UNIT-I

Overview- Introduction - Operating system objectives, user view, System view, Operating system defination, computer system Organization, computer system Architecture, OS structure; OS operation, process management, Memory Management, storage Management, protection and Security, Computing Environments.
Operating system services, user and OS. Interface, System calls Types of System calls, System programs, Operating system Design and Implementation, OS structure.

Introduction :

An operating system acts as an intermediary between The user of a computer and computer hardware. The purpose of an operating system is to provide an environment in which a user can execute programs in a convenient and efficient manner.

An operating system is software that manages the computer hardware. The hardware must provide appropriate Mechanisms to ensure the correct operation of the computer system and to prevent user programs from interfering with the proper operation of the system.

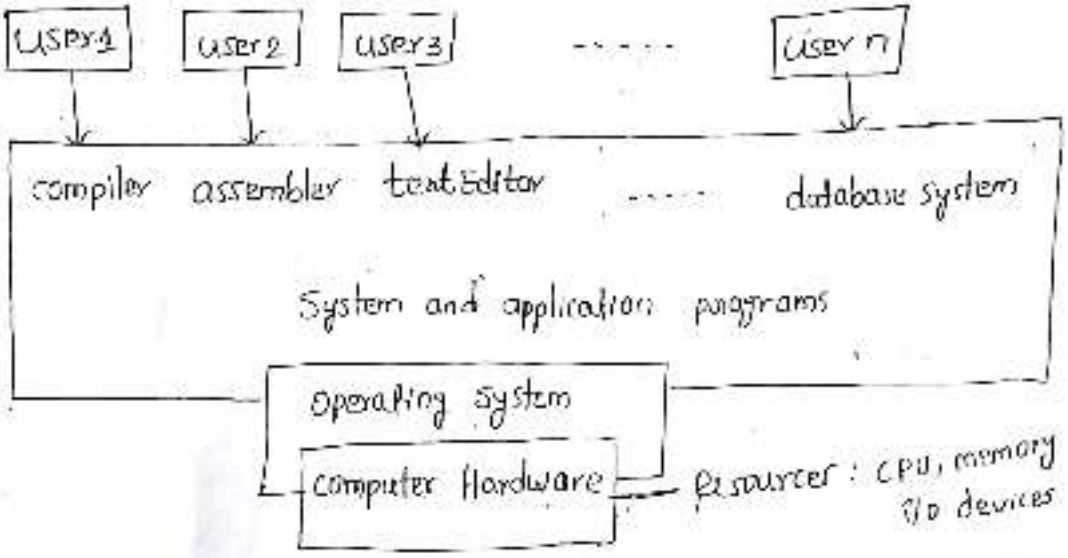
Operating System Definition:

An operating system is a program that controls the Execution of application programs and acts as an interface between the user of a computer and the computer hardware.

→ An operating system is concerned with the allocation of Resources and services, such as memory, processors, device and information.

→ The operating system correspondingly includes programs to Manage these resources, such as a traffic controller, a Scheduler, memory management module, I/o programs, and a file system.

→ Some popular operating systems include Linux, windows, MAC os, vms



Abstract view of the components of a computer system

→ The computer system can be divided into 4 components those are hardware, operating system, application programs & users

→ hardware: cpu, memory, & I/o devices provides the basic computing resources for the system.

→ Application programs: such as word processors, spreadsheets, compilers & web browsers, by using these resources solve the users computing problems.

Operating System Objectives:

The main objectives of operating system is

→ Convenience: An operating system makes a computer more convenient to use.

→ Efficiency: An OS allows the computer system resources to be used in an efficient manner.

→ Ability to Evolve: An OS should be constructed in such a way as to permit the effective development, testing and introduction of new system functions without at the same time interfering with service.

Operating system functions:

- process Management
- Memory Management
- I/o device Management
- file management
- Network management
- security & protection

→ Operating system: It controls the hardware and co-ordinates its users among the various application programs for the various users.

→ An operating system is a construct that allows the user Application programs to interact with the system hardware. OS by itself does not provide any function but it provides an atmosphere in which different applications and programs can do useful work.

User view:

The user view depends on the system interface that is used by the users.

→ if the user is using a personal computer the OS is largely designed to make the interaction easy. Some attention is also paid to the performence of the system, but there is no need for the OS to worry about resource utilization. this is because the pc uses all the resources available and there is no sharing.

→ if the user is using a system connected to a mainframe or a minicomputer, the OS is largely concerned with resource utilization. This is because there may be mutiple terminals connected to the mainframe and the OS makes sure that all the ~~OS~~ mak resources such as CPU, memory I/O devices etc. are divided uniformly between them.

