
6. Restart the instruction that caused the page fault.

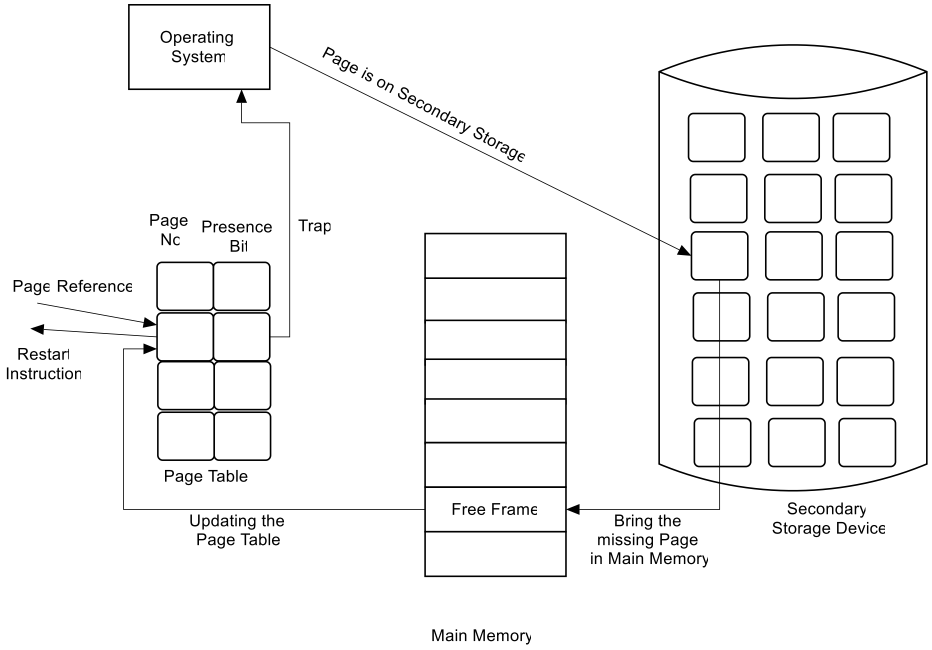


Figure: Steps in Handling Page Fault

Performance of Demand Paging

Demand paging can have a significant effect on the performance of a computer system. Let us calculate effective access time for a demand paged memory. Let P be the probability of a page fault $0 \leq P \leq 1.0$ if $P = 0$ no page faults and if $P = 1$ every reference is a fault.

Effective Access time =

$(1 – p) \times ma + pft$

p = page fault

ma = memory access time

pft = p × (page fault overhead + swap page out + swap page in + restart overhead)

EAT is directly proportional to the page fault rate. If the page fault rate is low then EAT is decreased otherwise.

Example: Let memory access time = 50 nanoseconds

Average page fault service time = 4 ms

Then EAT = $(1 – p) \times ma + pft$

$= 1 – p \times 50 + p \times (4\ ms)$

$= 50 - 50p + 4000000p$

$= 50 + 3999950p$ (nanosecond)

Page Replacement

As the number of processes and the number of pages in the main memory for each process increase, at some point in time, all the page frames become occupied. At this time, if a new page is to be brought in, the OS has to overwrite some existing pages in the memory. The page to be chosen is selected by the page replacement policy.

A **page** is a fixed-length block of memory that is used as a unit of transfer between physical memory and external storage like a disk, and a **page fault** is an interrupt (or exception) to the software raised by the hardware when a program accesses a page that is mapped in address space but not loaded in physical memory. The hardware that detects this situation is the memory management unit in a processor. The exception handling software that handles the page fault is generally part of an operating system. The operating system tries to handle the page fault by making the required page accessible at a location in physical memory or kills the program in case it is illegal to access.

The hardware generates a page fault for page accesses where:

- The page corresponding to the requested address is not loaded in memory.
- The page corresponding to the memory address accessed is loaded, but its present status is not updated in hardware.

Example

Let program A have four pages i.e. P1, P2, P3, P4. Currently, P1, P2, P3 are in memory and it requests page 4 i.e. P4. Program B has four pages i.e. P8, P9, P10, P11.

Currently, P8, P10, P11 are in memory. Only six pages can be stored in memory and currently, P1, P2, P3, P8, P10, P11 are in memory so there is no memory space. To load P4, one page must be swapped out. Let it be P2, P2 store in frame 3 which is swapped out, and P4 is loaded in frame 3 which is currently free. The figure shows memory position before swapping page 4 i.e. P4 and the next figure shows the memory position after swapping page 4.

0	Page 1
1	Page 2
2	Page 3
3	Page 4

Logical Address Space of Program A

Page No	Frame No	Valid/ Invalid Bit
Page 1	4	v
Page 2	3	v
Page 3	6	v
Page 4		i

Page Table of Program A

Operating System	
Page 11	1
Page 8	2
Page 2	3
Page 1	4
Page 10	5
Page 3	6

Secondary Memory

0	Page 8
1	Page 9
2	Page 10
3	Page 11

Logical Address Space of Program B

Page No	Frame No	Valid/ Invalid Bit
Page 8	2	v
Page 9		i
Page 10	5	v
Page 11	1	v

Page Table of Program B

Figure showing the page table, logical address, and main memory before swapping

0	Page 1
1	Page 2
2	Page 3
3	Page 4

Logical Address Space of Program A

Page No	Frame No	Valid/ Invalid Bit
Page 1	4	v
Page 2		i
Page 3	6	v
Page 4	3	v

Page Table of Program A

0	Page 8
1	Page 9
2	Page 10
3	Page 11

Logical Address Space of Program B

Operating System	
Page 11	1
Page 8	2
Page 4	3
Page 1	4
Page 10	5
Page 3	6

Secondary Memory

Page No	Frame No	Valid/ Invalid Bit
Page 8	2	v
Page 9		i
Page 10	5	v
Page 11	1	v

Page Table of Program B

Figure showing the page table, logical address, and main memory after swapping

Page Replacement Algorithms

When a page fault occurs, the operating system has to choose a page to remove from memory to brought a demanded page in memory.

A page replacement algorithm is logic or policy regarding how to select a page to be swapped out from main memory to create space for the page, known as the requested page, which has caused a page fault. These are several page replacement algorithms such as :

- First In First Out (FIFO)
- Optimal Page Replacement Algorithm
- Least Recently Used (LRU)
- Clock Algorithm (Second Chance Algorithm)
- Counting Based Algorithms
- Least Frequently Used (LFU)
- Most Frequently Used (MFU)

First In First Out (FIFO) Algorithm

The FIFO algorithm is the simplest of all the page replacement algorithms. It conveys a basic idea that when a page fault occurs, the oldest page in the main memory is to be swapped out of the main memory to create a room or the required page that needs to be executed. It replaces the page that has been in the memory longest.

> **The oldest page in the main memory is one that should be selected for replacement first. (if the number of frames is 3 then page repeated 3 times will replace)**

For example, let there are four pages i.e. 14, 21, 18, 36. Pages number 14, 21, and 18 are present in the memory and page number 21 was loaded first, followed by 14 and then 18. If page number 36 is required to be accessed by the CPU, it results in a page fault because page number 36 is not present in the main memory. The page loaded first, that is page number 21 is swapped out of the main memory and stored in the secondary memory to create room for page number 36. Then page 36 is loaded in the memory. One possible implementation is a FIFO queue of existing pages in memory. The oldest page will be at the FRONT of the queue. Whenever a page fault occurs, the page at the FRONT of the queue is mode victim and the new page is put at the REAR of the queue.

Advantages

It is very simple and easy to implement.

Programmers can easily code this algorithm.

Disadvantages

This algorithm has a severe drawback. If the oldest page is accessed frequently, the performance of this algorithm declines, since whenever the page swapped out will be required, it will result in another page fault, which degrades the performance.

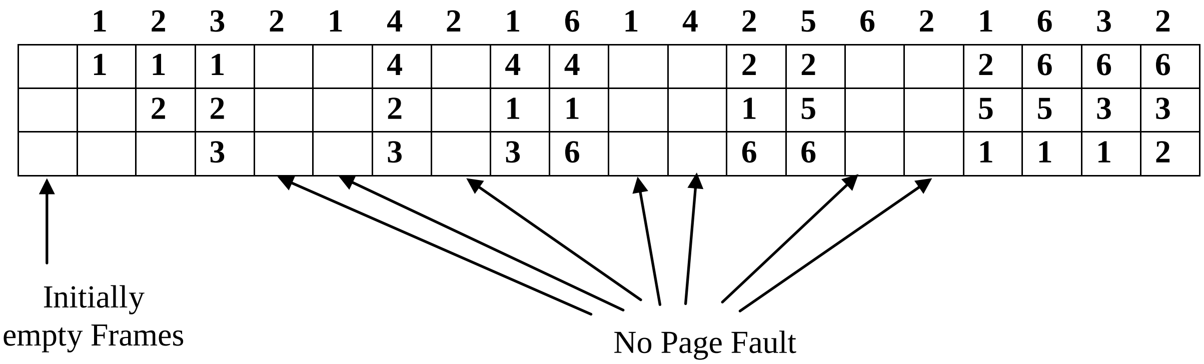
Reference String

A reference string refers to the sequence of page numbers referenced by a program during its execution.

Assume a reference string:

1, 2, 3, 2,1,4, 2,1, 6, 1, 4, 2,5, 6, 2,1, 6, 3, 2

Let frames 3 are available for allocation page frame. Find number of page faults.



	1	**2**	**3**	**2**	**1**	**4**	**2**	**1**	**6**	**1**	**4**	**2**	**5**	**6**	**2**	**1**	**6**	**3**	**2**
	1	**1**	**1**			**4**		**4**	**4**			**2**	**2**			**2**	**6**	**6**	**6**
		2	**2**			**2**		**1**	**1**			**1**	**5**			**5**	**5**	**3**	**3**
			3			**3**		**3**	**6**			**6**	**6**			**1**	**1**	**1**	**2**

Initially, all 3 slots are empty, so when 1, 2,3 came they are allocated to the empty slots —> **3 Page Faults.**

Next page 2 refer, it is already in memory so —> **0 Page Faults.**

next page 1 refers, it is already in memory so —> **0 Page Faults.**

next page 4 refer, it is not available in memory and no free frame so it replaces the oldest page slot i.e 1. —>**4th Page Fault**

Next page 2 refer, it is already in memory so —> **0 Page Faults.**

Next page 1 refers, it is not available in memory and no free frame so it replaces the oldest page slot i.e 2. —>**5th Page Fault**

next page 6 refer, it is not available in memory and no free frame so it replaces the oldest page slot i.e 3. —>**6th Page Fault**

Next page 1 refers, it is already in memory so —> **0 Page Faults.**

Next page 4 refer, it is already in memory so —> **0 Page Faults.**

next page 2 refer, it is not available in memory and no free frame so it replaces the oldest page slot i.e. 4 —>**7th Page Fault**

next page 5 refer, it is not available in memory and no free frame so it replaces the oldest page slot i.e. 1 —>**8th Page Fault**

Next page 6 refer, it is already in memory so —> **0 Page Faults.**

Next page 2 refer, it is already in memory so —> **0 Page Faults.**

next page 1 refers, it is not available in memory and no free frame so it replaces the oldest page slot i.e. 6 —>**9th Page Fault**

next page 6 refer, it is not available in memory and no free frame so it replaces the oldest page slot i.e. 2 —>**10th Page Fault**

next page 3 refer, it is not available in memory and no free frame so it replaces the oldest page slot i.e. 5 —>**11th Page Fault**

next page 2 refer, it is not available in memory and no free frame so it replaces the oldest page slot i.e. 1 —>**12th Page Fault**

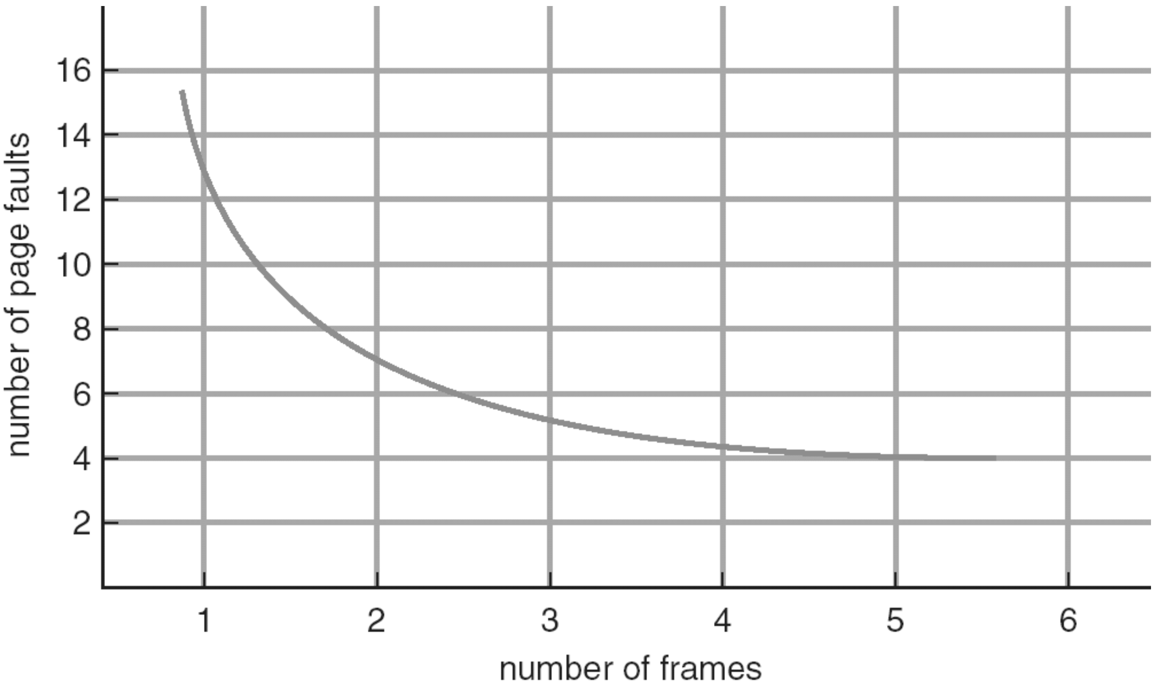
The number of page faults in FIFO is 12

Let frames 4 are available for allocation:

	1	**2**	**3**	**2**	**1**	**4**	**2**	**1**	**6**	**1**	**4**	**2**	**5**	**6**	**2**	**1**	**6**	**3**	**2**
	1	**1**	**1**			**1**			**6**	**6**		**6**	**6**					**3**	
		2	**2**			**2**			**2**	**1**		**1**	**1**					**1**	
			3			**3**			**3**	**3**		**2**	**2**					**2**	
						4			**4**	**4**		**4**	**5**					**5**	

Page fault in FIFO with 4 frames is 9.

By observing these two examples it states that if the number of frames will increase then the number of page fault must be decrease



Graph of Page Faults Versus The Number of Frames

The above graph reveals that if the number of frames increases then the number of page fault decreases but Belady’s anomaly proves that if the number of frames increases then it may be increased in page fault also while using the First in First Out (FIFO) page replacement algorithm.

Let string is 5, 6, 7, 8, 5, 6, 9, 5, 6, 7, 8, 9, 5, and frameset is 3.

	5	**6**	**7**	**8**	**5**	**6**	**9**	**5**	**6**	**7**	**8**	**9**	**5**
	5	**5**	**5**	**8**	**8**	**8**	**9**			**9**	**9**		**5**
		6	**6**	**6**	**5**	**5**	**5**			**7**	**7**		**7**
			7	**7**	**7**	**6**	**6**			**6**	**8**		**8**

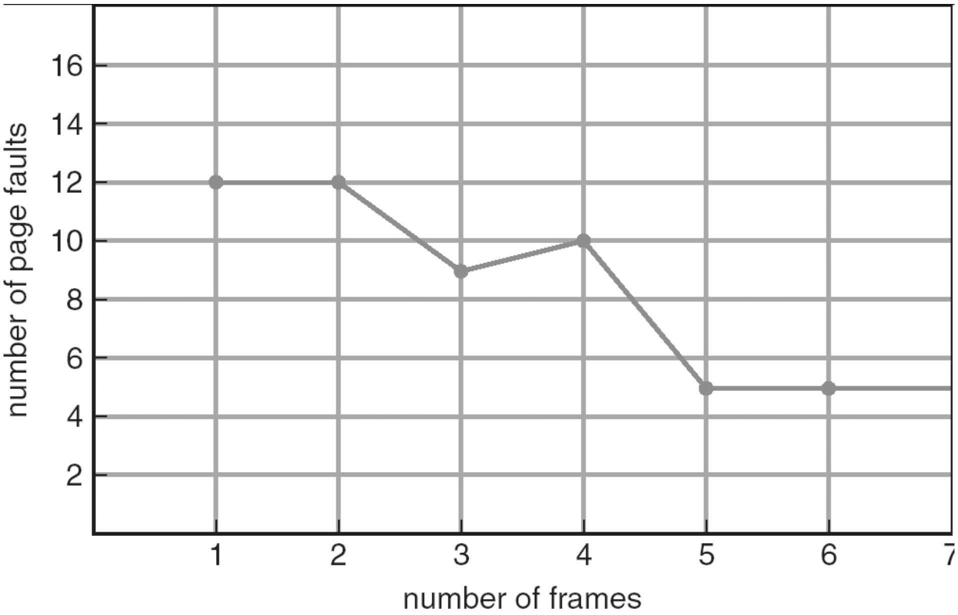
There are 10-page faults

Now with 4 frames set

	5	**6**	**7**	**8**	**5**	**6**	**9**	**5**	**6**	**7**	**8**	**9**	**5**
	5	**5**	**5**	**5**			**9**	**9**	**9**	**9**	**8**	**8**	**8**
		6	**6**	**6**			**6**	**5**	**5**	**5**	**5**	**9**	**9**
			7	**7**			**7**	**7**	**6**	**6**	**6**	**6**	**5**
				8			**8**	**8**	**8**	**7**	**7**	**7**	**7**

there are 11-page faults.

So by the increasing number of frames, it may be the increase of the number of page faults like in the above example where with 3 frames set there are 10-page faults and with 4 framesets there are 11-page faults. This is the Belady’s Anomaly.



FIFO Illustrating Belady’s Anomaly

The Optimal (OPT) Page Replacement Algorithm

Optimal page replacement algorithm is considered the best possible page replacement policy in a virtual memory theoretically but it is difficult to implement. According to the optimal page replacement policy, the page in the main memory, which will not be referred to for the longest time is swapped out from the main memory to create room for the requested page. A page should be replaced, which is to be referenced in the most distant future. Since, it requires knowledge of the future reference string, which is not practical. In the optional page replacement algorithm, the number of page faults is minimum as compared to another algorithm.

