SYLLABUS

Subject : Operating Systems **Marks of FA : 20**

Subject Code : C.A-303 **Marks of SA : 80**

Periods/Week : 04

Total No. of Periods : 60

S.No.	Major Topics	No. of Periods	CO's Mapped
1.	Introduction to Operating System	12	CO1
2.	Process Management	12	CO2, CO6
3.	Synchronization and Deadlocks	12	CO3, CO6
4.	Memory Management	12	CO4, CO6
5.	Disk Scheduling and File Management	12	CO5, CO6
	Total	**60**	

LEARNING OUTCOMES

CHAPTER-1

1.0 Introduction to operating systems

1.1 Define an operating system

1.2 Discuss history of operating system

1.3 Discuss about various types of operating systems

1.4 Distinguish spooling and buffering

1.5 Explain the concepts multiprogramming and timesharing

1.6 Differentiate between distributed and real time systems

1.7 Describe multiprocessor systems

1.8 Describe the operating system components

1.9 Discuss operating system services

1.10 Define system call with an example

1.11 List and explain different types of system calls

1.12 Define single user, multi user operating system structure

CHAPTER-2

CHAPTER-3

CHAPTER-4

5.7 List and explain various access methods

5.8 List and explain various allocation methods

5.9 List and explain directory structure

5.10 Explain disk organization and structure

MODEL BLUE PRINT

S.No.	Chapter/ Unit Title	No. of periods	Weightage Allocated	Marks Wise Distribution of Weightage				Question Wise Distribution of Weightage				CO's Mapped
				R	U	Ap	An	R	U	Ap	An	
1.	Introduction to Operating system	12	14	6	8			2	1			CO1
2.	Process management	12	14	6	8			2	1			CO2
3.	Synchronization & Deadlocks	12	24	3	3	8	*	1	1	1	*	CO3
4.	Memory management	12	14	6	8			2	1			CO4
5.	Disk scheduling and File management	12	14	3	1 1		*	1	2		*	CO5
	Total	**60**	**70 +10 ***	**24**	**38**	**8**	**10 ***	**8**	**6**	**1**	**1**	

CONTENTS

CHAPTER-1

INTRODUCTION TO OPERATING SYSTEM 1.1 – 1.26

CHAPTER-2

PROCESS MANAGEMENT 2.1 – 2.24

CHAPTER-4

MEMORY MANAGEMENT 4.1 – 4.36

CHAPTER

1

INTRODUCTION TO OPERATING SYSTEMS

CHAPTER OUTLINE

1.0 INTRODUCTION

As you know computer is capable of doing so many thing. However, a computer is useless without software. In association with software computer can store, process and retrieve information.

Software can be divided into Two Types. They are :

- System softwares of system programs.
- Application software.

System software purpose is to improve the performance of the system. Example for system software are operating system, compiler, interpreters, linkers, loaders, assemblers etc. The purpose of these software's is to ease the task of computing. Generally these programs are not developed by the user. Application programs are developed for a particular application. For example payroll package is developed to calculate salary details of employees of an organization. Application programs will be written by users. Let us assume that you are writing a "C" program, the compiler you are using to run this program is an example for system software. The "C" program which you have prepared is an example for application program.

The most important system software is the operating system, which is present in all computers. Operating system is the first software we see when turn on the computer and the last software we see when the computer is turned off.

A modern computer system consists of one or more processor, some main memory, disks, printers, network cards, and other input / output devices. If it is the responsibility of the programmer to control the working of hardware then task of programming will become complex. So system designers thought of protecting the programmer from the complexity

of the hardware. The way that was evolved is to put a layer of software on top of hardware to manage all parts of the system. This layer of software is operating system. This is shown in following Fig. 1.1.

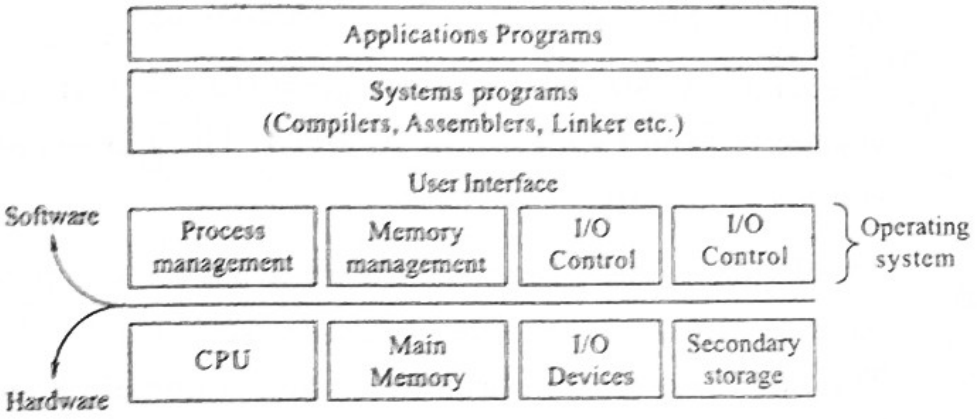


FIG 1.1 : Operating System

1.1 DEFINE OPERATING SYSTEM

Operating system can be defined as *"system software which acts as interface between user and the system and manages the resources in efficient way"*.

The primary goal of an operating system is :

- To make the computer system easy to use.
- Secondary goal is hardware resource management.

The operating system is the most basic program in a computer. All computers have an operating system that us used for starting the computer and running other programs (application programs). The operating system performs important tasks like receiving input from the keyboard and mouse, sending information to the screen, keeping track files and directories on the disk, as well as controlling the various units such as disk printers etc.

1.2 HISTORY OF OPERATING SYSTEM

The history of operating system is closely linked with the history and development of various generations of computer systems.

1940s and 1950s : The first generation of computers was a time of vacuum tube technology and computers are bulky. Each computer was unique in structure and purpose. There was a little need for standard operating system because computer was restricted to few professionals.

The General Motors Labs implemented the first operating system in early 1950s for the IBM 701. The systems of 1950s run only one job.

The 1960s : The systems of the 1960 were batch processing systems. They are capable of running several jobs at a once. The operating system supported many peripheral devices like tape drives, card readers etc.

IBM introduced 0S/360 for its 360 family of computers during this period. Time sharing systems were developed.

The 1970s : The operating systems of 1970s were time sharing that support batch processing, multi tasking and real time applications. The time sharing systems which were in initialstage during 1960s were advanced and they were commercially successful. Personal computing was in its initial stage.

The 1980s : The 1980 was the decade of personal computers and the workstation. During this stage computing was distributed rather than concentrated at the central computer. Real time systems were developed during this stage.

The 1990s and Beyond : During this stage graphical user interface system developed. The purpose of graphical user interface system is to provide simple interface to the user.

During this stage network and distributed operating systems are developed. Multimedia supported operating are another milestone in this period.

1.3 VARIOUS TYPES OF OPERATING SYSTEMS

Operating Systems can be classified in number of ways :

Based on the type of interface they provide OSs are classified as Graphical User Interface Systems or Character Based Systems.

In GUI system user interfacing is through a set of icons. Examples for GUI system Windows, Macintosh and Linux.

In character based system, interfacing is through a set of commands. Examples for this type of systems are UNIX, DOS

Operating systems are classified as single user or multi user operating systems based on no of users it support simultaneously.

Single User Systems are Classified as :

- Single User – Single Task
- Single User – Multi Tasking

Single User – Single Task : as the name indicates, this operating system is designed to manage the computer so that one user can effectively do one thing at a time.

Single User – Multi Task : This is the type of operating system most people use of their desktop and laptop computers today. Microsoft's Windows and Apple's Mac OS are example of operating systems that allow a user to initiate several programs in operation at the same time. Generally multi user operating systems are multi tasking.

Operating systems can be classified into the following based on response time and how data is entered into system.

- Batch
- Real time.
- Hybrid.

1. **Batch Operating System :** These operating systems were used with the older computers. The jobs are executed in batches. In a batch operating system environment users submit jobs to a central place where these jobs are collected into a batch, and subsequently placed on an input queue at the computer where they will run. The user submits the jobs and these jobs queued up and executed in a batch one after other, so the name batch operating system came.

2. **Time Sharing :** Time sharing as the name suggests, causes the CPU time to be shared among many users. Thus multiple users can be connected to a computer and the operating system moves from one users program / commands to another in quick succession. The turn around time is so short that every user gets a feeling that he is the only user connected to the system at that time.

3. **Real Time Operating Systems (RTOS) :** Real time operating systems are used to control machinery, scientific instruments, industrial systems and used in special applications like air port traffic control, space shuttle etc. The applications expects an immediate response from the computer. In this type of situation general operating systems are not suitable because they take some time to respond for user actions. A very important part of and RTOS is managing the resources of the computer so that a particular operation executes in precisely the same amount of time every time it occurs.

Hybrid systems are combination of batch and time sharing.

1.4 CONCEPT OF BUFFERING AND SPOOLING

What is buffer ? A buffer is an area of main memory for holding data during iput and output data transfers.

The buffering technique is used to reduce difference in speed between input/output device and the CPU.

The input/output devices are electro mechanical devices and their speed is very slow when compared the speed of the CPU. So to handle mismatch in speeds buffering technique is introduced.

Buffer is an interim storage area. When input device reads some data, the data is placed in the buffer and when read operation is over the data which is in buffer is transferred to the CPU at very faster rate. If data is to be transferred to the output device CPU places the data in the buffer and from the buffer data will go to output device.

Buffering can also be used for data transfer between two devices.

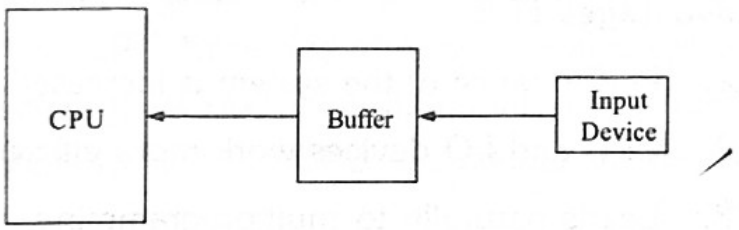


FIG 1.2 : Data Transfer for Input Operation with Buffering Technique

Advantage with Buffering : CPU is not idle during data transfer.

Spooling :

- Simultaneous Peripheral Operation On-Line.
- A spool is a buffer that holds output for a device, such as a printer, that cannot accept interleaved ata streams.
- Spooling overlaps input of one job with the computation of other jobs. For example, spooler may be reading the input of one job while printing the output of another and executing a third job.
- The spooler may be reading the input of one job while printing the output of a different job as shown in Fig. 1.3.

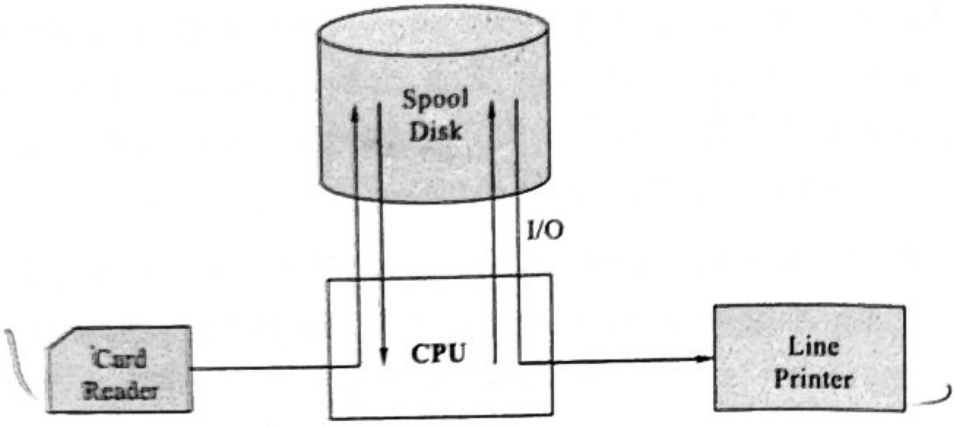


FIG 1.3 : Spooling

- Spooling increases the performance of the system by allowing both a faster CPU and slower I/O devices to work at higher operating rates.

Advantages :

1. Performance of the system is increased.
2. CPU and I/O devices work more efficiently.
3. Leads naturally to multiprogramming.
4. Also used for processing data at remote sites.

1.5 MULTIPROGRAMMING AND TIME SHARING

Multiprogramming : A multiprogramming operating system is a system that allows more than one user program (or part of user program) to be stored in main memory simultaneously. In multi programming number of process reside in main memory at a time. The operating system picks up and begins to executes one of the jobs in the main memory.

Operating System
Program 1
Program 2
Program 3
Program 4

FIG 1.4 : Multi Programming

Following Fig. 1.4 shows the layout of the multiprogramming system.

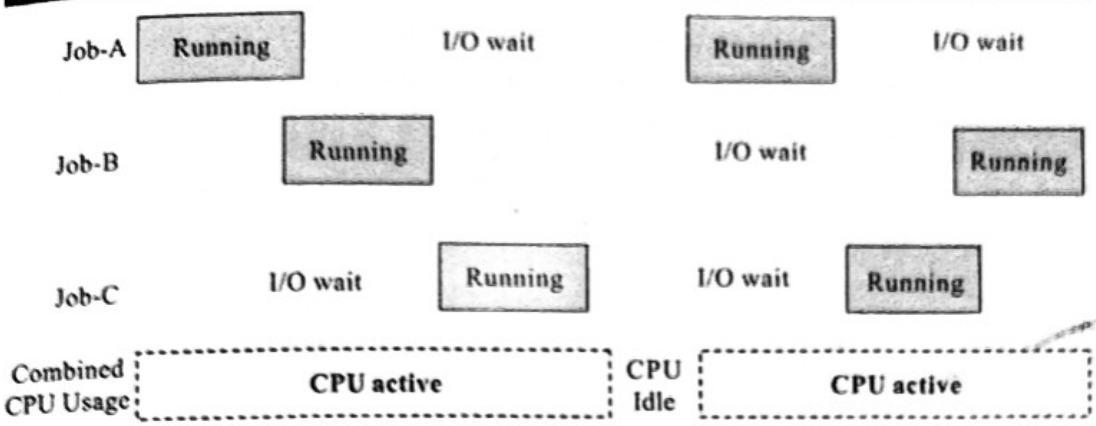


FIG 1.5 :

In the above Fig. 1.5 main memory consists of 4 programs (jobs), the CPU executes one by one.

In non-multiprogramming system, the CPU can execute only one program at a time, if the running program waiting for any I/O, the CPU becomes idle, so it will effect on the performance of the CPU. In multiprogramming systems, any program waits for I/O the CPU switches to another waiting program.

Advantages of Multiprogramming :

- No memory wastage. Efficient memory utilization.
- CPU is never idle, so the performance will increase.

Time Sharing Systems :

- Time sharing is the logical extension of Multiprogramming.
- Time sharing executes multiple jobs by switching among them.
- Time sharing as the name suggests, causes the CPU time to be shared among many users. Thus, multiple users can be connected to a computer and the operating system moves from one user program/ commands to another in quick succession.

- The turn around time is so short that every user gets a feeling that he is the only user connected to the system at that time.

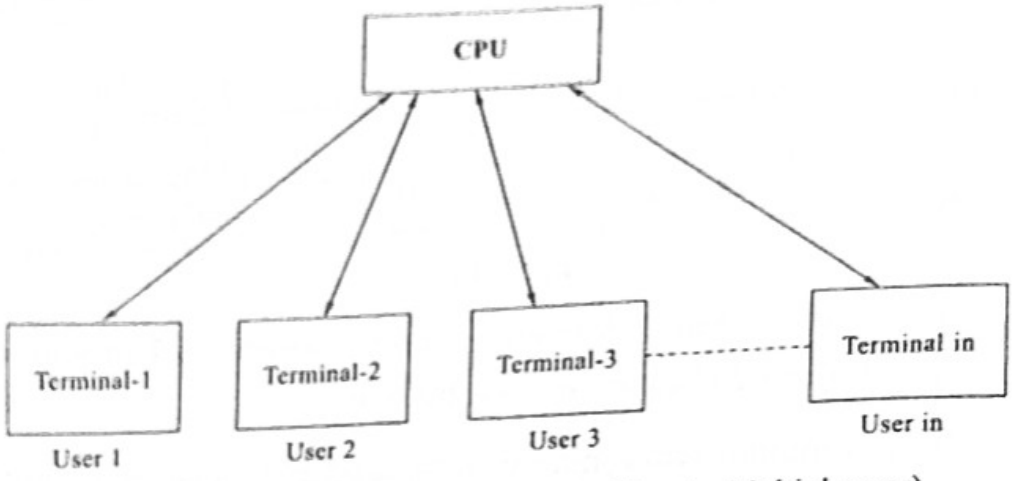


FIG 1.6 : Time Sharing System (Multitasking for Multiple user)

1.6 DISTRIBUTED SYSTEM AND REAL TIME SYSTEMS

Distributed System :

- Distributed systems are loosely coupled systems.
- A distributed computer system is a collection of autonomous computer systems
- Distributed systems communicate with one another through various communications lines, like high speed buses or telephone lines.
- Distributed computer systems developed from computer network.
- Distributed system makes the whole network transparent to the users.
- The database, files, printers and other resources are shared among a number of users actually working on different machines.
- Distributed systems allow parallelism i.e., a program can be divided into different tasks which can run on different machines.

- Distributed systems provide high levels of fault tolerance. So that one computer is down, the operating system could schedule the task to other computers. They are also known as loosely coupled systems.

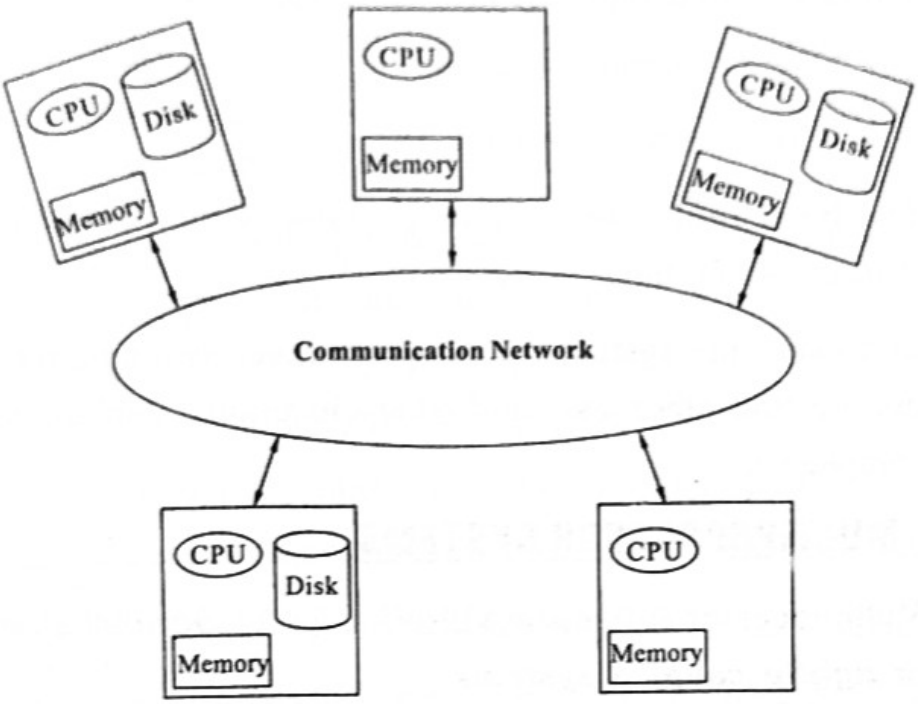


FIG 1.7 : Architecture of a Distributed System

Real Time Systems :

- Real time systems has well defined, fixed time constraints.
- Processing must be done with in the defined constraints, or the system will fail. ***For example :*** It would not do for a robot arm to be instructed to halt after it has smashed into the car it was building.
- Real time operating systems are used to control machinery, scientific instruments, industrial system and used in special applications like air port traffic control, space shuttle etc.
- The applications expects an immediate response from the computer. In this type of situation general operating systems are not suitable because they take some time to respond for user actions.

- A important part of an RTOS is managing the resources of the computer so that a particular operation executes in precisely the same amount of time every time it occurs.

There are two types of real time systems :

- Hard real time systems.
- Soft real time systems.

The **hard real time system** guarantees the completion of critical task on time.

Soft real time systems are less restrictive. Real time task gets priority over other tasks and retains its priority until the task is complete.

1.7 MULTIPROCESSOR SYSTEMS

Multiprocessor systems are also known as ***parallel systems*** or ***tightly coupled systems***.

Multiprocessor system is a system with more than one CPU working in close communication with one another.

Processors share memory and a clock; communication usually takes shared memory.

Advantages :

- **Increased through put :** More work will be done in less time.
- **Economical :** Multi processor systems can save more money than multiple single processor systems, because they share peripherals, mass storage, and power supplies.
- **Increased Reliability :** If functions can be distributed properly among several processors, then the failure of one processor will not half the system.

- **Flexibility :** A multi processor system can be made to dynamically reconfigure itself so as to optimize different objectives for different applications such as through put, application speed up or fault tolerance.
- **Functional Specialization :** Specialized processors can be added to improve performance in particular applications.

Based on the relationship between process and memory, multiprocessor systems can be classified as :

1. **Tightly Coupled :** Individual processors within the multi processor system share global shared memory.
2. **Loosely Coupled :** Individual processors within the multi processor system access their own private/local memory.

1.8 OPERATING SYSTEM COMPONENTS

Operating system is a large and complex program. It can be divided into the following modules based on their functioning.

- Process Management.
- Main Memory Management.
- File Management.
- I/O Management.
- Secondary Memory Management.
- Security System.
- Command Interpreter system.
- Network Support System.
- Protection System.

1. **Process Management :** A process can be defined as a program in execution. If more than one program is executing at the same time on the computer, then it is the responsibility

of the operating system to allocate CPU and other resources to the process so that they can execute properly. Operating system has to keep track of all processes (all the running programs) and ensure that the execution of one of them does not alter the other process.

2. **Main Memory Management :** The operating system has to allocate the main memory to the various programs.

3. **File Management :** The responsibility of file management system includes : creation and deletion of files, creating and deletion of directories, storing files on secondary media, facilities for backup of file. It also helps in the copy/ movement of files from the one disk to another.

4. **I/O Management :** The operating system performs the coordination between various input/output devices, such as keyboard, mouse, screen, printer etc., and the programs that are running.

5. **Secondary Storage Management :** This module is responsible for management of secondary memory. The tasks of this module are : free space management, storage allocation and disk scheduling (decides how to moves the read/write head over the required track to access the data).

6. **Security System :** The operating system provides appropriate security and integrity mechanisms which ensures the safety of the data and programs on the computer.

7. **Command Interpreter System :** One of the important module for an operating system is the command interpreter, which is the interface between the user and the operating system.

8. **Network Management :** The responsibilities of this module are routing and connection strategies, problems of conflict and security.

9. **Protection System :** In multi programming environment one process must be protected from one another activities.

1.9 OPERATING SYSTEM SERVICES

The operating system provides certain services to programs and to the users of those programs.

There Services Include :

- **Program Execution and Handling :** Starting of programs, managing their execution and communicating their results.
- **I/O Operation :** Mechanism for initiating and managing I/O operation.
- **File System Management :** Creating, maintaining and manipulating files.
- **Communications :** Between process of the same user, between different users.
- **Exception Detection and Handling :** Protection related issues.

  Safety in the case of power failures via backups.

  Detecting undesirable state such as printers out of papers.
- **Resources Allocation :** Includes processor and I/O scheduling, memory management.
- **Accounting :** To track users usage of resources for billing and statistical reasons.
- **Protection :** One process must be protected from another.
- **Command Interpretation :** User entered commands will be interpreted and appropriate action will be performed.
- **Resource Management :** The various resources of the system are managed in an optimal way.

1.10 SYSTEM CALL

System Call : The mechanism used by an application program to request services from the operating system. System calls often use a special machine code instruction which causes the processor to change mode (**Eg :** To ***"supervisor mode"*** or ***"protected mode"***). This allows the OS to perform restricted actions such as accessing hardware devices or the memory management unit.

Example : for() system call(UNIX operating system) can be used to start a new process the old process unaltered.

Consider the following program

```
main()
{
  printf("The name of our institute is/n");
  fork();
  printf("GITAM UNIVERSITY");
}
```

When compiled and executed the program gives the result.

```
The name of our institute is
GITAM UNIVERSITY
GITAM UNIVERSITY
```

The line GITAM UNIVERSITY gets printed twice. Fork() causes the process to split into two identical processes.

1.10.1 System Call with an Examples

The system call provides an interface to the operating system services When a program in user mode requires access to RAM or a hardware resource, it must ask the kernel to provide access to that resource. This is done via something called a

system call. When a program makes a system call, the mode is switched from user mode to kernel mode. This is called context switch. Then the kernel provides the resource which the program requested. After that, another context switch happens which results in change of mode from kernel mode back to user mode.

Generally, system calls are made by the user level programs in the following situations :

- Creating, opening, closing and deleting files in the file system.
- Creating and managing new processes.
- Creating a connection in the network, sending and receiving packets.
- Requesting access to a hardware device, like a mouse or a printer.

Examples of System Calls :

The fork() system call is used to create processes. When a process (a program in execution) makes a fork() call, an exact copy of the process is created. Now there are two processes, one being the parent process and the other being the child process.

The process which called the fork() call is the parent process and the process which is created newly is called the child process. The child process will be exactly the same as the parent. Note that the process state of the parent i.e., the address space, variables, open files etc. is copied into the child process. This means that the parent and child processes have identical but physically different address spaces.

```
#include<stdio.h>
Void main()
{
Printf("the  name of our publisher is ");
Fork();
Printf("this is falcon publisher");
}
```

When compiled and executed the program gives the result the name of publisher is

this is falcon publisher

this is falcon publisher

When the above example code is executed, when line A is executed, a child process is created. Now both processes start execution from line B. To differentiate between the child process and the parent process, we need to look at the value returned by the fork() call.