System view:

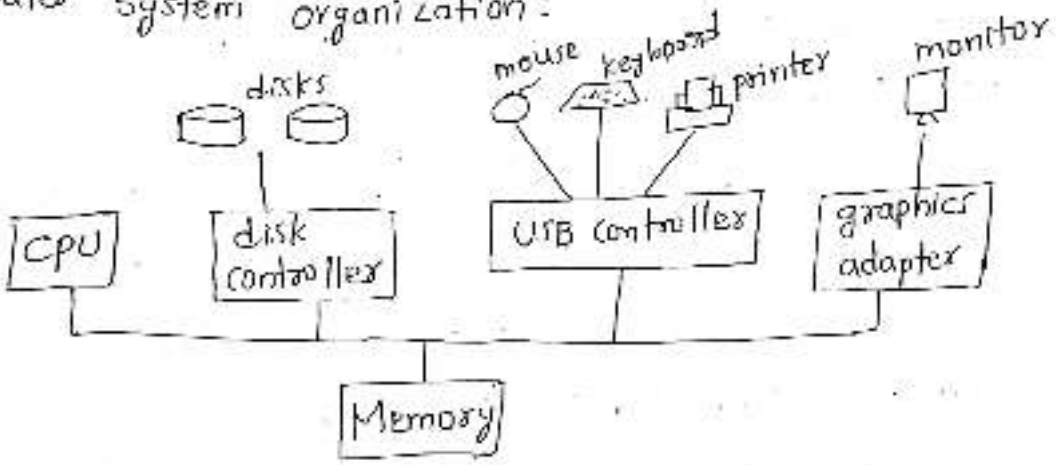
According to the computer system, the OS is the bridge between applications and hardware.

→ The system views the operating system as a resource allocator. There are many resources such as CPU time, memory space, file storage space, I/O device etc. that are required by the processer for execution. it is the duty of the OS to allocate these resources to the processes so that the computer system can run smoothly as possible

→ OS can also be viewed as a way to make using hardware easier.

→ computers were required to easily solve user problems. However it is not easy to work directly with the computer hardware. So, OS were developed to easily communicate with the hardware.

Computer system organization:



A modern computer system

How the system components are organized inorder to work the system properly.

- Computer system operation
- storage structure
- I/o structure

Computer system Operation:

A modern general-purpose computer system structure consists of one or more CPUs and a number of device controllers connected through a common bus that provides access to shared memory

→ CPU.. central processing unit, which is is called as brain of computer.
→ The cpu and the device controllers can execute concurrently competing for memory cycles.
→ To ensure orderly access to the shared memory, a memory controller is provided whose function is to synchronize access to the memory.

Bootstrap program:

→ bootstrap program is loaded at power-up or reboot.
→ it stored in ROM or EEPROM (electrically erasable programmable read-only memory, known by the general term 'firmware'.
→ It must know how to load the OS and start executing that system.
→ Loads OS kernel & starts execution.

Interrupt:

→ The occurence of an event is usually signalled by an interrupt from either the hardware or the software.
→ H/w may trigger an interrupts at any time by sending a signal to the CPU, usually by way of the system bus.
→ System call (Monitor call): S/w may trigger an interrupt by executing a special operation called system call.

When the CPU is interrupted, it stops what it is doing and immediately transfer execution to a fixed location (the fixed location usually contains the starting address where the service routine of the interrupt is located).

Storage Structure:

→ We cannot store the data permanently in main memory (RAM) because RAM is volatile, it is to small to store all needed programs & data permanently.

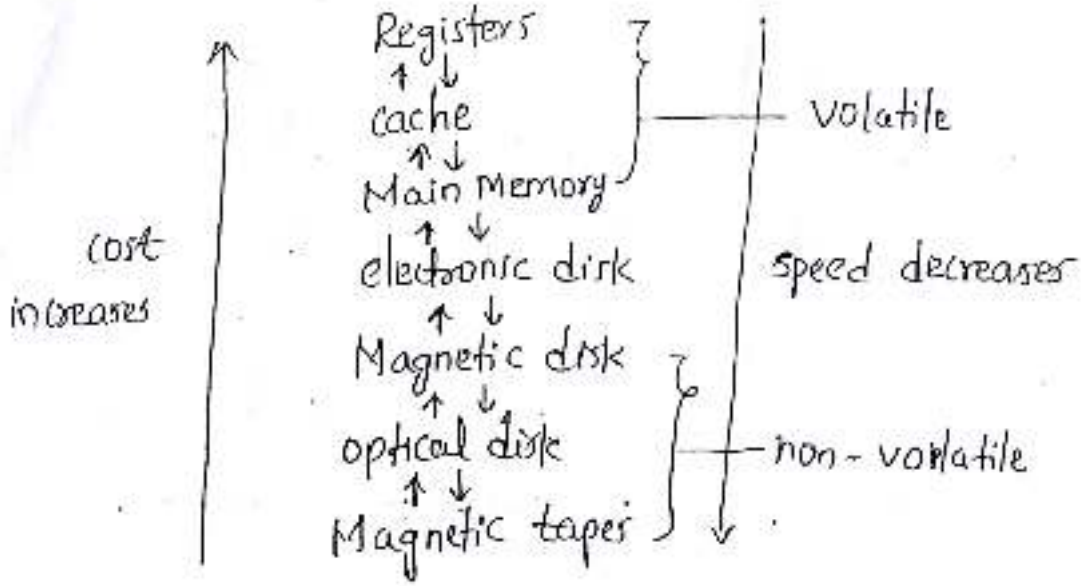
→ For this reason we are using secondary storage devices which are able to hold large quantities of data permanently.