Practically, implementation of the optimal page replacement algorithm is done as follows:

Usually, all pages are labeled with the number of instructions that will be executed before this page will be used again in the future. When a page fault occurs the page with the highest number is replaced with the requested page that has caused a page fault.

For example, let pages 14, 21, and 18 are present in the main memory. Page 82 is required to be loaded resulting in a page fault since page 82 is not there in the memory at present. Page 14 is not required till the next 2000 instruction. Page 21 is not required till the next 1500 instructions. Page 18 is not required till the next 1700 instructions. This means page 14 is the one that will not be accessed by the CPU for the longest time. Page 14 is swapped out of the main memory to create room for page 82.

Advantages

It has the lowest rate of occurrence of page faults.

It improves the system performance by reducing overhead for several page faults and swapping pages in and out when a page fault occurs.

Disadvantages

It is very difficult to implement.

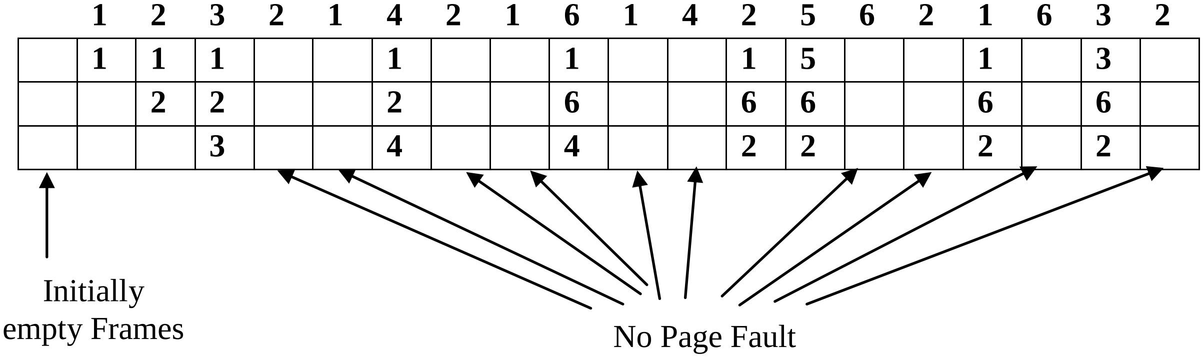
The situation is very similar to that of implementing the SJF algorithm in process management. It becomes very difficult for an operating system to calculate after what interval a page is to be referred to.

Example:

Let a reference string as follow:

1, 2, 3, 2,1,4, 2,1, 6, 1, 4, 2,5, 6, 2,1, 6, 3, 2

Let frames 3 are available for allocation page frame. Find number of page faults.



	1	2	3	2	1	4	2	1	6	1	4	2	5	6	2	1	6	3	2
	1	1	1			1			1			1	5			1		3	
		2	2			2			6			6	6			6		6	
			3			4			4			2	2			2		2	

There is a 9-page fault in the optional page replacement algorithm but in FIFO there are 12-page faults.

Least Recently Used (LRU)

The LRU algorithm uses information about the pages accessed in the recent past to predict the near future. The LRU algorithm is when a page fault occurs, the page that has not been referred to for the longest time is swapped out of the main memory to create space for the requested page that has caused the page fault. It replaces the page which has been used least recently.

Implementation of LRU can be done in various ways. One of the common methods to apply the LRU in a scheme for virtual management is using an array. The array stores the information about the page present in the main memory. The front end of the array stores the page accessed recently. The rear end of the array stores the page that has not been accessed for the longest time.

Whenever a page, that is present in the main memory, is accessed, the information about the page in the array is shifted to the front end of the array. If a page fault occurs the page indicated by the rear end of the array is swapped out of the main memory and the requested page is swapped in the main memory. Information about the page swapped in is stored in the front end of the array.

For example, the array stores four-page numbers {12, 56, 27, 61}. Page 12 is at the front end of the array. This indicates that page 12 has been accessed recently. Page 61 is at the rear end of the array. This indicates that the page that has not been referenced for the longest time is page 61. Page 27 is accessed. No page fault occurs because page 27 is present as the main memory. Information about page 27 is shifted at the front end of the array. The array becomes { 27, 12, 56, 61}. Page 43 is required to be accessed. A page fault occurs since page 43 is not in the main memory. Page 61 is at the rear end of the array, which indicates that it has not been accessed for the longest time. Page 61 is swapped out of the main memory and information about page 61 is removed from the array. Page 43 is swapped in the main memory and information about page 43 is inserted at the front end of the array. The array becomes { 43, 27, 12, 56}.

Advantages

It is very feasible to implement.

This is not as simple as the FIFO algorithm but not as complicated to implement as the optimal page replacement algorithm.

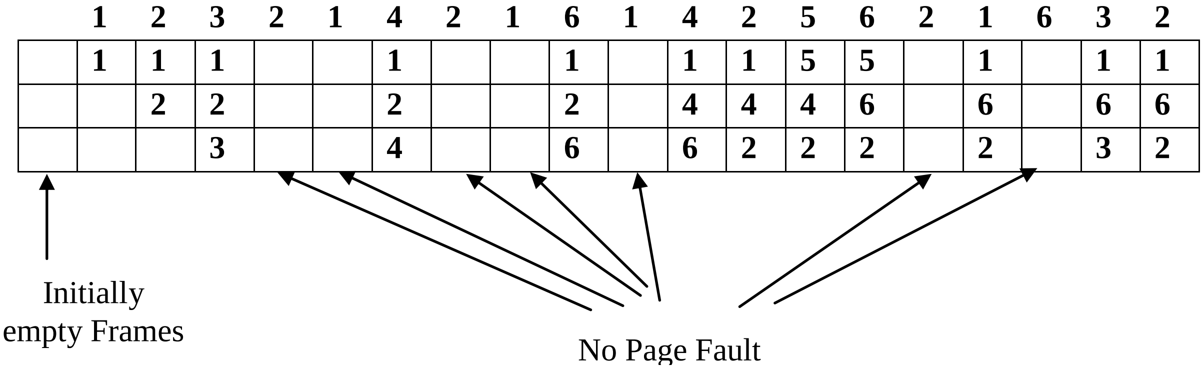
Disadvantage

The LRU algorithm requires additional data structure and hardware support for its implementation.

Let a reference string as follow:

1, 2, 3, 2,1,4, 2,1, 6, 1, 4, 2,5, 6, 2,1, 6, 3, 2

Let frames 3 are available for allocation page frame. Find number of page faults.



	1	**2**	**3**	**2**	**1**	**4**	**2**	**1**	**6**	**1**	**4**	**2**	**5**	**6**	**2**	**1**	**6**	**3**	**2**
	1	**1**	**1**			**1**			**1**		**1**	**1**	**5**	**5**		**1**		**1**	**1**
		2	**2**			**2**			**2**		**4**	**4**	**4**	**6**		**6**		**6**	**6**
			3			**4**			**6**		**6**	**2**	**2**	**2**		**2**		**3**	**2**

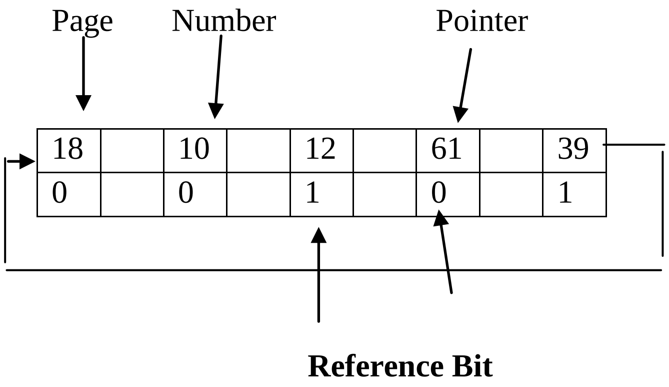
Page fault in LRU is 12. In this case, it is higher than optimal because a page that swap out is needed after one page so, it increases the page fault but in normal case page fault in LRU is less than FIFO but greater than the optional page replacement algorithm.

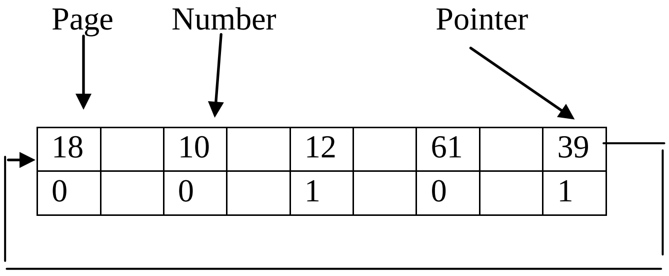
Clock algorithm

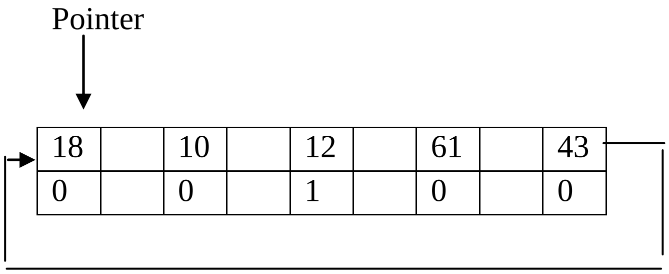
The clock algorithm also known as the second chance algorithm is a variation of the FIFO algorithm. In this, the page present for the longest time in the memory is given a second chance to remain loaded in the main memory. When that page is encountered for the second time, is swapped out to create room for the page that has caused a page fault. It is also referred to as **Not Recently Used (NRU)**. It replaces a resident page, which has not been accessed in the near past.

Working

This is implemented using the concept of a circular queue. Each cell in the circular queue contains two values: a page number and its corresponding **reference bit**. The value of the reference bit can be either 0 or 1. If the value of a reference bit of a page is 1 it means that the page was encountered as the oldest page and followed a second chance. If the value of the reference bit is 0 this indicates that this page has not been encountered as the oldest page yet. When a page is found the oldest page present in the memory for the first time, its reference bit is set from 0 to 1. The next time when that page is found, it is swapped out of the main memory for creating free space in the main memory.







Page 43 swapped in from the secondary memory replacing page 39

A circular chain or pull is organized to implement the clock algorithm for selecting a page to be swapped out in a virtual memory scheme. The pointer is directing presently at page number 61. a page fault occurs. Page 61 has not been given a chance to remain loaded in the memory since its references bit is 0. The reference bit of page 61 is made 1 and the pointer is moved to the next page in the pull that is page 39.

The reference bit on page 39 is 1. This indicates that previously page 39 has been given a chance to be remain loaded in the memory. Thus this page is swapped out of the memory to create space for the new to swapped on page 43.

Counting Based Algorithms

Some page replacement algorithms apply the logic based on how many times a page has been accessed. Usually, these types of algorithms are implemented using an array that stores information about a page number and the number of times it has been accessed. The type of counting algorithms is:

Least Frequently Used (LFU) page replacement algorithm

Most Frequently Used (MFU) page replacement algorithm

When a page fault occurs, the operating system verifies how many times it has been accessed and proceeds according to the logic provided by the counting algorithm adopted for page replacement in the virtual memory by the operating system.

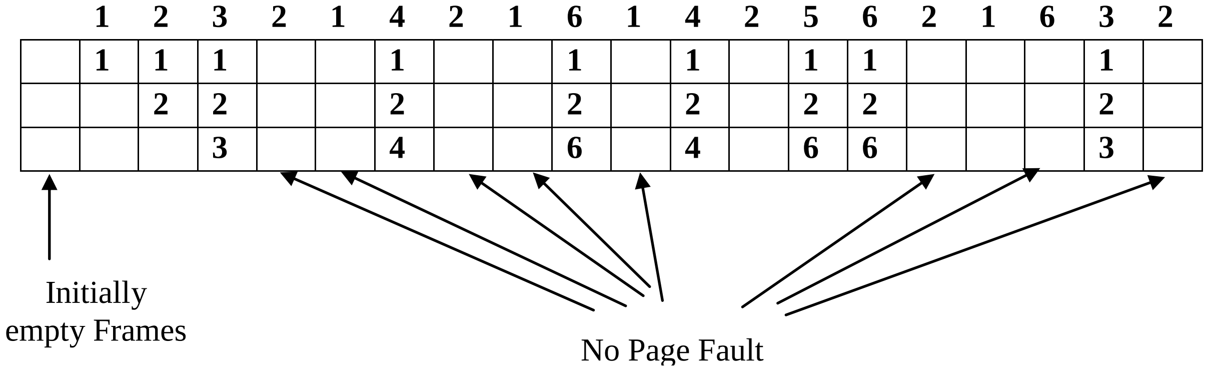
Least Frequently Used (LFU)

In the LFU algorithm, the page that has been accessed for the lowest/fewest number of times from the time when the page is loaded in the memory is replaced when a page fault occurs. The logic behind applying this algorithm is that some pages are accessed more frequently than others. Counting how many times a page has been accessed is used as an estimate of the probability of a page being referenced. Whenever a replacement is necessary, a page with the least count is replaced. The main drawback of this algorithm is that some pages may have a high usage initially and may build a high count but they have low usage subsequently, would remain in memory due to high count.

Let a reference string as follow:

1, 2, 3, 2,1,4, 2,1, 6, 1, 4, 2,5, 6, 2,1, 6, 3, 2

Let frames 3 are available for allocation page frame. Find number of page faults.



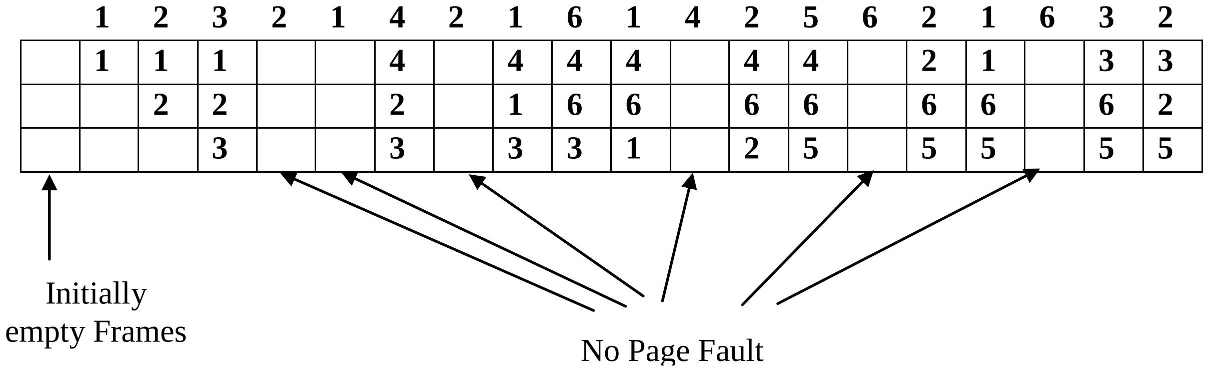
	1	2	3	2	1	4	2	1	6	1	4	2	5	6	2	1	6	3	2
	1	1	1			1			1		1		1	1				1	
		2	2			2			2		2		2	2				2	
			3			4			6		4		6	6				3	

There are 9-page faults is LFU

Note: If the counter of all the pages is the same then we adopt FIFO.

Most Frequently Used (MFU)

This algorithm replaces the page with the largest usage count. It is based on the assumption that the pages, with a smaller count, have been brought in recently and would need to be resident.



	1	2	3	2	1	4	2	1	6	1	4	2	5	6	2	1	6	3	2
	1	1	1			4		4	4	4		4	4		2	1		3	3
		2	2			2		1	6	6		6	6		6	6		6	2
			3			3		3	3	1		2	5		5	5		5	5

There are 13-page fault is MFU

Note: If the counter of all the pages is the same then we adopt FIFO.

Thrashing

In certain situations, a system spends more time processing page fault by swapping in and out pages than executing an instruction of processes. In other words, sometimes handling the pages faults becomes a huge overhead. This situation is called **thrashing**. Thrashing degrades the performance of a system. When multiprogramming increase more which degrades the performance is known as thrashing.

In a situation, where too many pages are active and required frequently, a page is required that is not present in the memory resulting in a page fault. One of the pages loaded in the main memory, which is also accessed frequently, is swapped out to create room for the required page. The required page is swapped in; the page swapped out is required for execution causing another page fault. Thus, a series of pages fault occurs and swapping becomes a large overhead.

In case a large number of processes are running in a multitasking operating system simultaneously, the memory becomes over-committed and thrashing occurs. In another word, the degree of multiprogramming is directly related to thrashing. The term, degree of multiprogramming, indicates many processes that are being simultaneous. This figure shows how system performance declines because of thrashing:

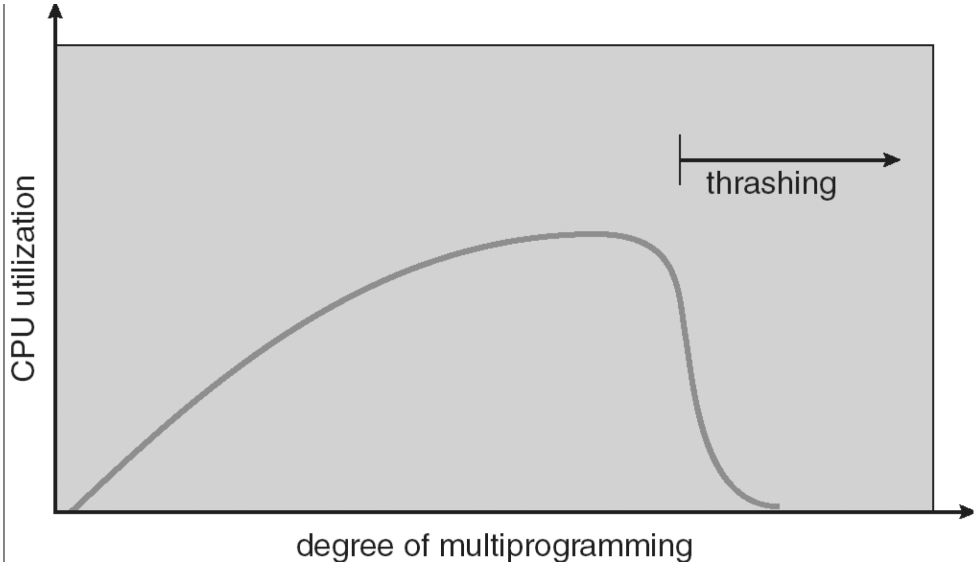


Figure: Effect of Thrashing on System Performance

This figure shows that system performance in terms of throughput reaches an optimal level to an extent if the degree of multiprogramming is increased. If the degree of multiprogramming is further extended thrashing occurs and system performance degrades. The peek of the graph represents the optimal performance of the system.