Exec() : The exec() system call is also used to create processes. But there is one big difference between fork() and exec() calls. The fork() call creates a new process while preserving the parent process. But, an exec() call replaces the address space, text segment, data segment etc. of the current process with the new process. It means, after an exec() call, only the new process exists. The process which made the system call, wouldn't exist. There are many flavors of exec() in UNIX, one being execl() which is shown below as an example:

```
#include<stdio.h>
Void main()
{
Exec1("/bin/ls","ls",0);
Printf("this text wont'be printed unless an error occurs in exec().");
}
```

As shown above, the first parameter to the execl() function is the address of the program which needs to be executed, in this case, the address of the ls utility in UNIX. Then it is followed by the name of the program which is ls in this case and followed by optional arguments. Then the list should be terminated by a NULL pointer (0).

When the above example is executed, at line A, the ls program is called and executed and the current process is halted. Hence the printf() function is never called since the process has already been halted. The only exception to this is that, if the execl() function causes an error, then the printf() function is executed.

1.11 DIFFERENT TYPES OF SYSTEM CALLS

Whenever users wants to write a program for a specific purpose, they do so by using a specific language. The program that is written uses the hardware as an aid to accomplish the task. The kernel (part of operating system and considered as heart of operating system) controls and streamlines the access to the hardware. The applications that a user writes normally run on top of an operating system. In most operating systems, a program cannot access the resources directly. The programs run by the user request the operating system to do a specific task on behalf of the program. For example, if the program wants to open a file, it request the operating system to "**Please open the file with file name**" "**/usr/read.txt**". The kernel replies to the request suitably. It opens the file if the calling application has permissions. If the application does not have permissions, the operating system simply ignores the request, giving back an error. The process of calling the kernel is referred to as **system call invocation** and the call to the kernal "**Please open the file**" is known as a ***system call***. The kernal calls appropriate device drivers to access the hardware.

Following Fig. 1.8 shows working of a system call.

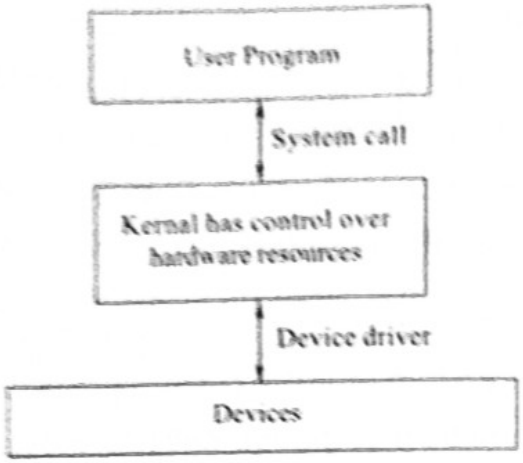


FIG 1.8 : Working of System Call

So a system call can be defined as a method by which a program makes a request to the operating system.

There are different system calls for performing different kinds of tasks :

- File manipulation system calls for example are open, close, read, write, lseek etc.
- There can be system calls for process control. These can be like abort, execute, create process, terminate a process, allocate and free memory for a process etc.
- For device management, the system calls that can be there are - request device, release device, read, write, reposition, get and set device attributes etc.
- For managing the information, the system calls that can be there are get and set time, get and set file or device attributes etc.

1.12 SINGLE USER, MULTI USER OPERATING SYSTEM STRUCTURE

Operating system is a large and complex program. It has to be designed carefully so that it will function normally and can be modified easily. Generally operating system will not be implemented as a single system. OS will be divided into

number of modules. Each module will occupy some portion of the system. Each module will have set of inputs, outputs and definite task to perform. While performing this task the module may take assistance of other modules.

The structure of an operating system is dictated by the model employed in building them. An operating system model is a broad frame work that unifies the many features and services the operating system provides and tasks it performs.

Operating systems are broadly classified into two categories based on their structure.

1. Monolithic Operating System
2. Layered Operating System.

1. **Monolithic Operating System (Single User) :** The components of monolithic operating system are organized randomly and any module can call any other module. The applications in the monolithic operating systems are separated from the operating system itself. That is the operating system code runs in privileged mode (kernel mode) with access to system data and to the hardware; applications runs in non - privileged mode (called user mode), with a limited set of interfaces available and with limited access to the system data. The monolithic operating system structure with separate user and kernel processor is shown in the Fig. 1.9.

Example Systems : CP/M and MS - DOS

The Key Features of Monolithic Kernel are :

- Monolithic kernel interacts directly with the hardware.
- Monolithic kernel can be optimized for particular hardware architecture.
- Monolithic kernel is not very portable.

- Monolithic kernel gets loaded completely into memory at boot time.

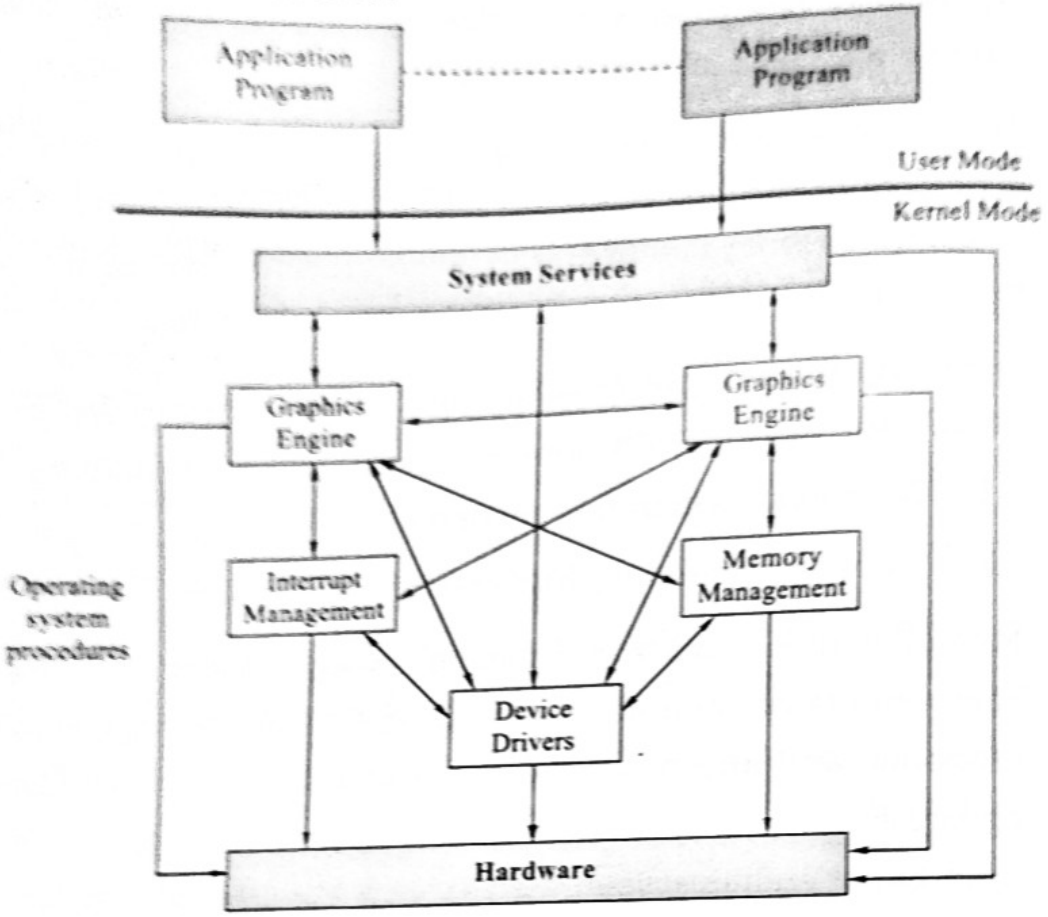


FIG 1.9 : Monolithic Operating System

2. **Layered Operating System (Multi User) :** Edsger W. Dijkstra introduced the layered architecture for operating sytem when he developed in the 60's an operating system called THE.

   The main advantage of this approach is modularity. All the benefits of modular programming can be achieved with this architecture.

   The components of layered operating system are organized into modules and layers them one on top of the other. Each module provide a set of functions that other module can call. Interface functions at any particular level can invoke services provided by lower layers but not the other way around. The layered operating system structure with hierarchical organization of modules is shown in Fig. 1.10.

One advantage of a layered operating system structure is that each layer of code is given access to only the lower - level interfaces (and data structures) it requires. That is in this approach, the N^{th} layer can access services provided by the $(N - 1)^{th}$ layer and provide services to the $(N + 1)^{th}$ layer. This structure also allows the operating system to be debugged starting at the lowest layer, adding one layer at a time until the whole system works correctly. Layering also makes it easier to enhance the operating system; one entire layer can be replaced without affecting other parts of the system. Layered operating system deliver low application performance in comparison to monolithic operating system.

Example Systems : VAX/VMS, Multics, UNIX

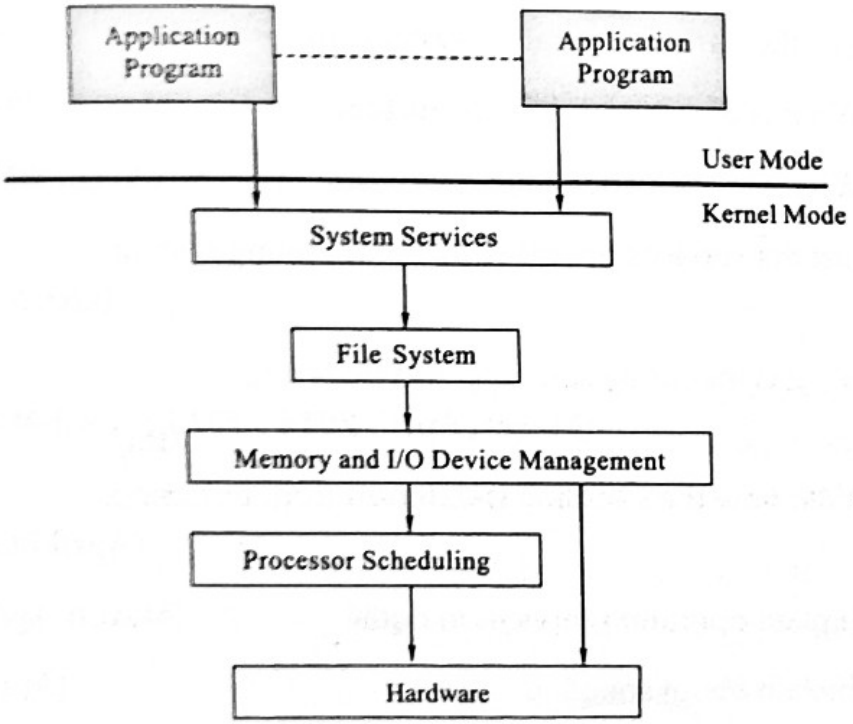


FIG 1.10 : Layered Operating System

REVIEW QUESTIONS

Part-A

1. Define operating system. **(March. 2018, 2016)**
2. What is buffering ? **(Oct/Nov. 2019)**
3. What do you mean by multiprocessors systems. **(April. 2012)**
4. List the advantages of multiprocessor systems. **(March. 2018)**
5. What is a system call ?
6. List the operating system services.

Part-B

1. Write how multiprogramming is different from timesharing. **(March/April. 2014)**
2. List the various operating system components. **(April. 2008)**
3. What is a system call ? List its features **(March. 2016, 2013)**
4. What are the various types of system calls ? **(April. 2011, 2007)**
5. List the services provided by an operating system. **(Oct/Nov. 2019)**
6. What is meant by spooling and buffering. **(March/April. 2014 ; 2013 ; April/May. 2011)**
7. Write how the spooling is different from buffering. **(April/May. 2012)**
8. Explain operating services in detail. **(March/April. 2016)**
9. Explain about spooling. **(April. 2008)**
10. Explain about (i) Multiprogramming (ii) Time sharing. **(Oct/Nov. 2019)**

Part-C

1. With neat sketch, explain the concept of spooling. **(April. 2007)**
2. Explain the services provided by operating system. **(March/April. 2018)**

3. What are the services of operating system ? Explain.
(March/April. 2013 ; April/May. 2012)

4. Explain about the various components of operating system.
(April/May. 2011)

5. Write about different types of system calls. **(March/April. 2014)**

6. Explain using user operating system structure and multi user operating system structure.

7. What are the distributed and real systems. **(April. 2008, 2007)**

CHAPTER

2

PROCESS MANAGEMENT

CHAPTER OUTLINE

2.0 INTRODUCTION TO PROCESS MANAGEMENT

One of the important tasks of the operating system is to manage resources. Most important resource of the system is CPU (Processor). So operating system must use CPU in an effective way. In single user system, the processor is busy only when the user is executing a job (program) - at all other times it is idle. Managing processor is very simple here. However, when there are many users with many jobs on the system, the processor must be allocated to each job in an effective manner, which can be a complex task. The CPU can execute only one instruction at a time and that instruction can belong to only one of the programs residing in the memory. The operating system will have to allocate the CPU time to various users based on a certain policy. This is done by process management.

In order to understand process management, let us understand what a process is and how it is different from program. As you know program is a set of instructions and the program will be in secondary memory. Whenever you try to execute this program it will be brought into primary memory. At this moment a program is said to be a process. So a process **is a program under execution which is requesting for the CPU time and other resources.** A process is not a program. A process is one instance of a program in execution. Many process can be running the same program.

To understand process and program, let us assume that you are at a construction site. You will observe that some of the workers are working and remaining workers are taking rest. The working workers will use some tools to do the task. The supervisor will supervise these workers. The resting workers are similar to a program. The working workers are similar to a process. Similar to working workers needs tools for doing work process needs resources like memory, CPU time etc., for execution. Supervisor is operating systems processor management.

S.No.	Program	Process
1.	Consists of instructions in any programming language	Instruction execution in machine code
2.	Resides in secondary storage	Resides in main memory
3.	Static object	Dynamic object
4.	Not active entity	Active entity.
5.	—	Many process can be running the same program

2.1 DEFINE PROCESS

The term process was first used by the designer of MULTICS system. Process can be defined in many ways.

Some of these are :

- A program in execution.
- The entity to which processors are assigned.
- Is a program under execution which competes for the CPU time and other resources.

The components of a process are shown in the Fig. 2.1

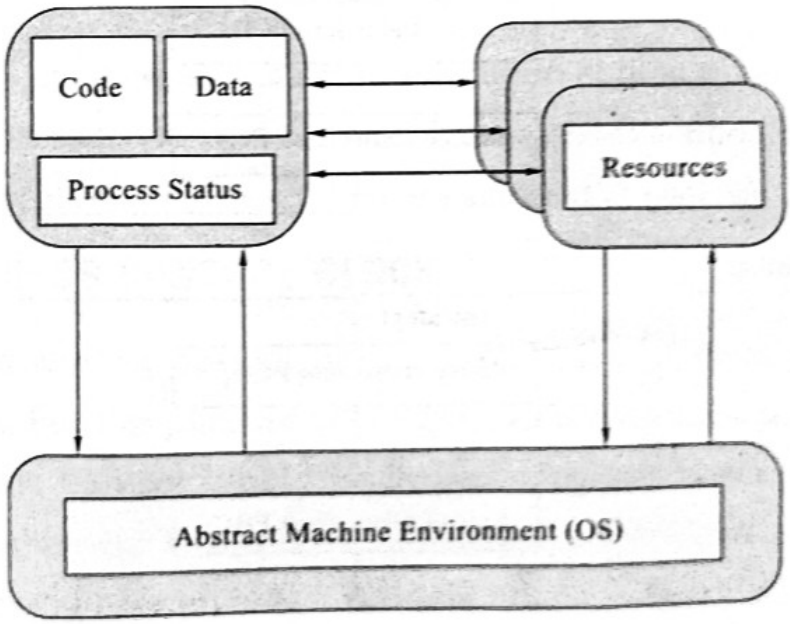


FIG 2.1 : A Processor

- The code to be executed.
- The data on which the program will execute.
- Resources required by the program.
- The status of the process's execution.

For process to execute, it must have a suitable OS to manage the sharing and isolation of resources among the community of processes.

2.2 SEQUENTIAL PROCESSES

A process is usually sequential and consists of sequence of operations that take place one at a time. The operations include creation of process, running a process, destroying a process, suspend a process etc. When a set of process is running on a single CPU, using time sharing to multi tasking this is referred as concurrent sequential processing.

The Fig. 2.2 shows sequential execution of two process P_0 and P_1.

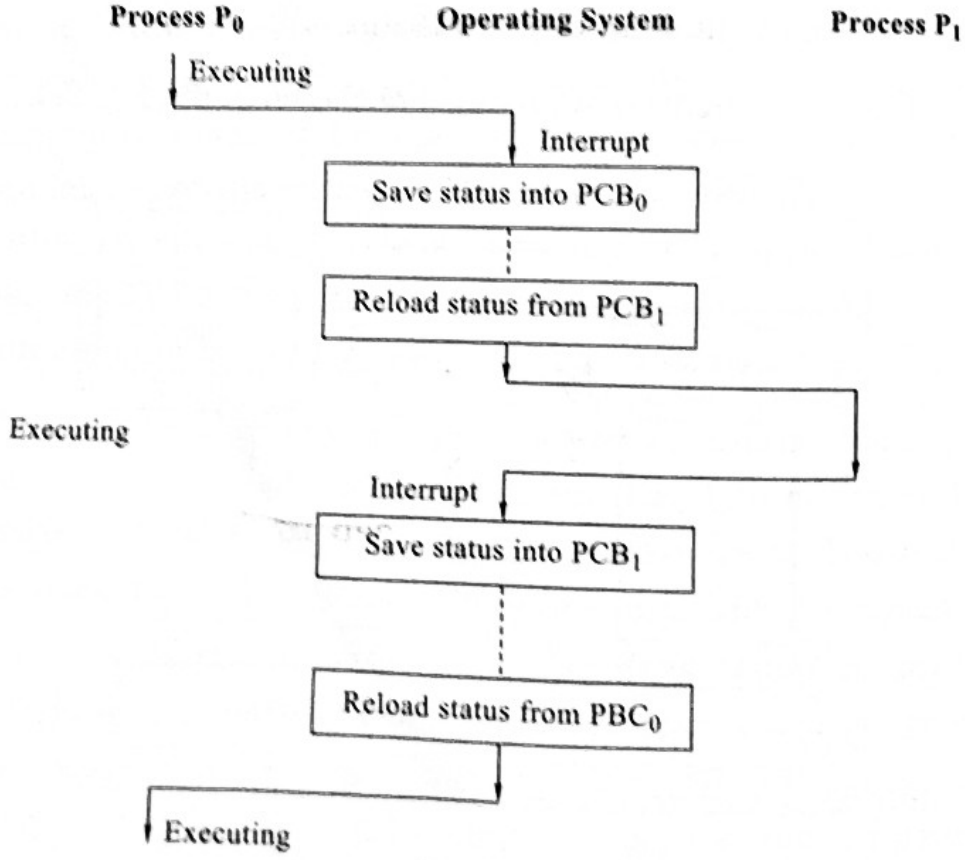


FIG 2.2 :

Let us assume that the process are time shared. Initially P_0 is executing and after predefined time interval processor (CPU) time will have to be given for P_1 so operating systems process management module will take processor from P_0 and processor will be allocated for P_1 and after predefined time interval once again processor will go to process P_0.

Two process are said to be sequential if the execution of one must be complete before the execution of other can be start. If two process are said to be concurrent, they are not serial, and their execution can overlap in time. See Fig. 2.3.

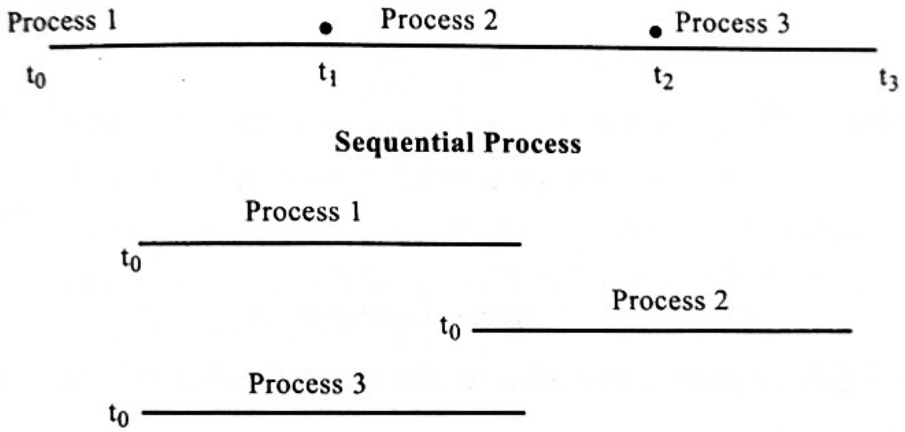


FIG 2.3 : Concurrent Process

In Fig. 2.3 process 1 starts its execution at time t_0, process p_2 starts its execution only after p_1 finished its execution. P_3 starts its execution after p_2 so they are sequential process.

2.3 PROCESS CONTROL BLOCK

The operating system has to keep track of each process status. For this purpose O.S maintains a data structure called process control block. Each user process has s process control block. This is created when a user creates a process (generally user creates a process by executing a program). And it is removed from the system when the process is killed / terminated.

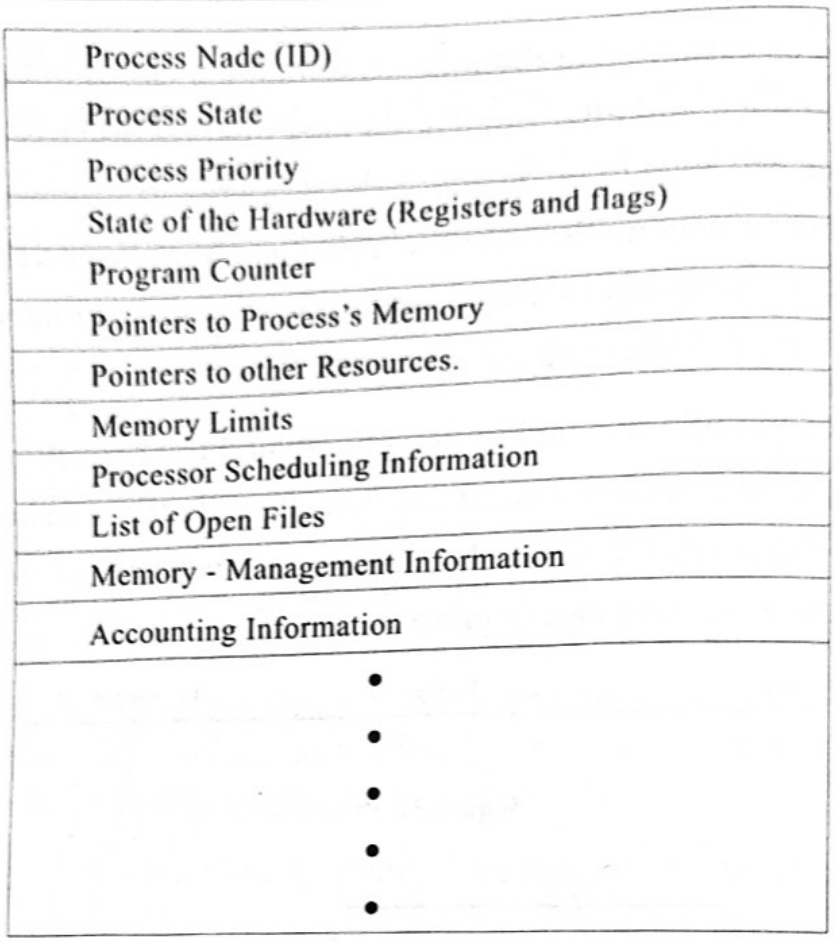


FIG 2.4 : PCB (Process Control Block)

The PCB contains the above described fields which can be discussed now.

1. **Process Name (ID) :** This is the number allocated by the OS to the process on creation.
2. **Process State :** Each process may be in any of these states, New, Ready, Running, Suspended and Terminated.
3. **Process Priority :** Some of the process may be required to get completed quicker than others. So they need higher priority than other processes.
4. **Registers :** The Registers include General - purpose, as well as special registers needed for allowing the process to continue.
5. **Program Counter :** This program counter contains the address of the next instruction to be executed for a given process.

6. **Pointers to Process Memory :** This provides the pointers to the other structures (Data structures) maintained for this purpose.

2.4 PROCESS STATE DIAGRAM

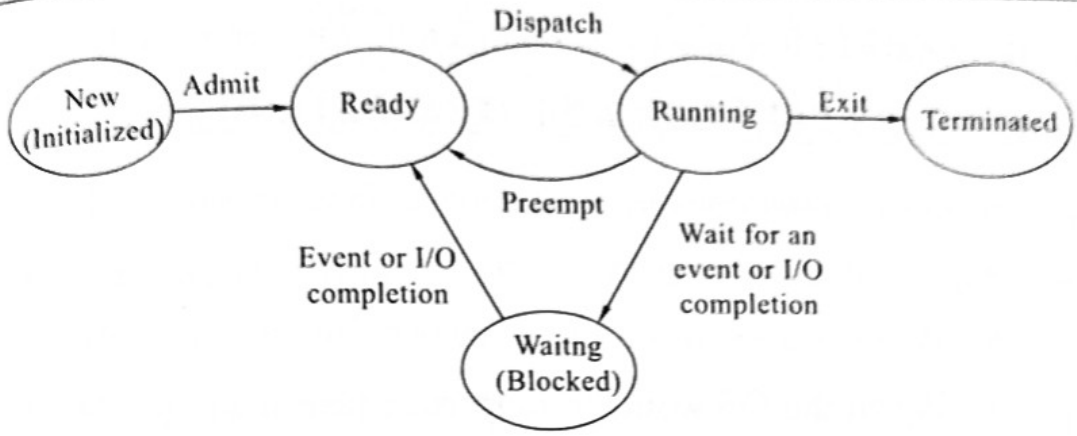


FIG 2.5 : Process State Diagram

1. **New :** When you start executing a program a new process is created. The new process can be admitted into ready process list depending on the load (i.e., no. of process in the ready list) if the load is heavy then the process may not be accepted. If the load is moderate then the process can be admitted into ready process list.

2. **Ready :** No. of process are waiting for the processor. When a process turn comes, OS dispatches it to the running state i.e., processor is allocated for this process.

3. **Running :** The process is executing the instruction of the corresponding process.

   Depending on the scheduling algorithm a process may be forced to release processor which is known as ***preemption***, if a process is preempted once again it will go to ready state.