→ Secondary storage - extension of main memory that provides large non-volatile storage capacity.

A storage structure contains registers, mainmemory Magnetic disks. storage system in a computer system can be organized in a hierarchy based on speed & cost.

Magnetic disks: it is rigid metal or glass platters covered with Magnetic recording material. disk surface is logically divided into 'tracks', which are subdivided into 'sectors'

The disk controller determine the logical interaction between the device and the computer.



Storage device hierarchy

→ Storage system in a computer system can be organized in a hierarchy based on speed & cost, volatility.

→ Semiconductor memory have become faster & cheaper.

→ Electronic disk: It is designed to be either volatile or non-volatile. it contains a hidden magnetic harddisk and a battery power. If the external power is interrupted, the electronic disk controller copies the data from RAM to magnetic disk when the power is restored, it copies data back to RAM.

→ caching: copying information into faster storage system; main memory can be viewed as a cache for secondary storage.

→ Information in use copied from slower to faster storage temporarily.

I/O structure:

→ A Large portion of operating system code is dedicated to Managing I/O.

→ OS has a device driver for each device controller, it is responsible for moving the data between peripheral devices that it controls and its local buffer storage.

→ Device controller maintance local buffer storage, special purpo registers.

Working of an I/O operation:

To start an i/o operation, the device driver loads the appropriate registers within the device controller.

→ The controller starts the transfer of data from the device to its local buffer.

→ one data transfer complete, the device controller informs the device driver via an interrupt that it has finished its operation.

→ storage capacity: The volume of data that a computer can store.

→ processing power: The no of instructions that a computer is able to process per second.

Based on storage capacity, processing power and cost computers are classified into 4 types.

- micro computers: pc, single user
- mini " : multiuser at a time
- mainframe " : high processing speed, huge storage capacity, it
- super " : support 100s of users.

↓ interconnect 100 of micro computers (eg: whether forecastin

• Increased Reliability - graceful degradation or fault-tolerance

Increased throughput:

→ By increasing the number of processors, we expect to get more work done in less time.

→ When multiple processors co-operate on a task, a certain amount of overhead is incurred in keeping all the parts working correctly. This overhead, plus using the shared resources, low the expected gain from additional processors.

Economy of scale:

→ Multiprocessor systems can cost less than equivalent multiple single-processor systems because they can share peripherals, mass storage, and power supplies.

→ if several programs operate on the same set of data, it is cheaper to store those data on one disk and to have all the processors share them than to have many computers with local disks and many copies of the data.

Increased Reliability:

→ if functions can be distributed properly among several proces then the failure of one processor will not halt the system, only slow it down.

→ if we have 10 processors and one fails, then each of the remaining nine processors can pick up a share of the work o the failed processor. Thus, the entire system runs only 10 percent slower, rather than failing altogether.

In the above figure, we show a dual-core design with two cores on the same chip. In this design, each core has its own register set as well as its own local cache. other designs might use a shared cache or a combination of local and shared caches.

→ The cpu design may have multiprocessor cores per chip or multiple chips with single cores

→ Multiprocessor cores per chip is more efficient because one-chip communication is faster than multiple chips communication by this we get less power than multiple single core chips.

→ It is well suited for server systems such as database & web servers.

Clustered systems:

→ The clustered computers share storage and are closely linked via LAN networking.

→ Like multiprocessor systems, but multiple systems working together.

→ It uses multiple CPU's to complete a task.