In certain cases, thrashing can be avoided. In most cases, an operating system attempt to recover from thrashing by suspending the execution of current processes and preventing the execution of new processes to start.

Selecting an appropriate page replacement policy to operate virtual memory plays an important role in the paging of thrashing. A page replacement strategy based on the local mode prevents the occurrence of thrashing to an extent. In local mode, while running a process, When a page fault occurs, swapping out of a page, which belongs to another process, is not allowed. A page belonging to the same running process is selected and swapped out to create room for the required page. In such cases, the thrashing of a particular process does not affect other processes. If other processes have occupied enough frames in the main memory, they can continue execution properly.

A page replacement policy based on the global model can lead the system to thrash. In global mode, when a page fault occurs, any page, regardless of which process it belongs to, is swapped out to create room for the required process that needs immediate execution. In global mode, a process may occupy a huge part of the main memory while other processes are competing for room in the main memory and hence resulting in page fault frequently.

Points to Remember

- The logical address is generated by the CPU.
- The logical address is also known as the virtual address.
- Logical and Physical addresses are different for execution time address binding.
- The base register is also called the relocation register.
- The user program deals with the logical address.
- Swapping is a technique to temporarily removing the inactive program from the memory of a system.
- Memory allocation methods are: first fit, Best fit, Worst fit.
- Paging is a memory management method.
- Logical memory divided it into similar size called pages.
- Physical memory divided it into similar size called frames.
- Paging eliminates fragmentation.
- In segmentation, the program is divided into variable size segments.
- Paging and segmentation can be shared.
- Virtual memory allows the execution of partially loaded processes.
- Virtual memory is the separation of user logical memory from physical memory.
- Demand paging is similar to a paging system with swapping.
- With the use of virtual memory, CPU utilization can be increased.
- In demand paging, pages are swapped in and out of the main memory.
- In demand paging, MMU also maintains an indexed table for address translation, known as a page table.
- This table is similar to that of a simple paging scheme.
- When the page is not present in the main memory a page fault occurs.
- Demand paging uses memory more efficiently.
- In demand paging, there is no limit on the degree of multiprogramming.
- Effective Access time calculated as

  $(1 - p) \times ma + pft$
  where p = page fault

  ma = memory access time

pft = p × (page fault overhead + swap page out + swap page in + restart overhead)

- EAT is directly proportional to the page fault rate.
- Demand paging can have a significant effect on the performance of a computer system.
- Page replacement algorithms are:
  - First In First Out (FIFO)
  - Optimal Page Replacement Algorithm
  - Least Recently Used (LRU)
  - Clock Algorithm (Second Chance Algorithm)
  - Counting Based Algorithms

  Least Frequently Used (LFU)
  Most Frequently Used (MFU)
- FIFO replaces the page at FRONT of Queue.
- Optimal it replaces the page that will not be used for the longest time.
- LRU replaces the page that has not been used for the longest time.
- LFU replaces the page, which has the smallest count.
- MFU replaces the page, which has the highest count.
- Belady anomaly state that page fault rate may increase as the number of allocated frames increases,
- FIFO replacement algorithm suffers from Belady anomaly.
- The degree of multiprogramming increases more which degrade the performance of the system is known as thrashing.
- Thrashing degrades the performance of a system.

4.7 PRACTICE EXERCISES

1. What is memory management?
2. How Operating system helps in memory management?
3. What are the different types of memory management techniques?
4. Explain the following terms:

   (i) Internal fragmentation
   (ii) External fragmentation.
   (iii) Compaction
5. Explain the strategies of the first fit, best fit, and worst fit in the multiple partition memory management techniques?
6. What is the basic difference between Single partition and Multiple partition memory management techniques?
7. What is Paging?
8. Why is page size always power by 2?
9. Explain the technique of Segmentation in memory management?

10. Why paging is used?

11. Describe the following:

    — Swapping

    — Structure of Page Table

    — First Fit

    — Best Fit

    — Worst Fit

    — Address Binding

    — Production and Sharing

12 Write a note on virtual memory.

13. Explain the demand paging.

14. What do you mean by page fault?

15. Write steps handling the page fault.

16. Explain the performance of demand paging.

17. Why we need the page replacement algorithm.

18. Describe the working of various page replacement algorithms.

19. What do you mean by reference string?

20. What is the Belady anomaly?

21. Write a short note on Thrashing.

22. Consider the following page reference string:

    1, 2, 3, 4, 7, 5, 1, 2, 1, 7, 5, 3, 2, 4, 2, 3, 1, 6, 4, 2, 1, 2, 3, 6

    How many page faults will occur for the following page replacement algorithms? Assume a set of four page-frames (initially all empty).

    FIFO

    LRU

    Optimal

B.Sc.(DATA SCIENCE)

SEMESTER-II

OPERATING SYSTEM

UNIT IV: FILE SYSTEM

STRUCTURE

5.5 File-System Implementation

5.6 Directory Implementation

5.7 Allocation Methods

5.8 Free-Space Management

5.9 Practice Exercises

5.0 OBJECTIVE

- Understanding Concept of File, different methods to access files.
- Using File system structure and their implementation,
- Concept of Directory and its allocation methods

5.1 FILE CONCEPT

A file is a group of similar records which is stored in memory. The file is treated as one unit by users and applications. It may be mentioned by name. The filename should be sole which means in the same location file's name should be unique. It may be created, deleted, appended, truncated. There should be file manager which provides a protection mechanism to allow machine user to administrator how processes executing on behalf of different users can access the information in a file.

5.1.1 File System

The file system in the operating system is assigned with the work of storing, controlling, and managing data that is stored on disks or secondary storage in the form of files. File system management is responsible for maintaining consistency in data when multiple users access files concurrently. It also provides measures for file protection at times when the system crashes.

5.1.2 Introduction

Files are stored permanently on secondary storage devices, such as hard disks. A file system is a part of the OS that is responsible for controlling secondary storage space. It hides device-specific complexities and provides a uniform logical view of users. A function of file system includes:

□ It allows users to provide a facility to give the name of the file as user-defined names, to create, append, truncate, and delete files.

□ It maps user-defined names with low-level spotters, names in a machine-understandable format so that the machine finds a file uniquely.

□ It provides a uniform logical view of data to users rather than a physical view i.e. internal structure of files in which they are stored on the disk, by giving user-friendly interface.

□ It controls transferring of data blocks between secondary storage and main memory and also between different files.

□ It offers semantics or the file-sharing rules and regulations between numerous processes and users.

□ It also allocates and manages space for files on secondary storage devices, such as disks or magnetic tapes. Space management is an important part of the file system.

□ It protects the files from system failures and applies measures for recovery and backup.

□ It provides security measures for confidential data such as electronic funds or criminal records.

- It also provides encryption and decryption facilities to users. Encryption is a mechanism of converting data into some code form that is unreadable to everyone except the recipient. Decryption is the reverse process of encryption.

A file system is implemented as a layer of OS and is placed between the kernel and the memory manager. It consists of utility programs that run as constrained applications that are used to control access to files. Users may access files with the help of the file system only.

5.1.3 File Attributes and Naming

The file is the named collection i.e. each unit is having a name, that is unique and the end-user identifies a file by its name, usually considered as a linear array i.e. sequentially arranged, of date and records. Almost all input/output operations are performed through files. All inputs are done via files and all outputs are recorded in files. Data or information stored in files is of many types. A file has a certain defined structure according to its types such as executable programs, numeric and textual data, pictures, images, and sound recordings. Each file is saved in a directory which is acts as a container for that file. Access to file is done through these directories. A user assigns a name with each file through which it can refer. Moreover, the file system requires more information about a file that is attached with the file in the form of attributes to maintain consistency and reliability of data and control multiple accesses.

5.1.4 File Attributes

File attributes are required by any file system to manage or maintain a file. It may differ from one operating system to another operating system. File attributes are the information about a file that is associated with every file. Some attributes are unique for each file in disks such as file locations, name and creation, date, and time. Few attributes are accessible for users, such as access privileges, name, or size of a file, whereas some of them are specifically assigned to a file for file system usage.

File attributes vary from one OS to another, but few of them are needed by every OS. The major types of attributes are:-

(1) **User-Defined attributes**

(2) **System-Defined Attributes**

5.1.4.1 User-Defined attributes

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

5.1.4.2 System-Defined Attributes

- **Low-level identifier:** Machine understandable names, usually in binary digits that are used by hardware to identify a file by mapping it with a user-defined name. This attribute also consists of numbers.
- **Location:** Address of sector or area where the file is stored on the disk. Usually, this information is used as a pointer; a value used by programs to find the location of a certain sector or file, to the location and consists of numbers.
- **File size:** Current file size in bytes, or words, or in blocks.
- **Creation date:** This contains the date when the file was created.
- **Allocated size:** Total allocated size of a file by the file system.
- **Volume:** Indicates which type of device is used for storing files.
- **Date of last modification:** Contains date and time of last update, insertion, and deletion in a file.

All the attributes of a file are stored in a directory where the file resides. Their storage takes more than 1 Kb space per file. Some attributes are stored in the header record associates with each file.

5.1.5 Naming

The most significant attribute of a file is its name that is given by the user who created it. Users should associate a name with every file to uniquely identify it and access files through these names. The file's name should be unique in its directory. In a shared system, it is recommended that the user must assign a unique name to a file.

Files are accessed by giving a complete path or address i.e., you have to specify the names of directory and subdirectories with the filename, all when combined makes a complete address for a file. It is possible, that a directory is also stored in another directory that is contained in the root directory, a directory at the top level. Therefore, a file in a system can be located by a giving-by-giving complete path from the root directory to the files, specifying all the intermediates directories, to the file system. All the directories, other than the root directory, are the subdirectories of the root directory. The root directory is denoted by '\'.

An example of the pathname for a file:

C: \ Rohit \ rsts \ os.doc

The pathname of a file is also called the complete address for a file. Slash is used to restricting names in a sequence and also indicate subdirectories of a directory. Two or more files are allowed to have similar names provided they have different pathnames i.e. their parent directory should be different.

The process that calls a file is associated with it a current directory also called the working directory. In this case, a complete pathname is not required rather; a user can access files giving pathname starting from this working directory.

5.1.6 File Operations

Basic operations on files are:

Create a File: For creating a file, address space in the file system is required. After creating a file, the entry of that file must be entered in the directory. Then directory entry contains the file name and location of that file which is stored in the file system.

Writing a File: A system call is used for writing into a file. It requires two parameters one file name and the second information which is to be written.

Reading a File: A system call is called to read a file. It requires two parameters: one the file name and its memory address.

Delete a File: System will search the directory with the file to be deleted. It releases all file space that can be reused by another file.

Truncating File: The user may need to remove the contents of the file but want to keep its attributes. It recreates a file.

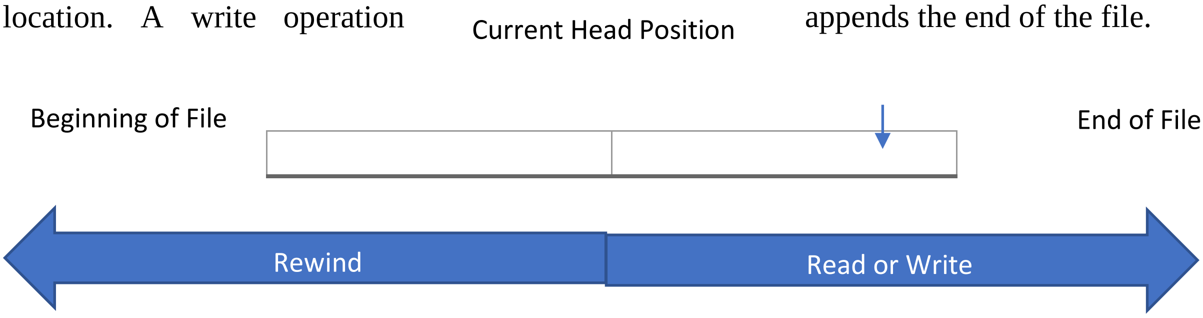
5.2 ACCESS METHODS

In the file, information can be accessed in various ways. Different types of file access methods are:

1. Sequential access
2. Direct access
3. Indexed Access

1. Sequential Access

It is a very simple method among the other methods. In the file, information is serially accessed which means one record after other records. For example: in a tape recorder, the tape plays in sequential mode. A read process reads the file's next portion and automatically advances a file pointer, that keeps tracks of Input/Output location. A write operation appends the end of the file.



2. **Direct Access**

It allows random access to any block. This model is based on a disk model of a file. It allows programs to read and write records in any order.

3. **Indexed Access**

The records in a logical sequence according to a key contained in each record. The system maintains an index containing the physical address of certain records. By using the key attributes, the records can be indexed directly.

Sequential and direct access method both are not supported by all operating system. Some operating system uses sequential access method, some operating system uses direct access method and some operating system used both access methods.

5.3 DIRECTORY AND DISK STRUCTURE

Directories are considered as symbolic tables of files that are store all the related information about the file it holds, with the content. This information includes file attributes, location, type, and access privileges. Directories are also known as containers of files.

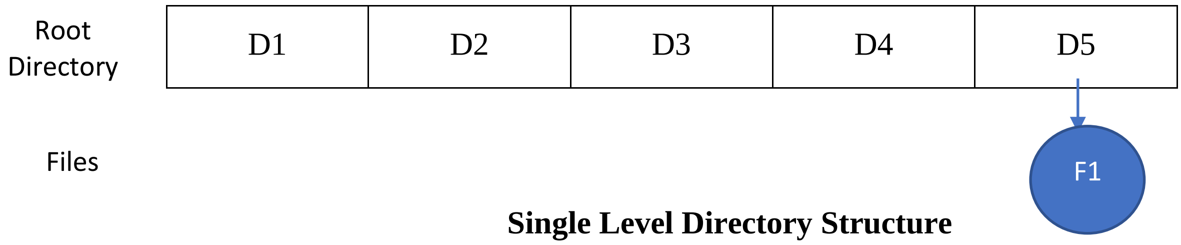
The directory is itself a file that is owned by the OS. Millions of files present in the system need to be managed. Directories provide means to organize files in a structure. Each entry in a directory contains information about a file. Similar to files, operations such as insertion, deletion, and searching can be performed on it. Operations performed on these entries are:

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

Based on these entries and their operations, the structure for directories can be organized in different ways. The three most common structures for organizing directory are, single-level, two-level, and hierarchical structures.

5.3.1 Single-Level Structure

It is the simplest form of directory structure having only one level of directories. The entire files are contained in the same directory. It appears like the list of files or sequential files having file names serving as the key. The logical structure of a single directory is given in the figure below:



Single Level Directory Structure

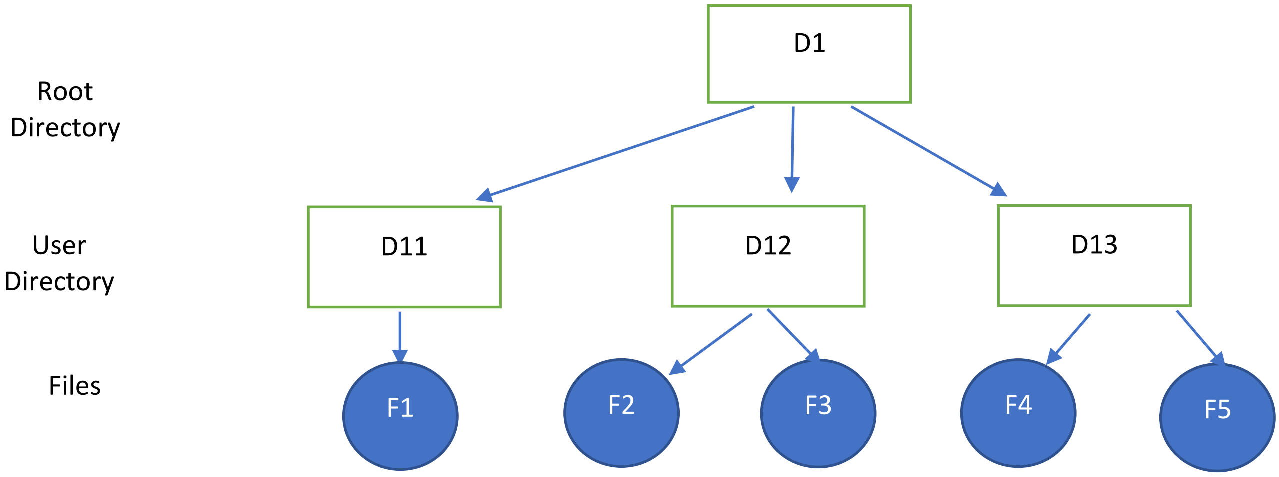
The directory structure is implemented in an earlier single-user system. It becomes outdated and inadequate in the multiple-user system. Even for a single user, it is difficult to keep track of the files if the number of files increases. Moreover, files are of different types, such as graphic files,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

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

5.3.2 Two-Level Structure

As the name suggests, this structure is divided into two levels of directories i.e. a master directory and user, and all these directories are contained and indexed in the master directory. The user directory represents a simple list of files of that user.