4. **Blocked :** While running a process needs some Input/Output operation, as you know I/O operations involves peripheral devices, which are electro mechanical devices so the operation takes more time. Instead of keeping processor idle (during this

I/O operation) processor is released form process and the process is blocked. Processor is allocated to some other process. Once the I/O operation is complete the process will go to ready state. (That means once again it will ask for processor).

5. **Terminated :** It a process is completed, it is terminated.

2.5 PROCESS CREATION AND TERMINATION

A process is created.

- When a user logs in
- When a user runs a program
- When the OS wants to perform a task
- A process may generate another process to perform an independent (Sub) task.

When a process is created, several steps take place. They are

- **Creation of PID :** The operating system creates a new Process ID (PID) for the new process.
- **Allocate Resources :** The operating system allocates the necessary resources to the process.
- **Initialize the PCB :** The OS adds the relevant information about this process to the PCB.
- **Update Lists :** The OS adds this process to the list of processes.

A process is terminated because of

- Normal exit (requested by itself)
- Critical error (memory violation, system error etc.)
- Termination by another process.

When a process is terminated then

- Freeing of resources.
- Deleting entries from the PCB.

2.6 RELATION BETWEEN PROCESSES

The concurrent process execution in the operating system my be either independent or cooperating. A process is independent if it cannot affect or affected by other process executing in the system. A process that does not share any data with any other process is independent. On the other hand, a process is cooperating if it can affect by the other process executing in the system.

There are several reasons for providing process cooperation.

1. **Information Sharing :** Several people may be interested in sharing same information.
2. **Computation Speed Up :** It we want a particular task to run faster, we must break it into subtasks, each of which will be executing in parallel with the others.
3. **Modularity :** We may want to construct the system in a modular fashion, dividing the system functions into separate processes.
4. **Convenience :** Even an individual user may have many tasks to work on at one time. For example, a user may be editing, printing and compiling in parallel.

2.7 THREADS, MULTITHREADING MODELS

Threads : A process may be divided into many smaller independent tasks (known as light weight processes) that can be executed simultaneously. The light weight process is known as ***thread***.

Threads share the resources allocated for a process. The threads created for a particular process are known as ***sibling threads***. Threads comprises a thread *ID*, a program counter, a register set, and a stack. Threads share with other threads belonging to the same process its code section, data section, and other operating system resources, such as open files and signals. Threads are not independent of one another. Just like process, threads will have different states. Threads are

introduced to improve the application performance through parallelism. A running thread has the processor (CPU) and is active. The processor switches among the threads giving illusion that all the threads are running.

Processor vs Threads :

- Like process, threads share CPU and only one thread active (running) at a time.
- Like processes, threads with in a process execute sequentially.
- Like processes, threads can create children.
- Like processes, if one thread is blocked another thread can run.
- Process switching needs interface with OS, where as threads need not.
- Unlike processes, threads are not independent of one another.
- Unlike processes, threads are designed to assist one other.

Advantages of Threads :

- Inter process communication is not required as threads can share common data.
- Because of its nature threads can take the advantage of multiprocessor.
- Resources required for a thread are less so they are cheap to create.
- Context switching (form one thread to another thread) is very fast.
- A process with multiple threads can be viewed as large server.

Multi Threading Model : Multi threading refers to the ability on an operating system to support multiple threads of execution within a single process.

Traditionally there is a single thread of execution per process.

Example : MSDOS supports a single user process and single thread.

UNIX supports multiple user processes but only support one thread per process.

Multithreading : Java run time environment is an example of one process with multiple threads.

Examples of supporting multiple processes, with each process supporting multiple threads.

Windows 2000, Solarix, Linux, Mac, and OS/2

Fig. 2.6 shows a One process one thread, One process multiple threads, Multiple processes one thread per process, Multiple processes multiple threads per process.

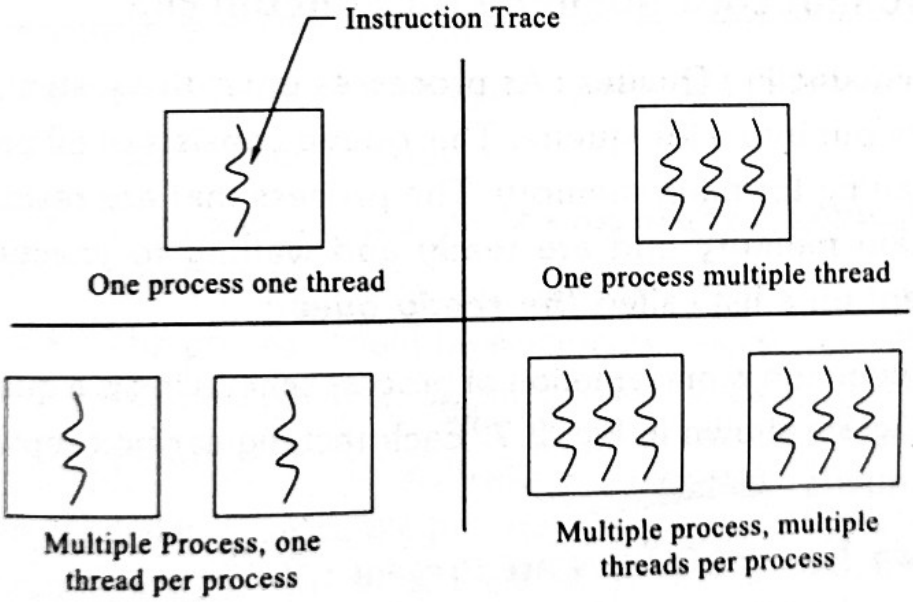


FIG 2.6 :

In multithreading CPU switches rapidly from one to other among the threads providing the illusion that the threads are running in parallel.

2.8 SCHEDULING CONCEPTS

Scheduling is the basis of multi programming operating system. By switching the processor among processes, the operating system can make the computer more productive.

In multiprogramming environment, where many process might be in ready state, it is important to schedule processes to the processor in order of their priority. A part of the operating system called the scheduler schedules the transition of process from ready state to running state and back. The scheduler tracks all the processes in memory.

When more than one process is in the ready state, the scheduler must decide which process should be assigned to the processor. The processes should be scheduled in such a manner that no process must be made to wait for a long time and the processor time is utilized to the maximum. This gives maximum efficiency and minimum wait time for each process.

2.9 SCHEDULING QUEUES AND SCHEDULERS

Scheducling Queues : As processes enter the system, they are put into a job queue. This queue consists of all process waiting for main memory. The process that are residing in main memory and are ready and waiting to execute are kept on a list called ***the ready queue.***

A common representation of process scheduling is a queueing diagram shown in Fig. 2.7. Each rectangular box represents a queue.

Two Types of Queues are Present :

- The ready queue and
- A set of device queues.

The circles represent the resources that serve the queues, and the arrows indicate the flow of processes in the system.

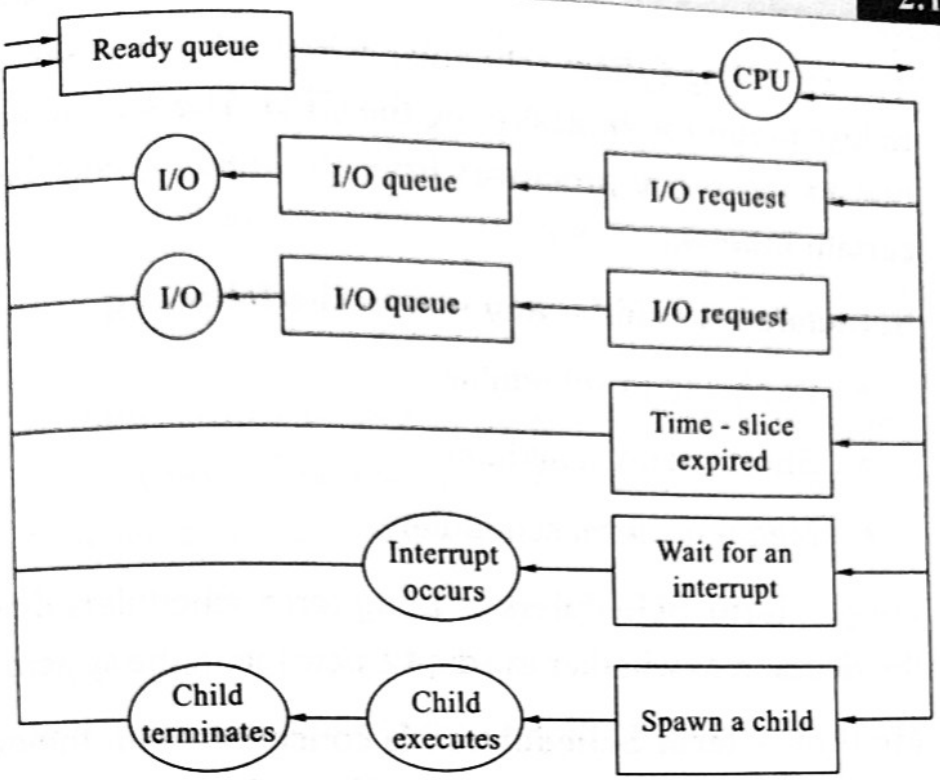


FIG 2.7 :

A new process is initially put into the ready queue. It waits in the ready until it is selected for execution and is given the CPU. Once the process is allocated the CPU and is executing, one of several events could occur :

- The process could issue an I/O request, and then be placed in an I/O queue.
- The process could create a new process and wait for its' termination.
- The process could be forcibly removed from the CPU, as a result of an interrupt, and be put back into the ready queue.

In the first two cases, the process eventually switches from the waiting state to the ready state and is then put back into the ready queue. A process continues this cycle until it terminates, at which time it exits from the system.

Schedulers : At any given instant of time, a process is in any one of the new, ready, running waiting or terminated state. Also a process moves from one to another as long as it is active.

The operating system schedular schedules processes from the ready queue for execution by the CPU. The schedular selects one to the many processes from the ready queue based on certain criteria.

Schedulers could be any one of the following :

- Long - term schedular
- Short - term schedular
- Medium - term schedular.

Long - Term Schedulers : Long term schedulers deals with the decision as whether to admit a new job to the system or not.

Medium - Term Schedulers : Is concerned with the decision to temporarily remove a process from the system in order to reduce the system load.

Short - Term Schedules : Is concerned with the decision of which ready process is to be assigned the processor.

2.10 CPU SCHEDULING AND SCHEDULING CRITERIA

CPU Scheduling : The **CPU scheduler** selects a process from among the ready processes to execute on the CPU. CPU scheduling is the basis for multiprogrammed operating systems. CPU utilization increases by switching the CPU among ready processes instead of waiting for each process to terminate before executing the next.

The idea of multiprogramming could be described as follows : A process is executed by the CPU until it completes or goes for an I/O. In simple systems with no multiprogramming the CPU is idle till the process completes the I/O and restarts execution. With multiprogramming, many ready processes are maintained in memory. So when CPU becomes idle as in the case above, the operating system switches to execute another process each time a current process goes into a wait for I/O.

Scheduling Criteria : Many algorithms exist for CPU scheduling. Various criteria have been suggested for comparing these CPU scheduling algorithms. Common criteria include :

1. **CPU Utilization :** This may range from 0% to 100 % ideally. In real systems it ranges from 40% for a lightly loaded systems to 90% for heavily loaded systems.
2. **Throughput :** Number of processes completed per time unit is throughput. For long processes may be of the order of one process per hour where as in case of short processes, throughput may be 10 or 12 processes per second.
3. **Turnaround Time :** The interval of time between submission and completion of a process is called turnaround time. It includes execution time and waiting time.
4. **Waiting Time :** Sum of all the times spent by a process at different instances waiting in the ready queue is called **waiting time.**
5. **Response Time :** In an interactive process the user is using some output generated while the process continues to generate new results. Instead of using the turnaround time that gives the differences between time of submission and time of completion, response time is sometimes used. Response time is thus the difference between time of submission and the time the first response occurs.

Desirable features include maximum CPU utilization, throughput and minimum turnaround time, waiting time and response time.

2.11 VARIOUS SCHEDULING ALGORITHMS

Scheduling algorithms differ in the manner in which the CPU selects a process in the ready queue for execution.

Scheduling algorithms can be classified into :

- Preemptive.
- Non preemptive

In **Preemptive** scheduling algorithms processor can be released forcibly from a process before its execution completes.

In **non-preemptive** scheduling algorithms processor will be with process throughout its execution. Processor can not be forcibly released from the process before its execution completes.

2.11.1 First Come First Serve (FCFS) Scheduling Algorithm

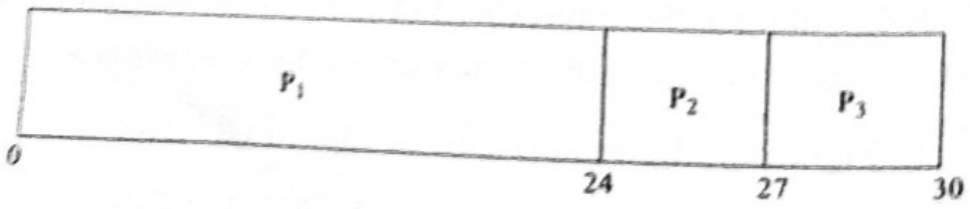
This is simplest CPU scheduling algorithm. A process that requests for the CPU first is allocated the CPU first. Hence the name ***first come first serve***. The FCFS algorithm is implemented by using a ***first-in-first-out*** (FIFO) queue structure for the ready queue. This queue has a head and a tail. When a process joins the ready queue its PCB is linked to the tail of the FIFO queue. When the CPU is idle, the process at the head of the FIFO queue is allocated the CPU and deleted from the queue. Since the algorithm is non preemptive in nature, it is not suited for time sharing systems.

Even though the algorithm is simple, the average waiting is often quite long and varies substantially if the CPU burst times vary greatly as seen in the following example.

Consider a set of three processes P_1, P_2, P_3 arriving at time instant 0 and having CPU burst times as shown below :

Process	Burst time (msecs) / CPU time required
P_1	24
P_2	3
P_3	3

The Gantt chart below shows the result.



Average waiting time and average turnaround time are calculated as follows :

Waiting time = Starting time – Arrival time

The waiting time for process

$$P_1 = 0 - 0 = 0 \text{ msecs}$$

The waiting time for process

$$P_2 = 24 - 0 = 24 \text{ msecs}$$

(P_2 can start its execution only after P_1 completes its execution. So P_2 will start 24.

The waiting time for process

$$P_3 = 27 - 0 = 27 \text{ msecs}$$

(P_3 can start its execution only after P_2 completes its execution. So P_2 will start at 27.

Average waiting time = (0 + 24 + 27) / 3 = 17 *msecs.*

P_1 completes at the end of 24 *msecs.* P_2 at the end of 27 *msecs* and P_2 at the end of 30 *msecs.*

Turn around time = Finish time – Arrival time

Turn around time for $P_1 = 24 - 0 = 24$

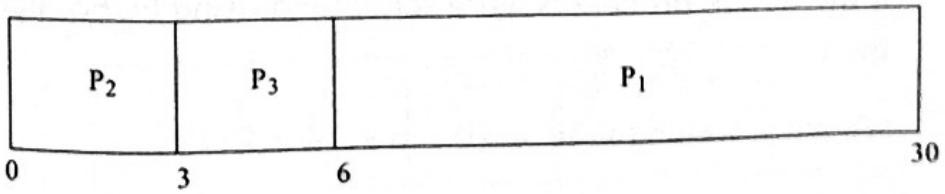
$P_2 = 27 - 0 = 27$

$P_3 = 30 - 0 = 30$

Average turnaround time = (24 + 27 + 30) / 3

= 81/3 = 27 msecs.

If the process arrive in the order P_2, P_3 and P_3, then the result will be as follows :



Average waiting time=(0 + 3 + 6) / 3 = 9/3 = 3 msecs

Average turnaround time = (3 + 6 + 30) / 3 = 39/3 = 13 msecs.

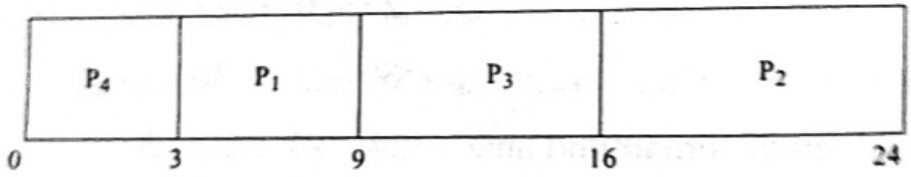
Thus if process with smaller CPU burst times arrive earlier, then average waiting and average turnaround times are lesser.

2.11.2 *Shortest Job First (SJF) Scheduling Algorithm*

Another approach to CPU scheduling is the shortest job first algorithm. In this algorithm, the length of the CPU burst is considered. When the CPU is available, it is assigned to the process that has the smallest next CPU burst. Hence the name shortest job first. In case there is a tie, FCFS scheduling is used to break the tie. As an example, consider the following set of processes P_1, P_2, P_3, P_4 and their CPU burst times :

Process	Burst Time (msecs)
P_1	6
P_2	8
P_3	7
P_4	3

Using SJF algorithm, the processes would be scheduled as shown below.



Average waiting time=(0 + 3 + 9 + 16) / 4 = 28/4 = 7 msecs.

Average turnaround time = (3 + 9 + 16 + 24)/4 = 52/4 = 13 msecs

If the above processes were scheduled using FCFS algorithm, then

Average waiting time = (0 + 6 + 14 + 21)/4

= 41/4 = 10.25 msecs

Average turnaround time = (6 + 14 + 21 + 24)/4
= 65/4 = 16.25 msecs.

The SJF algorithm produces the most optimal scheduling scheme. For a given set of processes, the algorithm gives the minimum average waiting and turnaround times. This is because, shorter processes are scheduled earlier than longer ones and hence waiting time for shorter processes decreases more than it increases the waiting time or long processes.

The main disadvantage with the SJF algorithm lies in knowing the length of the next CPU burst.

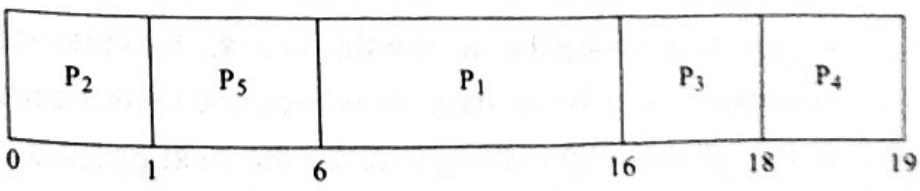
2.11.3 Priority Scheduling

Each process can be associated with a priority. CPU is allocated to the process having the highest priority. Hence the name priority. Equal priority processes are scheduled according to FCFS algorithm.

The SJF algorithm is a particular case of the general priority algorithm. In this case priority is the inverse of the next CPU burst time. Larger the next CPU burst, lower is the priority and vice versa. In the following example, we will assume lower numbers to represent higher priority.

Process	Priority	Burst time (msecs)
P_1	3	10
P_2	1	1
P_3	3	1
P_4	4	1
P_5	2	5

Using priority scheduling, the processes are scheduled as shown below :



Average waiting time = (6 + 0 + 16 + 18 + 1) / 5

= 41 = 8.2 *msecs.*

Priority based algorithms can be either preemptive or nonpreemptive. In case of preemptive scheduling, if a new process joins the ready queue with a priority higher than the process that is executing, then the current process is preempted and CPU allocated to the new process. But in case of nonpreemptive algorithm, the new process having highest priority from among the ready processes, is allocated the CPU only after the current process gives up the CPU.

2.11.4 Round - Robin (RR) Scheduling Algorithm

The round - robin CPU scheduling algorithm is basically a preemptive scheduling algorithm designed for time - sharing systems. One unit of time is called a time slice. Duration of a time slice may range between 10 *msecs.* and about 100 msecs. The CPU scheduler allocates to each process in the ready queue one time slice at a time in a round - robin fashion. Hence the name round - robin.

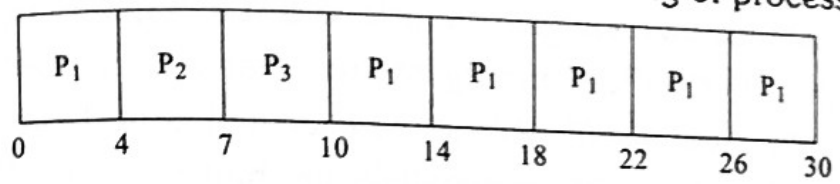
The ready queue in this case is a *FIFO* queue with new processes joining the tail of the queue. The CPU schedular picks processes from the head of the queue for allocating the CPU. The first process at the head of the queue gets to execute on the CPU at the start of the current time slice and is deleted from the fready queue. The process allocated the CPU may have the current CPU burst either equal to the time slice or smaller than the time slice or greater than the time slice. In the first two cases, the current process will release the CPU on its own and there by the next process in the ready queue will be allocated the CPU for the next time slice. In the third case, the current process is preempted, stops executing, goes back and joins the ready queue at the tail there by making way for the next process.

Consider the same example explained under FCFS algorithm.

Process	Burst Time (msecs)
P_1	24
P_2	3
P_3	3

Let the duration of a time slice be 4 msecs, which is to say CPU switches between processes every 4 msecs in a round - robin fashion.

The Gantt chart below shows the scheduling of processes.



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

= 17/3 = 5.66

2.11.5 Multilevel Queue Scheduling

Processes can be classified into groups. For example, interactive processes, system processes, batch processes, student processes and so on. Processes belonging to a group have a specified priority. This algorithm partitions the ready queue into as many separate queues as there are groups. Hence the name multilevel queue. Based on certain properties is process is assigned to one of the ready queues. Each queue can have its own scheduling algorithm like *FCFS* or *RR*. For example, Queue for interactive processes could be scheduled using *RR* algorithm where queue for batch processes may use *FCFS* algorithm.

An illustration of multilevel queue is shown in Fig. 2.8.

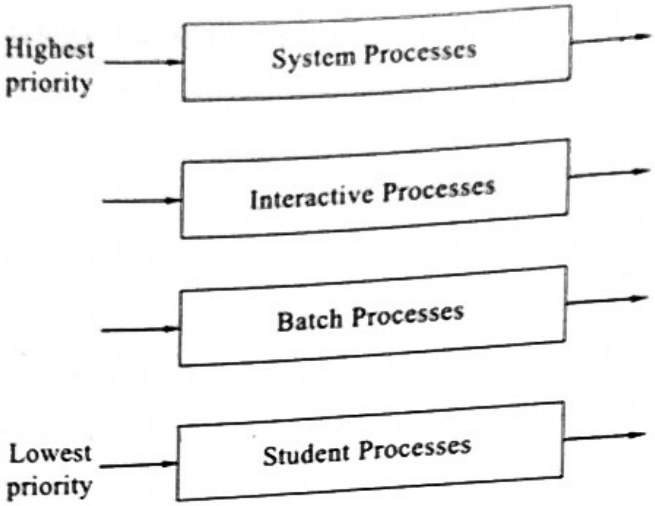


FIG 2.8 :

Queues themselves have priorities. Each queue has absolute priority over low priority queues, that is a process in a queue with lower priority will not be executed until all processes in a queue with higher priority have finished executing. If a process in a lower priority queue is executing (higher priority queues are empty) and a process joins a higher priority queue, then the executing process is preempted to make way for a process in the higher priority queue.

2.11.6 Multilevel Feedback Queue Scheduling

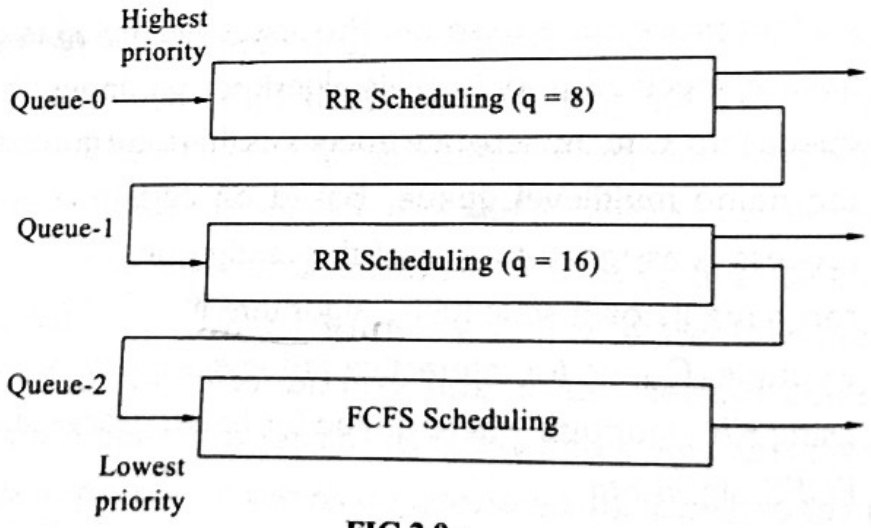


FIG 2.9 :

In the previous multilevel queue scheduling algorithm, processes one assigned to queues do not move or change

queues. Multilevel feed back queues allow a process to move between queues. The idea is to separate out processes with different CPU burst lengths. All processes could initially join the highest priority queue. Processes requiring longer CPU bursts are pushed to lower priority queues. *I/O* bound and interactive processes remain in higher priority queues. Aging could be considered to move processes from lower priority queues to higher priority to avoid starvation. An illustration of multilevel feedback queues is shown below.

A process entering the ready queue 0. *RR* scheduling algorithm with $q = 8$ is used to schedule processes in queue 0. If the CPU burst of a process exceeds 8 msecs., then the process preempted, deleted from queue 0 and joins the tail of queue 1. When queue 0 becomes empty, then processes in queue 1 will be scheduled. Here also *RR* scheduling algorithm is used to schedule process but $q = 16$. This will give processes a longer time with the CPU. If a process has a CPU burst still longer, then it joins queue 3 on being preempted. Hence highest priority processes (processes having small CPU busts, that is I/O bound processes) remain in queue 1 and lowest priority processes (those having long CPU bursts) will eventually sink down. The lowest priority queue could be scheduled using *FCFS* algorithm to allow processes to complete execution.