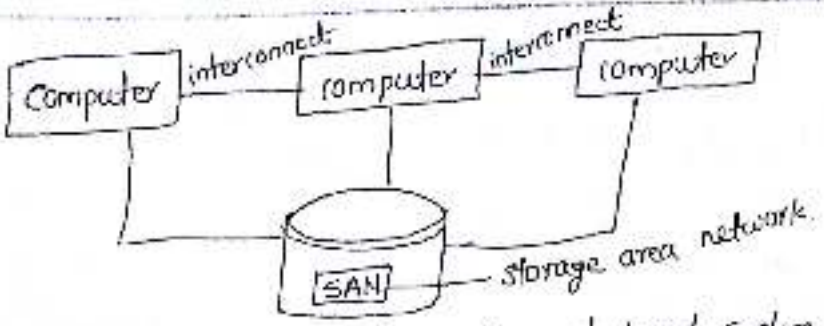
→ It provides a high availability; that is, a service will continue even if one or more systems in the cluster fail. Each node can monitor one or more nodes over the LAN.

Clustered systems can be of the following forms.

- Asymmetric clustering
- Symmetric clustering.

Asymmetric clustering: one machine is in standby mode and other machines are running the applications. the standby mode performs nothing it only monitors the server. it becomes the active server if server is fails

Symmetric clustering: Two or more machines run the applications. They also monitor each other at the same time. this mode is more efficient because it uses all available machines. it can be used only if multiple applications are available to be executed.



General structure of a clustered system

Operating System Structure:

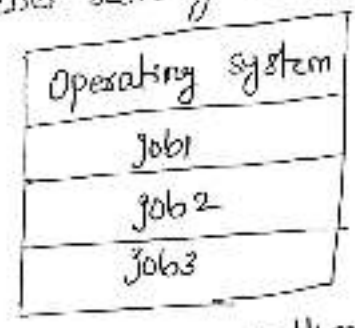
→ An operating system provides the environment within which programs are executed

→ An operating system has ability to run multiprograms at a time. A single user frequently have multiple programs running.

→ Multiprogramming increases CPU utilization by organizing jobs so that the CPU always has one to execute.

→ The operating system keeps several jobs in memory simultaneously but main memory is too small to accommodate all jobs.

→ So, the jobs are kept initially on the disk in the job pool. this pool consists of all processes residing on disk awaiting allocation of MM.



Memory layout for a multiprogramming system.

→ An operating system provides the environment for
- Multiprogramming
- Multiprocessing
- Multitasking / Timesharing

Operating system follows the different structures, but we mainly discussed the following structures.

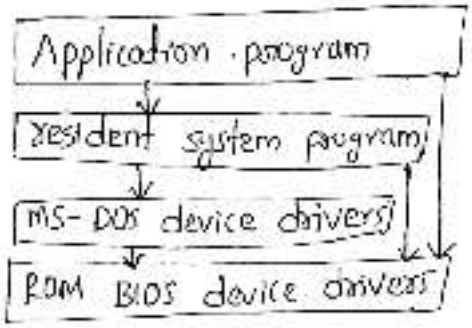
- Simple structure
- Layered Approach
- Microkernels

Simple structure:

→ Operating systems such as MS-DOS and the original UNIX did not have well-defined structures.

→ There was no CPU Execution mode (user & kernel), and so errors in Applications could cause the whole system to crash

M



MS-DOS layer structure

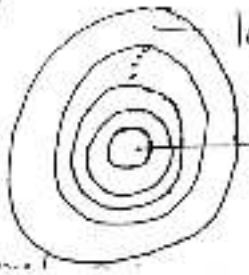
Layered Approach:

This approach breaks up the operating system into different layers.

→ This allows implementers to change the inner workings, and increases modularity.

→ As long as the external interface of the routines don't change, developers have more freedom to change the inner workings of the routines.

→ With the layered approach, the bottom layer (layer-0) is the hardware and highest layer is the user interface (layer-N).



→ The main advantage is simplicity of construction and debugging.

→ The main difficulty is defining the various layers

Microkernels:

This structures the OS by removing all non-essential portions of the kernel and implementing them as system and user level programs

→ Generally they provide minimal process and memory management, and a communications facility.

→ Communication between components of the OS is provided by message passing.

Advantages:

→ Extending the operating system becomes much easier.

→ It provide more security and reliability.

The main disadvantage is poor performence due to increased System overhead from message passing.

Operating system operations:

Modern operating systems are interrupt driven. if there are no processes to execute, then OS will sit idle and waits for an event to occur.

→ Events are signalled by interrupts or trap

→ A trap (or an exception) is a software generated interrupt caused either by an error or by a specific request from a user program. (which needs services of the operating system)

Generally two types of operations can be done by O.S

- Dual mode operation
- Timer operation.