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



Two Level Directory Structure

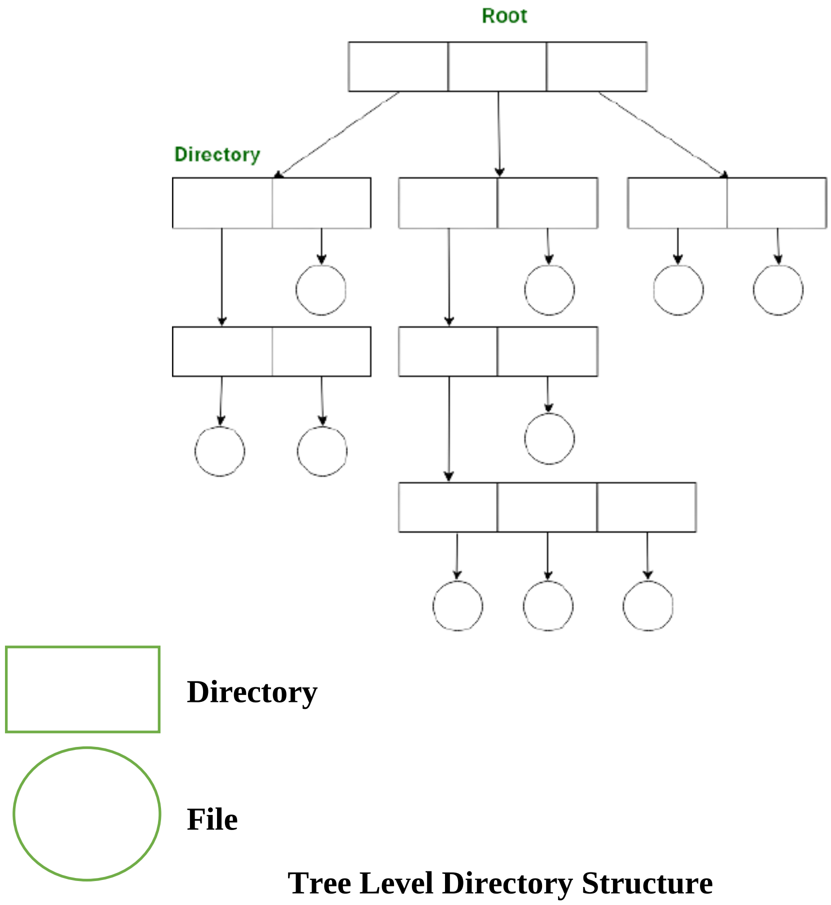
A user name and file name are the pathnames for a file. This structure solves the problem of unique names up to a certain extent i.e. user can assign duplicate names to files provided files are present in different directories. Names need to be unique in the user's directory because two file names cannot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

To create or delete a file, OS searches only the user's directory that initiates the command. OS uses a special system program or system calls, to create or delete a user directory. This system program creates or deletes a user directory entry from the master directory.

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

5.3.3 Hierarchical Structure or Tree-Structured Directories

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



Tree Level Directory Structure

Users can create their subdirectories to organize files of different types, such as separate subdirectories for graphic files, or separate subdirectories for text files. Special system calls are used to create or delete directories. Internal formats, i.e. the internal structure in which the details of the directory are stored, of each directory ha

s an entry that stores special bits representing a subdirectory or a file i.e. 0 bit represents a file and 1 bit represents a directory.

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

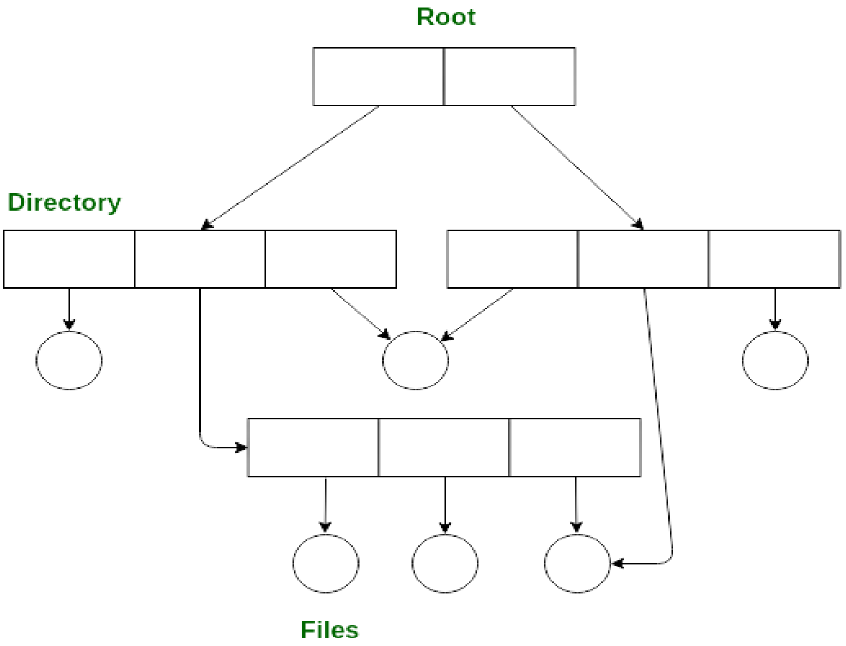
In a hierarchical structure, a file can be referenced in two ways, *absolute pathname,* and *relative pathname.*

Absolute pathname starts from the root and ends at the required file following a path of directories and subdirectories.

Relative pathname starts from the current directory to the file. Users can access another user file by giving its pathname. In a hierarchical structure, the pathname to a file can be longer than a two-level directory. This increases the search time for a file that resides in other user directories.

5.3.4 Acyclic Graph Directories

It always directories to shared subdirectories and files. Some files or directories may be in two different directories. Shared files and directories can be implemented by using links. Link is implemented as an absolute path or relative pathname. It is more flexible than a simple tree structure but sometimes it is more complain.



The acyclic graph directory structure

Characteristics

1. Files and sub-directories can be shared.
2. It is more flexible than three structures.
3. Deletion of a shared file is difficult.
4. Consistency should be taken care of.

5.3.5 General Graph Directories

In this structure, cycles are also permissible within a directory structure are from more than one parent directory, multiple directories can be made. The main disadvantage of the general graph directory structure is to compute the total size or memory required which has been taken by the files and directories.

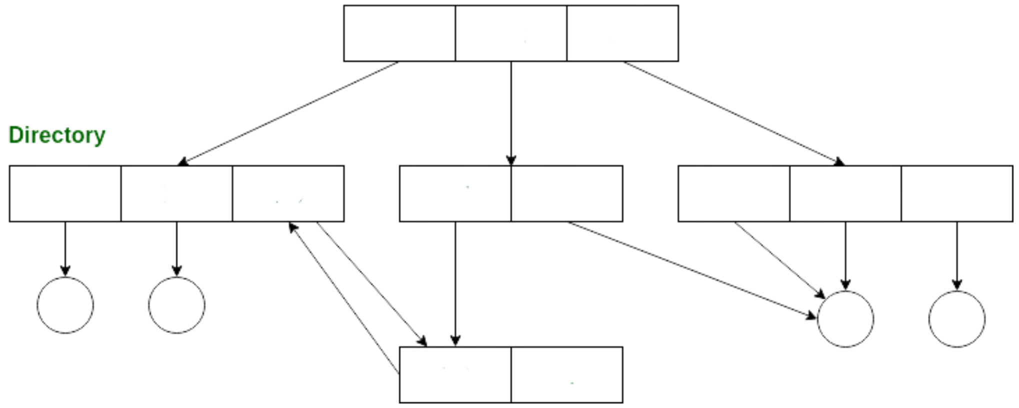
Characteristics

It permits more cycles.

It is more flexible as compared to the other directories structure.

It is more expensive than others directory structures.

It requires garbage collection.



A General Graph Directory structure

5.4 FILE-SYSTEM STRUCTURE

The file system resembles a complete directory structure that includes a root directory following subdirectories and files under it. The file system is required to enhance the function of retrieval and storage of files. We can mount more than one file system that creates a directory structure showing an image of a single file system. Commands used for file system operations are:

Chfs: Change the file system characteristics

Crfs: Add a new file system

Rmfs: Remove a file system

Mount: Make files system available for user access.

In the real world, different Operating systems use different file systems. Each file system has a different directory structure and different ways to represent files.

1. **CD ROM**

CD ROM is an acronym for Compact Disc-Read Only Memory. Data is stored on one side of it that is coated with the highly reflected material. Data is stored in the form of series of microscopic pits, i.e. data is stored in binary digits of 0 and 1 and a pit represents a binary 1, which is read by a low-powered laser in a drive unit. To read data, the laser rotates the disk at various speeds at a Constant Linear Velocity (CLV).

CD ROM is used for storing audio in digital format and is of size 6547.4 MB. CD ROM can store data equivalent to 500 floppy disks. Data on CD ROM can be accessed much faster than any other tape. Accessing speed of CD ROM is 75-150 sectors per second. To read a CD ROM, we need a CD ROM reader and an appropriate software driver that controls the functionality of CD ROM.

Data is stored in the form of sectors with error-correction codes i.e. codes that are used to debug any error if occurred in the stored data. Data on CD ROM can be stored and retrieved, as files for that we need a file system to manage the stored data. The file system allows you to access the contents of CD ROM using the interface of a normal file system i.e. the file system stored on the disk.

2. **FAT16**

To understand the FAT 16 file system, a discussion on File Allocation Table (FAT) is necessary. FAT is a table that has entries in the form of pointers. These pointers point to the files that are present in a partition. Partition is a logical division of the table. Each entry in this table consists of a block number, a 0 valued block number represents as unused block any new entry is stored on it. Partition of the disk depends on the file system used. The two most common file systems are FAT 16 and FAT 32.

FAT 16 is the old MS-DOS file system created by Microsoft in 1977 that used 16 bits table. FAT 16 support filenames of 8.3 standard i.e. 8 characters for the name and 3 characters for file extension.

The value stored in the FAT table is 16 bits. A 16-bit file allocation table means that the size of the address stored in the entry of FAT is limited to 16 bits. This implies that you can’t have storage allocation units i.e. the entries in the FAT 16 table are restricted to 65536 units. This is the main drawback of FAT 16.

Another drawback of FAT 16 is that its partition is of fixed size i.e. of 32k. That wastes a lot of disk space in storing files of small size such as pictures, HTML documents, etc.

3. **FAT32**

It is an advancement of FAT 16 that supports 32 bits addresses. Entries in FAT 32 tables can hold a value up to 32 bits. It divides the hard disk into smaller partitions of a small disk area than FAT 16 so that disk space is used more efficiently with increased storage capacity.

FAT 32 allows you to create a partition up to 2048 GB size. The number of clusters in FAT 32 ranges around 4 billion i.e. 4,294,967,296 inexact. FAT 32 supports long file names of a maximum of up to 255 characters. File names are not case sensitive i.e. no matter whether the names are in capital letters or small letters; it saves the case of filenames once created. Minimum disk space needed for FAT 32 is 8 GB. Os that supports FAT 32 are: OSR2, Windows 98, Windows 2000, and Linux.

4. **Unix**

Unix is a multi-user and multitasking OS that allows multiple users to access multiple files concurrently. It is a data structure that resides on disks. Unix file system has a tree-like structure having a root directory (/) at the top node. This hierarchical structure appears to users as a single file system having all types of files in one tree, the concept of drives such as A, C: etc is not found here. Each node, a vertex of the UNIX file system tree, represents a separate file system.

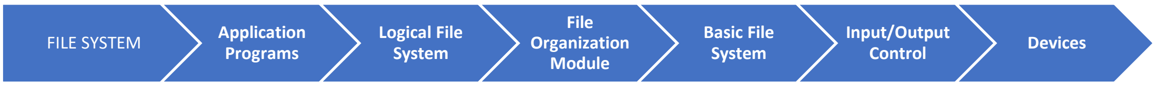
Thus, the separate device is mounted in one single file system. A file in UNIX is a collection of randomly addressable bytes, not sequentially; each byte has a different address, not in the continuity of a previously-stored byte. Files are allocated on the block randomly on the disk. The index method is used to keep track of each file. A filename can extent up to 255 characters excluding forward-slash (/) & Null and are case sensitive. File in Unix may contain holes i.e. no data blocks. The directory is also a file except the user can’t write on it. Unix associates its file with three access permissions: read, write, and executable. Users can access files giving absolute and relative pathname.

UNIX file system is made up of four parts:

- **Boot block:** Contains boot program and load kernel into memory when a user boots the system or switch on the system.
- **Superblock:** Contains all information about the attributes of a file.
- **I-list:** Contain a list of I-nodes of the file system. I-node is a data structure that holds information about an individual file, such as file size, access privileges, or file name. Each file is related to I-node and is identified by I-nodes number.
- **Data block:** Contains actual file contents.

5.5 FILE-SYSTEM IMPLEMENTATION

File system structured in the layers which are: Application Programs, Logical File System, File Organization Module, Basic File System, Input/Output Control, Devices



- **Input/Output Control level –**
  Device drivers perform as an interface between devices and Operating Systems, these drivers help to transfer data from disk to main memory and vice versa. It picks block numbers as input and as output, it returns low-level hardware-specific instruction.

- **Basic file system –**
  It refers to normal commands to the particular device driver to do operations: read and write on particular physical blocks on the disk. A block is part of memory in the buffer that can contain the contents of the disk block and cache stores frequently used file system metadata. It also handles the memory buffers and caches.

- **File organization Module –**
  It contains information about the files such as their location, and their logical blocks, and physical blocks. Physical blocks were entirely different so they do not match with logical numbers of the logical block which are numbered from 0 to N. It contains some free space that is tracked by unallocated blocks.
- **Logical file system –**
  It manages metadata information about a file i.e includes all details about a file except the actual contents of the file. It also keeps with the help of File Control Blocks. File Control Block (FCB) contains all the information about a file such as the owner of the file, size of the file, permissions granted, file location, and its contents.

Advantages :

1. Replication of code is lessened.
2. Each file system can have its independent logical file system.

Disadvantages :

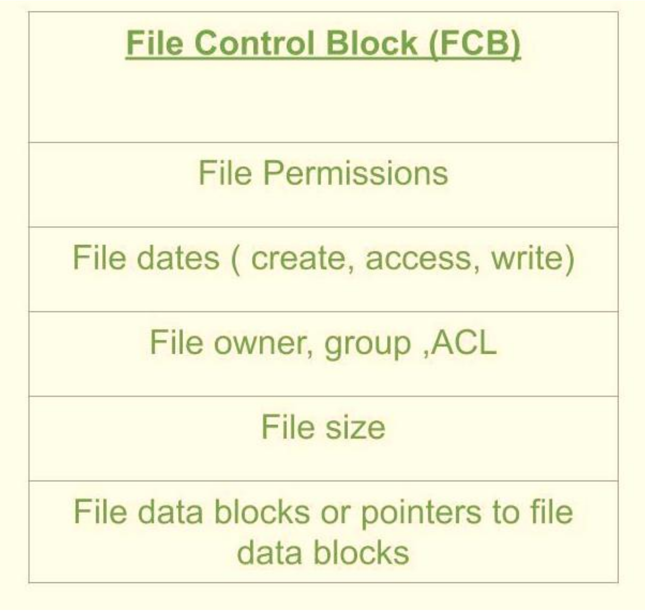
If someone accesses many files simultaneously then its outcomes will reduce the performance of the system.

We can **implement** a file system by using data structures which are:

1. **On-disk Structures –**
   Generally, it contains the following information:
   a. total number of disk blocks available
   b. free disk blocks available
   c. location of them.

Below given are different on-disk structures :

1. **Boot Control Block –**
   It is generally the 1st block of volume and it has information that is needed to boot any operating system. In UNIX, this block is known as a boot block and in NTFS it is known as a partition boot sector.
2. **Volume Control Block –**
   It contains information about a particular partition for example:- count of free blocks, size of blocks, and its pointers also. In UNIX it is known as superblock and in NTFS it is stored in the master file table.
3. **Directory Structure –**
   They kept file names and linked inode numbers. In UNIX, it contains file names and associated file names and in NTFS, it is stored in the master file table.
4. **Per-File FCB –**
   It holds details about files and it has an exclusive identifier number to allow association with the directory entry. In NTFS it is kept in the master file table.



File Control Block (FCB)
File Permissions
File dates (create, access, write)
File owner, group ,ACL
File size
File data blocks or pointers to file data blocks

Structure of File Control Block (FCB)

In-Memory Structure :
They are maintained in the main memory and these are helpful for file system management for caching. Several in-memory structures given below:

1. **Mount Table:** It is a table in which information about every mounted volume is stored.
2. **Directory-Structure cache –** It contains the directory information about which directory is freshly accessed.
3. **System-wide open-file table –**It holds the copy of the File Control Block (FCB) of every open file.

4. **Per-process open-file table** – It holds information on which process opened the file and it also maps with appropriate system-wide open-file.

5.6 DIRECTORY IMPLEMENTATION

A directory can be implemented in two ways:

1. Linear List
2. Hash Table

Linear List

The linear list is the simplest method. In this linear list of filenames used. For searching purposes, a linear search technique is used to find a particular entry. It is very simple but time-consuming to execute. Directory information is used often. Users also notice the slow implementation of access to it.

Hash Table

With the hash table, it decreases the directory search time. Insertion and deletion are very simple. Hash table takes the value which is compiled from the file name and then it returns a pointer to a file name in the form of a linear list. It always uses a fixed size.

5.7 ALLOCATION METHODS

The space allocation strategy is often closely related to the efficiency of file accessing and of logical to physical mapping of disk addresses. Three major methods of allocating disk space are widely used which are:

Contiguous Allocation Method

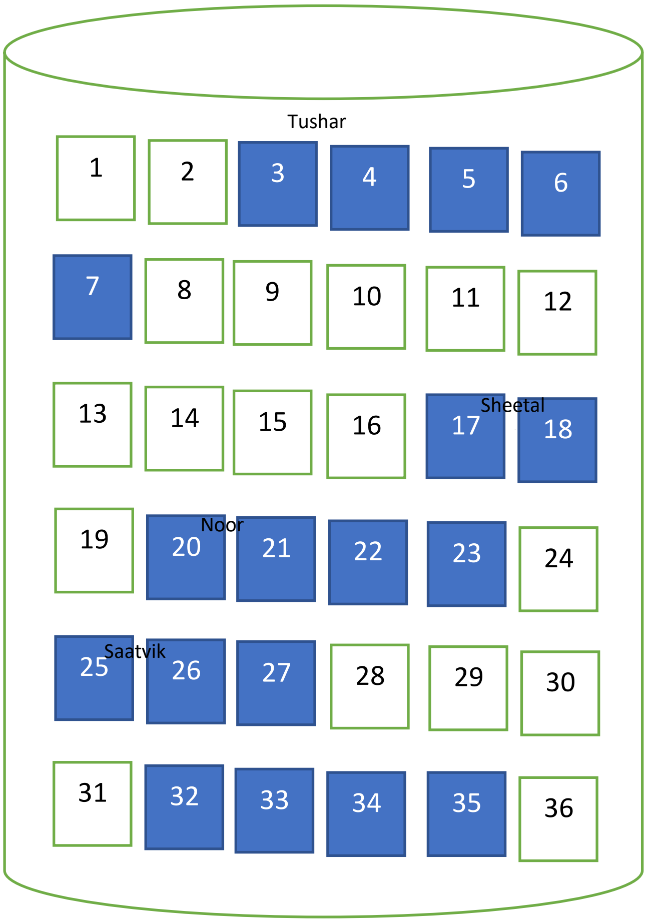
The contiguous allocation method always requires every file to occupy a set of adjoining addresses or blocks on the disk. Disk addresses describe in a linear order on the disk. The important thing to be noted is that, in this ordering, the block of memory represents as B, accessing block where is B+1 after block B normally requires no head movement. When head movement is needed, it is only one track. Thus, the number of disks seeks required for accessing contiguous allocated files is minimal.