Multilevel feedback scheduler will have to consider parameters such as number of queues, scheduling algorithm for each queue, criteria for upgrading a process to a higher priority queue, criteria for downgrading a process to a lower priority queue and also the queue to which a process initially enters.

REVIEW QUESTIONS

Part-A

1. What do you meant by process management?
2. What is program and process?
3. Define a process. (March/April. 2014)
4. What is Process Control Block? (Oct. 2020)
5. What is sequential process?
6. What is a scheduler ?
7. What is FCFS ? (March/April. 2013)

Part-B

1. What are the three major activities of an operating system in regard to process management ? (April/May. 2011)
2. What are the turn around time and response time. (March/April. 2016)
3. Differentiate between program and process. (March. 2018)
4. List three examples of deadlocks that are not related to a computer-system environment. (March/April. 2013)
5. What three issues must be considered in the case of preemption. (March/April. 2014)

Part-C

1. Explain the process state diagram. (March/April. 2013 ; 2012)
2. Explain process creation and termination. (April/May. 2011)
3. Explain the threads. (March/April. 2013)
4. Explain CPU scheduling and scheduling criteria. (March/April. 2014 ; April/May. 2012 ; April. 2008)
5. Explain the queuing diagram. (March/April. 2013)
6. Discuss about process creating and termination. (Oct. 2020)

CHAPTER

3

SYNCHRONIZATION AND DEADLOCKS

CHAPTER OUTLINE

3.1 PROCESS SYNCHRONIZATION

Process Synchronization is the task of coordinating the execution of processes in a way that no two processes can have access to the same shared data and resources.

It is specially needed in a multi-process system when multiple processes are running together, and more than one processes try to gain access to the same shared resource or data at the same time.

This can lead to the inconsistency of shared data. So the change made by one process not necessarily reflected when other processes accessed the same shared data. To avoid this type of inconsistency of data, the processes need to be synchronized with each other.

3.2 SEMAPHORE

A semaphore is a variable and is used for restricting access to a shared resource in multiprogramming environment. A semaphore value can be altered by *P* and *V* operations. Semaphores were invented by **Edsger W. Dijkstra.** *P* operation is also known as ***wait operation*** and *V* operation is also known as ***signal operation***. Any process wants to access a resource, has to perform *P* operation. Any process while releasing the resource has to perform *V* operation.

P Operation : Wait operation

```
if s > 0 then
Set s to s-1
else
block the calling process (i.e., wait for resource)
endif
```

V Operation : Signal operation :

If any processes are waiting on s

```
Start one of these processes
else
  Set s to s+1
Endif
```

Initially semaphore s value is set to '1'. When a process wants to access a resource it performs P operation. Suppose a process p_0 wants to access resource, as S value is 1 which is greater than 0 so process can access the resource. Now process p_1 wants resource. Now as s value is less than 0 the calling process i.e., p_1 will be blocked. When p_0 wants to release the resource it will perform v operation. As p_1 waiting for resource p_1 will get resource otherwise the s value will be incriminated by 1 which indicates resource is free.

3.3 INTER PROCESS COMMUNICATION

Inter process communication is a set of techniques for the exchange of information between two or more process. Processes may be running in one computer or on two or more computers connected by a network.

The IPC mechanisms can be classified into the following categories.

- Pipes
- Fifos
- Shared memory
- Message passing
- Sockets

1. **Pipes :** Pipes provide unidirectional flow of communication between process within the same system. Pipes are half duplex, that is data flows in only one direction. A pipe is created by invoking the pipe system call. A pipe operation creates a temporary file into which the file process writes information and form it second process reads its input.

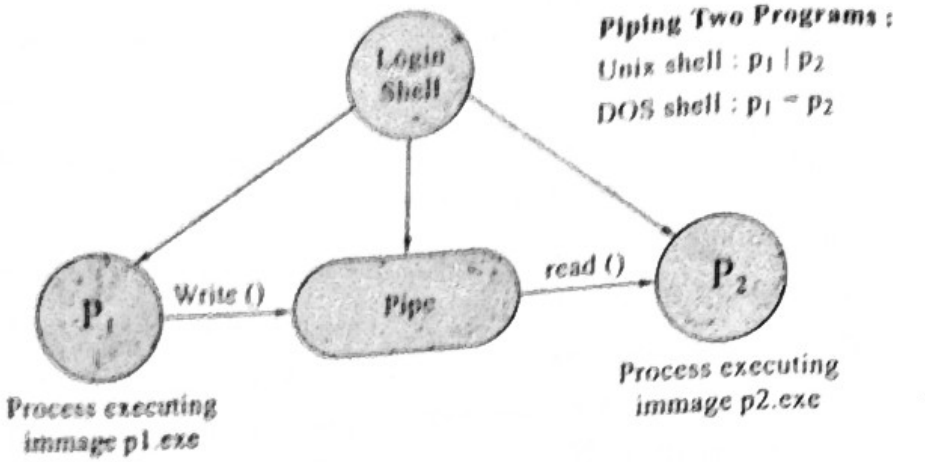


FIG 3.1 :

2. **FIFOS :** FIFOS (first in first out) are similar to the working of pipes. FIFOS also provide half duplex flow of data just like pipes. The difference between *fifos* and pipes is that the former is identified in the file system with a name, while the later is not. That is *fifos* are named pipes.

3. **Shared Memory :** Shared memory facility enables an area of memory to be shared by two or more processes. Use of this facility requires computer hardware support for shared memory segments.

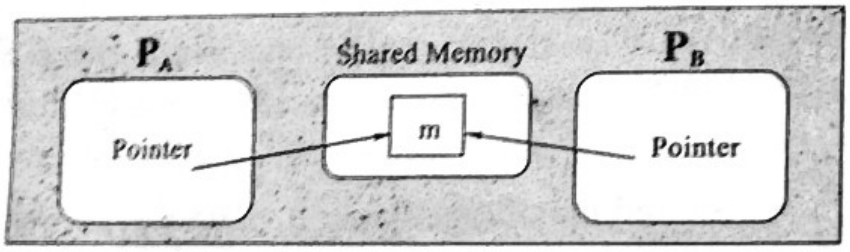


FIG 3.2 : Shared Memory Approach

4. **Message Passing :** A message includes a sequence of characters that are sent to and receive by processes. The message exchange system uses two system calls send and receive. The send call sends a message to the destination process, and the receive call receives a message from the source process.

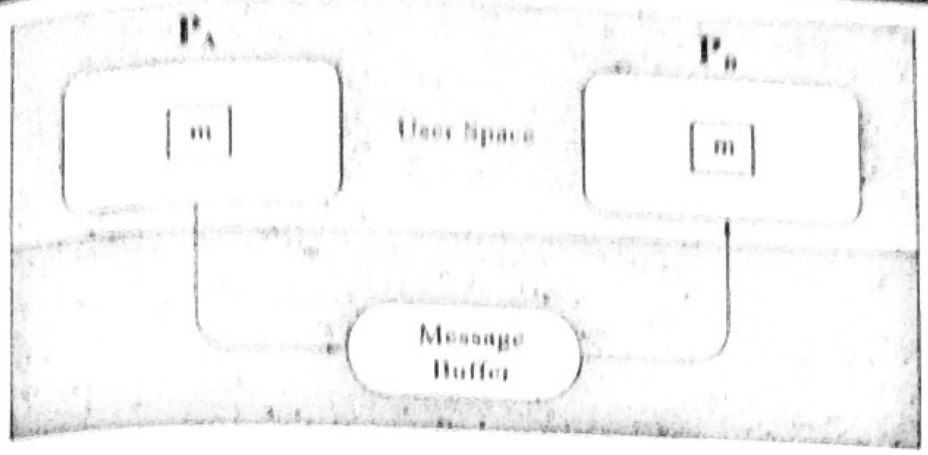


FIG 3.3 : Message Passing Approach

5. **Sockets** : Sockets are the most popular of all the IPC mechanisms because : other mechanisms do not support communication between two machines. Sockets provide a full - duplex communication between processes that do not necessarily run on the same Operating systems.

   Generally, sockets are associated with the concept of network communication in the form of client - server programming; a pair of processes of which one will be a client and one a server. The client process will send requests to the server, and the server process will send replies as an acknowledgment to the client requests.

3.4 DEAD LOCK

In a multi programming environment a process is said to be in a state of dead lock if it is waiting for an event that will never occur. In a system deadlock, one or more processes are deadlocked. A process requests resources, and if the resources are not available at that time, the process enters a wait state. It may happen that waiting process will never change state, because the resources they have requested are held by other waiting processes. This situation is called ***deadlock***.

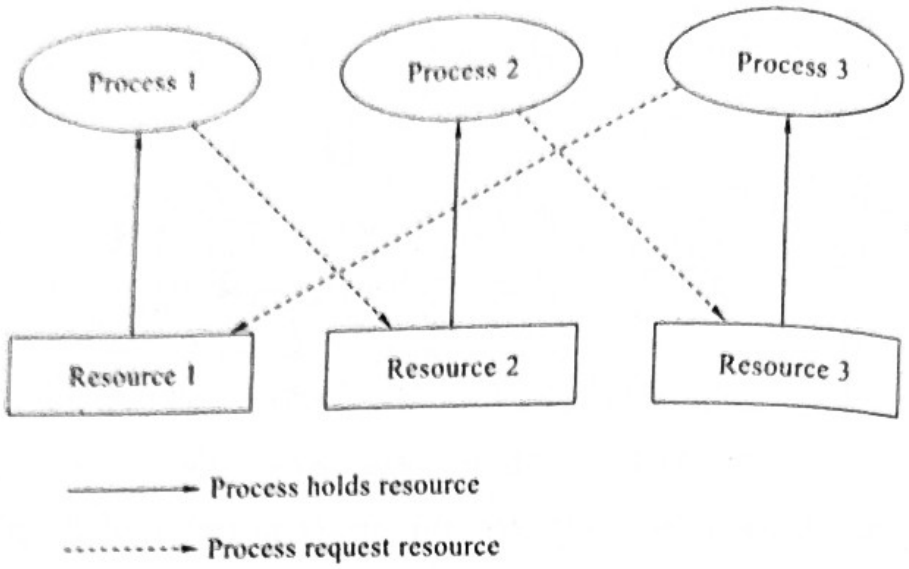


FIG 3.4 :

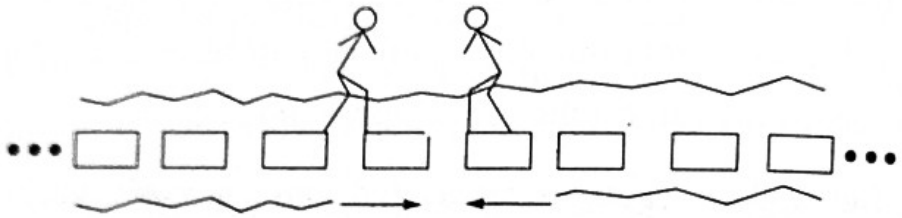


FIG 3.5 : River Crossing with a Deadlock Situation

3.5 NECESSARY CONDITIONS FOR ARISING DEADLOCKS

A deadlock occurs in a system if the following four conditions hold simultaneously :

1. **Mutual Exclusion :** At least one of the resources is non-sharable that is only one process at a time can be use the resource.

2. **Hold and Wait :** A process exists that is holding on to at least one resource and waiting for an additional resource held by another process.

3. **No Preemption :** Resources cannot be preempted that is a resource is released only by the process that is holding it.

4. **Circular Wait :** There exist a set of processes $P_0, P_1 \ldots, P_n$ of waiting processes such that P_0 is waiting for a resource held by P_1, P_1 is waiting for a resource held by P_2, . . .

P_{n-1} is waiting for a resource held P_n and P_n is in turn waiting for a resource held by P_0.

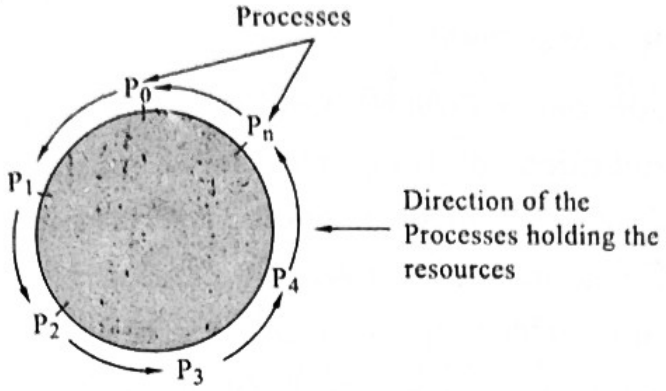


FIG 3.6 : Circular Wait

3.6 VARIOUS TECHNIQUES FOR DEADLOCK PREVENTION

Deadlock prevention means placing restrictions on resource requests so that deadlocks cannot occur. Denying at least any one of the conditions of deadlock can prevent deadlock.

1. **Mutual Exclusion :** Only for non - sharable resources the mutual exclusion concept should be applied.

   For example printer is a non - sharable resource. If one program is using printer then other programs must wait for the printer. This waiting may lead to indefinite waiting but to overcome this the non - sharable resources may be made sharable because for sharable resource, the concept of mutual exclusion is not required. However, by making the printer as sharable we get errors output hence printer cannot be made sharable. In general, it is not possible to prevent deadlock by denying mutual exclusion principle since some resources should be maintained as non - sharable resource.

2. **Preventing Hold and Wait :** There are two techniques to prevent hold and wait :

   (i) Request all resources before starting an execution : any program must request all the resources before starting the execution, acquire them, use them and release them

once execution is completed. In this way no process will wait for the resources during execution. Thus hold and wait is prevented.

(ii) Any process will request for new resources only when it has none i.e., if a process has 2 resources and requires 3 more resources, then the process can request for these 3 resources after releasing 2 resources it was holding. After releasing the resources, it will request for new resources. If they are busy, the process waits without holding any resources. Suppose if the request resource are free, then they can be allocated to the process.

3. **Preventing Non-Preemption :** The third condition is that there is no preemption of resource that have already been allocated To ensure that this condition does not hold, we can use th following protocol :

   If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being hold are preempted i.e., these resources are implicity released. The preempted resources are added to the list of resources for which the process is waiting. The process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting.

4. **Preventing Circular Wait :** To prevent circular wait, we impose ordering of all resource type i.e., a unique integer number is assigned to each resource. No two resources will have same *id*. The concept is request only for resources of higher numbers or higher ids. If a process is requesting for resources of higher "*ids*", it must be released. If a process acquires all of the resources it needs at one time, then it will never be in a situation where it is holding a resource and waiting for another resource. This will prevent deadlock.

Thus, we can give each resource a unique, positive integer and only acquire resources in ascending, numerical order. This will prevent any circular wait.

3.7 DEADLOCK AVOIDANCE AND DETECTION

Deadlock prevention algorithms ensure that at least one of the four necessary conditions for deadlocks namely mutual exclusion, hold and wait, no preemption and circular wait do not hold. This disadvantage with prevention algorithms is poor resource utilization and thus reduced system throughout.

An alternate method is to avoid deadlocks. In this case additional a prior information about the usage of resources by processes is required. This information helps to decide on whether a process should wait for a resource or not. Decision about a request is based on all the resources available, resources allocated to processes, future requests and releases by processes.

A deadlock avoidance algorithm requires each process to make known in advance the maximum number of resources of each type that it may need. Also known is the maximum number of resources of each type available. Using both the above a prior knowledge, deadlock avoidance algorithm ensures that a circular wait condition never occurs.

Safe State : A system is said to be in a safe state if it can allocate resources upto the maximum available and is not in a state of deadlock. A safe sequence of processes always ensures a safe state. A sequence of processes $< P_1, P_2, \ldots . P_n >$ is safe for the current allocation of resources to processes if resource requests from each P_i can be satisfied from the currently available resources and the resources held by the P_j where $j < i$. If the state is safe then P_i requesting for resources can wait till P_j's have completed. If such a safe sequence does not exist, then the system is in an unsafe state.

A safe state is not a deadlock state. Conversely a deadlock state is an unsafe state. But all unsafe states are not deadlock states as shown in Fig. 3.7.

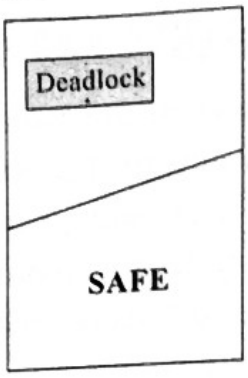


FIG 3.7 : Safe, Unsafe and Deadlock State Spaces

If a system is in a safe state it can stay away from an unsafe state and thus avoid deadlock. On the other hand, if a system is in an unsafe state, deadlocks cannot be avoided.

Illustration : A system has 12 instances of a resource type and 3 processes using these resources. The maximum requirements for the resource by the processes and their current allocation at an instance say t_0 is as shown below :

Process	Maximum	Current
P_0	10	5
P_1	4	2
P_3	9	2

At the instant t_0, the system is in a safe state and one safe sequence is $< P_1, P_0, P_2 >$.

This is true because of the following facts :

- Out of 12 instances of the resource, 9 are currently allocated and 3 are free.
- P_1 needs only 2 more, its maximum being 4, can be alloted 2.
- Now only 1 instance of the resource is free.

- When P_1 terminates, 5 instances of the resource will be free.
- P_0 needs only 5 more, its maximum being 10, can be allotted 5.

Now resource is not free.

- Once P_0 terminates, 10 instances of the resource will be free.
- P_3 needs only 7 more, its maximum being 9, can be allotted 7.
- Now 3 instances of the resource are free.
- When P_3 terminates, all 12 instances of the resource will be free.

Thus the sequence $< P_1, P_0, P_3 >$ is a safe sequence and the system is in a safe state. Let us now consider the following scenario at an instant t_1. In addition to the allocation shown in the table above, P_2 requests for 1 more instance of the resource and the allocation is made. At the instance t_1, a safe sequence cannot be found as shown below :

- Out of 12 instances of the resource, 10 are currently allocated and 2 are free.
- P_1 needs only 2 more, its maximum being 4, can be alloted 2.
- 7 Now resource is not free.
- Once P_1 terminates, 4 instance of the resource will be free.
- P_0 needs 5 more while P_2 needs 6 more.
- Since both P_0 and P_2 cannot be granted resources, they wait.

The result is a deadlock.

Thus the system has gone from a safe state at time instant t_0 into an unsafe state at an instant t_1. The extra resource that was granted to P_2 at the instant t_1 was a mistake. P_2 should have waited till other processes finished and released their resources.

Since resource available should not be allocated right away as the system may enter an unsafe state, resource utilization is low if deadlock avoidance algorithms are used.

Deadlock Detection : If a system does not employ either is deadlock prevention or a deadlock avoidance algorithm, then a deadlock situation may occur. In this situation the system must provide.

- An algorithm that examines the states of the system to determine whether a deadlock has occurred.
- An algorithm to recover from the deadlock.

Resource allocation graph is one of the important technique used to detect deadlock. The resource allocation graph is a directed graph consisting of vertices and directed edges. The vertex set is partitioned into two types, a subset representing processes and another subset representing resources. Pictorially, the resources are represented by rectangles with dots within, each dot representing an instance of the resource and circles represent processes.

A directed edge from a process to a resource ($P_i \rightarrow R_i$) signifies a request from a process P_i for an instance of the resource R_j and P_i is waiting for R_j. A directed edge from a resource to a process ($R_j \rightarrow P_i$) indicates that an instance of the resource R_j has been allotted to process P_i. Thus a resource allocation graph consists of vertices which include resources and processes and directed edges which consist of request edges and assignment edges. A request edge is introduced into the graph when a process requests for a resource. This edge is

converted into an assignment edge when the resource is granted. When the process releases the resource, the assignment edge is deleted.

Consider the Following System : There are 3 processes P_1, P_2 and P_3.

Resources R_1, R_2, R_3 and R_4 have instances 1, 2, 1, and 3 respectively.

- P_1 is holding R_2 and waiting for R_1.
- P_2 is holding R_1, R_2 and is waiting for R_3.
- P_3 is holding R_3.

The resource allocation graph for a system in the above situation is as shown in Fig. 3.8.

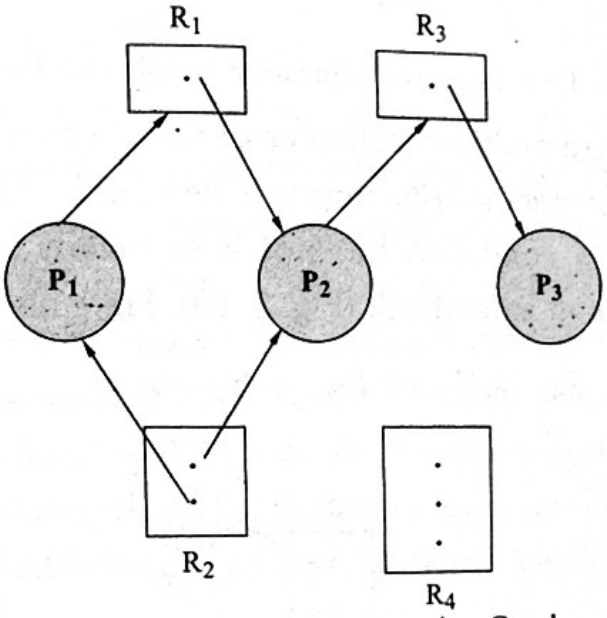


FIG 3.8 : Resource Allocation Graph

If a resource allocation graph has no cycles (a closed loop in the direction of the edges), then the system is not in a state of deadlock. If on the other hand, there are cycles, then a deadlock may exist. If there are only single instances of each resource type, then a cycle in a resource allocation graph is a necessary and sufficient condition for existence of a deadlock (Fig. 3.9). Here two cycles exits.

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

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

Processes P_0, P_1 and P_3 are deadlock and are in a circular wait. P_2 is waiting for R_3 held by P_3. P_3 is waiting for P_1 or P_2 to release R_2. So also P_1 is waiting for P_2 to release R_1.

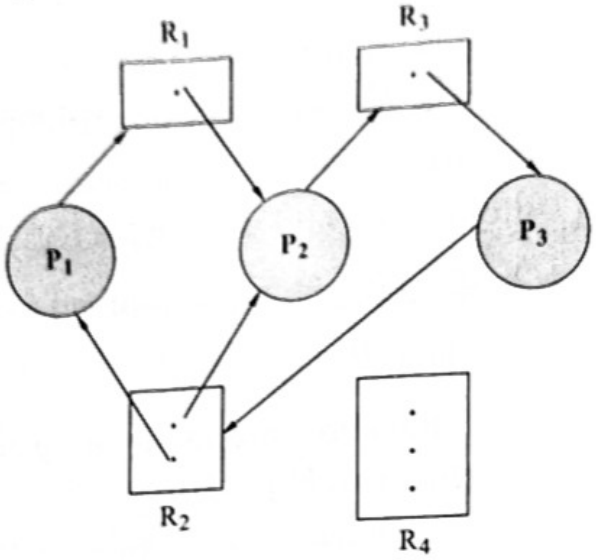


FIG 3.9 : Resource Allocation Graph with Deadlock

If there are multiple instances of resources types, then a cycle does not necessarily imply a deadlock. Here a cycle is a necessary condition but not a sufficient condition for the existence of a deadlock (Fig. 3.10). Here also there is a cycle.

$$P_1 \rightarrow R_1 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$$

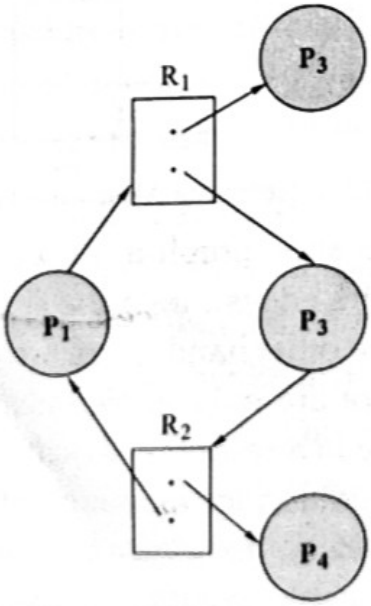


FIG 3.10 : Resource Allocation Graph with A Cycle but no Deadlock

The cycle does not imply a deadlock because an instance of R_1 released by P_2 could be assigned to P_1 or an instance of R_2 released by P_4 could be assigned to P_3 there by breaking the cycle.

3.8 RECOVERING FROM DEADLOCK

Once a deadlock has been detected, it must be broken. Breaking a deadlock may be manual by the operator when informed of the deadlock or automatically by the system.

There exist two options for breaking deadlocks.

1. Abort one or more processes to break the circular - wait condition causing deadlock.
2. Preempting resources from one or more processes which are deadlocked.

In the first option, one or more processes involved in deadlock could be terminated to break the deadlock. Then either abort all processes or abort one process at a time till deadlock is broken. The first case guarantees that the deadlock will be broken. But processes that have executed for a long time will have to restart all over again. The second case is better but has considerable overhead as detection algorithm has to be invoked after terminating every process. Also choosing a process that becomes a victim for termination is based on many factors like

- Priority of the processes.
- Length of time each process has executed and how much more it needs for completion.
- Type of resources and the instances of each that the processes use
- Need for additional resources to complete.
- Nature of the processes, whether iterative or batch.

Based on these and many more factors, a process that incurs minimum cost on termination becomes a victim.

In the second option some resources are preempted from some processes and give to other processes until the deadlock cycle is broken. Selecting the victim whose resources can be preempted is again based on the minimum cost criteria. Parameters such as number of resources a process is holding and the amount of these resources used thus far by the process are used to select a victim. When resources are preempted, the process holding the resource cannot continue. A simple solution is to abort the process also. Better still is to rollback the process is required restart later. To determine this safe state, more information about running processes is required which is again an overhead. Also starvation may occur when a victim is selected for preemption, the reason being resources from the same process may again and again be preempted. As a result the process starves for want of resources. Ensuring that a process can be victim only a finite number of times by having this information as one of the parameters for victim selection could prevent starvation.

Prevention, avoidance and detection are the three basic approaches to handle deadlocks. But they do not encompass all the problems encountered. Thus a combined approach of all the three basic approaches is used.