Dual mode operation:

The O.S is designed in such a way that it is capable of distinguish between the execution of OS code and user defined code

→ since the OS and the users share the hardware and sw resources we need to make sure that one program failure doesn't effect the other programs.

→ To avoid those failures OS play dual mode.

dual modes are - user mode
- kernel mode/supervisor/privileged/system mode

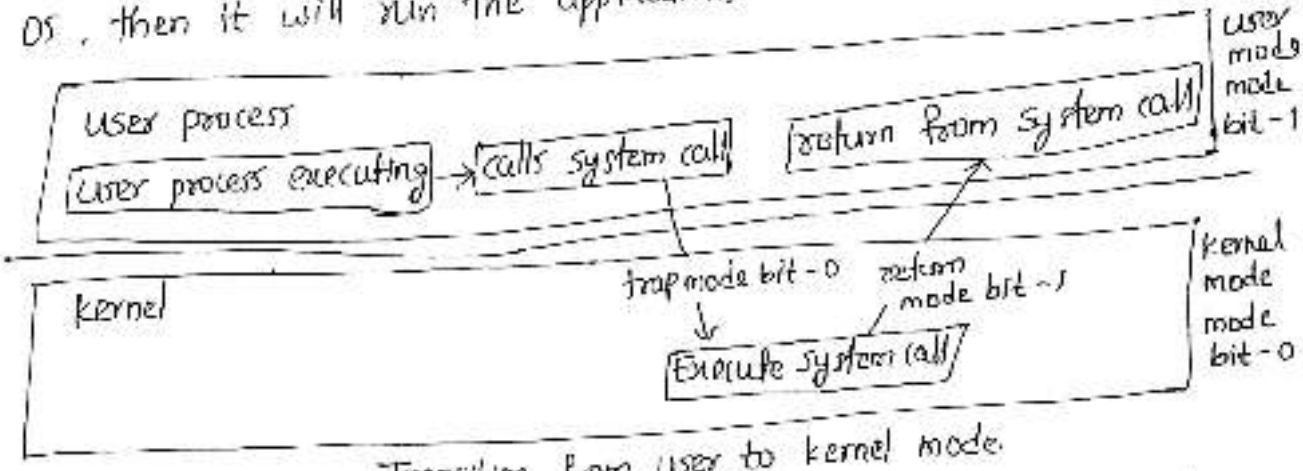
→ A mode bit of computer hardware is used to indicate the current mode.

mode bit - 0 kernel mode
- 1 user mode.

→ Dual mode of operation protects the OS from errant users.

→ We can transfer the user mode to kernel mode if any services needed by user.

→ In system boot time, hardware starts in kernel mode & load the OS, then it will run the applications.



Transition from user to kernel mode.

Timer:

→ Since OS operates in dual mode it should maintain control over CPU. Sometimes a user program gets stuck in an infinite loop or fail to call system services and never return control to the OS. To avoid this OS uses timer.

OS timers are — fixed timer
- variable timer

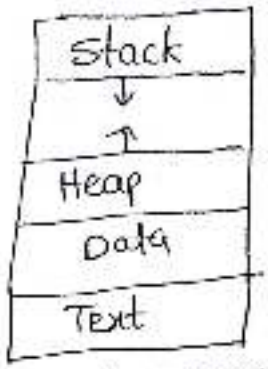
→ This timer control mechanism will interrupt the system at a specific period.

Text Section: A process, sometimes known as the text section, also include the current activity represented by the value of the program counter.

Stack: The stack contains the temporary data, such as parameters, return addresses & local variables.

Data Section: contains the global variable.

Heap Section: Dynamically allocated memory to process during its run time.



process in memory

Memory Management:

Every process needs some memory to store its variables and code. But if there is more than one process in memory at any one time, then the operating system must manage memory to prevent processes from reading / damaging each other's memory, and to ensure that each process has enough memory.

→ main memory refers to a physical memory that is the internal memory to the computer.

→ What ever the files we want to access and execute that must be copied from a storage device into main memory.

The operating system is responsible for the following activities in the connection with memory management.

→ keeping track of which parts of memory are currently being used and by whom.

→ Deciding which processes & data to move into and out of memory.

→ Allocating & deallocating memory space as needed.

Storage Management:

To make the computer system convenient to users, the O.S provides a uniform, logical view of information storage.

1 File system management:

It is one of the basic and important feature of the O.S. O.S is used to manage files of computer system.

A file is a collection of specific information stored in the Memory of computer system. all files with different extensions are managed by OS. a computer can store files on the disk which provide long term storage. a file system is normally organized into directories to make ease of their use.