Contiguous allocation of a file is defined by the disk address or blocks and the length of the first block. Suppose, the file has a size of N blocks, and its starting location is location B, then it must occupy consecutive blocks namely B, B+1, B+2,….., B+n-1. The directory entry must contain for each file indicates the address of the starting block of the file and the total length of the file in terms of blocks. This is the simplest method of allocation. Performance is good because the entire file can be read from the disk in a single operation.

The difficulty with a contiguous allocation is finding space for a new file. If the file to be created is n blocks long, then the OS must search for n free contiguous blocks. The most common approaches used to select a free hole among the set of available holes are First-fit, best-fit, and worst fit. Both first-it and best-fit are a better option as compare to worst fit in terms of both criteria which is time and storage utilization. Neither first-fit nor best fit is best in terms of storage utilization, but first-fit is generally faster.

These algorithms also have a disadvantage which is external fragmentation. As files are allocated and also deleted, the free disk space is broken into tiny chunks. External fragmentation exists when enough total disk space exists to satisfy a request, but this space not contiguous; storage is fragmented into a larger number of small holes.

Another problem with a contiguous allocation is determining how much disk space is needed for a file. At the file creation time, how much space it requires must be known and allocated. One question arises here, how does the creator know about the size of the file which is to be created? In some cases, this determination may be fairly simple, but generally, the estimation of the size of an output file is a little difficult. A directory containing the file information is maintained. Starting block address and a total number of blocks in a file are read from this directory to read/write a file.



Directory		
File	Start	Length
Tushar	3	5
Saatvik	25	3
Noor	20	4
Sheetal	17	2
Rohit	32	4

🟦 USED

□ FREE

Contiguous File Allocation

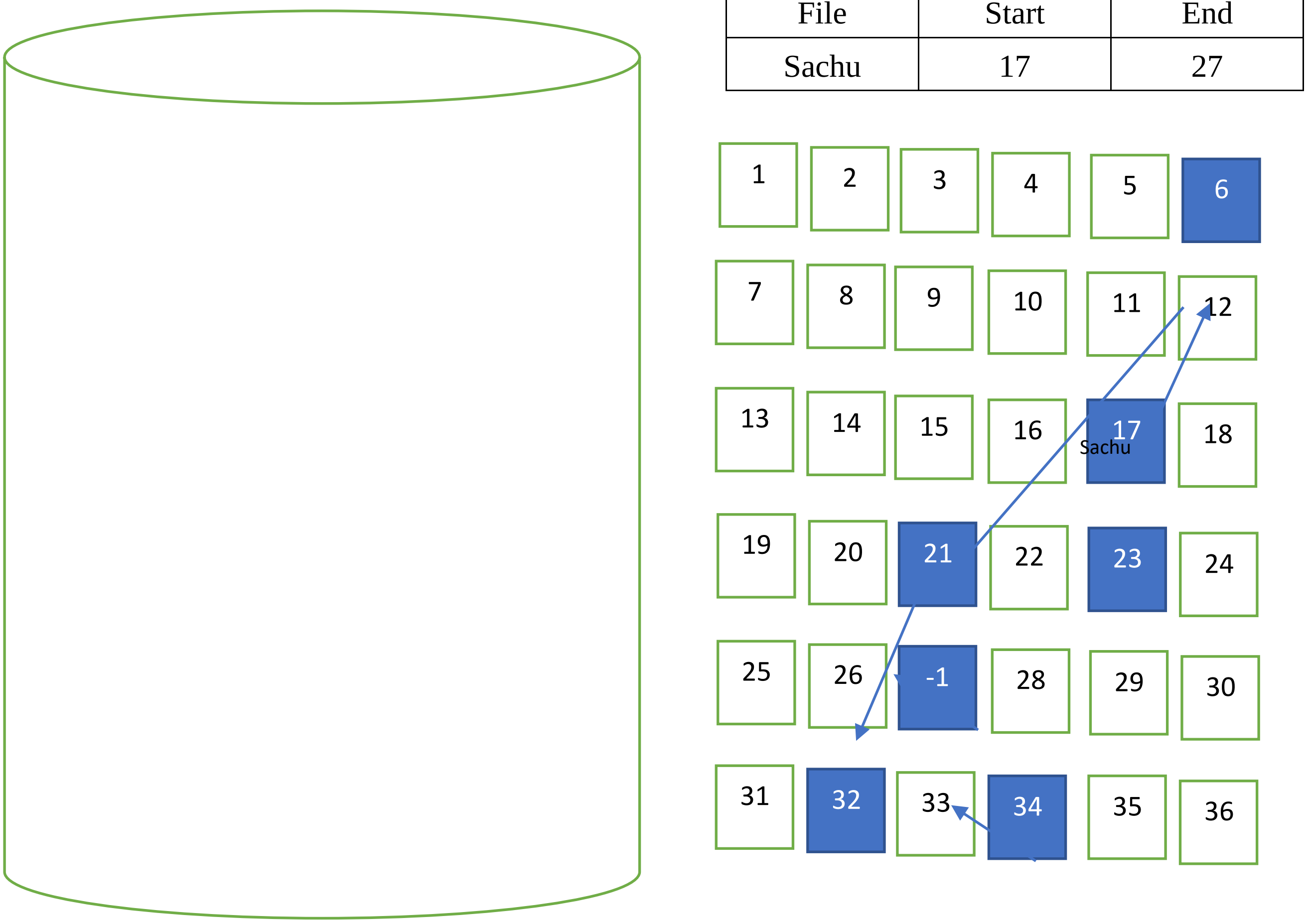
Characteristics

1. It supports flexible size portions.
2. It requires Pre-allocation.
3. It needs only a single entry of a file.
4. Allocation frequency is only once.

Linked List Allocation method

The problems in contiguous allocation can be traced directly to the requirement that the spaces be allocated contiguously and that the files that need these spaces are of different sizes. These requirements can be avoided by using linked allocation.

Directory		
File	Start	End
Sachu	17	27



Link List Allocation

In linked allocation, every file is fragmented into Blocks and makes a linked list of these disk blocks. The directory stores a pointer to the first Block of the file. For example, a file of 7 blocks starts at block 17, they might be next block 6, then block 21, block 32, block 23, block 34, and finally last block 27. Each block holds a pointer to the next block and the last block contains a NULL pointer which shows that end of the link list. The value -1 may be used for NULL to differentiate it from block 0.

In the linked allocation method, every directory entry contains a pointer to the file's first disk block. Initially, the pointer is initialized with a null value which shows that it is an empty file. A write to a file removes the first free block from the free block list and writes to that block. Then the address of this new block is associated with the last block of the file and then linked to this block at the end of the file. To read a file, the pointers are used by a followed pointer from block to block. The problem of the **Contiguous File** allocation method i.e., external fragmentation is resolved in the linked allocation. Any free block can be utilized to fulfill a request. Notice also that there is no need to declare the size of a file when that file is created. A file will still grow as possible as there are free blocks.

The linked allocation has some disadvantages. The foremost problem is that it is inefficient to support direct access; it is effective only for sequential-access files. To locate the i^{th} block of a file, it should begin at the start of that file and follow the pointers till the i^{th} block is reached. Note that every access to a pointer needs a disk read.

Another severe problem is reliability. A bug in OS or disk hardware failure might result in the pointer being lost and damaged. The effect of which could be picking up a wrong pointer and linking it to a free block or into another file.

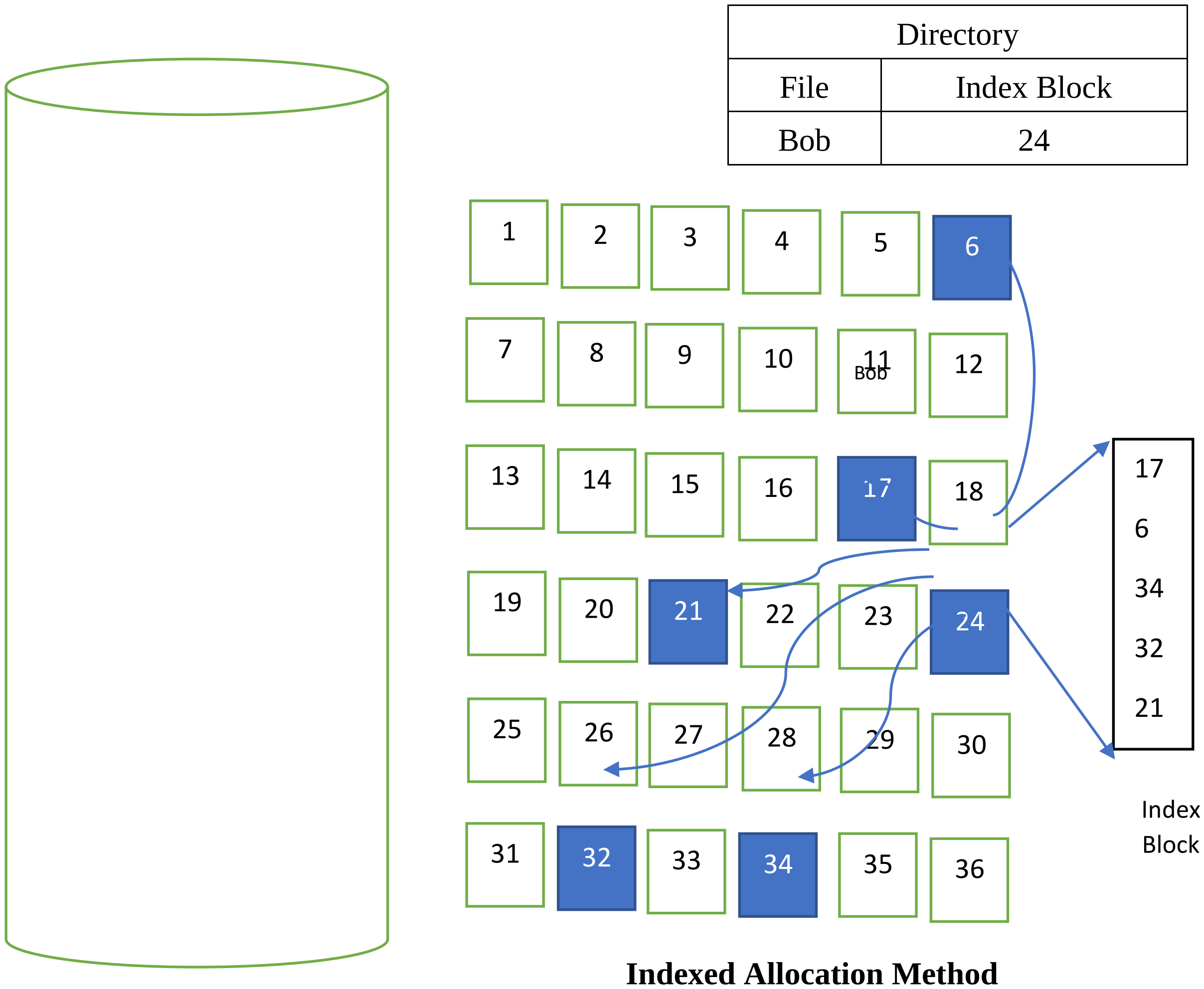
Characteristics

1. For a File there is one entry in the File allocation table size.
2. It's supports fixed size portions.
3. In this allocation Pre-allocation is feasible.
4. Allocation occurrence is Low to High.

Indexed Allocation Method

The problem of both contiguous and linked allocation is resolved in the indexed allocation method. In this allocation, one block is used as an Index block in which all the pointers bring together into one location.

Of course, the index block will occupy some space and thus could be considered as an overhead of the method. In indexed allocation, every file has its induvial index block, which is nothing an array of disk sectors of addresses. The j^{th} sector of the file is pointed by the j^{th} entry in the index block. The directory keeps the address of the index block of a file only. To read the j^{th} sector of the file, from the index block, the pointer in the j^{th} entry is read to locate the desires sector. Indexed allocation supports direct access, it resolves the external fragmentation. Any free block anyplace on the disk might satisfy a request for extra space.



Directory	
File	Index Block
Bob	24

Indexed Allocation Method

Characteristics

1. It supports both sequential and direct access.
2. There is no external fragmentation.
3. It does suffer wasted space.
4. Manage the Pointer is overhead.
5. It is faster than the other two approaches.

5.8 Free-Space Management

Since there is only a restricted amount of disk space, it is necessary to utilize the space from deleted files for new files. To keep track of free disk space, a free-space list is maintained by the system. This list keeps records of all disk blocks which are free. Whenever a file is created, the free space list has to be searched for the required amount of space and allocate space to a newly created file. After that from the free-space list that space is removed. Whenever a file is deleted then the free space list is updated by deleted file space added to the free-space list.

Bitmaps or Bit – Vector

In the bitmap method a table, called bit map is maintained for keeping track of the information about which part of memory is allocated to which process and which portion of memory is free. The bitmap is dynamic. This means the table is updated each time when a new process is allocated memory space, an existing process is swapped out of the main memory, a process is swapped in the main memory, or a process is completed and released from memory.

In the bitmap method, the memory is divided into certain equal size units or blocks. While allocating memory space to a program an OS can only allocate a complete unit of memory space and not a part of the unit. There is a cell in the bitmap for each unit of memory. If a unit is allocated to any process the value of the cell corresponding to the unit is set to 1 otherwise if the memory unit is free, the value is set to 0.

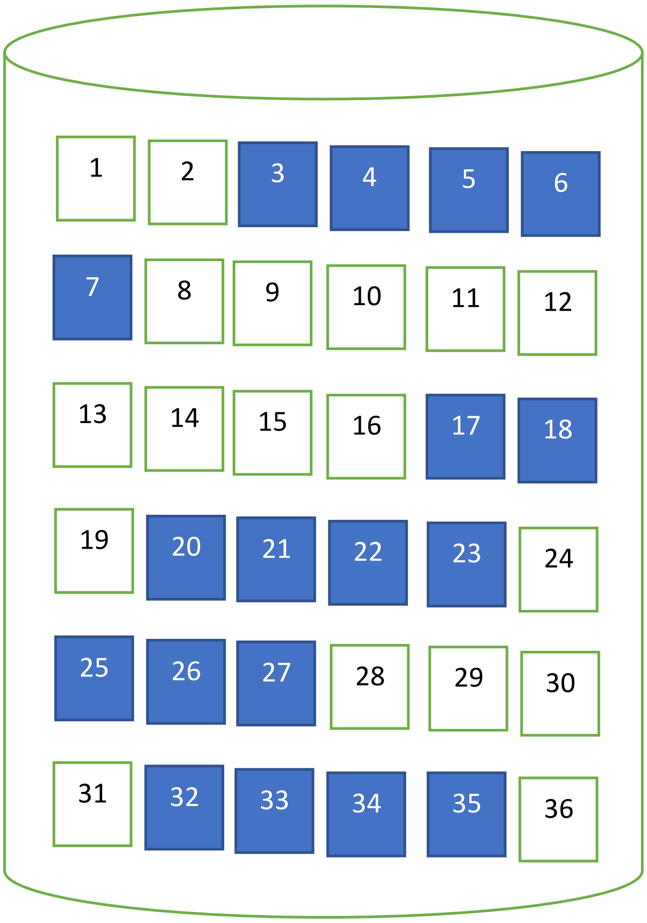


Figure Bit Vector

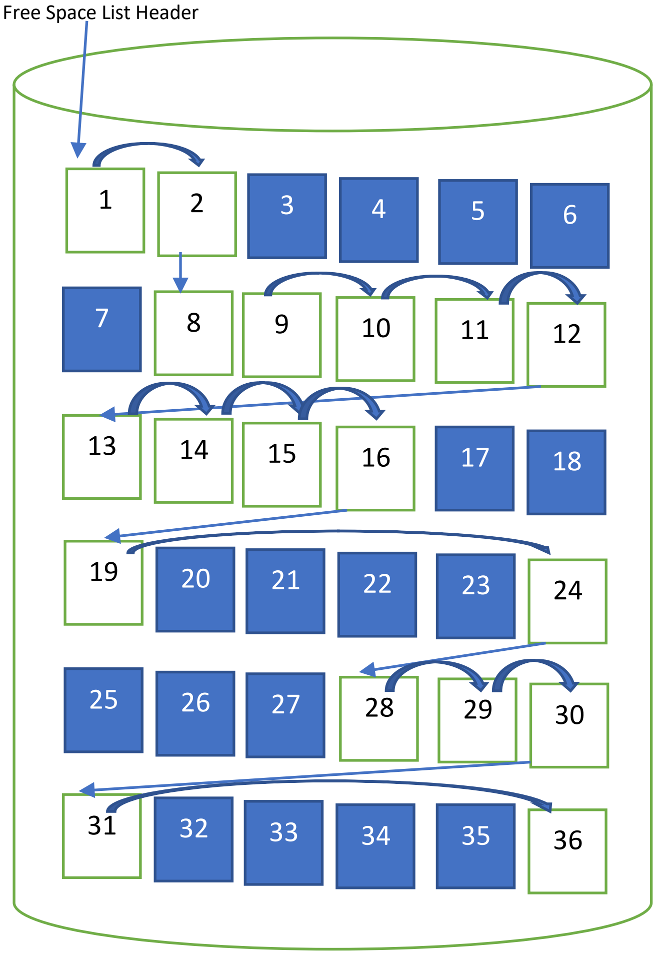
1, 2, 8, 9, 10, 11, 12,13,14,15,16,19,24,28,29,30,31,36 are flocks free so bit map of this example is 0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,1,1,0,1,1,1,1,0,1,1,1,0,0,0,0,1,1,1,1,0

There is a cell in the bit map corresponding to every memory unit. The value of the 3 cell in the bitmap is 1 because its corresponding memory unit is in use by the process. Value of the first and second cell is 0, because it's free.

An **advantage** of the bitmap method is that it is very simple to implement. On the other hand, the **disadvantage of this method is** it is very slow. OS needs much time to search for a hole, when allocating memory space to a new process or an existing process swapped in.

Linked List

In this method, a pool or chain of nodes is created to keep track of memory that is free and maintain a linked list i.e. the first block contains the address of the next free block and so on. The address of the very first free block is stored in a special node called as START node. The last block of the linked list contains NULL, which implies that there is no other free block.



Link Free Space List on Disk

For accessing the information in the linked list, it is loaded into physical memory. Whenever any allocation is to be done, blocks are removed from the head of the list and allocated. It occupies a large space. Traversing is also time-consuming. The following list shows a free block list.