REVIEW QUESTIONS

Part-A

1. What is seamaphore? **(March/April. 2018)**
2. Give an example of seamaphore.
3. Write the uses of semaphores. **(April/May. 2012, 2011)**
4. What is mutual exclusion **(Oct. 2020)**
5. Define deadlock. **(March/April. 2008)**
6. Write about deadlock prevention. **(March/April. 2014)**

Part-B

1. List three options for breaking an existing deadlock. **(April/May. 2011)**
2. List three overall strategies in handling deadlocks. **(March/April. 2013)**
3. List the necessary conditions for arising deadlock. **(March/April. 2018, 2014)**
4. List two ways that we can break the second condition to prevent deadlock. **(April/May. 2012)**
5. What is a starvation problem with respect to CPU scheduling? **(April/May. 2012)**
6. What is a circular wait in deadlocks ? **(April/May. 2011)**

Part-C

1. Explain about semaphase and its operation.
2. Explain about inter process communication. **(April. 2007)**
3. Define deadlock. Explain deadlocks prevented. **(March/April. 2016)**
4. Explain deadlock recovery process.

5. Explain how deadlocks can be avoided and detected. **(Mach/April. 2018, April. 2008)**

6. Write in detail about process control block. **(April. 2008)**

7. Describe the differences among short-term, medium-term and long-term scheduling. **(April/May. 2011)**

CHAPTER 4

MEMORY MANAGEMENT

CHAPTER OUTLINE

4.0 INTRODUCTION · MEMORY MANAGEMENT

Memory is the essential part of the computer system as both CPU and I/O devices accesses it. Due to this reason the memory must be efficiently managed for maximum utilization of computer system.

Memory manager is defined as the part of operating system that manages memory. The job of memory manager is to keep track of which parts of main memory are in use and which parts of are not in use. It also allocates memory to process when they need it and deal locate memory for the processes when they have executed completely i.e., when the process finishes their job. It also manages the swapping between main memory and disk when the main memory is not big enough to store all the processes. Memory manager uses algorithms and techniques while dealing with allocation and deallocation of memory to various processes or programs.

In uniprogramming systems, main memory is divided into two parts : one part is for the operating system and the other part is for the program currently being executed.

In multiprogramming environments, the user part is further subdivided to accommodate multiple processes.

Following are the important functions of operating systems memory manager :

- Provide the memory space for the process to execute.
- Reclaim the allocated memory space when the process completes its execution.
- To provided a satisfactory level of performance (i.e., process execution speed) for the system users.
- To protect each process from each other.
- When desired, to enable sharing of memory space between processes.

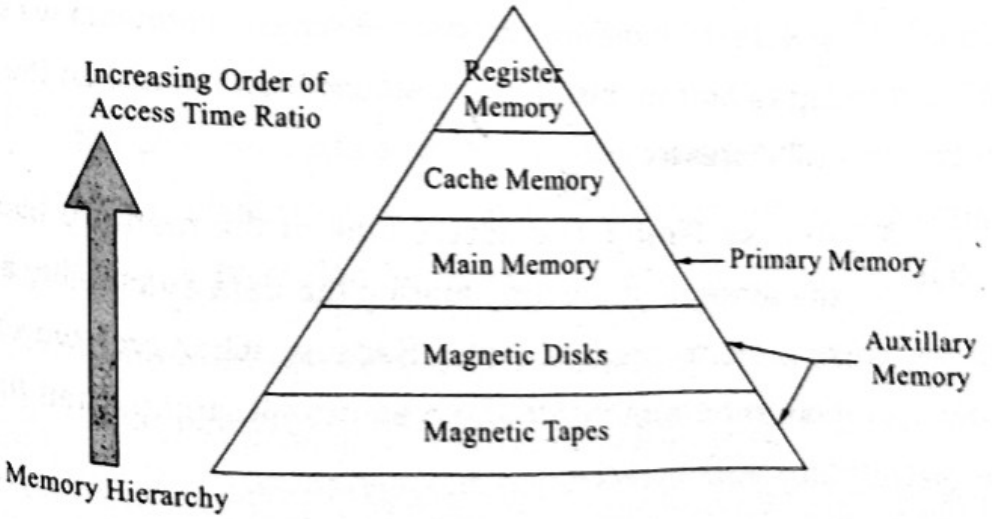
4.1 MEMORY HIERARCHY

What is Memory Hierarchy? The memory in a computer can be divided into five hierarchies based on the speed as well as use. The processor can move from one level to another based on its requirements. The five hierarchies in the memory are registers, cache, main memory, magnetic discs, and magnetic tapes. The first three hierarchies are volatile memories which mean when there is no power, and then automatically they lose their stored data. Whereas the last two hierarchies are not volatile which means they store the data permanently.

A memory element is the set of storage devices which stores the binary data in the type of bits. In general, the storage of memory can be classified into two categories such as volatile as well as non- volatile.

Memory Hierarchy in Computer Architecture : The memory hierarchy design in a computer system mainly includes different storage devices. Most of the computers were inbuilt with extra storage to run more powerfully beyond the main memory capacity.

The following **memory hierarchy diagram** is a hierarchical pyramid for computer memory. The designing of the memory hierarchy is divided into two types such as primary (Internal) memory and secondary (External) memory.



1. **Primary Memory :** The primary memory is also known as internal memory, and this is accessible by the processor straightly. This memory includes main, cache, as well as CPU registers.

2. **Secondary Memory :** The secondary memory is also known as external memory, and this is accessible by the processor through an input/output module. This memory includes an optical disk, magnetic disk, and magnetic tape.

4.1.1 Characteristics of Memory Hierarchy

The memory hierarchy characteristics mainly include the following.

1. **Performance :** Previously, the designing of a computer system was done without memory hierarchy, and the speed gap among the main memory as well as the CPU registers enhances because of the huge disparity in access time, which will cause the lower performance of the system. So, the enhancement was mandatory. The enhancement of this was designed in the memory hierarchy model due to the system's performance increase.

2. **Ability :** The ability of the memory hierarchy is the total amount of data the memory can store. Because whenever we shift from top to bottom inside the memory hierarchy, then the capacity will increase.

3. **Access Time :** The access time in the memory hierarchy is the interval of the time among the data availability as well as request to read or write. Because whenever we shift from top to bottom inside the memory hierarchy, then the access time will increase.

4. **Cost Per Bit :** When we shift from bottom to top inside the memory hierarchy, then the cost for each bit will increase which means an internal Memory is expensive compared with external memory.

4.1.2 Memory Hierarchy Design

The memory hierarchy in computers mainly includes the following.

1. **Registers :** Usually, the register is a static RAM or SRAM in the processor of the computer which is used for holding the data word which is typically 64 or 128 bits. The program counter register is the most important as well as found in all the processors. Most of the processors use a status word register as well as an accumulator. A status word register is used for decision making, and the accumulator is used to store the data like mathematical operation. Usually, computers like complex instruction set computers have so many registers for accepting main memory, and RISC- reduced instruction set computers have more registers.

2. **Cache Memory :** Cache memory can also be found in the processor, however rarely it may be another IC (integrated circuit) which is separated into levels. The cache holds the chunk of data which are frequently used from main memory. When the processor has a single core then it will have two (or) more cache levels rarely. Present multi-core processors will be having three, 2-levels for each one core, and one level is shared.

3. **Main Memory :** The main memory in the computer is nothing but, the memory unit in the CPU that communicates directly. It is the main storage unit of the computer. This memory is fast

as well as large memory used for storing the data throughout the operations of the computer. This memory is made up of RAM as well as ROM.

4. **Magnetic Disks :** The magnetic disks in the computer are circular plates fabricated of plastic otherwise metal by magnetized material. Frequently, two faces of the disk are utilized as well as many disks may be stacked on one spindle by read or write heads obtainable on every plane. All the disks in computer turn jointly at high speed. The tracks in the computer are nothing but bits which are stored within the magnetized plane in spots next to concentric circles. These are usually separated into sections which are named as sectors.

5. **Magnetic Tape :** This tape is a normal magnetic record which is designed with a slender magnetizable covering on an extended, plastic film of the thin strip. This is mainly used to back up huge data. Whenever the computer requires accessing a strip, first it will mount to access the data. Once the data is allowed, then it will be unmounted. The access time of memory will be slower within magnetic strip as well as it will take a few minutes for accessing a strip.

Advantages of Memory Hierarchy : The need for a memory hierarchy includes the following.

- Memory distributing is simple and economical.
- Removes external destruction.
- Data can be spread all over.
- Permits demand paging and pre-paging.
- Swapping will be more proficient.

4.2 ADDRESS BINDING, DYNAMIC LOADING, DYNAMIC LINKING

4.2.1 Address Binding

To reference any particular memory address, one needs a location of that memory address. These are physical addresses. In program, the memory locations that are referenced are called ***logical addresses*** because an address specified by a program may or may not be the correct physical address. There are basically three different ways that logical addresses are mapped to physical address, which is called ***address binding***. These address binding techniques depend on the memory management methods that the OS of your computer use.

Compile Time Address Binding : generates absolute code, in other words, the addresses are already in place, and when the program is run, the logical address or either absolute where logical locations 000 - 999 actually correspond to physical locations 000 - 9999, or where an offset is added to the logical locations to generate the physical locations.

Load Time Address Binding : If the starting location of the program isn't known at compile time of the program, then the compiler will generate re-locatable code, i.e., the locations would be changed during the loading of the program.

Run Time Address Binding : If the starting location of the program is to be moved during execution of the program, then the binding must be delayed until run - time. Physical addresses are generated during execution by adding the value of logical addresses to an offset register.

4.2.2 Dynamic Loading

Dynamic loading involves loading routines into memory into 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.

Help from the operating system is not essential for dynamic loading to take place. Although, the operating system may provide library routines to aid dynamic loading.

4.2.3 *Dynamic Linking*

Dynamic linking is often for libraries. Only a **"stub"** of the library is kept in the programs image.

When a program calls one of these routines, the routine is loaded and linked into memory.

All programs share the one copy of the same library routine.

Dynamic linking requires the operating systems intervention as sharing between processes is required.

4.3 OVERLAYS

Following figure illustrates the overlay concept.

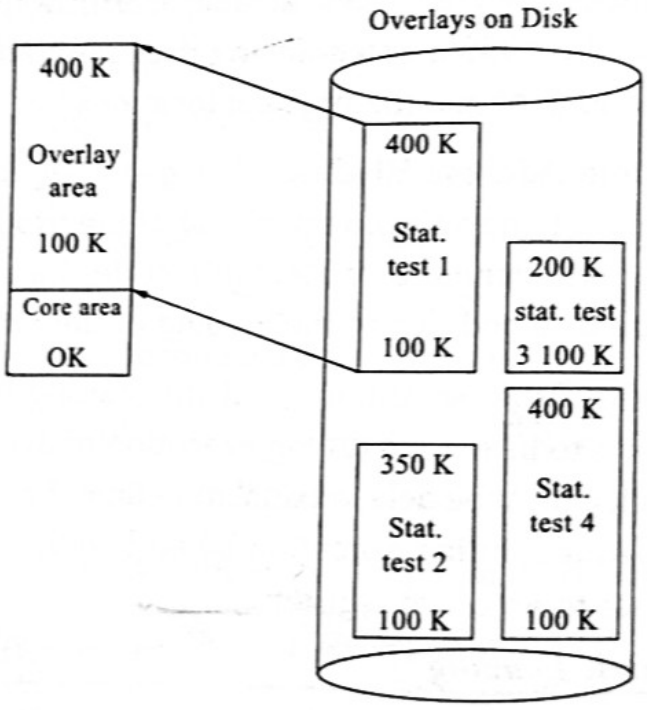


FIG 4.1 : Overlays

IBM *PCs* when introduced they did not have enough memory. Memory was very expensive at that time. So *PC* developers introduced a concept known as overlay. Overlays are software

method of running a large program on a small address space (memory). The concept of overlaying that the program is constructed as a number of separate modules called ***overlay*** which can be loaded individually and selectively into small memory area.

Test 1, 2, 3 and 4 are overlays. In overlay area one of this overlay will be loaded. Overlays allow to run a program that is larger than the address space (memory) running it in.

4.4 SWAPPING

A process needs to be in memory to be executed. A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution. For example, assume a multiprogramming environment with a round robin CPU scheduling algorithm. When a quantum expires the memory manager will start to swap out the process that must finished, and to swap in another process to the memory space that has been finished, and to swap in another process to the memory space that has been freed. In the mean time the CPU scheduler will allocate a time slice to some of the process in memory. When each process finishes its quantum, it will be swapped with other process.

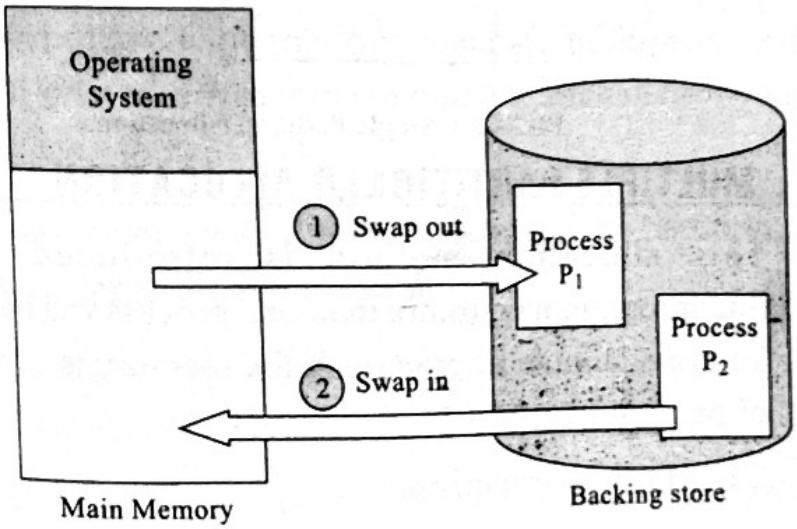


FIG 4.2 : Swapping of Two Processes Using a Disk as a Backing Store

4.5 SINGLE PARTITION ALLOCATION

In single partition allocation all the available memory is available as one partition. If a process wants a memory of 1k ad the partition is 1000k then also the entire partition will be allocated to the process.

If another process comes and wants memory then it as to wait. This type of memory management technique is used in days of computing.

Advantage :

- Its simple to implement.

Disadvantage :

- Poor utilization of memory.
- Multi programming is not possible.

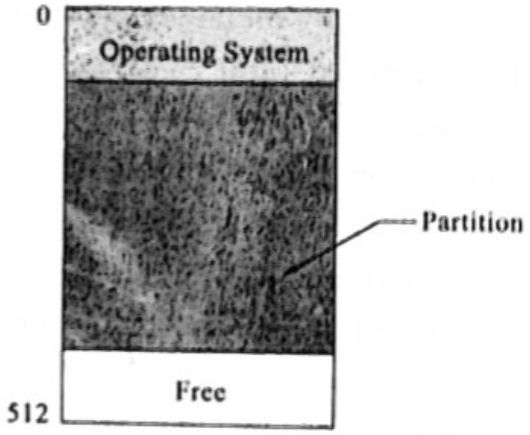


FIG 4.3 : Single Partition Allocation

4.6 MULTIPLE PARTITIONED ALLOCATION

This allocation method is introduced to provide multiprogramming (more than one process will be in memory). To provide multiprogramming the memory is divided into no. of partitions.

1. Fixed partition size.
2. Variable partition size.

In case of fixed partition the memory is divided into no. of partitions and all the partitions are of same size. A process can be loaded into one of the partition for execution and on termination the partition will be freed for another process to be loaded.

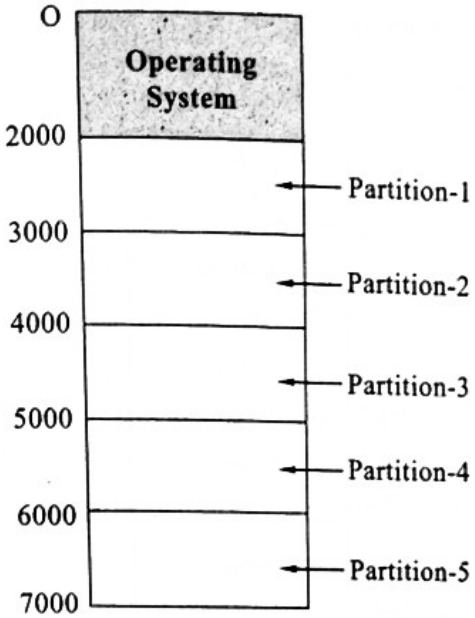


FIG 4.4 : Multiple Partition Allocation

Let us assume that the available memory is 5000 *k* it can be divided into 5 partitions of 1000 *k* size. If a process whose memory requirement is less than 1000 *k* then it can be accommodated in any of the partition.

The advantage with this approach is multiprogramming support. That means more than one process can be in memory at the same moment.

The disadvantage with this approach is, if the partition size is small then large programs can not run.

If the partition size is large than there is wastage of space in partitions.

For example, if partition size is 1000 *k* then a process with above this size say 1050 comes it can not be accommodated in any of the partitions. If partition size is made large say 2000

then this process can run but the wastage in partition is 950 k. If partition size is more degree of multi programming is less otherwise it is more.

4.7 CONCEPT OF FRAGMENTATION

Space at the end of each partition is unused by a process is wasted. Wasted space with in a partition is called ***internal fragmentation***.

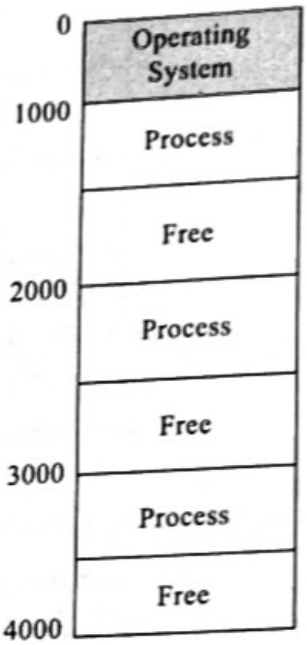


FIG 4.5 : Internal Fragmentation

To minimize internal fragmentation problem variable partitioned technique is introduced. In this method memory is divided into no. of partitions and the partition are not of same size. A process can be placed into a partition based on following algorithms.

- Best fit.
- First fit.
- Worst fit.

1. **Best Fit :** An incoming process is placed in partition in which it fits most tightly i.e., for all the choices of partitions, the difference between partition size and process size is least.
2. **First Fit :** An incoming process is placed in the first available free partition which can accommodate it.

3. **Worst Fit :** An incoming process is placed in the partition which leaves the maximum amount of unused space which logically must be current maximum partition.

Consider the following Fig. 4.6.

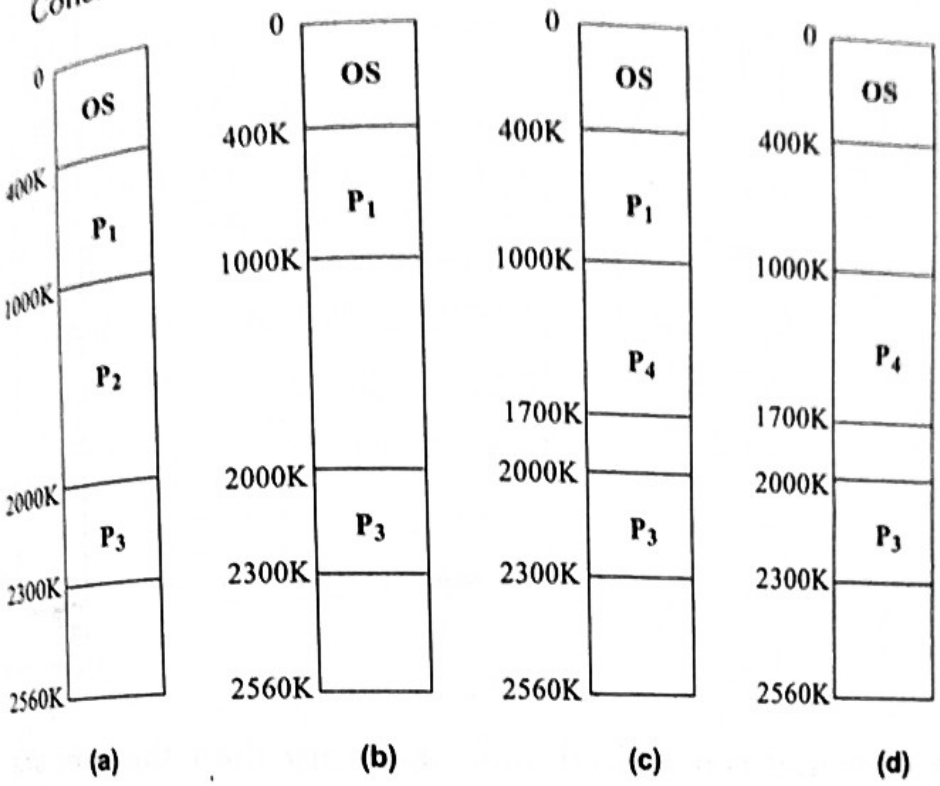


FIG 4.6 : Memory Allocation and Job Scheduling

Consider the Fig. 4.6 (a) here only one partition is free.

In Fig. 4.6 (b) process P_2 completed.

In Fig. 4.6 (c) New process P_4 is entered.

In Fig. 4.6 (d) Process P_1 is completed.

Now partition 1 of size 600 K is free and partition 4 of size 260 K is free

Now if a process comes of size 700 K we cannot accommodate it in any of the partitions even though the process size is less than the available memory. This type of problem is known as ***external fragmentation.***

4.8 PAGING CONCEPT

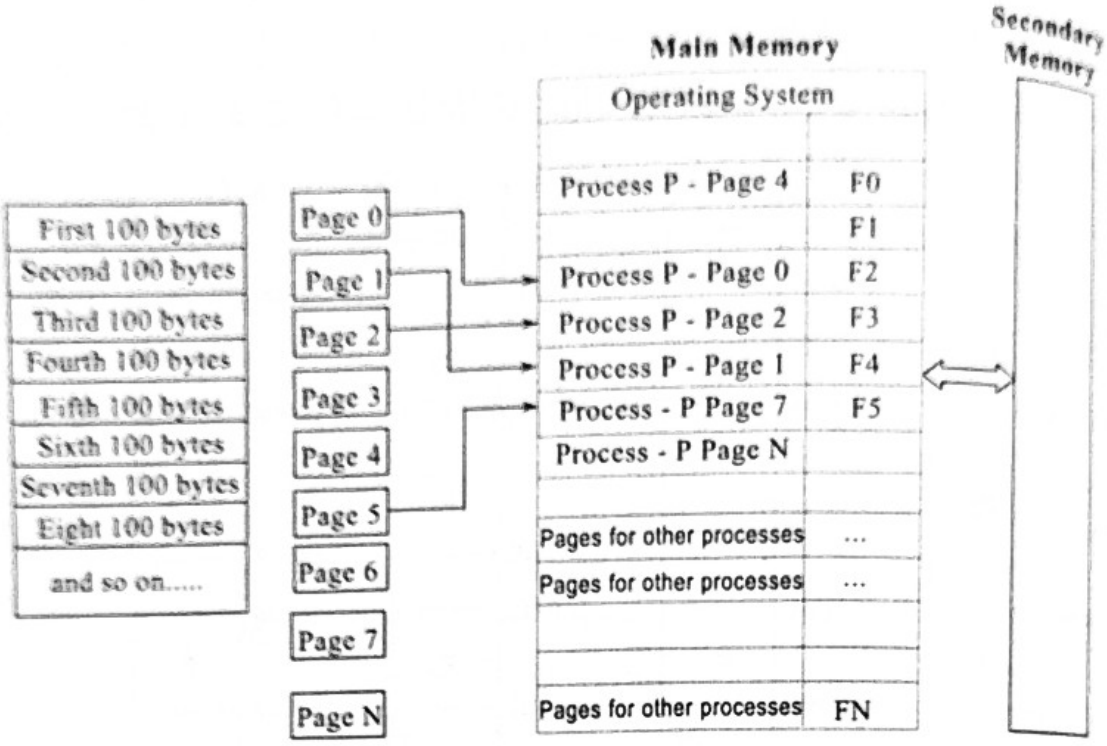


FIG 4.7 :

A computer can address more memory than the amount physically installed on the system. This extra memory is actually called virtual memory and it is a section of a hard that's set up to emulate the computer's RAM. Paging technique plays an important role in implementing virtual memory.

Paging is a memory management technique in which process address space is broken into blocks of the same size called pages (size is power of 2, between 512 bytes and 8192 bytes). The size of the process is measured in the number of pages.

Similarly, main memory is divided into small fixed-sized blocks of (physical) memory called **frames** and the size of a frame is kept the same as that of a page to have optimum utilization of the main memory and to avoid external fragmentation.

4.9 HOW LOGICAL ADDRESS IS TRANSLATED INTO PHYSICAL ADDRESS

Address Translation : Page address is called ***logical address*** and represented by page number and the offset.

Logical address = Page number + Page offset

Frame address is called physical address and represented by a frame number and the offset.

Physical address = Frame number + Page offset

A data structure called page map table is used to keep track of the relation between a page of a process to a frame in physical memory.

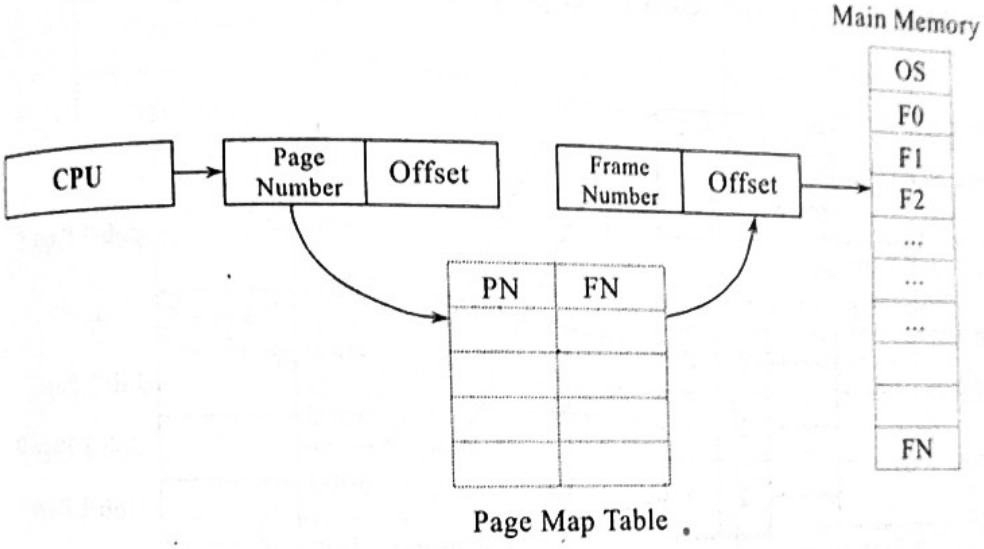


FIG 4.8 :

When the system allocates a frame to any page, it translates this logical address into a physical address and create entry into the page table to be used throughout execution of the program.