The following tasks are performed by file system management of OS

→ Creating and deleting files
→ Creating & deleting directories to organize files
→ Supporting primitives for manipulating files and directories.
→ mapping files onto secondary storage
→ Backing up files on stable storage media.

2. Mass - Storage management:

Main memory is too small to accomodate the data and the contents are lost when the power is off. so the computer system must provide secondary storage to back up the main memory.

Most of the programs are stored on a disk until loaded into memory and then use the disk as both the source & destination of their processing. hence proper management of disk storage is of central importance to computer system.

The entire speed of computer operation depends on speed of disk subsystem.

The tasks performed by the os in storage management are as follows

→ free space management
→ storage allocation
→ Disk scheduling.

3. Caching:

It is a temporary storage area which is very nearest to the cpu. All the recent instructions are stored into the cache memory. The cache memory lies in the path between the processor and the memory. it has lesser access time than memory and faster than main memory. The need for the cache memory is due to the mismatch between speeds of main memory & cpu

4. I/o systems:

OS must provide programer to manage the hardware resource of computer like disk, memory, cpu.

It is a part of the os that provides an environment for the better interaction blw system & I/o devices. The os hides the user internal details of the hardware.

Protection & security:

Modern computer systems allows multiple users to execute their multiple processes concurrently in the system. These multiple processes may access data simultaneously, but only valid users can access the data. it is a job of os to apply protection and security.

Protection is a mechanism of controlling access of computer resources by users. the mechanism should provide tools so that the admin can define restrictions on users. security refers to the process of preventing the system from attacks. their are several types of attacks like internal, external, viruses and authentication information.

File-system manipulation:

os provides an interface to the user to create and delete files by specific name along with extension. Search for a given file and list file information. and it gives permissions to users to perform the operation on it

Communication:

process needs to exchange information with other process. processes executing on same computer system or on different computer system can communicate using operating system support by using shared memory or via message passing.

Error detection:

Errors may occur within CPU, memory hardware, I/O devices and in the user program. for each type of error, the os takes appropriate action for ensuring correct and consistent computing.

System with multiple users can gain efficiency by sharing the computer resources among the users.

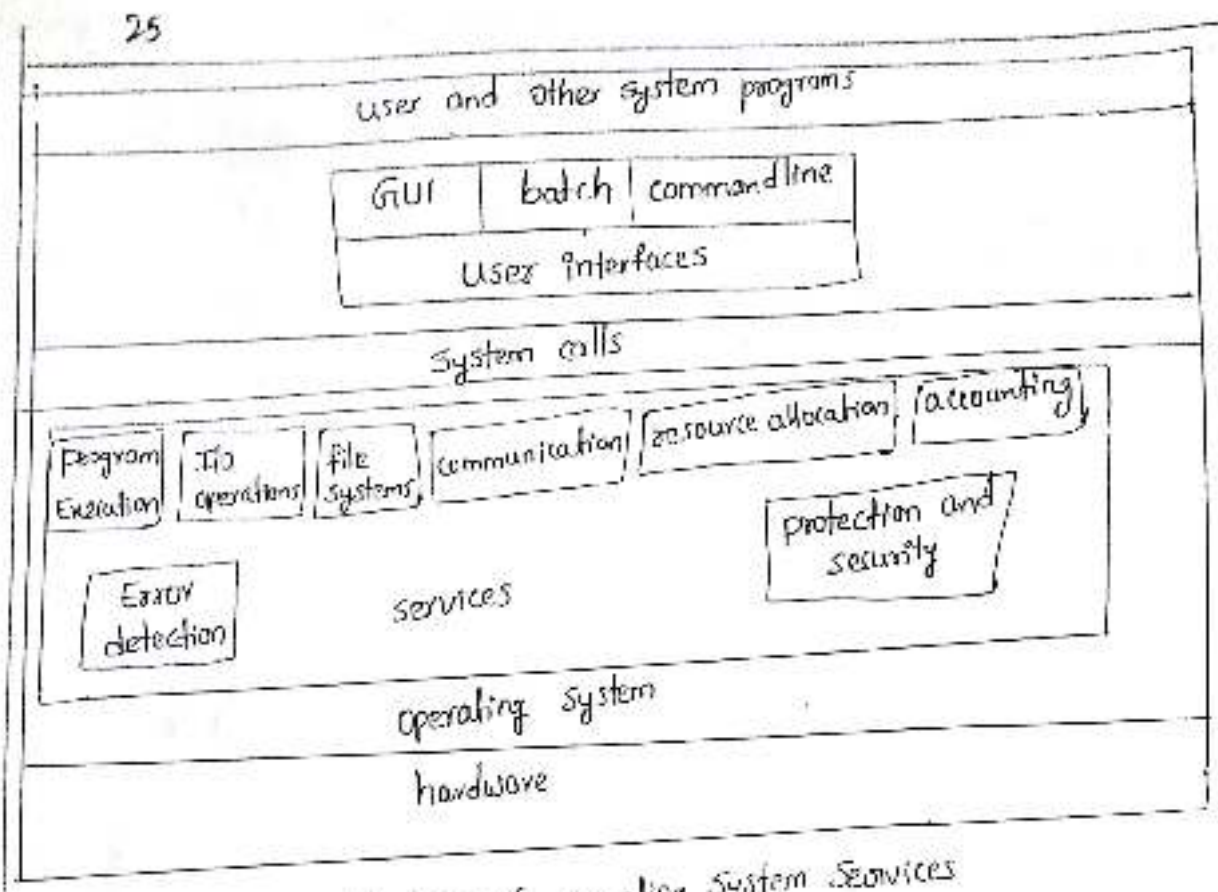
Resource allocation:

In the multitasking environment when multiple jobs are running at a time, it is the responsibility of os to allocate the required resources (CPU, mainmemory, file storage) to each process for its better utilization. The os manages all kinds of resources using schedulers.

Accounting:

This service of the os keeps track of which users are using how much and what kinds of computer resources have been used for accounting or simply to accumulate usage statistics.

protection & security: protection and security includes in ensuring all access to system resources in a controlled manner. for making a system secure The user needs to authenticate him or her to the system before using (usually via login id & pwd).



A view of operating system services

User and operating System interface:

There are two fundamental approaches for users to interface with the operating system.

1. command Line interface (CLI) / command Interpretter
2. Graphical user interfaces

Commandline interface: it allows users to directly enter commands that are to be performed by the OS.

→ Some operating systems include the command interpreter in the kernel others, such as windows xp and UNIX, treat the command interpreter as a special program.

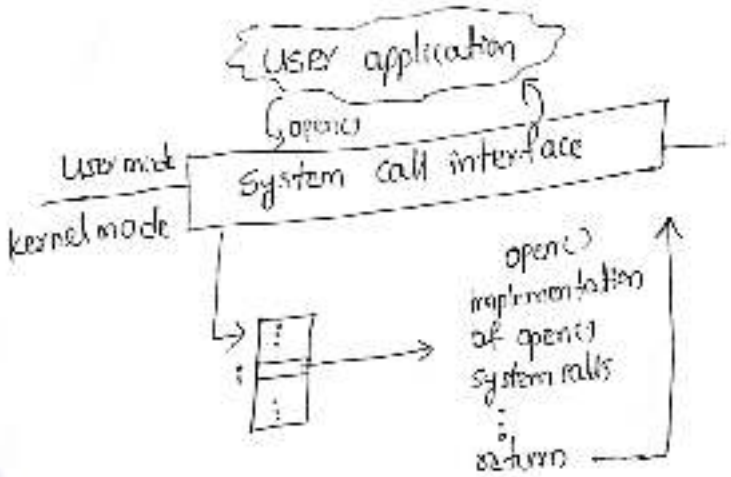
→ On systems with multiple command interpreters to choose from, the interpreters are known as shells.

eg: Bourne shell
C shell
Bourne Again shell (BASH)
korn shell etc.

System calls:

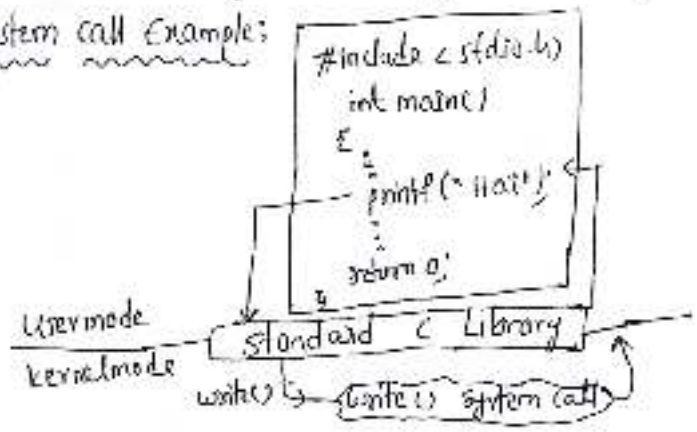
System calls provide an interface to the services and mode available by an os. system call is the programmatic way in which a computer program requests a service from the kernel of the os.