Grouping

A modification of the free-list approach, in the first free block, keeps the addresses of n free blocks. There are only the first n-1 blocks that are free. The last one is the disk address of another block containing addresses of another n free block. The importance of this implementation is that a block contains the addresses of a larger number of free blocks so it can be found quickly if we need the number of free blocks.

Example: Block 1 Contains the address of Block 2. Block 2 contains the address of the first block of the next group block i.e., 8, Block 8 stores the address of 9 to 16 and the last block is 16 so it contains the address of the first block of the next group 19. As 19 is a single free block so, it contains the next group address i.e. 28. Block 28 keeps the address of 29 and 30. Block 30 contains the address of 31 and again Block 31 is the only single block in Grouping so it contains the address of 36. 36 is the last block os it stores nothing

Counting

Another method is to take benefit of the fact that, generally, many contiguous blocks may be allocated or freed contiguously, mainly when we used the contiguous allocation method. Thus, rather than keeping the entire list of free disk addresses, we noted the address of the first free block along with it the number n of free contiguous blocks which follow the first block. Each entry in the free-space list then contains the disk address along with the count. Although each entry needs more space as compared to a simple disk address, the overall list will be shorter, as long as the count is generally greater than 1.

First Free Block	Count
1	2
8	9
19	1
28	3
31	1
36	1

Points to Remember

The file system must identify and locate the selected file before operation on a file.

The file control block contains information about the file.

A disk can be divided into multiple partitions.

The file can be accessed by

— Sequential Access

— Direct Access

— Indexed Access

Three major methods of allocating disk space are widely used which are

Contiguous Allocation Method

Linked Allocation Method

— Indexed Allocation Method

The directory can be implemented in two ways:

— Linear List

— Hash Table

Free space management techniques are:

— Bit Vector

— Linked List

— Grouping

— Counting

5.9 PRACTICE EXERCISES

1. What is a File system?
2. Write a note on file attributes and naming?
3. What is File Organization?
4. Discuss the various methods of file allocation?
5. What are the disadvantages of the contiguous file allocation method?
6. What is the difference between linked allocation and indexed allocation methods?
7. What is a directory?
8. What are the different levels of the directory structure?
9. Discuss the various file systems?
10. Discuss the concept of free space management?
11. What are the various methods of Free Space Management?

B.Sc.(DATA SCIENCE)

SEMESTER-II

OPERATING SYSTEM

UNIT VI: INTRODUCTION TO LINUX

STRUCTURE

6.0 OBJECTIVES

- To Understand Linux's shell, Kernel, and file system of Linux.
- Introduction to different types of directories: Parent, Subdirectory, Home directory; rules to name a directory, Important directories in Linux File System

6.1 INTRODUCTION TO LINUX

Linux which is pronounced as Lin-nucks is an operating system similar to UNIX. Some people also called it as descended form of UNIX. Linux is first released in 1991 by Linus Torvalds at the University of Helsinki. Since then it has gain huge popularity among the programmers. The Linux is an open-source operating system. It means it comes with the source code, so that one can change and customize the operating system according to the requirements. By the term open source it is meant that

- The independence to run the program in operating system for any function.
- The freedom to study any program that how it works and to alter it to make it work according to your need.
- The liberty to redistribute copies of the program to facilitate your neighbours.
- The free will to share copies of your modified versions to others for use.

Initially, Linux was created for personal computers and gradually it was used in other machines like servers, mainframe computers, supercomputers, etc. Nowadays, Linux is also used in embedded systems like routers, automation controls, televisions, digital video recorders, video game consoles, smart watches, etc. The biggest success of Linux is Android (operating system) it is based on the Linux kernel that is running on smart phones and tablets. Due to android Linux has the largest installed base of all general-purpose operating systems. Linux is generally packaged in a Linux distribution.

6.2 LINUX DISTRIBUTIONS

Linux has a numeral of dissimilar versions to ensemble any kind of user. Roughly all the editions of the Linux can be downloaded free of charge. These editions are labelled allocations (or, in the small form, “distros”). There are additional 600 distributions of Linux**.** Some of the general distributions of the Linux are:

- LINUX MINT
- MY LINUX
- MANJARO
- DEBIAN
- UBUNTU
- DEEPIN

- ANTERGOS
- ARCH LINUX
- GENTOO LINUX
- KALI LINUX
- SOLUS
- FEDORA
- ELEMENTARY OS
- OPENSUSE

Every distribution has a dissimilar take on the desktop. A small amount of the Server distributions of the Linux comprises:

- Red Hat Enterprise Linux
- Ubuntu Server
- Centos
- SUSE Enterprise Linux

A few of the above server distributions are at no cost (such as Ubuntu Server and CentOS) and a little have an associated cost (such as Red Hat Enterprise Linux and SUSE Enterprise Linux). Those with an associated price also comprise support.

6.3 CHARACTERISTICS OF LINUX OPERATING SYSTEM

Following are a few of the significant characteristics of Linux Operating System.

- **Portable** – Portability means software be able to work on dissimilar types of hardware in similar way. Linux kernel and application programs sustain their setting up on any kind of hardware stage.
- **Open Source** – Linux source code is liberally accessible and it is community based growth project. Multiple teams carry out task in collaboration to augment the potential of Linux operating system and it is constantly evolving.
- **Multi-User** – Linux is a multiuser scheme means numerous users can access network resources like memory/ ram/ application programs at similar time.
- **Multiprogramming** – Linux is a multiprogramming arrangement means numerous functions can run at similar time.
- **Hierarchical File System** – Linux offers a standard file arrangement in which organization files/ user files are ordered.

- **Shell** – Linux provides a particular interpreter program that can be employed to carry out commands of the operating system. It can be employed to do a variety of kinds of processes, call application programs etc.
- **Security** – Linux offers user security by means of authentication characteristics like password protection/ controlled admittance to particular files/ encryption of information.

Linux is fast, free and easy to use, power laptops and servers around the world. Linux has numerous more characteristics to amaze its user such as:

- **Live CD/USB:** Approximately all Linux distributions have Live CD/USB characteristic by which user can run/try the OS even with no mounting it on the scheme.
- **Graphical user interface (X Window System):** People think that Linux is a command line OS, wherever its precise also but not essentially, Linux have packages that can be mounted to construct the whole OS graphics based as Windows.
- **Support's** most national **or customized keyboards:** Linux is engaged worldwide and consequently obtainable in many languages, and supports on the entire of their custom national keyboards.
- **Application Support:** Linux has its own software ordnance through which users can download and set up thousands of desires just by concerning a command in Linux Terminal or Shell. Linux can3 besides run Windows demand if essential.

6.4 HISTORY OF LINUX OPERATING SYSTEM

The History of Linux commenced in 1991 with the beginning of a personal project by a Finland student Linus Torvalds to create a new open operating scheme kernel. From then, the resulting Linux kernel has been perceptible by fixed expansion during history.

- In the year 1991, Linux was started by a Finland student Linus Torvalds.
- Hewlett Packard UNIX(HP-UX) 8.0 was enlightened.
- In the year 1992, Hewlett Packard 9.0 was liberated.
- In the year 1993, NetBSD 0.8 and FreeBSD 1.0 liberated.
- In the year 1994, Red Hat Linux was initiated. Caldera was originated by Bryan Sparks and Ransom Love and NetBSD1.0 liberated.
- In the year 1995, FreeBSD 2.0 and HP UX 10.0 were liberated.
- In the year 1996, K Desktop Environment was initiated by Matthias Ettrich.
- In the year 1997, HP-UX 11.0 was liberated.
- In the year 1998, the fifth invention of SGI Unix i.e IRIX 6.5, Sun Solaris 7 operating system, and Free BSD 3.0 was liberated.
- In the year 2000, the conformity of Caldera Systems with the SCO server software division and the professional services division was declared.
- In the year 2001, Linus Torvalds liberated the Linux 2.4 edition source code.

- In the year 2001, Microsoft ordered a trademark suit besides Lindows.com
- In the year 2004, Lindows name was altered to Linspire.
- In the year 2004, the first liberate of Ubuntu was discharged.
- In the year 2005, The project, open SUSE begin a complimentary allocation from Novell's community.
- In the year 2006, Oracle liberated its own allocation of Red Hat.
- Dell initiated allocated laptops with Ubuntu pre-installed in it, in the year 2007.
- The Linux kernel 3.0 version was liberated in the year 2011.
- Google Linux-based Android declared 75% of the smart phone market share, in form of the number of phones dispatched, In the year 2013
- Ubuntu asserted 22,000,000 users in the year 2014.

6.5 ARCHITECTURE OF LINUX

Linux architecture has the subsequent components:

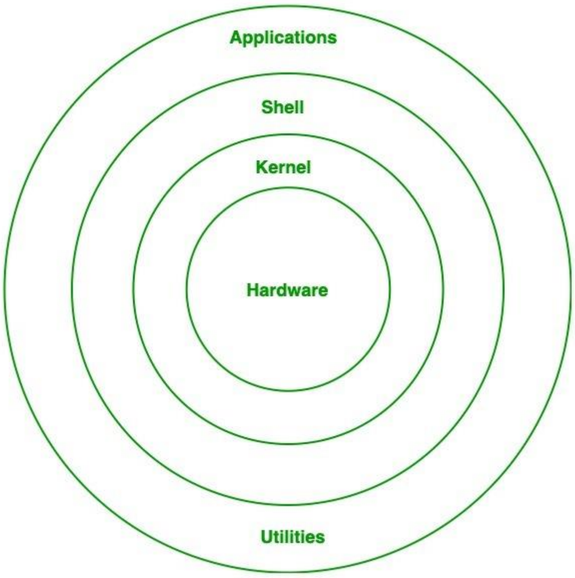


Figure 6.1: Architecture of Linux Operating System

I. **Kernel:** Kernel is the heart of the Linux supported operating system. The kernel is the mainly significant part or mind of every operating system as it handles the conversation among a machine's hardware and its software. In easy words, the kernel acts as a spirit of the network and handles the memory, peripheral apparatus, and CPU. The kernel places at the "lowest" point of the OS. This creates the method seem as if it is the only procedure running on the device. The kernel too well is accountable for avoiding and justifying clashes among diverse processes. Particular kernel types are as

- Monolithic Kernel,
- Hybrid Kernels,
- Micro Kernels and

- Kernels from Exo.

A few of the key design ethics executed by Linus kernel are:

- In Linux, the whole thing is a file philosophy
- Multi-users ability
- Multitasking capability
- Portability among GPU architecture
- Modularity
- Security
- Configurability

II. **Shell:** It is an interface to the kernel which covers the difficulty of the kernel's tasks from the client. It takes instructions from the user and carry out the kernel's functions. Linux include two kinds of command keys: text mode comparable to those originated in mainly UNIX systems (example the bourne shell, the bourne over shell, the c shell, the turbo C shell and the korn shell) and graphical user interface (GUI) such as the KDE (K Desktop Environment) and GNOME (GNU Network Object Model Environment). **Terminal** is the purpose that brings it all collectively, in the sense that it offers a visual illustration of the shell for the client to penetrate commands. In other words, in a GUI (graphical user interface), where requests and other descriptions are visually symbolized by images that the client can manoeuvre by ticking on them with a cursor, a terminal request release a window where the client can type in instructions for the shell to infer into binary communication for the kernel.

III. **System Library:** It is the exceptional kind of tasks that are used to execute the functionality of the operating system. These pieces carry out the background facility of the Linux operating system similar to scheduling, printing, sound, etc that each start-up during boot of the operating scheme, or after you register into your system.

IV. **Hardware Layer:** This layer comprises of all the peripheral apparatus like RAM/ HDD/ CPU etc. Some of the substantial hardware that is incorporated in any tool comprises of mouse, keyboard, the graphics chipset, display, and, but you do have one, your system interface card

V. **System Utility:** It offers the functionalities of an operating system to the client. A Linux-based scheme will typically appear with such a set of established Unix-like utility services; these were all normally basic tasks used in day-to-day operating system procedures, and also precise devices and requests. This is frequently software that was available and liberated under the open-source license by the GNU Project, consequently the software can be explicitly downloaded, restructured, and reallocated to all.

Advantages of Linux

The main benefit of Linux, is it is an open-source operating system. This means the source code is simply obtainable for everybody and you are authorized to give, alter and allocate the code to anybody without any consent.

- In terms of security, Linux is further secure than any other operating scheme. It does not signify that Linux is 100 percent safe it has a little malware for it but is fewer susceptible than any other operating scheme. So, it does not necessitate any anti-virus software.
- The software renewed in Linux are simple and recurrent**.**
- A variety of Linux distributions are accessible so that you be able to utilize them according to your desires or according to your flavour.
- Linux is liberally available to employ on the internet.
- It has huge community support.
- It offers high constancy. It seldom slows down or freezes and there is no necessitating rebooting it after a small time.
- It preserves the privacy of the client.
- The presentation of the Linux scheme is much elevated than other operating scheme. It permits a large quantity of people to employee at the similar time and it grips them resourcefully.
- It is network pleasant.
- The flexibility of Linux is elevated. There is no necessitate to establish a complete Linux suit; you are authorized to establish only required apparatus.
- Linux is companionable with a huge number of file set-ups.
- It is quick and simple to fit from the web. It can also mount on any hardware even on your previous computer structure.
- It carry out all tasks correctly even if it has partial liberty on the hard disk.

Disadvantages of Linux

- It is not extremely user-friendly. So, it might be puzzling for beginners.
- It has tiny marginal hardware drivers as contrast to windows.

6.6 Widows Vs Linux

Few Differences between Linux and Windows

S.NO	Linux	Windows
1.	Linux is a open source operating scheme.	While windows are the not the open source operating scheme.
2.	Linux is at no cost.	While it is expensive.

S.NO	Linux	Windows
3.	It's file name case-sensitive.	While it's file name is case-insensitive.
4.	In linux, monolithic kernel is employed.	While in this, micro kernel is employed.
5.	Linux is more competent in comparison of windows.	While windows are fewer competent.
6.	There is forward slash is employed for separating the directories.	While there is back slash is employed for isolating the directories.
7.	Linux offers more protection than windows.	While it offers less safety than linux.
8.	Linux is extensively employed in hacking point based systems.	While windows does not offers much competence in hacking.

6.7 UNIX VS LINUX

Now, we will observe what is the distinction between linux and unix:

Key Differences	Linux	Unix
Cost	Linux is liberally dispersed, downloaded during magazines, Books, website, etc. There are waged versions also accessible for Linux.	Diverse flavors of Unix have dissimilar pricing depending upon the kind of vendor.
Development	Linux is Open Source, and thousands of programmers work together online and supply to its expansion.	Unix systems have dissimilar versions. These editions are primarily created by AT&T as well as other business vendors.
User	Everybody. From home customer to developers and computer aficionado alike.	The UNIX can be employed in internet servers, workstations, and PCs.
Text made	BASH is the Linux evasion shell. It presents support for numerous command	Initially prepared to job in Bourne Shell. Though, it is now

Key Differences	Linux	Unix
interface	interpreters.	well-suited with numerous others software.
GUI	Linux offers two GUIs,viz., KDE and Gnome. Although there are numerous substitutes such as Mate, LXDE, Xfce, etc.	General Desktop atmosphere and also has Gnome.
Viruses	Linux has had concerning 60-100 virus listed today which are at present not scattering.	There are among 80 to 120 viruses informed till date in Unix.
Threat detection	Threat finding and way out is extremely quick since Linux is chiefly community driven. So, if any Linux clients post any type of threat, a team of competent developers commence working to overcome this risk.	Unix client necessitate longer wait instance, to acquire the appropriate bug fixing patch.
Architectures	Originally urbanized for Intel's x86 hardware processors. It is obtainable for over twenty dissimilar kinds of CPU which also comprises an ARM.	It is obtainable on PA-RISC and Itanium machines.
Usage	Linux OS can be fitted on a variety of kinds of plans like mobile, tablet computers.	The UNIX operating scheme is employed for internet servers, workstations & PCs.
Best feature	Kernel renewed with no reboot	Feta ZFS - next age group file system DTrace - dynamic Kernel Tracing
Versions	Dissimilar editions of Linux are OpenSuse ,Redhat, Ubuntu, etc.	Dissimilar editions of Unix are BSD ,HP-UX, AIS, etc.
Supported file type	The File arrangement maintained by file kind like devpts, xfs, nfs, cramfsm ext 1 to 4, ufs, NTFS.	The File arrangement sustained by file kinds are vxfs, zfs, hfx, GPS, xfs.
Portability	Linux is moveable and is booted as of a USB Stick	Unix is not moveable

Key Differences	**Linux**	**Unix**
Source Code	The source is obtainable to the common public	The source code is not accessible to everyone.

6.8 TYPES OF FILES

In Linux and UNIX, the whole thing is a file. Directories are records; files are files, and tools like keyboard, Printer, mouse, etc. are files.

Let's look into the File types in more detail.

General Files

General Files also called as Ordinary files. They can include image, video, program or simply text. They can be in ASCII or a Binary format. These are the most commonly used files by Linux Users.

Directory Files

These files are a warehouse for other file types. You can have a directory file within a directory (sub-directory).You can take them as 'Folders' found in Windows operating system.

Device Files

In MS Windows, devices like Printers, CD-ROM, and hard drives are represented as drive letters like G: or H:. In Linux, these are represented as files. For example, if the first SATA hard drive had three primary partitions, they would be named and numbered as /dev/s*da1*, /dev/sda2 and /dev/sda3.

Note: All device files reside in the directory /dev/

All the above file types (including devices) have permissions, which allow a user to read, edit or execute (run) them. This is a powerful Linux/Unix feature. Access restrictions can be applied for different kinds of users, by changing permissions.