When a process is to be executed, its corresponding pages are loaded into any available memory frames. Suppose you have a program of 8 KB but your memory can accommodate only 5 KB at a given point in time, then the paging concept will come into picture. When a computer runs out of RAM, the operating system (OS) will move idle or unwanted pages of

memory to secondary memory to free up RAM for other processes and brings them back when needed by the program.

In paged memory management each process address space is divided into equal pieces, called ***pages*** and physical memory is divided into pieces of the same size called ***frames***. By providing a suitable hardware mapping facility any page can be placed into any frames.

The page remain logically contiguous but the corresponding blocks are not necessarily contiguous.

For the hardware to perform the mapping, page map tables are used.

Mapping means loading a page into frame.

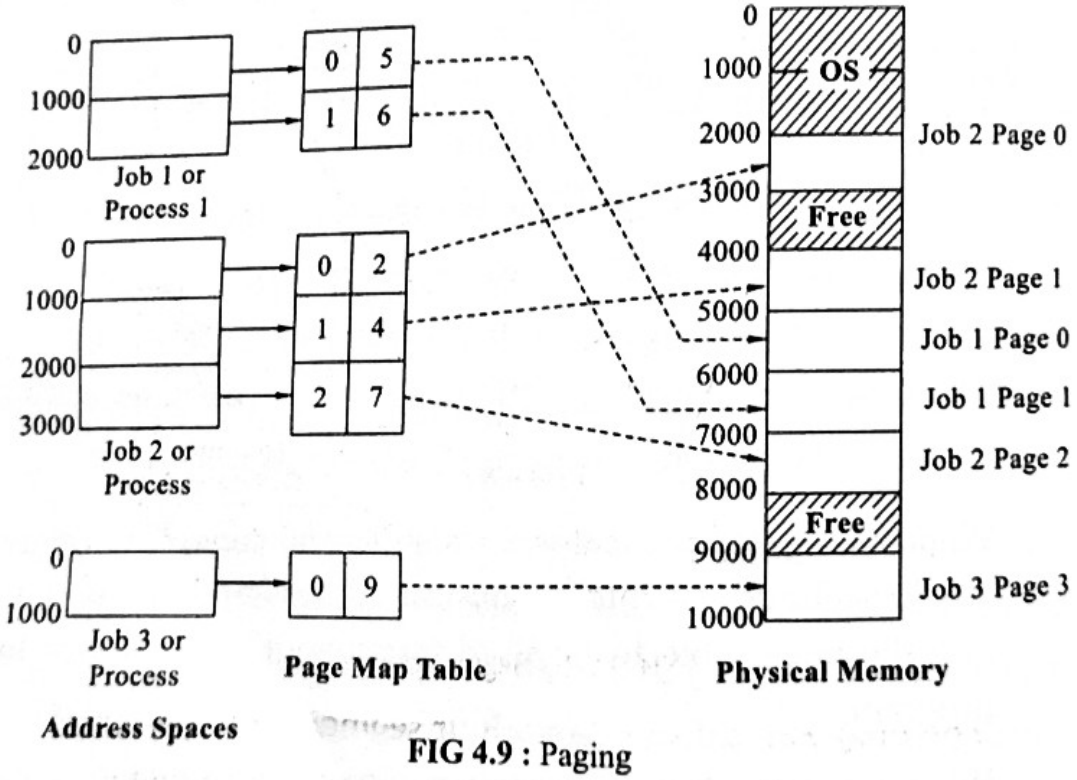


FIG 4.9 : Paging

A simple example, using a 1000 byte page size is shown in the above Fig. 3.9. Job 2 which has address space of 3000 bytes, is divided into three pages. The page table associated with job 2 indicates the location of its page. Page 0 is in block 2, page 1 is in block 4, page 2 is in block 7.

4.10 SEGMENTATION

Segmentation divides a program into a number of smaller blocks called segments. A segment can be defined as a logical grouping of instructions such as subroutine, array or a data area. Each segment is allocated to memory independently. Responsibility of dividing a program into segment lies with the user or compiler, the operating system is not involved.

Segmentation is similar to variable partition allocation, with the improvement that the process in divided into several portions.

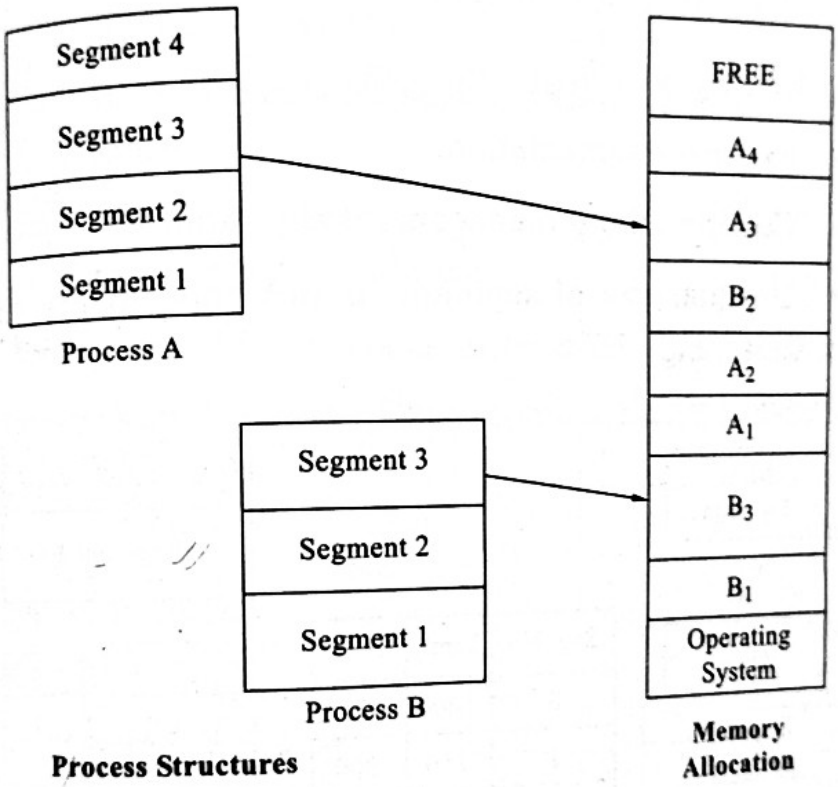


FIG 4.10 : Segmentation

The program is divided into four segments namely segment 0 . . . . segment 3. The segments are loaded into memory using segment table.

Limit indicates size of the segment.

Base indicates starting address of the segment.

Advantages of Segmentation :

- No internal fragmentation.
- Segment table consume less memory than page tables (only one entry per actual segment as opposed to one entry per page in paging method.
- Because of the small segment table, memory reference is easy.
- Lends itself to sharing data amoung processes.
- Lents itself to protection.
- Supports virtual memory.

Disadvantages of Segmentation :

- External fragmentation.
- Costly memory management algorithm..
- Unequal size of segments is not good in the case of swapping.

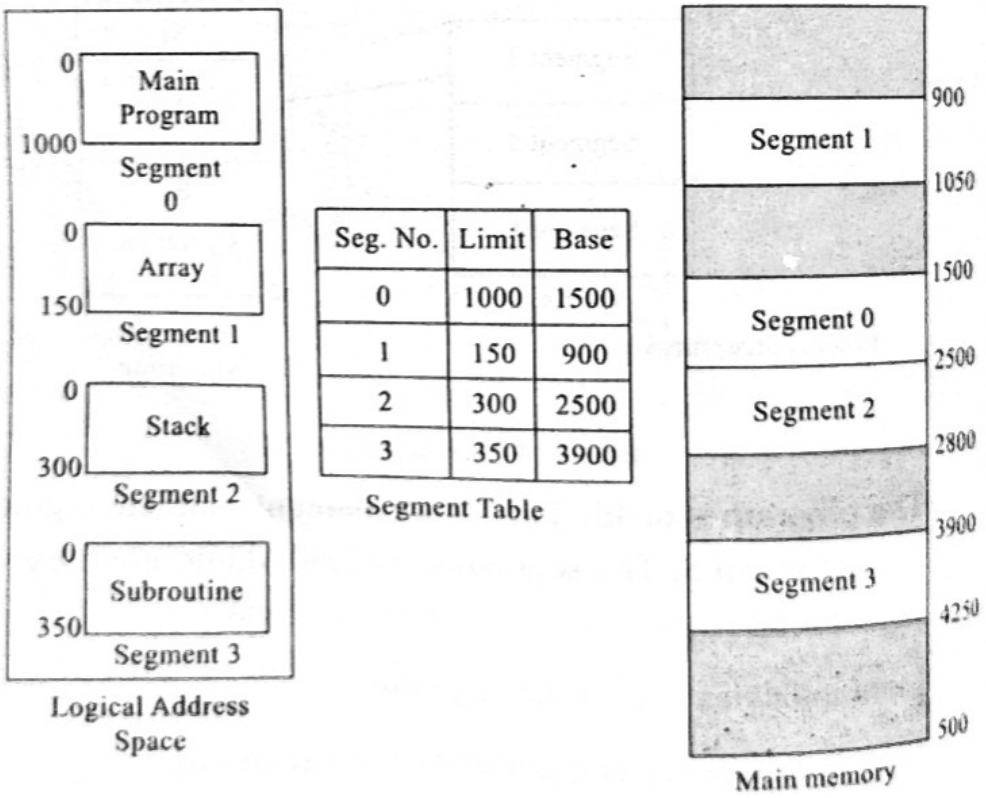


FIG 4.11 : Example of Segmentation

4.11 SEGMENTATION WITH PAGING

In a combined paging/segmentation system a user's address space is broken up into a number of segements. Each segment is broken up into a number of fixed-sized pages which are equal in length to a mian memory frame.

- Segmentation is visible to the programmer.
- Paging is transparent to the programmer.

- Most architectures support segmentation and paging.
- Basic idea,
  - Segements exist in virtual address space.
  - Base address in segment descriptor table is a virtual address.
  - Use paging mechanism to translate this virtual address into a physical address.
- Now an entire segment does not have to be in memroy at one time.
  - Only the part of the segment that we need will be in memory.

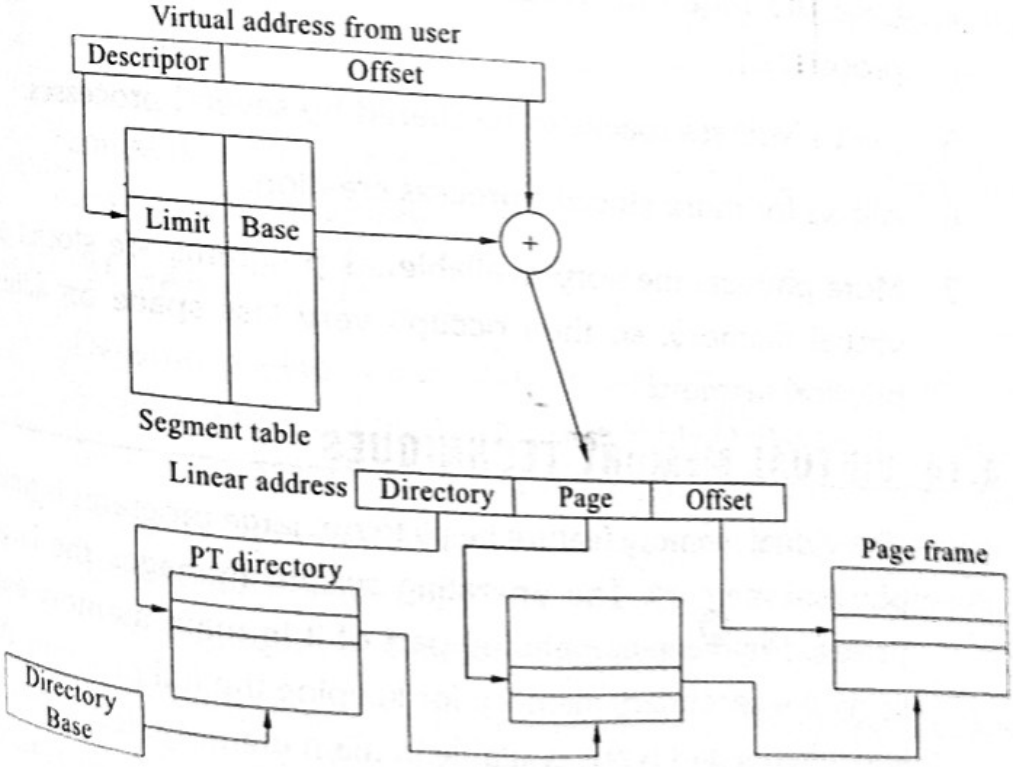


FIG 4.12 :

4.12 VIRTUAL MEMORY

Virtual memory is a common part of operating system on desktop computers. The term virtual memroy refers to something which appears to be present but actually it is not.

Virtual memory is a space where large programs can be store

The virtual memory technique allows users to use more memory for a program than the real memory of a computer.

4.13 BENEFITS OF VIRTUAL MEMORY

1. The whole program does not need to be in memory for it to be executed; only parts of the program currently used need to be in memory.
2. Large programs can be written, as virtual space available is huge compared to physical memory.
3. More programs could be run concurrently, because the only parts of the programs need to be in memory.
4. Less I/O required, leads to faster and easy swapping of processes.
5. Allows address spaces to be shared by several processes.
6. Allows for more efficient process creation.
7. More physical memory available, as programs are stored on virtual memory, so they occupy very less space on actual physical memory.

4.14 VIRTUAL MEMORY TECHNIQUES

The virtual memory feature helps to run large programs is small physical memory. The operating system manages the large program by keeping only the part of it in main memory and using the secondary memory for keeping the full program. As and when a part is not available in main memory, it is brought from secondary memory. The CPU hardware and operating

system collaborate in achieving this while program is being run. The programmer is not aware of the physical memory size. Virtual memory gives an illusion to the programmer that a large pool of memory is available.

In virtual memory systems, the operating system automatically manages the large programs. The program can be larger than the main memory capacity. The operating system stores the entire program on a hard disk whose capacity is much larger than the main memory size. At a given time, only some portion of the program are brought into the main memory from the hard disk. As and when needed, a portion which is presently not in main memory is sent - in from the hard disk, and at the same time, a portion of the program which is presently in the main memory is released out of the main memory and stored on the hard disk. This process is known as ***swapping***. In a virtual memory system, when a program is executed, swapping is performed as per the requirement on a continuous basis.

The CPU fetches instruction and data from the main memory. Whenever the required instruction or data is not presently available in the main memory, a page fault occurs. In response, the operating system loads the section of the program which contains the required instruction or data from the hard disk drive to main memory.

4.15 DEMAND PAGING

Demand paging is a concept in which the pages are loaded into memory only on demand i.e., whenever a page needs to be executes, that page will be requested by the CPU. The operating system will bring that page from disk and transfers it into memory. Thus pages are loaded into memory only when they are required, the remaining pages are in disk.

In demand paging the program or process is divided into several pages. Memory is divided into several frames. Instead of loading

the entire program into memory, only essential pages are loaded into memory i.e., all the pages of the program are present in secondary storage and required page will be transferred to main memory. Pages are transferred to memory only when a request to that page is generated.

EXAMPLE :

Consider a program of 5 pages (Page 0, 1, 2, 3, 4) initially P_0 is loaded in main memory. The moment a page is transferred into main memory, the page table pertaining to that page is updated. The page table contains information about the details of allocation of pages i.e., which page is present in which frame. Further since all pages are not present in main memory, another entry named validity bit is maintained. If the validity bit is 1 for a page, then that page is present in main memory at the specified frame. The format of the page table is as follows :

P.No.	Frame	Validity Bit
P_0	F_2	1
P_1	–	0
P_2	–	0
P_3	–	0
P_4	–	0

The validity bit for page P_0 is 1 which means P_0 is present in main memory at frame 2. The validity bit for the remaining pages are zeros which means P_1 thru P_4 are not present in main memory i.e., they are on disk.

When CPU executes the instruction present in P_0, it generates some address, these addresses are called logical addresses. Now CPU refers to page table and find whether this address falls in page 0 or not. Suppose if it falls in page 0 then CPU

continues execution smoothly. Suppose it does not fall in page 0 then there is a page fault i.e., the required page is not present in main memory and required page is brought into main memory.

Advantages of Demand Paging :

- Does not load the pages that are never accessed, so saves the memory for other programs and increases the degree of multiprogramming.
- Less loading latency at the program startup.
- Less disk over head because of fewer page reads.
- Ability to run large programs on the machine, even though it does not have sufficient memory to run the program. This method is better than an old technique called overlays.
- Does not need extra hardware support than what paging needs, since protection fault can be used to get page fault.

Disadvantages of Demand Paging :

- Individual programs face extra latency when they access a page for the first time. So preparing, a method of remembering which pages a process used when it last executed and preloading few of them, is used to improve performance.
- Memory management with page replacement algorithms become slightly more complex.

Comparisons Between Paging and Segmentation : Paging and segmentation both scheme having advantages and disadvantages, some times paging is useful and some times segmentation is useful. Consider the below table for better compression.

S.No.	Paging	Segmentation
1.	The main memory partitioned in to frames (or) blocks	The main memory partitioned in to segments.
2.	The logical address space divided into pages by compiler (or) memory management unit (MMU)	The logical address space divided into segments, specified by the programmer.
3.	This scheme suffering from internal fragmentation or page breaks.	This scheme suffering from external fragmentation.
4.	The operating system maintains a free frames list need not to search for free frame.	The operating system maintain the particulars of available memory.
5.	The operating system maintains a page map table for mapping between frames and pages.	The operating system maintains a segment map table for mapping purpose.
6.	This scheme does not supports the the users view of memory.	It supports users view of memory
7.	Processor uses the page number and displacement to calculate absolute address (P.D.)	Processor uses the segment number and displacement to calculate the absolute address (S.D)
8.	Multilevel paging is possible.	Multi level segmentation is also possible, But no use.

4.16 PAGE REPLACEMENTS

In multiprogramming there will be several processes competing for memory space. Consequently, the available real memory will become full of pages belonging to these processes. Fig. 4.13 shows three processes each with number of pages resident in memory. All the available memory pages have been utilized.

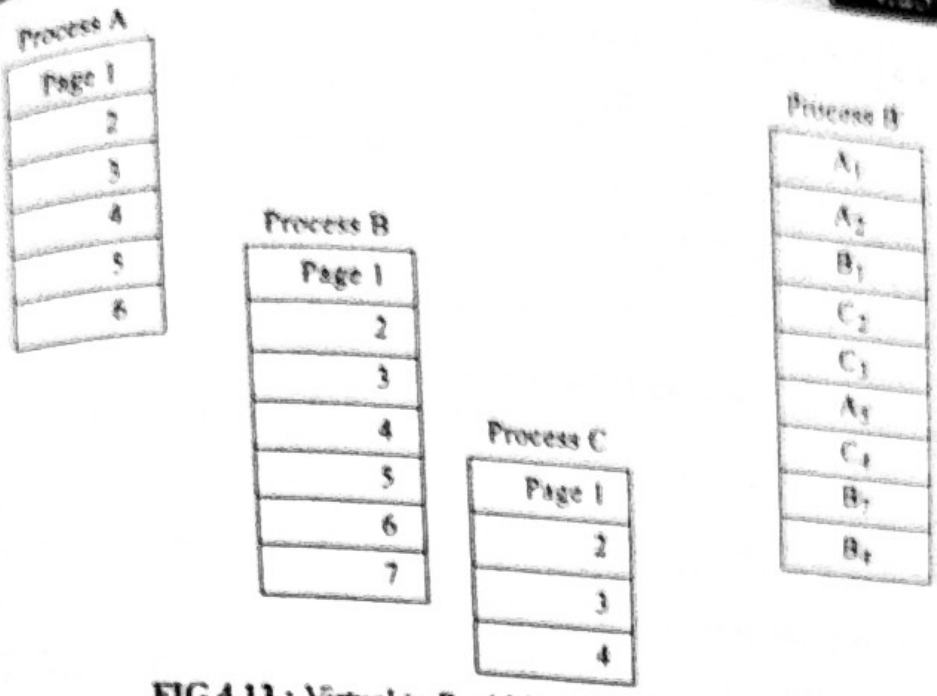


FIG 4.13 : Virtual to Real Memory Allocation

While executing code in A_1 makes a call transfer to the code in A_6. But A_6 is not loaded in memory. This is known as ***page fault.***

If a page fault occurs and no free memory space is available, the operating system has to remove a currently loaded page in order to accommodate the new page. This activity is knows as ***page replacement***. The operating system will have a routine handle this page replacement.

The routine will work like this :

1. Find the location of the desired page on the disk.
2. Find a free frame :
   (a) If there is a free frame, use it.
   (b) Otherwise use a page replacement algorithm to select a victim frame.
   (c) Write the victim page to the disk, change the page and frame tables accordingly.
3. Read the desired page into the free frame, change the page and frame tables.
4. Restart the user process.

4.17 PAGE REPLACEMENT ALGORITHMS

The algorithms that selects the page to be swapped out is called the ***page replacement algorithms.***

There are three basic algorithm are there :

- First in firstout (FIFO).
- Least recently use (LRU).
- Optimal.

4.17.1 First in Firstout (FIFO)

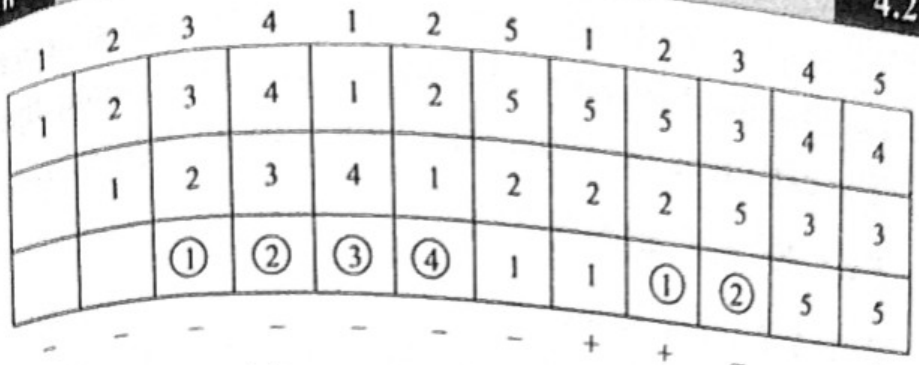
This algorithm is easy to implement, all we require is a simple queue. The queue contains page numbers in the order in which CPU refers the pages. The top of the queue contains first page that comes in and the bottom has the last page that comes in whenever there is a page fault and all the frames are full then operating system swaps out that page which entered into memory first and the new page is swapped into that frame that is first freed.

Although this algorithm is simple, its performance is not good. If a page that is going to be accesses in the near future is at the top of the queue, the FIFO algorithm marks the page for replacement. When the swapped out page is again required in the near future than again a page fault is generated. Thus the number of page faults may increase. Hence it is not considered to be an efficient algorithm.

EXAMPLE

Consider following page reference string and number of frames = 3

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



- Indicates page failures
+ Indicates page success
○ Indicates page selected for removal

FIG 4.14 :

Initially all the three frames are free. Now reference is for page 1. As the page 1 is not in memory it gives page fault and 1 is loaded into memory. Same way 2 and 3 are loaded. Now memory is full. Page reference is 4 which is not in memory and all memory frames are full. So one of the page is to br removed.

FIFO says the page first loaded into the system is to removed. So page 1 is selected for removal. If a referred page is in memory it indicates page success. For the above page sequence with frames 3, FIFO gives a page failures and 3 page success. If a referred page is not in memory it indicate **'page failure'.**

4.17.2 *Least Recently Used (LRU)*

The LRU policy selects the page whose time since last reference is greatest for removal. This algorithm keeps track of transactions in physical memory. Whenever a page fault occurs, operating system checks for that page which is not used since long time but is present in memory. Once such a page is found it is marked for replacement. Suppose there are 3 pages present in memory. One page say page 1 referred about 2 *m sec.* back and second page is page 2 is referred 10 *m* sec back another page 3 is referred 20 *m sec.* back. Then page 3 is marked for replacement whenever page fault occurs. The idea is that a page which is not in use since long time may not be used in

the near future. In other words a page which is just now referred may be used in the near future and hence retain that page. The LRU sees the past and decides about future.

Ex : No.of Frames = 3

Page reference string

1	2	3	4	1	2	5	1	2	3	4	5
1	2	3	4	1	2	5	1	2	3	4	5
	1	2	3	4	1	2	5	1	2	3	4
		①	②	③	④	1	2	⑤	1	②	3
–	–	–	–	–	–	–	+	+	–	–	–

FIG 4.15 :

No. of page failures = 10 and page success = 2.

In LRU algorithm if page success occurred, the referred page is removal to being frame which indicates that this page is just referred.

4.17.3 Optimal Algorithm

The optimal policy selects for replacement that page for which the time to the next reference is the longest. This algorithm is almost impossible to implement as it would require the operating system to have perfect knowledge of future events.

EXAMPLE :

Consider following page reference string.

2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2.

		2	3	2	1	5	2	4	5	3	2	5	2
Frame	0	2	3	3	①	5	5	5	5	5	5	5	5
	1		2	2	3	3	3	3	3	③	2	2	2
	2				2	2	②	4	4	4	4	4	4
		–	–	+	–	–	+	–	+	+	–	+	+

Page failures = 6
Page sucess = 6

FIG 4.16 :

When the memory is full, one of the page is to be removal. The page which is not going to be referred soon that will be removed. For example in the Fig. 4.16 memory is full with pages 1, 3, 2 so one of it is to removed.

From page reference string we can find out that page 2 is referred after page 5 and after 5, 2, 4, 5 pages page is referred. So out of the pages 1, 3, 2 page 1 is selected for removal as it is not going to be referred soon.

4.18 CONCEPT OF THRASHING

The virtual memory concept appears to give the opportunity for the system to sustain an indefinite number of processes simultaneously. However, if too many processes are running will cause frequent occurrence of page faults. The point can be reached when the processor is spending most of its time swapping pages and doing little productive work; this condition is known as ***thrashing***.

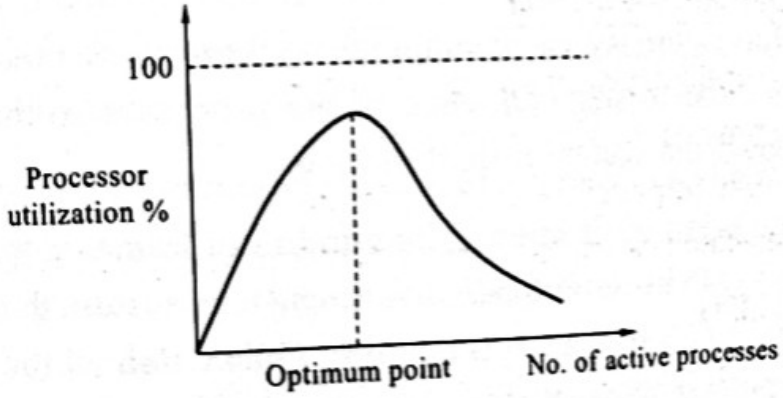


FIG 4.17 : Thrashing

This condition also suffers from a ***'positive feedback'*** characteristics as the level of swapping increases, the greater will be the page faults as more active pages are replaced. The overall effect can be visualized graphically in the following Fig. 4.17.

4.18.1 Cause of Thrashing

1. **High Degree of Multiprogramming :** The CPU scheduler sees the decreasing CPU utilization and increases the degree of multiprogramming as a result. The new process tries to get started by taking frames from running processes, causing more page faults and a longer queue for the paging device. As a result, CPU utilization drops even further, and the CPU scheduler tries to increase the degree of multiprogramming even more.

2. **Lack of Frames :** If a process has less number of frames then less pages of that process will be able to reside in memory and hence it would result in more frequent swaping. This may lead to thrashing. Hence sufficient amount of frames must be allocated .

   Recovery : Do not allow the system to go into thrashing by instructing the long term scheduler not to bring the processes into memory after the threshold.

   If the system is already in thrashing then instruct the mid term schedular to suspend some of the processes so that we can recover the system from thrashing.

   Allocation of Frames : The number of frames a process can use has to be determined. In a single user system the entire set of frames is available to the user. Only when all the available frames are exhausted, a page replacement algorithm will be used to bring in a page. The free frame list has the list of free frames from which allocation can be made.

   The problem is difficult in a multiprogramming environment with demand paging because there are more than one user processes in memory at the same time. The maximum number of frames that can be allocated is equal to the maximum number of frames available. The minimum number of frames that can be allocated is less than this maximum. The number

of frames allocated to a process determines the number of page faults. Lesser the number of allocated frames, more the number of page faults and slower is the execution.

The minimum number of frames that can be allocated to a process is dictated by the system architecture, more precisely by the instruction set architecture. Whenever a page fault occurs during instruction execution, the instruction has to be re - executed. Therefore the minimum number of frames allocated must be enough to hold all pages that any single instruction can reference.

4.19 WORKING SET MODEL AND PAGE FAULT FREQUENCY

Working set is the set of physical memory pages currently dedicated to a specific process.

The effect of working set size is important and should be neither too large nor too small. If the working set is too large, then fewer processes can be ready at any one time. If the working set is too small, then additional requests must be made of the swapping space to retrieve required pages.

The working set model states that a process can be in RAM if and only if all the pages that it is currently using (the most recently used pages) can be in RAM. The model is an all or nothing model, meaning if the pages it needs to use increases, and there is no room in RAM, the process is kicked from memory to free the memory for other processes to use.

The working set model uses a parameter, W, to define the working set window. It examines the most recent page references in the window size W. The set of pages in the most recent W page references is the working set. If a page is in active use, it will be in the working set. If it is no longer being used, it will be dropped from the working set W time units after its last reference. Thus the working set is an estimation of the program's locality of pages.

Consider the sequence of memory references shown in following Fig. 4.18, if W = 12 memory references, then the working set of time t_1 is (1, 2, 3, 4, 5). By time t_2 the working set has changed to (1, 3, 6 8).

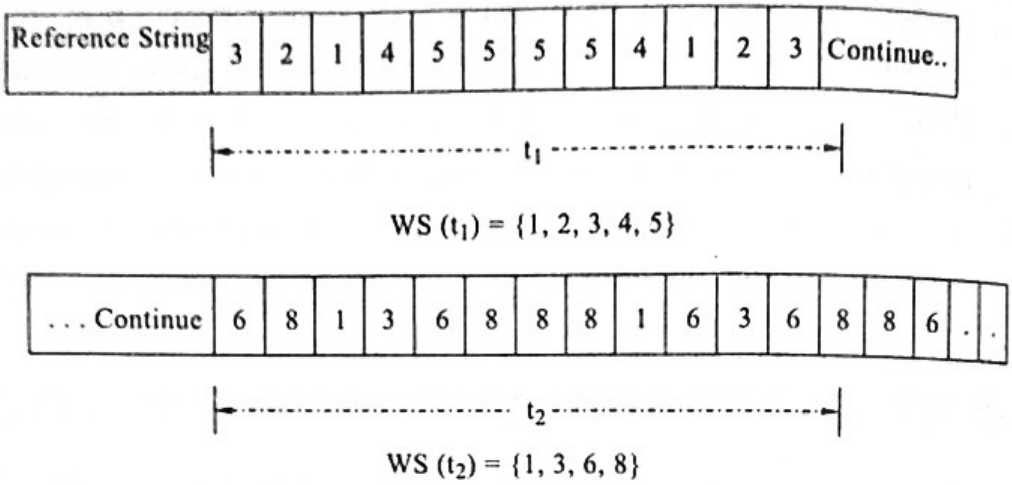


FIG 4.18 : Working Set Model

The correctness of the working set depends upon the selection of W.

This working set strategy prevents trashing while keeping the degree of multiprogramming as high as possible. Thus it tries to optimize CPU utilization.

Page Fault Frequency : The working set can be useful to control thrashing. As thrashing is basically the problem of having page fault rate at high frequency. Thus in this approach the basic aim is to control the page fault rate. When it is too high, it means that the process needs more frames. Similarly, if the page fault rate is too low, then the process may have too many frames. Thus upper and lower bounds on the page fault rate can be established as shown in Fig. 4.18. If the actual page fault exceeds the upper limit, process is allocated more frames, if the page fault rate falls below the lower limit, then frames can be freed from that process. Thus thrashing can be measured and controlled directly by controlling the page fault rate to prevent thrashing.

As with the working set strategy, the system has to suspend a process. If the page fault rate increases and no free frames are available, a process can be selected to be suspended. The freed frames are then distributed to processes with high page fault rates.

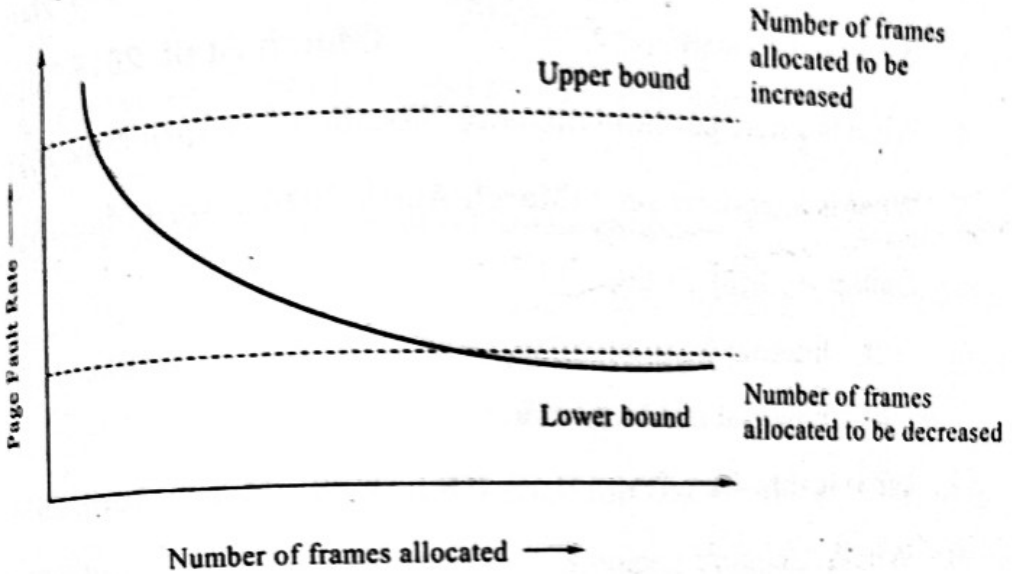


FIG 4.19 : Page Fault Rate Frequency

REVIEW QUESTIONS

Part-A

1. What is memory management ?
2. What is overlay? **(April. 2008, 2007)**
3. What is the swapping ? **(March/April. 2018, 2013)**
4. What is single partition memory allocation ? **(April/May. 2011)**
5. What is a working set ? **(March/April. 2014 ; April/May. 2011)**
6. Define
   (i) Internal fragmentation.
   (ii) External fragmentation.
7. What is thrashing ? **(Oct. 2020 ; March/April. 2018)**
8. What is demand paging ? **(March/April. 2014)**
9. List the advantages of demand paging.
10. Define a page and a frame. **(March/April. 2016)**
11. What is a page fault **(Oct. 2020 ; March/April. 2013)**
12. Write any three difference betwen paging and segmentation.
13. List the page replacement algorithms. **(Oct. 2020)**
14. List the different ways of implementing the page table. **(April/May. 2012)**

Part-B

1. Define and explain about : **(April. 2007)**
   (i) Address binding. **(Oct. 2020)**
   (ii) Dynamic loading.
   (iii) Dynamic linking.
2. Explain single partition technique ?
3. Give the principal of FIFO disk scheduling algorithm.

4. How many frames are needed for each page ? Why ?
(March/April. 2013)

5. What is the cause of thrashing ? Briefly explain.
(April/May. 2012)

Part-C

1. Explain single partition and multiple partition techniques.
(April. 2008)

2. Write about the single partition allocation.
(March/April. 2013)

3. Explain FIFO page replacement algorithm.
(Oct. 2020 ; March/April. 2016, 2014)

4. Write about FIFO and LRU page replacement algorithms, with appropriate examples.
(March/April. 2018)

5. Explain the concept of page replacement. (Oct. 2020)

4. Explain the working of demand paging. (Oct. 2020)

5. Explain optimal page replacement algorithm. (Oct. 2020)

6. Explain LRU page replacement algorithm. (Oct. 2020)

7. Explain about virtual memory technique. (Oct. 2020)

3. Discuss abou the multiple partition allocation.
(March/April. 2014)

4. Explain about (March/April. 2016; April. 2007)
   (i) Paged management.
   (ii) Segmentation.

5. Explain about
   (i) Demand paged technique.
   (ii) Page replacement. (April. 2007)

6. With help of examples explain page replacement algorithms. **(March/April. 2013)**

7. Explain about
   (i) Working set model
   (ii) Page fault frequency.

8. Explain about paging concept **(April. 2008)**

9. Describe the demand paging. **(April/May. 2011)**

10. Explain about fragmentation and its types. **(March. 2016)**

11. Explain the difference between internal and external fragmentation. **(April/May. 2012)**

12. Explain the difference between logical and physical address. **(April/May. 2012)**

13. Explain how logical address is translated into physical address. **(Oct. 2020)**

CHAPTER

5

DISK SCHEDULING AND FILE MANAGEMENT

CHAPTER OUTLINE

5.1 INTRODUCTION

Disk Structure : Disks provide the bulk of secondary storage for modern computer systems. Magnetic tape was used as early secondary storage medium, but access time is much slower than disks. Thus tapes are mainly used for backup, for storing infrequently used information, as a medium transferring information from one system to another, and for storing large quantities of data which are too large for disks systems.

Modern disk drives are addressed as large one - dimensional arrays of logical blocks where logical block is the smallest unit of transfer. The size of a logical block is usually 512 bytes or 1024 bytes.

The one-dimensional array of logical blocks is mapped onto the sectors of the disk sequentially. Sector 0 is the first sector of the first track on the outer most cylinder. The mapping then proceeds in order through that track, then through the rest of the cylinders from outermost to innermost. By using this mapping, it should be possible to convert logical block number into an disk address that consists of a cylinder number, a track number with in that cylinder, and a sector number with in that track.

The number of sectors per track has been increasing as disk technology improves, and it is common to have more than 100 sectors.

5.2 DISK PERFORMANCE PARAMETERS

The main measures of the qualities of a disk are capacity, Latency time, Seek time, data transfer rate, reliability and average transfer time

Capacity : Refers to the maximum amount of memory the disk can hold.

Latency Time : Time take by the arm to make rotations

Seek Time : The time from when a read or write request is issued to when data transfer actually begins. The time for repositioning the arm is called ***seek time***, and it increases with the distance the arm must move.

Data Transfer Rate : The rate at which data can be retrieved from or stored to the disk.

Reliability : Refers to the trustworthyness of data transfer.

Average Transfer Time : Average of all access's transfer time.

Latency time, Seek time and Transfer time of a disk structure will be ranging in **Nano seconds.**

Transfer rate will be measured in Mbps (mega bits per second) in disks.

Average Access time can be calculated as :

Average access time = Average seek time + Average rotational delay + Transfer time + Controller overhead + Queuing delay

Consider a situation like,

Calculate the average access time for transferring 512 bytes of data with the data rate 40 kbps. The average seek time is 5ms, disk rotation rate is 6000 rpm and the controller overhead is 0.1 ms.

Solution :

Given data :

Average seek time = 5 msec

Disk rotation = 6000 RPM

Data rate = 40 KB/sec

Controller overhead = 0.1 m-sec

Calculate time Taken For One Full Rotation :

= (60 / 6000) sec

= (1 / 100) sec

= 0.01 sec

= 10 msec

Average rotational delay :

$$= \frac{1}{2} \times \text{Time taken for one full rotatio}$$

$$n = \frac{1}{2} \times 10 \text{ m-sec}$$

= 5 m-sec

Transfer time :

= (512 bytes / 40 KB) sec

= 0.0125 sec

= 12.5 m-sec

Average Access Time :

= Average seek time + Average rotational delay + Transfer time + Controller overhead + Queuing delay

= 5 m-sec + 5 m-sec + 12.5 m-sec + 0.1 m-sec + 0

= 22.6 m-sec

Therefore 22.6 milli seconds is the average access time.

5.3 DISK SCHEDULING POLICIES

One of the responsibilities of the operating system is to use the hardware efficiently. For the disk drives this means having a fast access time and disk bandwidth. The access time has two major components. The seek time is the time for the disk arm to move the heads to the cylinder containing the desired sector. The rotational latency is the additional time waiting for the disk to rotate the desired sector to the disk

head. The disk band width is the total number of bytes transferred, divided by the total time between the first request for service and completion of the last transfer. The access time and bandwidth can be improved by scheduling the servicing disk I/O requests in a good manner.

The various disk scheduling algorithms are :

- First in-First Out (FIFO)
- Shortest Seek Time First (SSTF)
- SCAN scheduling
- C-SCAN scheduling

5.3.1 *First in First Out (FIFO)*

The simplest form of scheduling is first-in-first out (FIFO) scheduling, which processes items from the queue in sequential order. This strategy has the advantage of being fair, because every request is honored and the requests are honored in the order received.

Fig. 5.1 illustrates the disk arm movement with FIFO.

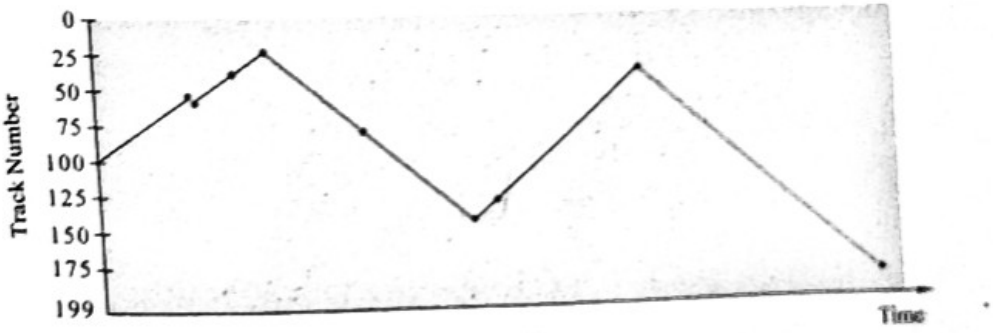


FIG 5.1 : FIFO

With FIFO, if there are only a few processes that require access and if many of the requests are to clustered file sectors, then we can hope for good performance. However, this technique will often approximate random scheduling in performance, if there are many processes competing for the disk.

5.3.2 Shortest Seek Time First (SSTF)

In this scheduling all the request close to the current head position are served before moving the head far away to service other requests. The SSTF algorithm selects the request with the minimum seek time from the current head position. Since seek time increases with the number of cylinders traversed by the head, SSTF chooses the pending request closest to the current head position.

Consider the previous example, the closest request to the initial head position (53) is at cylinder 65. Once we are at cylinder 65, the next closest request is at cylinder 67. From there, request at cylinder 37 is closer than 98 so 37 is served next. Continuing, we service the request at cylinder 14, then 98, 122, 124 and finally at 183. This scheduling method results in a total head moment of only 236 cylinders. This algorithm gives a substantial improvement in performance.

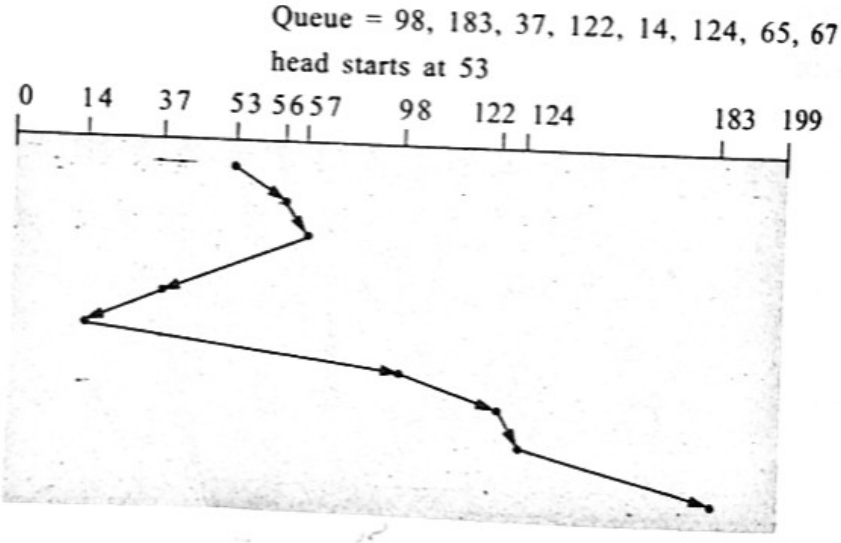


FIG 5.2 : SSTF Disk Scheduling

5.3.3 SCAN Scheduling

In the SCAN algorithm, the disk arm starts at one end of the disk, and moves toward the other end, servicing requests as it reaches each cylinder, until it gets to other end of the disk. At the other end, the direction of the head movement is

reversed and servicing continues. The head continuously scans back and forth across the disk.

Before applying SCAN to schedule the requests on cylinders 98, 183, 37, 122, 14, 124, 65 and 67 we need to know the direction of head movement, in addition to the head's current position (53). If the disk arm is moving toward 0 the head will service 37 and then 14. At cylinder 0, the arm will reverse and move toward the other end of the disk, servicing the request at 65, 67, 98, 122, 124 and 183. If a request arrives in the queue just in front of the head, it will be serviced almost immediately; a request arriving just behind the head will have to wait until the arm moves to the end of the disk, reverses direction, and comes back.

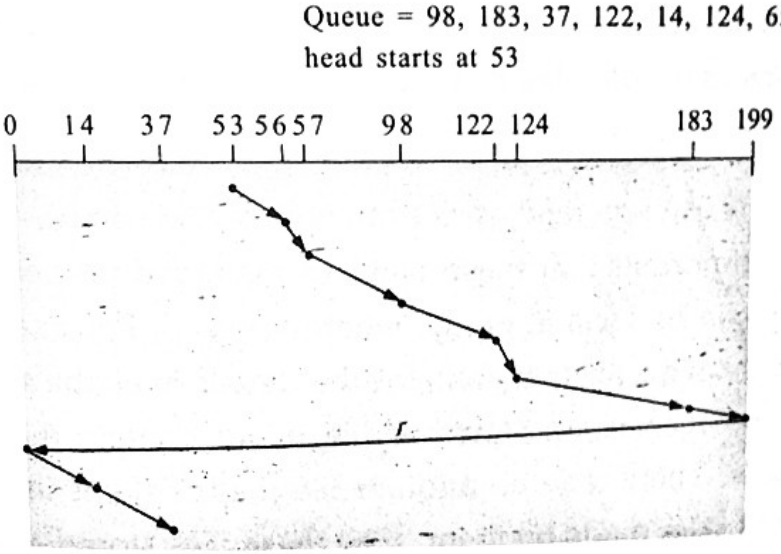


FIG 5.3 : Scan Disk Scheduling

5.3.4 C-Scan

Circular SCAN is variant of SCAN. Like SCAN, C-SCAN moves the head from one end of the disk to the other, servicing requests along the way. When the head reaches the other end, it immediately returns to the beginning of the disk, without servicing any requests on the return trip. The C-SCAN scheduling algorithm essentially treats the cylinders as a circular list that wraps from final cylinder to the first one.

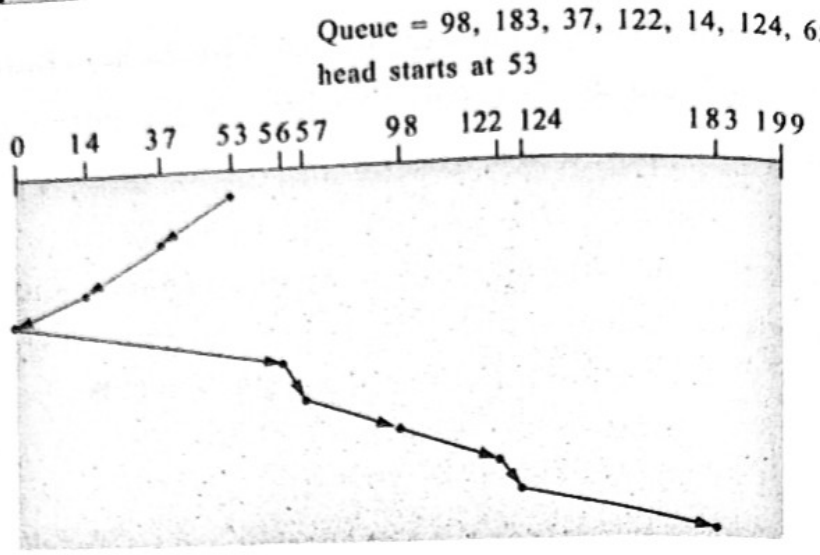


FIG 5.4 : C-SCAN Disk Scheduling

5.4 INTRODUCTION TO FILE SYSTEM AND PROTECTION

A file is a named collection of logically related information or we can say that files deal with storage and retrieval of data in a computer.

We will use several files to store information on our computer. It is important that these files are managed properly so that they can be located easily, when required. For example you might have a file that contains the details of all the employees of an organization. Another file might contain their official details. There may be another file that contains information about sales made by them. If all these files stored at different locations on hard disk, it will be difficult to find appropriate file and it is time consuming. So, it is important to store these files together at a location that is distinct from that of other files. This helps in locating the files easily. File management helps in doing so. The way in which an operating system organizes, names, protects, accesses, and uses files is called a file management system. File protection means the information stores in files should be protected from physical damage or unauthorized access.

5.5 FILE MANAGEMENT

A file management is that set of system software which provides services to users and applications related to the use of files. The only way that a user or application may access files is through the file management system. A file manager provides a view of the hierarchy in which the directories and subdirectories are organized on the computer.

File management system function include :

- Provide user friendly interface.
- Provide facilities to create, modify and delete files.
- Ability to share each others files without any disturbance or difficulty.
- Ability for the user to structure their files.
- Provides facilities for backup and recovery.

5.6 VARIOUS FILE OPERATIONS

A file is an abstract data type. Six basic operations are possible on files. They are :

1. **Creating a File :** Creation of file includes two steps. First space allocation for the file and second an entry to be made in the directory to record the name and location of the file.
2. **Writing a File :** Parameters required to write into a file are the name of the file and the contents to be written into it. Give the name of the file the operating system makes a search in the directory to find the location of the file. The write pointer enable to write the contents at a proper location in the file.
3. **Reading a File :** To read information stored in a file the name of the file specified as a parameter is searched by the operating system in the directory to locate the file. An read pointer helps Read information from a particular location in the file.

4. **Repositioning with in a File :** A file is searched in the directory and a based on new value current file position is moved to the desired location.

5. **Deleting a File :** The directory is searched for the particular file. If it is found, file space and other resources associated with that file are released and the corresponding directory entry is erased.

6. **Truncating a File :** The structure of the file remains the same, but the contents of the file are deleted. So the file size is reduced.

Other common operations are combinations of these basic operations. They include ***append, rename*** and ***copy.***

5.7 VARIOUS ACCESS METHODS

When you want to use the information stored in a file, it needs to be accessed and read into the computer memory. There are different ways to access the information in a file.

1. **Sequential Access Method :** The sequential access method is the simplest and most common access method. In this method, information is accessed in proper order, moving from one block of information to the next.

In this method, a read operation reads the next block in a file and automatically moves the file pointer ahead. In the same way, the write operation writes information to the end of the file and, then, moves the pointer to the new end of the file. This method also allows the rewinding of a file to move to the beginning of the file. (A file pointer indicates the current of information in the file when the file is open. The pointer always points to the next block of the file that is being used).