6.8.1 Linux Directory Structure

In **Linux**, everything is a file. A directory is nothing but a special file that contains details about all the files and subdirectories housed within it. The Linux contains following directories for different purpose:

i. **Root Directory (/)**
The **root directory** also represented as " / " is the topmost level of the system drive. Every other directory in your Linux system is located under the root directory. You may imagine the / directory similar to the C:\ directory or drive on your Windows system. This is not at all true but you may understand it as imaginary vision because

Linux doesn’t have drive letters. It is commonly used to represent a filesystem. A filesystem is the hierarchy of directories that is used to organize directories and files on a computer.

ii. **Essential User binaries (/bin)**

This folder contains the essential user programs which are also called as binaries whichare present when the system is mounted in single-user mode. It also contain the files for common commands likes ls, cat, cd, cp etc. The '/**bin**' **directory** also contains executable files

iii. **Static Boot Files (/boot)**

It contains the files which are required to boot the system such Linux kernel files, GRUB configuration files.

iv. **Historical mount point for CD-ROMs (/cdrom)**

/mnt/cdrom and /cdrom are the mount point directories for the CD-ROM drive. The /cdrom directory isn’t part of the Filesystem Hierarchy Standard (FHS), but you’ll still find it on Ubuntu and other operating systems.

v. **Device Files (/dev)**

In Linux, the /dev directory contains device and other special files. To see a list of these items, run the following command in a terminal session.

```
ls –l /dev
```

These include terminal devices, usb, or any device attached to the system.

vi. **Configuration Files (/etc)**

The /etc/ directory contains system-wide configuration files — user-specific configuration files are located in each user’s home directory.

vii. **Home Folder (/home)**

The home directory for each user takes the form /home/username (where username is the name of the user account). For example, if your user name is john, the system will have a home folder located at /home/john

viii. **Essential Shared Libraries (/lib)**

The lib folder is a library files directory which contains libraries needed by the essential binaries to be used by the system. These are supportive files which are used by an application or a process for their proper execution. The commands in /bin or /sbin dynamic library files are located just in this directory.

ix. **Recovered Files (/lost+found)**

Whenever a there is crashed in the file system, the system boots again and check is performed. Any file which got corrupted by crash will be found in lost+found folder.Every disk partition has a lost+found directory.

x. **Removable Media (/media)**

This Directory is used for mounting files systems on removable media like Zip drives, floppy drives, CD-ROM drives.

xi. **Temporary Mount Points (/mnt)**

This directory is used for temporarily mounted filesystem. For example, if you’re mounting a FAT partition for some operations, it might be mounted at /mnt/fat.

xii. **Optional Packages (/opt)**

It contains subdirectories for optional software packages which are written as a result of copy/install operations.

xiii. **Kernel and Process Files (/proc)**
It is a special directory for virtual filesystem. Instead of standard files it contains special files that represent system and process information.

xiv. **Root Home Directory (/root)**
This directory is distinct from / directory which is used for the system root directory. It is the home directory for root user. Instead of using the /home/root, it uses the location at /root.

xv. **Application State Files (/run)**
This folder contains data which describe the system since it has been booted. It gives applications a standard place to store temporary/transient files containing data related to sockets and process Ids.

xvi. **System Administration Binaries (/sbin)**
It contains files related to administrative commands. This directory includes essential binaries that might be needed to run by the root user for system administrator of the system.

xvii. **SELinux Virtual File System (/selinux)**
Ubuntu doesn't use SELinux. The /selinux directory contains special files used by SELinux for security (example Fedora and Red Hat). It is similar to /proc folder.

xviii. **Service Data (/srv)**
This holds the data for system services like Apache HTTP, FTP, etc.

xix. **Temporary Files (/ tmp)**
When the system or user executes the applications, the temporary files are stored in this folder. These temporary files are usually deleted automatically when the system gets started again.

xx. **User Binaries and Read Only Data (/usr)**
This directory is meant to contain the applications and files which belong to users, dissimilar to applications and files used by the system.

xxi. **Variable data Files (/var)**
This folder holds numerous system files which may be related to log, mail directories, print spool, etc. that may change over time in numbers and size.

6.8.2 Parent, Subdirectory, Home Directory

Root Directory (/) holds the top position as the main directory. All the directories mentioned in above section from sr. No. 2 to sr. No. 21 are the subdirectory of the root directory. /Home and /root are the home directories for the normal user and root user respectively.

6.8.3 Naming Rules for Directory and Files

1. The file names in the Linux are case sensitive. For example a filenamesarea.txt, Area.txt and AREA.txt all are considered as different files.

2. The filename can be created either by using uppercase letters, lowercase letters, numeric numbers, "_" (underscore) and "." (dot) symbols.

3. In Linux, other special characters such as blank space can also be used, but the rules for using other symbols are hard and it is heal their not to use them.

4. As mentioned in above point filenames may contain any character but one cannot use slash sign“/” (root directory), because it is reserved character and can only be used as the separator between files and directories while mentioning the pathname. One cannot use the null character.

5. .(dot) sign should not be used in the filename. Some time dot sign increases the readability of filenames but this may add confusion in understanding the extension of files. Dot sign should be used to identify the extension of the file. For example:

    - .tar.gz = zipped or Compressed files
    - .sh = used to identify the shell file

6. Older versions of the Unix operating system uses filenames upto 14 characters long. But in today’s era latest version of Linux and UNIX can use filename length upto 255 characters (255 bytes).

7. A filename should be unique in a particular directory. For example, in directory /home/john, one cannot give two file name as try.txt and try.txt. However, different directory can have same filenames inside them. For example, you can create try.txt in /tmp directory and try.txt inside /home/john directory.

6.9 PRACTICE QUESTIONS

Q1. What are the basic components of Linux?
Q2. Write down the name of some Linux variants.
Q3. Which popular office suite is available free for both Microsoft and Linux?
Q4. Suppose your company is recently switched from Microsoft to Linux and you have some MS Word document to save and work in Linux, what will you do?
Q5. What is the difference between Linux and Unix?
Q6. What is Linux Kernel? Is it legal to edit Linux Kernel?
Q7. What is the difference between BASH and DOS?
Q8. What are the process states in Linux?
Q9. What is the advantage of open source?

B.Sc.(DATA SCIENCE)

SEMESTER-II

OPERATING SYSTEM

UNIT VII: LINUX COMMANDS

STRUCTURE

7.0 OBJECTIVE

- Understanding Linux commands

7.1 OVERVIEW OF LINUX COMMANDS

The command line is one of the most powerful features of Linux. There are a countless number of commands in Linux. In this chapter, we will introduce few of most commonly and regularly used Linux commands for easy learning.

cal / ncal command

The **cal** command is used to print the calendar for a specified month and/or year on the monitor i.e. standard output device.

$cal: This will print the calendar for the current

$cal 2020: This command will print the twelve month calendar for the year 2020.

$cal 7 2020: This command will print the calendar July 2020.

mkdir command

This command is used to make the subdirectory under the current directory

$mkdir student: This command will create the directory student under the current directory

cd command

The cd command is used to change the current directory to the specified directory

$cd student: This command will change the current directory to student directory.

mv command

This command is used move the files from one folder to another folder.

$mv /home/john/try.txt /home/merry/. This command will move the try.txt file from /home/john directory to home/merry folder

$mv /home/merry/*.* /home/john/. This command will copy all the files in /home/merry folder to /home/john folder.

cp command

This command is used to copy the files from one folder to another folder.

$cp /home/john/try.txt /home/merry/. This command will copy the try.txt file from /home/john directory to home/merry folder

$cp /home/merry/*.cpp /home/john/. This command will copy all the files with extension .cpp from /home/merry folder to /home/john folder.

date command

The date command displays the current day, date, time, and year.

$ date: It will display current system date like “**Mon May 24 17:14:57 IST 2021**”

echo command

The echo command prints whatever you will write except quote. This command is used to insert the text on a file as shown below.

$echo "Welcome to The Linux Class": This command will print the line Welcome to The Linux Class.

rm command:

One can use “rm” command to delete any file or directory. To delete directories recursively we can use rm command along with -r , -R

$rm –r fname: This command will remove the folder with name fname.

$rm rea*.* : This command will remove all files starting with name prefix rea in current folder

rmdir command:

one can use rmdir to delete a directory. This command can only be used to delete an empty directory.

$rmdir dir_name: This command will remove the folder with name dir_name.

cat command:

cat is used to display the contents of the file whose name is mntioned along with the command. Example to see the text inside the file area.txt we may write the command as follows.

$cat area.txt: It will display the contents inside the filename area.txt

It is also used to merge the contents of the two files. The output is printed on the standard output device.

$cat filename1 filename2: It will display the contents of both the files together after merging.

pwd command

pwd command is used to display the name of present working directory. With the help of pwd command we are able to know about the directory in which we are working with. It gives us the absolute path, which means the path that starts from the root.

$pwd : It will the complete path of current directory. Example if you are working under user john under home directory then we will get output as /home/john.

who command

who command used to find out last system boot time, the system's current run level, List of the users who logged in the system and more. The different variations of who command are as follows:

Description	**Example**
The who command displays the following information for each user currently logged in to the system if no option is provided	$who
To display host name and user associated with standard input such as keyboard	$who -m –H
To show all active processes which are spawned by INIT process	$who -p –H
To show status of the users message as +, – or ?	$who -T –H
To show list of users logged in to system	$who –u
To show time of the system when it booted last time	$who -b –H
To show details of all dead processes	$who -d –H
To show system login process details	$who -l -H
To count number of users logged on to system	$who -q –H
To display current run level of the system	$who -r

To display all details of current logged in user	$who –a
To display system's username	$whoami
To display list of users and their activities	$ w
To display user identification information	$ id

pwd command

pwd command is used to display the name of present working directory. With the help of pwd command we are able to know about the directory in which we are working with. It gives us the absolute path, which means the path that starts from the root.

$pwd : It will the complete path of current directory. Example if you are working under user john under home directory then we will get output as /home/john.

ls command

This is called list command. It is used to display the files stored in a particular directory.

ls: To view the brief, multi-column list of the files in the current directory.

ls –a: To also see files starting with symbol "dot" (configuration files that begin with a period, such as .login).

ls –la: To see the permissions, owners and size of the files along with their names.

ls -la | less: If the listing is enough long that it is not fit into one screen view then reading becomes difficult. This command combines ls with the less utility for the same.

bc Command

This command is used for arbitrary precision CLI calculator which can be used like this: **$ echo 30.07 + 16.00 | bc**

more Command

This command enables us to view the large text data one screen display at a time.

gzip Command

gzip command is used to compress a file, replaces it with a file with .gz extension. The example of usage of gzip command is as shown below:

```
$ gzip passwds.txt

$ cat filename1 filename2 | gzip > try.gz
```

tar Command

This powerful utility command is a used for archiving files in Linux as follows.

```
$ tar -czf  demo.tar.gz
```

7.2 FILE PERMISSIONS IN LINUX

Linux is a **multi-user operating system** which means it can be accessed by many users simultaneously. The multi-user concern of Linux imposes security concerns as an unsolicited or **malign user** can **corrupt, change or remove crucial data**. Linux categorizes authorization into 2 levels.

1. Ownership
2. Permission

Ownership in Linux

Every file and directory in Linux operating system can be assigned 3 different types of owner, given below.

User

The person who creates the file is by default the owner of the file. Such user is the owner of the file. Therefore, a user is also sometimes called an owner.

Group

A group or user- group consists of multiple users. All the users which belong to a same group will have the same access permissions to the file. For example there is a project where the number of people/users needs access to a file. You could add all users to a single group and assign the permissions to the group instead of assigning the permissions to each user individually. No one other than the group member can read or modify the files.

Other

In this category, any general user who can access to a file is included in other category. This person has neither produced the file, nor do he / she belong to a particular usergroup who could own the file. Practically, it means everybody else. Therefore, this category is also called as the set permissions for the world.

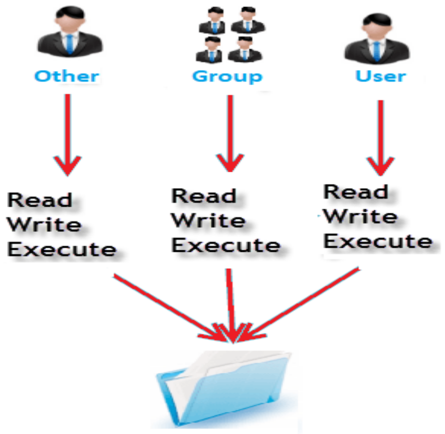
It is matter of the question that how Linux distinguishes between the three category of the users. One user 'A' cannot affect a file that contains the information of some other user 'B'. It means that you do not want another colleague in your organization to access your data. This is where the role of setting permissions comes in, and this also defines user behavior.

Let us understand the role of setting permissions in the Linux OS.

Permissions

Every file and directory in your Linux operating system has three types of permissions defined for all the 3 types of owners.

- **Read:** The read permission gives the authority to read as well as open the file. This permission on a directory gives you the capability to view its contents.
- **Write:** This permission gives a person the authority to modify the contents inside the file. You can rename, remove and add files present in the directory. For example if you don't have write permission on a folder but have the write permission on one of its file then you will be able to change the contents of the file but you will not be able to remove, move or rename the file placed inside the directory.
- **Execute:** Similar to Windows where files with ".exe" extension are executable. But in Linux OS, one cannot execute any file without having permission on it. If the execute permission is not granted on file, then you will not be able to execute the file. However you can see/modify the contents inside the file.



Categorization of Permissions that can be assigned to different types of Users

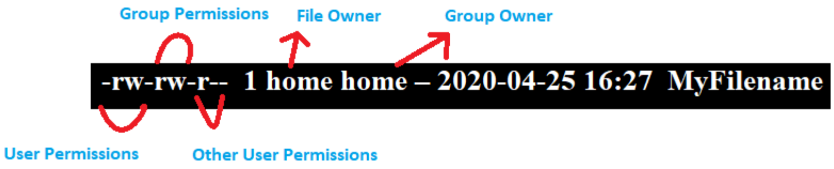
Let us see view this with the help of commands:

Issue command ls - l on terminal gives you the filenames with file types and access permissions.

```
ls – l
```

```
home$  ls –l
```

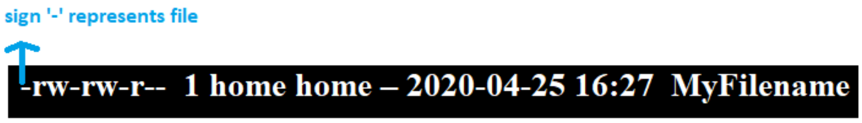
```
-rw-rw-r--  1 home home – 2020-04-25 16:27  MyFilename
```



```
-rw-rw-r--  1 home home – 2020-04-25 16:27  MyFilename
```

The string **'-rw-rw-r--'** in the output describes about the permissions given to the owner, user group and the world.

Here, the first '**-**' represents that this line indicates information about file

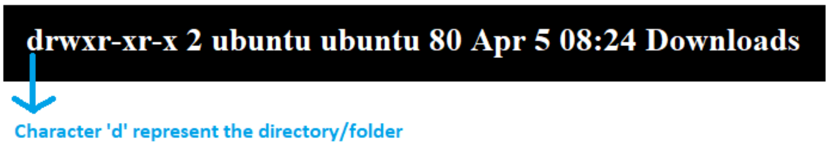


```
-rw-rw-r--  1 home home – 2020-04-25 16:27  MyFilename
```

If it could have been a directory then **d** would have been shown as below:

drwxr-xr-x 2 ubuntu ubuntu 80 Apr 5 08:24 Downloads

In the above output, the first character ‘d’ represent the directory.



```
drwxr-xr-x 2 ubuntu ubuntu 80 Apr 5 08:24 Downloads
```

The characters shown in the above example are easy to remember.

- = have no permission
x = the permission to execute
w = the permission to write
r = the permission to read
Let us explore the meaning of the output as follows:

The first part in the output line is **'rw-'**. This indicates that the owner 'home' can have two permissions i.e. read and write the file but the owner is not having the permission to execute that is why sign ‘-‘ has been used.

However, many distributions of the Linux like Ubuntu, CentOs and Fedora etc. can add users to the same group name which is the name of the user. Therefore a username 'john' may be added to a group having name as 'john'.

The second part of the output is **'rw-'** which indicates that group having named as ‘**home’** have the permission to read and write the file but don’t have the permission to edit the file**.**

Third part is used to represent the permissions for the other person from the outside world. It means there could be any other user. The string **'r--'** indicates that other users have the permission to only read the file but these persons may not be able to write and execute the file.

Using ‘chmod’ command to change the directory/file permissions

Permissions help you to restrict the others in performing operations on files/directories in form of read, write and execute. This can be accomplished by changing permissions on files/directories.

We can use the '**chmod'** command which stands for 'change mode'. Using the command, we can set permissions (read, write, execute) on a file/directory for the owner, group and the world. The Syntax of command is as follows:

chmod permissions filename

There are 2 types of modes for using chmod command -

1. **Absolute mode**
2. **Symbolic mode**

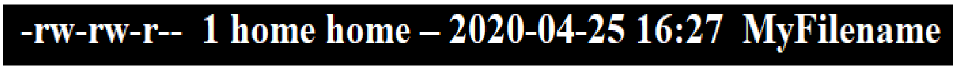
Absolute Mode / Numeric Mode

In this technique/mode, the permissions on files are represented as three-digit octal number instead as characters. The following table represents the different numbers for each type of permissions.

Number	**Permission Type**	**Symbol**
0	No Permission	---
1	Execute	--x
2	Write	-w-
3	Execute + Write	-wx
4	Read	r--
5	Read + Execute	r-x
6	Read +Write	rw-
7	Read + Write +Execute	Rwx

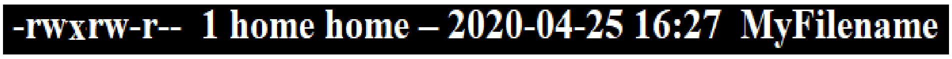
Let us look at the examples of using chmod command.

Command **“ls –l MyFilename”** shows the current permissions on the MyFilename



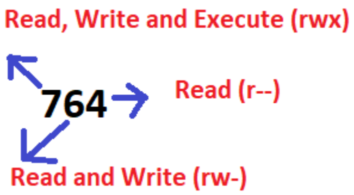
```
-rw-rw-r--  1 home home – 2020-04-25 16:27  MyFilename
```

If we issue the command **"chmod 764 MyFilename"** then we will see the following results



```
-rwxrw-r--  1 home home – 2020-04-25 16:27  MyFilename
```

In the above-given terminal window, we have changed the permissions of the file 'sample to '764'.



'764' absolute coding can be interoperated as follows:

- Owner can read, write and execute
- User group can read and write
- World can only read

This can be seen in the output as '-rwxrw-r-

In this way; you can change the permissions on file by assigning an absolute number.

Symbolic Mode

In the Absolute mode, you change permissions for all 3 owners. In the symbolic mode, you can modify permissions of a specific owner. It makes use of mathematical symbols to modify the file permissions.

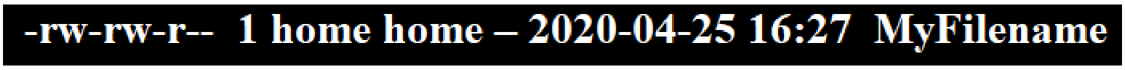
Operator	**Action of applying the operator**
+	Adds the particular permission to the file / directory
-	Removes the particular permission from file / directory
=	Sets the new permission by overriding the previous permissions.