Sequential access method is based on the tape model. In the tape model, the information is written on the tape and the directory lists the name and location of each file on the tape.

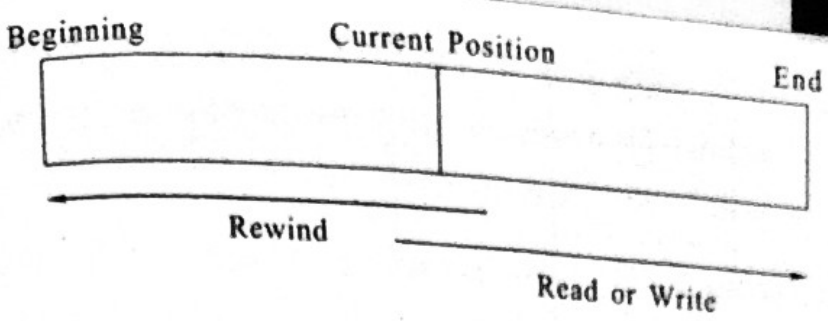


FIG 5.5 : Sequential Access File

2. **Direct Access Method :** The direct access method allows read and write operations in random blocks. This means that the blocks can be read or written quickly without following any specific order. This method is also called the ***relative access method.***

   In this method it is considered that the sequential blocks in a file are numbered and there is no order to be followed for reading or writing information. So, it is possible to read block 11 then read block 28 and then write block 4.

   This access method is especially useful when you need to quickly access information from about customer accounts in different blocks that are identified by their account numbers. The direct access method makes it easy to locate the account details of a particular customer by directly accessing that block without having to read all the other blocks.

   The direct access method is based on the disk model. A disk is divided into tracks, and each track is divided into blocks. The information to be accessed is stored in each block and can be accessed randomly. Sequential access of a direct access file is possible but direct access of a sequential file is not.

3. **Indexed Access Method :** This method is based on the direct access method. In this method, an index is created for a file. The index is just like the index of a book and contains pointers to the various blocks of information. To find any information in the file, the index is searched for a pointer to the information. This pointer is then used to directly access the block of information.

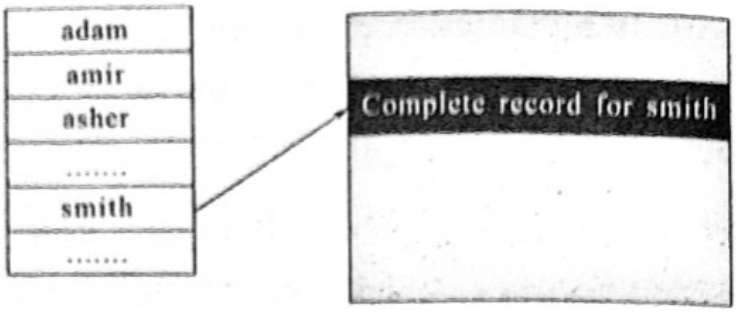


FIG 5.6 : Indexed Access Method

5.8 VARIOUS ALLOCATION METHODS

An allocation method refers to how disk blocks are allocated for files.

Allocation methods are for

- Effective disk space utilization.
- Allow fast file access.

Three major methods of allocation of disk space are :

- Contiguous.
- Linked.
- Indexed.

1. **Contiguous Allocation :** In this, files are assigned to contiguous areas of secondary storage. A user specifies in advance the size of the area needed to hold a file to be created. If the desired amount of contiguous space is not available, the file cannot be created.
   - Starting block and length of the file.
   - Two most common strategies are
     - First-fit
     - Best-fit

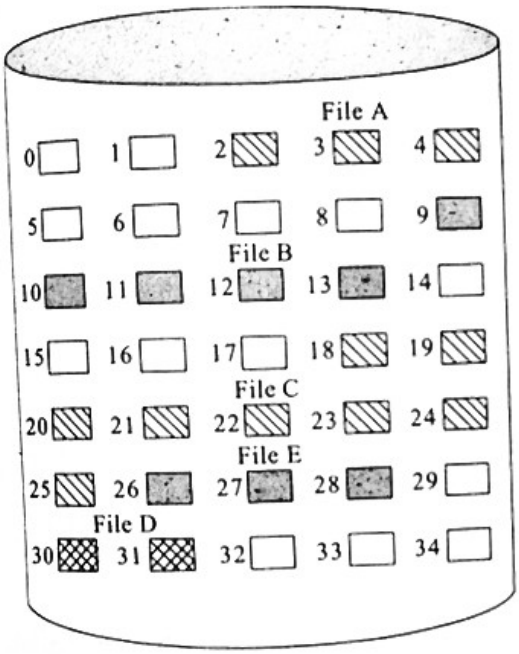
First-Fit : In this case as soon as the first hole (that is big enough) is encountered, searching is stopped and

memory is allocated for creating a file. Searching can start either at the beginning of the set of holes or where the previous first-fit search ended.

Best-Fit : In this case the entire list is searched for and the smallest hole, that is big enough, is allocated for creating a file.

Neither first-fit nor best-fit is clearly best in terms of storage utilization, but first-fit is generally faster.

External fragmentation will occur.



File Allocation Table

File Name	Start Block	Length
File A	2	3
File B	9	5
File C	18	8
File D	30	2
File E	26	3

FIG 5.7 : Contiguous Allocation

2. Linked Allocation :

- In linked list allocation each file is linked list of disk blocks.
- These disk blocks may be scattered through the disk.
- A few bytes of each disk block contains the address of the next block.
- A single entry in the file allocation table.
  - Starting block and length of file.

- No external fragmentation.
- Best for sequential files.

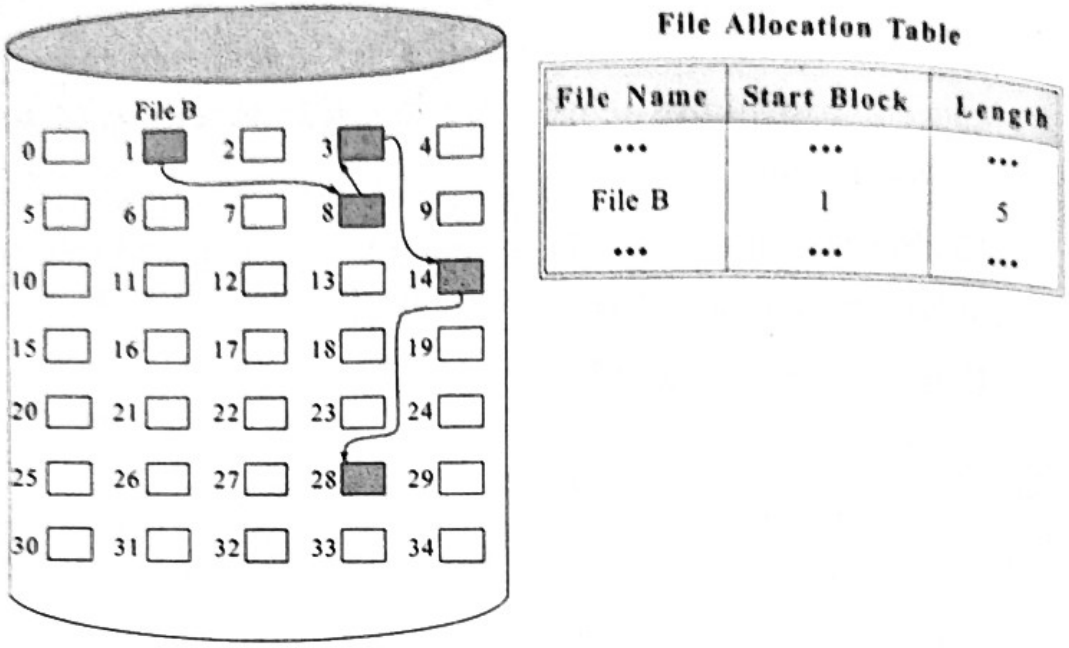


FIG 5.8 : Linked Allocation

3. Index Allocation :

- In this scheme each file is provided with its own index block, which is an array of disk block pointers (addresses).
- The K^{th} entry in the index block points to the k^{th} disk block of the file.
- The file allocation table contains block number for the index.
- Indexed allocation solves this problem by bringing all the pointers together into one location known as the index block.
- Each file has its own index block, which is an array of disk-block addresses. The 'i' entry in the index block points to be 'i' block of the file.

The advantages of indexed file allocation include the absence of external fragmentation and the efficiency of random accessing.

Indexed allocation requires lots of space for keeping pointers.

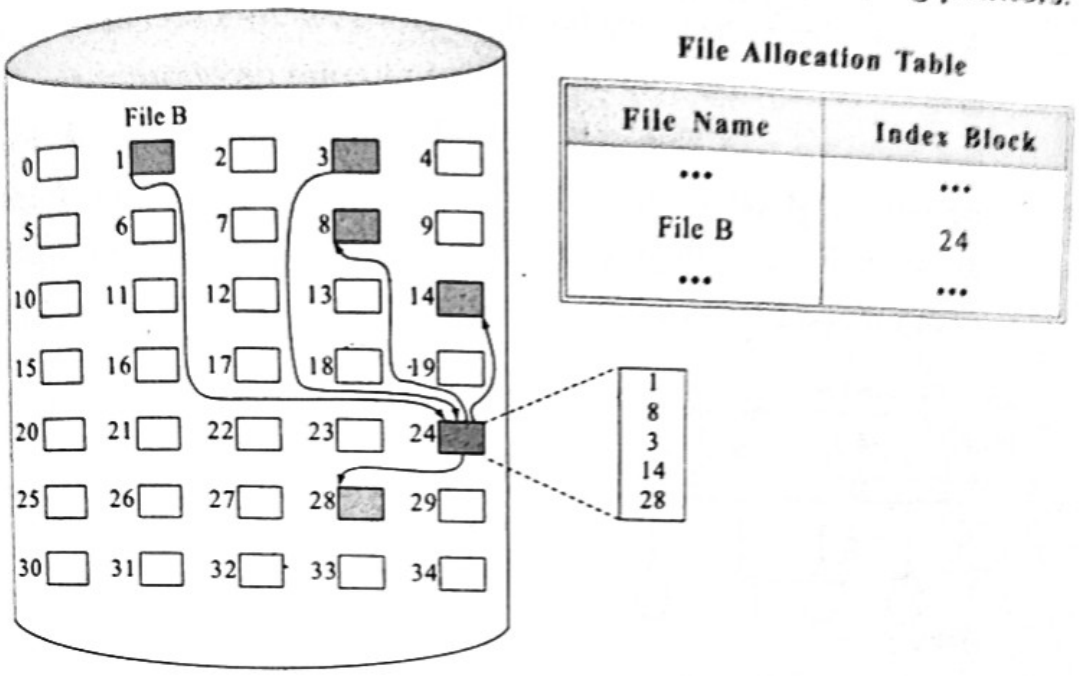


FIG 5.9 : Index Allocation

5.9 DIRECTORY STRUCTURE ORGANIZATION

Files systems are very large. Files have to be organized. Usually a two level organization is done :

- The file system is divided into partitions. Default there is at least one partition. Partitions are nothing but virtual disks with each partition considered as a separate storage device.
- Each partition has information about the files in it. This information is nothing but a table of contents. It is known as a directory.

The directory maintains information about the name, location, size and type of all files in the partition. A directory has a logical structure. Directories are files which are under control of the OS. An interactive user can only see some of the attributes by using commands ls (UNIX) or dir (MS - DOS).

The following are some common ways of organizing directories i.e., these are the various directory structures.

This is a simple directory structure that is very easy to support. All files reside in one and the same directory.

A single - level directory has limitations as the number of files and users increase. Since there is only one directory to list all the files, no two files can have the same name, that is, file names must be unique in order to identity one file from another. Even with one user, it is difficult to maintain files with unique names when the number of files becomes larger.

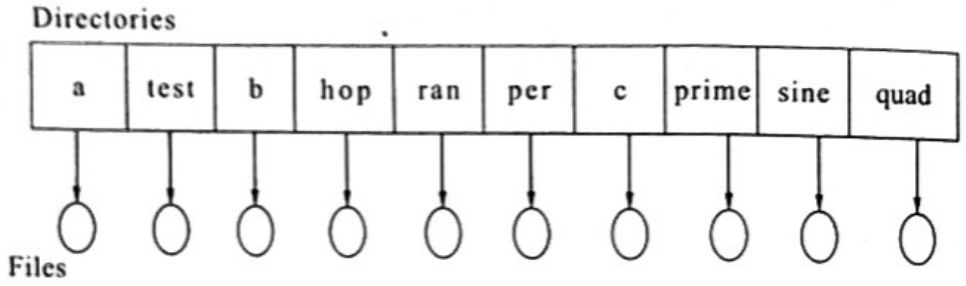


FIG 5.10 : Single - Level Directory Structure

1. **Two-Level Directory :** The main limitation of single-level directory is to have unique file names by different users. One solution to the problem could be to create separate directories for each user.

   A two - level directory structure has one directory exclusively for each user. The directory structure of each user is similar in structure and maintains file information about files present in that directory only. The operating system has one master directory for a partition. This directory has entries for each of the user directories.

   Files with same names exist across user directories but not in the same user directory. File maintenance is easy. Users are isolated from one another. But when users work in a group and each wants to access files in another users directory, it may not be possible.

Access to a file is through user name and file name. This is known as a path. Thus a path uniquely defines a file.

For example : In MS-DOS if 'C' is the partition the C"/user1/ Test, C:\User2\test, C:\user3-C are all files in user directories. Files could be created, deleted, searched and renamed in the user directories only.

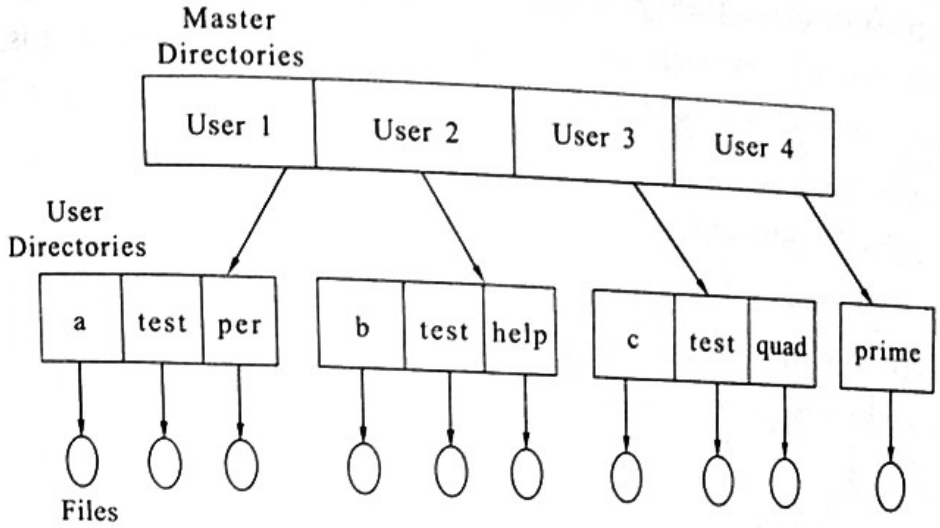


FIG 5.11 : Two - Level Directory Structure

2. **Three Structure Directories :** A two - level directory is a tree of height two with the master file directory at the root having user directories as descendants that in turn have the files themselves as descendants. Tree-structured directories area generalization of this two-level directory structure to have a directory structure of arbitrary height. This generalization allows users to organize files within user directories into sub directories. Every files has a unique path. Here the path is from the root through all the sub directories to the specific file.

   Usually the user has a current directory. User created sub directories could be traversed. Files are usually accessed by giving their path names. Path names could be either absolute or relative. Absolute path names begin with the root and give the complete path down to the file. Relative path names begin with the current directory. Allowing users to define sub

directories allows for organizing user files based on topics. A directory is treated as yet another file in the directory, higher up in the hierarchy. To delete a directory it must be empty. Two options exist : delete all files and then delete the directory or delete all entries in the directory when the directory is deleted. Deletion may be a recursive process since directory to be deleted may contain sub directories.

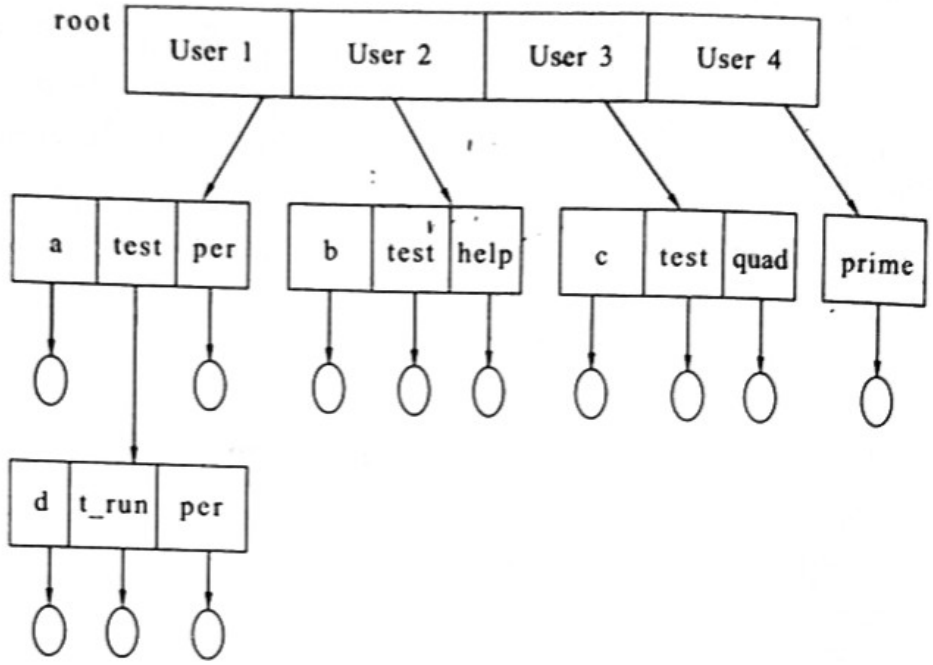


FIG 5.12 : Tree - Structure Directory Structure

5.10 FILE PROTECTION

When information is kept in a computer system, a major worry is its protection from both physical damage (reliability) and improper/unauthorized access (protection).

Reliability is generally provided by duplicate copies of files. Many computers have system programs that automatically copy files. File systems can be damaged by hardware problems (such as errors in reading or writing), power problems, head crash, dirt, temperature extremes etc. File may be deleted accidentally. Bugs in the file system software can also cause file contents to be lost.

Protection can be provided in many ways. For a single user system, we can provide protection by physically removing the floppy disks and locking them in safe custody. In multi user system, other mechanisms are needed.

Files need to be protected from unauthorized users. The problem of protecting files is more acute in multi - user systems. Some files may have only read access for some users, read / write access for some others, and so on. Also a directory of files may not be accessible to a group of users, for example, student users do not access to any other files except their own. Like files devices, database, processes also need protection. All such items are grouped together as objects. Thus objects are to be protected from subjects who need access to these objects.

The operating system allows different access rights for different objects. For example UNIX has read, write and execute (rwx) rights for owners, groups and others.

Possible Access Rights are Listed Below :

- No access.
- Read only.
- Update.
- Delete.
- Execute only.
- Append only.
- Modify protection rights.

A hierarchy of access rights is identified. For example, if update right is granted then it is implied that all rights above update in the hierarchy are granted. This scheme is simple but creation of a hierarchy of access rights is not easy. It is easy for a process to inherit access rights from the user who has created it. The system then need maintain a matrix of access rights for different files for different users.

Encryption is an important tool in protection, security and authentication.

The process involves two steps :

1. **Encryption :** The original message is changed to some other form.
2. **Decryption :** The encrypted message is restored back to the original.

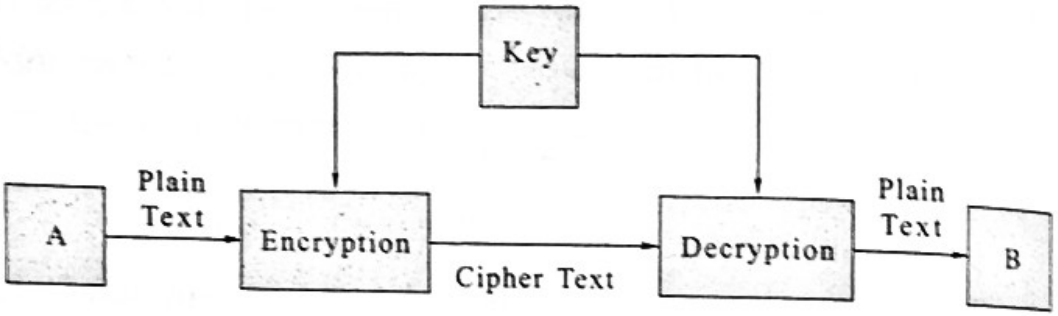


FIG 5.13 : Encyrption

Data before encryption is called plain text and after encryption is called cipher text. Usually the above operations are performed by hardware.

There is another approach to the protection problem, which is to associate a password with each file. Just as access to the computer system itself is often controlled by a password, access to each file can be controlled by a password.

REVIEW QUESTIONS

Part-A

1. What are four various disk scheduling algorithms ?
2. Define disk structure. **(March/April. 2013 ; April. 2007)**
3. Define file management. **(March/April. 2018)**
4. List various file operations. **(April. 2008)**
5. List file management functions.
6. What do you meant by file protection. **(Oct. 2020)**

Part-B

1. How is information on the disk referenced. **(April/May. 2011)**
2. List various methods of disk space allocation. **(March. 2018)**
3. List various file access method. **(March/April. 206)**
4. Define a file. List any four file types. **(April. 2007)**
5. List the types of system directories and write briefly how the files are organized in the directories. **(April/May. 2011)**
6. List operations to be performed on directories. **(Oct. 2020)**
7. How to protect files on a single-user system ? **(April/May. 2012)**
8. Write a short note on FIFO disck scheduling algorithm. **(Oct. 2020)**

Part-C

1. Explain space allocation methods.
2. Explain disk scheduling algorithms. List the various types of disk scheduling algorithms. **(March/April. 2016)**
3. Write about the SCAN disk scheduling. **(March/April. 2018, 2014, 2012)**
4. Write about the C-SCAN disk scheduling. **(March/April. 2018, 2014 ; April/May. 2012)**

5. List and explain varous disk performance parameters. (Oct. 2020)
6. What are the file operation ? Explain them. (March/April. 2014)
7. Explain various file access method in detail. (Oct. 2020)
8. Explain the differences among the file access methods. (March/April. 2013 ; April/May. 2011)
9. Explain directory structure organization.
10. Discuss about single level directory and two level directory. (April. 2007)
11. Why rotational latency is not considered in disk scheduling? (Oct. 2020)
12. Explain the following disk scheduling methods. (Oct. 2020)
    (a) SCAN (b) SSTF

BOARD DIPLOMA EXAMINATION

FEB/MARCH-2022

OPERATING SYSTEMS

D.CM.E II Year III Semester

Time : 3 Hours Total Marks : 80

PART – A 10 x 3 = 30

Note : Answer all question. Each Question carries 3 Marks.

1. Define the terms spooling and buffering.
2. List any three applications which does not need OS.
3. Draw the process state diagram.
4. State the need of process scheduler.
5. List two operations that can be performed on a semaphore.
6. Define the term deadlock.
7. Write any two differences between logical and physical address space.
8. Define the term swapping.
9. Define the terms capacity, latency time, seek time and reliability with respect to disks.
10. List any three file access methods.

PART – B 5×8 = 40

Note : Answer all questions. Each question carries 8 Marks. Answer should be comprehensive and the criterion for valuation is the content but not the length of the answer.

11. (a) Compare distributed system with real time system with respect to online gaming.

(or)

11. (b) Identify the challenges of designing operating system for mobile devices compared with designing operating systems for traditional PCs.

12. (a) Apply SJF scheduling algorithm to calculate the average waiting time for processes given below

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

(or)

12. (b) Compare preemptive and non-preemptive scheduling algorithms with respect to round robin scheduling algorithm.

13. (a) A counting semaphore was initialized to 12, then the following sequences of operations were performed on this semaphore : 10P (wait), 4V (signal), 2P (wait) and 5V (signal). Calculate the final value of counting semaphore.

(or)

13. (b) Apply the bankers algorithm to check the sytem is in safe state or not for the following snapshot of a system.

Process	Allocation A B C D	Max A B C D	Available A B C D
P0	0012	0012	1520
P1	1000	1750	
P2	1354	2356	
P3	0632	0652	
P4	0014	0656	

14. (a) Initially usage of memory is allocated as specified in figure below. After receiving additional requrest : 10K; 25 K; and 15 K, at what starting address will each of the additional requests be allocated by applying first-fit allocation method?

Used	Hole	Used	Hole	Used	Hole	Used	Hole	Used	Hole
10K	5K	20K	20K	10K	30K	30K	20K	10K	15K

(or)

14. (b) Consider the following page reference string :

2, 1, 3, 0, 4, 1, 2, 3, 0, 1, 2, 4, 3, 0, 3, 2, 1, 4, 3, 1

How many page faults would occur when make use of the LRU replacement algorithm, assume frame size is four?

15. (a) Calculate the average access time for transferring 512 bytes of data with the data rate 40 kB per second. The average seek time is 5 msec, the disk rotation is 6000 RPH, and the controller overhead is 0.1 msec.

(or)

15. (b) A disk queue with requests for I/O to blocks on cylinders 88, 18, 44, 50, 45, 24, 65, 96. The head is initially at cylinder number 53. The cylinders are numbered from 0 to 100. Starting from the current head position, calculate the total distance (in number of cylinders) that the disk arm moves to satisfy all the pending requests by applying SCAN disk scheduling policy.

PART – C 1×10 = 10

16. A shared variable x, initialized to zero, is operated on by four concurrent processes W, X, Y, Z as follows. Each of the processes W and X reads x from memory, increments by one, stores it to memory and then terminates. Each of the processess Y and Z reads x from memory, decrements by two, stores it to memory, and then terminates. Each process before reading x invokes the P operation (i.e., wait) on a counting semaphore S and invokes the V operation (i.e., signal) on teh semaphore S after storing x to memory. Semaphore all the processess complete execution?