The various types of owners of file and directories are given as follows:

User Denotations	
U	user/owner
G	Group
O	Other
A	All

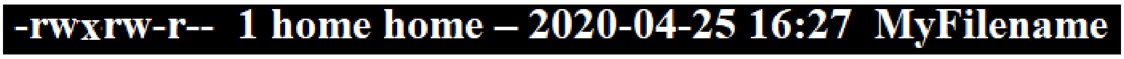
In symbolic mode we shall be setting permissions by making use of characters like rwx unlike numbers in absolute mode eg 764. Let us apply example for symbolic mode using the previous file permissions on MyFilename:

Command **“ls –l MyFilename”** shows the current permissions on the MyFilename



```
-rw-rw-r--  1 home home – 2020-04-25 16:27  MyFilename
```

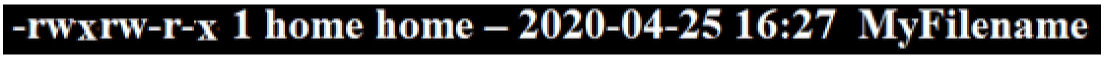
After applying the command **“Chmod u = rwx MyFilename”**, we shall get the following result:



```
-rwxrw-r--  1 home home – 2020-04-25 16:27  MyFilename
```

In above command “u” stands for user. It means new permissions will be set for the user. The command will assign the user permissions of read, write and execute to the MyFilename.

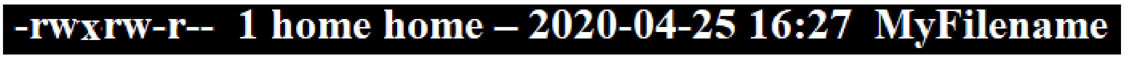
Let us apply another command as **“chmod o+x MyFilename”**. Then the permissions will be changed as follows.



```
-rwxrw-r-x 1 home home – 2020-04-25 16:27  MyFilename
```

String **“o+x”** in above command will add the execute permission for the other user.

Let us again apply command as **“chmod o-x MyFilename”**. Then the execute permission which was given just now will be removed. The output of **“ls –l MyFilename”** command will look as follows after applying the command.



```
-rwxrw-r--  1 home home – 2020-04-25 16:27  MyFilename
```

7.3 CHANGE IN OWNERSHIP AND THE ASSOCIATED GROUP

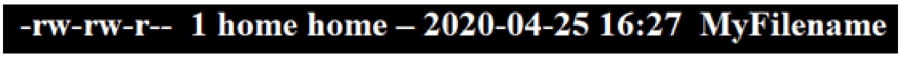
To change the ownership of particular file/directory, we may use the **“chown”** command as follows:

chown new_user_name

If there is need to change the group along with the user for a file or directory, then we may use the following command

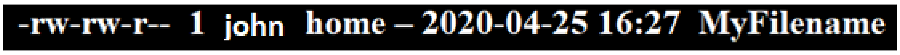
chown new_user_name:new_group_name filename

Let us look at the output of **“ls – l MyFilename”** command



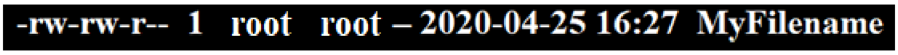
```
-rw-rw-r--  1 home home – 2020-04-25 16:27  MyFilename
```

Now apply the “**chown john MyFilename”** command and then apply **“ls – l MyFilename”** command. It you are not logged in as root user then you may use **“sudo chown john MyFilename”** command to act as super user. The system may ask you the password for the super user in such case.



```
-rw-rw-r--  1  john  home – 2020-04-25 16:27  MyFilename
```

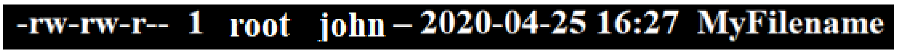
Let us change the user and group both to root by applying **“sudo chown root:root MyFilename”** command. The output of the file may look as follows after applying the command.



```
-rw-rw-r--  1   root   root – 2020-04-25 16:27  MyFilename
```

If you want to change only the group of the user then may also apply **“chgrp”** command. Command “**chgrp”** stands for change group.

Let us apply the command **“sudo chgrp john MyFilename”** to change the group from root to john. Then we will get output as follows:



```
-rw-rw-r--  1   root   john – 2020-04-25 16:27  MyFilename
```

<u>7.4 ADVICE BEFORE USING COMMANDS</u>

- The file stored in the **/etc/group** contains the name of all the groups defined in the system at any moment of time.
- We may use the command **"groups"** on terminal to find the name of all the groups where we are a members.
- We may use the command **“newgrp”** to work as a member a group other than your default group
- No file/directory can own to two groups simultaneously.
- Linux do not have nested groups.
- x- eXecuting a directory means Being allowed to "enter" a dir and gain possible access to sub-directories.

7.5 SUMMARY

- Linux being a multi-user system uses permissions and ownership for security.
- There are three user types on a Linux system viz. User, Group and Other
- Linux divides the file permissions into read, write and execute denoted by r,w, and x
- The permissions on a file can be changed by 'chmod' command which can be further divided into Absolute and Symbolic mode
- The 'chown' command can change the ownership of a file/directory. Use the following commands: chown user file or chown user:group file
- The 'chgrp' command can change the group ownership **chrgrp group filename**
- What does x - executing a directory mean? A: Being allowed to "enter" a dir and gain possible access to sub-dirs.

7.6 PRACTICE QUESTIONS

Q1.What are the basic commands for user management?

Q2.Which command is used to uncompressed grip files?

Q3.What are the modes used in VI editor?

Q4.What are the file permissions in Linux?

Q5.Which are the Linux Directory Commands?

Q6.What are inode and process id?

Q7.Explain Process Management System Calls in Linux

Q8.Explain the redirection operator.

Q9.How to copy a file in Linux?

Q10. How to terminate a running process in Linux?

Q11. How to create a new file or modify an existing file in vi?

Q12. Explain Regular Expressions and Grep

B.Sc.(DATA SCIENCE)

SEMESTER-II

OPERATING SYSTEM

UNIT VIII: SHELL SCRIPTING

STRUCTURE

8.0 OBJECTIVE

To learn the usefulness of shell scripting

8.1 INTRODUCTION TO SHELL

- The shell is a user program or it is an environment provided for user interaction.
- It is a command language interpreter that executes commands read from the standard input device such as keyboard or from a file.
- The shell gets started when you log in or open a console (terminal).
- Quick and dirty way to execute utilities.
- The shell is not part of system kernel, but uses the system kernel to execute programs, create files etc.

Different shells are available in Linux which are as under:

- **BASH (Bourne-Again SHell)** - Most common shell in Linux. It's Open Source.
- **CSH (C SHell) -** The C shell's syntax and usage are very similar to the C programming language.
- **KSH (Korn SHell) -** Created by David Korn at AT & T Bell Labs. The Korn Shell also was the base for the POSIX Shell standard specifications.
- **TCSH** - It is an enhanced but completely compatible version of the Berkeley UNIX C shell (CSH).

8.2 SHELL SCRIPTING

A shell script is just a normal Linux file which contains Linux and shell commands. The simplest shell scripts simply group together commonly used sequences of commands.

More complex scripts use the shell's programming syntax to perform more advanced tasks. Shell scripts are not compiled but interpreted. This means that each time they are run a shell is executed to read the file and run the commands it contains. Languages which are compiled, such as C, will produce a compiled code which will run faster than an equivalent shell script.

Shell scripts usually begin with a #! and a shell name.
For example: #!/bin/sh
If they do not, the user's current shell will be used

8.3 SHELL PROMPT

There are various ways to get shell access:

- **Terminal** - Linux desktop provide a GUI based login system. Once logged in you can gain access to a shell by running X Terminal (XTerm), Gnome Terminal (GTerm), or KDE Terminal (KTerm) application.
- **Connect via secure shell (SSH) -** You will get a shell prompt as soon as you log in into remote server or workstation.
- **Use the console** - A few Linux system also provides a text-based login system. Generally you get a shell prompt as soon as you log in to the system.

You may type the following into the Terminal to find out the shells available in your system.

cat /etc/shells

Why write shell scripts?

– To avoid repetition: If you do a sequence of steps with standard Linux commands over and over, why not do it all with just one command?
– To automate difficult tasks: Many commands have subtle and difficult options that you don't want to figure out or remember every time.
– Creating your own power tools/utilities.
– Automating command input or entry.
– Customizing administrative tasks.
– Creating simple applications.
– Since scripts are well tested, the chances of errors are reduced while configuring services or system administration tasks such as adding new users.

Advantages

- Easy to use.
- Quick start, and interactive debugging.
- Time Saving.
- Sys Admin task automation.
- Shell scripts can execute without any additional effort on nearly any modern UNIX / Linux / BSD / Mac OS X operating system as they are written an interpreted language.

Disadvantages

- Compatibility problems between different platforms.
- Slow execution speed.
- A new process launched for almost every shell command executed.

8.4 VARIABLE NAMES

Variables names may comprise upper and lower case alphabetic characters, digits and underscores. A user defined variable name cannot start with a digit. As with most things in Unix variable names are case sensitive, e.g. FOO and foo are two distinct variables.

In Linux, there are two types of variable
1) System variables - Created and maintained by Linux itself. This type of variable defined in CAPITAL LETTERS.
2) User defined variables (UDV) - Created and maintained by user. This type of variable defined normally by using lower case LETTERS.

Some System variables

You can see system variables by giving command like $ set, Some of the important System variables are

System Variable	**Meaning**
BASH=/bin/bash	Our shell name
BASH_VERSION=1.14.7(1)	Our shell version name
COLUMNS=80	No. of columns for our screen
HOME=/home/vivek	Our home directory
LINES=25	Number of columns for our screen
LOGNAME=students	Our logging name
OSTYPE=Linux	Our o/s type : -)
PATH=/usr/bin:/sbin:/bin:/usr/sbin	Our path settings
PS1=[\u@\h \W]\$	Our prompt settings
PWD=/home/students/Common	Our current working directory
SHELL=/bin/bash	Our shell name
USERNAME=vivek	User name who is currently login to this PC

Special Shell Variables

Some variables have special significance to the shell. For example, the PATH variable is used by the shell to contain the list of directories to search for commands. Other variables have special meaning to other Linux utilities, for example the TERM variable contains the current terminal type.

```
bash$ variable1=23
```

Use echo command to display variable value.

```
bash$ echo variable1
        variable1
bash$ echo $variable1
        23
```

To display the program search path, type:

```
bash$ echo "$PATH"
```

To display your prompt setting, type:

```
bash$ echo "$PS1"
```

All variable names must be prefixed with $ symbol, and the entire construct should be enclosed in quotes. Try the following example to display the value of a variable without using $ prefix:

```
bash$ echo "HOME"
```

To display the value of a variable with echo $HOME:

```
bash$ echo "$HOME"
```

You must use $ followed by variable name to print a variable's contents.

The variable name may also be enclosed in braces:

```
bash$ echo "${HOME}"
```

This is useful when the variable name is followed by a character that could be part of a variable name:

```
bash$ echo "${HOME} work"
```

To define User Defined Variable (UDV) use following syntax

Syntax: *variablename=value*

NOTE: Here 'value' is assigned to given 'variablename' and Value must be on right side = sign For e.g.

```
$ no=10 # this is ok
$ 10=no # Error, NOT Ok, Value must be on right side of = sign.
```

To define variable called 'vech' having value Bus

```
$ vech=Bus
```

To define variable called n having value 10

```
$ n=10
```

Q.1.How to Define variable x with value 10 and print it on screen

```
$ x=10
$ echo $x
```

Linux Shell Script Tutorial

Q.2.How to Define variable xn with value Rani and print it on screen

```
$ xn=Rani
$ echo $xn
```

Q.3.How to print sum of two numbers, let's say 6 and 3

```
$ echo 6 + 3
```

This will print 6 + 3, not the sum 9, To do sum or math operations in shell use expr, syntax is as follows Syntax: *expr op1 operator op2*

Where, op1 and op2 are any Integer Number (Number without decimal point) and operator can be

+ Addition
- Subtraction
/ Division
% Modular, to find remainder For e.g. 20 / 3 = 6 , to find remainder 20 % 3 = 2, (Remember its integer calculation)
* Multiplication

```
$ expr 6 + 3
```

Now It will print sum as 9 , But

```
$ expr 6+3
```

will not work because space is required between number and operator (See Shell Arithmetic)

Q.4.How to define two variable x=20, y=5 and then to print division of x and y (i.e. x/y)

```
$x=20
$ y=5
$ expr x / y
```

Q.5.Modify above and store division of x and y to variable called z

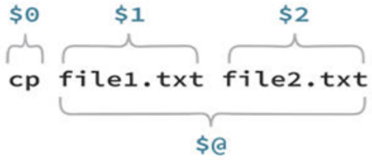
```
$ x=20
$ y=5
$ z=`expr x / y`
$ echo $z
```

Note : For third statement, read Shell Arithmetic.

8.5 POSITIONAL PARAMETER / COMMAND LINE ARGUMENTS

Command line arguments are important part of writing scripts. Command line arguments define the expected input into a shell script. For example, we may want to pass a file name or folder name or some other type of argument to a shell script.

A **positional parameter** is a variable within a **shell** program; its value is set from an **argument** specified on the **command** line that invokes the program. **Positional parameters** are numbered and are referred to with a preceding ``$": $1, $2, $3, and so on. A **shell** program may reference up to nine **positional parameters**.



Consider the following bash command. Let the command name is mycommand. The command line has three parameters: one, two, and three four.

$ mycommand one two “three four”

Variable name	Value
$0	Mycommand
$1	One
$2	Two
$3	three four
$#	3
$@	one two three four
$*	one two three four
$!	This parameter represents the process number of the background that was executed last.
$?	This parameter represents exit status of the last command that was executed. Here 0 represents success and 1 represents failure.
$_	This parameter represents the command which is being executed previously.
$-	This parameter will print the current options flags where the set command can be used to modify the options flags.

```
$ cat  program.sh
echo "The File Name is: $0"
echo "The First argument is: $1"
echo "The Second argument is: $2"

$ sh program.sh ab cd
The File Name: program.sh
The First argument is: ab
The Second argument is: cd
```

If any arguments are supplied, they become the positional parameters when filename is executed. Otherwise, the positional parameters remain unchanged.

if-then-else

Like most programming languages, shell script supports the if statement, with or without an else. The general form is below:

```
if (condition)
        simple commands

if (condition)
then
        commands
        …
fi

if (condition)
then
        commands
…
Else
        Commands
…
fi
```

You can have nested if-else-fi

The *condition* used as the predicate can be any program or expression. The results are evaluated with a 0 return being true and a non-0 return being false.

If ever there is the need for an empty if-block, the null command, a :, can be used in place of a command to keep the syntax legal.

The following is a nice, quick example of an if-else:

```
if [ "$LOGNAME"="guna" ]
then
        printf "%s is logged in" $LOGNAME
else
        printf "Intruder! Intruder!"
fi
```

The elif construct

Shell scripting also has another construct that is very helpful in reducing deep nesting. It is unfamilar to those of us who come from languages like C and Perl. It is the elif, the "else if". This probably made its way into shell scripting because it drastically reduces the nesting that would otherwise result from the many special cases that real-world situatins present -- without functions to hide complexity (shell does have functions, but not parameters -- and they are more frequently used by csh shell scripters than traniditonalists).

```
if condition
then
        command
        command
        ...
        command
elif condition
then
        command
        command
        ...
        command
elif condition
then
        command
        command
        ...
        command
fi
```

The switch statement

Much like C, C++, or Java, shell has a case/switch case statement. The form is as follows:

```
case “$variable” in

pattern1)
command
```

```
command
...
command
;;   # Two ;;'s serve as the break
Pattern2)
command
command
...
command
;; # Two ;;'s serve as the break
Pattern3)
command
command
...
command
;;  # Two ;;'s serve as the break
esac
```

Here's a quick example:

```
#!/bin/sh
case "$char"
in
"+")
        ans=`expr $1 + $3`
        printf "%d %s %d = %d\n" $1 $2 $3 $ans
        ;;
"-")
        ans=`expr $1 - $3`
        printf "%d %s %d = %d\n" $1 $2 $3 $ans
        ;;
"\*")
        ans=`expr "$1 * $3"`
        printf "%d %s %d = %d\n" $1 $2 $3 $ans
        ;;
"/")
        ans=`expr $1 / $3`
        printf "%d %s %d = %d\n" $1 $2 $3 $ans
        ;;
# Notice this: the default case is a simple *
*)
        printf "Don't know how to do that.\n"
        ;;
esac
```

The for Loop

The for loop provides a tool for processing a list of input. The input to for loop is a list of values. Every iteration through the loop extracts one value into a variable and then enters the body of the loop. The loop stops when the extract fails because there are no more values in the list.

for loop operates on a list and repeats commands in the block for each element on the list.

```
for x in [ list ]
do
        commands
done
```

Let's consider the following example which prints each of the command line arguments, one at a time. We'll extract them from "$@" into $arg:

```
for var in "$@"
do
        printf "%s\n" $var
done
```

Much like C or Java, shell has a *break* command, also. As you might guess, it can be used to get out of a loop. Consider this example which stops printing command line arguments, when it gets to one whose value is "quit":

```
for var in "$@"
do
        if [ "$var" = "quit" ]
then
        break
fi
        printf "%s\n" $var
done
```

Similarly, shell has a *continue* that works just like it does in C or Java.

```
for var in "$@"
do
        if [ "$var" = "me" ]
then
        continue
elif [ "$var" = "mine" ]
then
        continue
elif [ "$var" = "myself" ]
```

```
then
        continue
fi
if [ "$var" = "quit" ]
then
        break
fi
printf "%s\n" $var
done
```

The while and until Loops

Shell has a while loop similar to that seen in C or Java. It continues until the predicate is false. And, like the other loops within shell, break and continue can be used. Here's an example of a simple while loop:

```
# This lists the files in a directory in alphabetical order

# It continues until the read fails because it has reached the end of input

ls | sort |

        while read file
        do
                echo $file
        done
```

In the above code, | called a “pipe” directs output from one process to another process. For

example, ls | sort takes the output from ls command, and sort the output stream received. Pipe is a form of inter process communication which we will discuss later.

There is a similar loop, the *until* loop that continues *until* the condition is successful -- in other words, while the command failes. This will pound the user for input until it gets it:

```
        printf "ANSWER ME! "
        until read $answer
        do
                printf "ANSWER ME! "
        done
```

8.6 PRACTICE QUESTIONS

Program1: Write a shell script program to display “HELLO WORLD”.

Program2: Write a shell script to find the factorial of given integer

Program3: Write a shell Script program to check whether the given number is even or odd.