- It provides an interface to the os service when the program in user mode require access to the RAM or a hardware resources.
- it must request kernel to provide access to that resource, this is done using system call.
- Generally their are two modes user mode and kernel mode. If the program is executed in an user mode, their if no direct access to the memory or hardware. so it switched to the user mode to kernel mode at particular time this is called "context switching".
- it switched kernel mode to user mode also. If the programs are executed kernel mode, their is direct access to resources but their is a chance to crash the system this is called "privileged mode."



The handling of a user application invoking the open() system call.

System call Example:



Types of System calls:

- Process Control
- File Management
- Device Management
- Information Maintainance
- Communication
- protection

Process Control:

To control the processes of a running program may halt (stop) its execution either normally or abnormally.

→ end, abort
→ load, execute
→ Create process, terminate process
→ get process attributes, set process attributes
→ wait for time
→ wait event, signal event
→ allocate and free memory

File Management:

→ Create file, delete file
→ open, close
→ read, write, reposition
→ get file attributes, set file attributes

Device Management:

→ request device, release device
→ read, write, reposition
→ get device attributes, set device attributes
→ logically attach or detach devices

Information maintenance:

→ get time or date, set time or date
→ get system data, set system data
→ get process, file or device attributes
→ set process, file or device attributes

Communications:

→ create, delete communication connection
→ send, receive messages
→ transfer status information
→ attach or detach remote devices.

System programs / system utilities:

System programs provide a convenient environment for the program execution and development. these programs are design to run computer hardware and application programs.

Types of system programs:

File Management
Status information
file modification
programming - language support
program loading and execution
communications.

File Management: These programs create, delete, copy, rename, print dump, list and manipulate files and directories

Status information: it ask the systems for date, time, available memo no. of users etc. Some systems support a registry which is used for storing and retrieving the configuration information.

File Modification:

Text editors are used for creating and modifying the contents of the files stored on a disc (or) storage devices.

programming- Language support:

compilers, assemblers, debuggers and interpreter for common programming language (C, C++, Java, PERL) provided to the user with the os.

Programming loading and execution:

once a program is compiled (or) assembled the programs must be loaded into memory for execution, for this system provides absolute loaders, relocatable loaders, linkage loaders.

communication:

These programs provide the mechanism for creating connection among processes. they allow users to send messages (or) to transfer file from one machine to another.

Operating System Design and implementation:

The design of the os depends on goals and specifications The goals and specifications will give the clear idea about the system. The problems that are to be considered while designing of a os are

- choice of hardware
- Type of os

Requirements: The requirements of os are divided into two categories

1. User requirements:
   - easy to use and learn
   - reliable
   - Safe and fast

System requirements:

- Easy to design and implement, maintain & operate
- implement it should be flexible
- Reliable
- Error free and efficient

Policy and Mechanism:

→ Mechanism determines 'how to do something'

→ policy determines 'what to do' or it decide what will be done.

Eg: Timer construct is a mechanism for ensuring cpu protection
How long the timer is to be set is a policy / decision of time allottment is considered as a policy.

→ one important principle is that "separation of policy from mechanism" Change in policy will not effect to the mechanism is good for flexible os.

Implementation:

→ once an os is designed it must be implemented by using assembly level language (or) high-level language.

→ Traditionally os have been written in assembly-level language.

→ Now, high-level languages are used for implementing the os. because the high-level languages are easier to coding and debug easy to understand, easy to port i.e, we can easily move the os to any hardware platform.

Eg: MS-DOS - implemented by using assembly-level language
Windows, LINUX, Unix are developed in high-level language

Computing Environments:

A computer system uses many devices, arranged in different ways to solve many problems. this constitutes a computing Environment where many computers are used to process and exchange information to solve various types of computing problems.

Traditional computing:

→ Stand alone general purpose machines
→ But blurred at most systems interconnect with others.
→ portals provide web access to internal systems
→ Network computers are are like web terminals.

Client server computing:

It is a computing model in which client & server computers communicate with each other over a network. in this a server takes requests from client computers and shares its resources, applications or data with one/more client computers on the network

Peer to peer network:

All the devices are connected together but there is no central computer. workload/task is divided among the peers. peers are equally privileged.

Types of operating systems:

- Batch operating system
- Multiprogramming operating system
- Multiprocessor operating system
- Realtime operating system
- Distributed operating system

Batch operating system: there is no direct interaction between user and the computer the user has to submit a job to a computer operator. Then Computer operator places a batch of several jobs on an input device.

Multiprogramming OS: the os picks up and begins to execute one of the jobs from memory. if several jobs are ready to run at the same time then the system chooses which one to run through the process of CPU scheduling.

Multiprocessor: it contain several processors that share a common physical Memory. It provides higher computing power and speed.

Distributed os:

Distributed system use multiple central processors to serve Multiple real-time applications and multiple users. Data processing jobs are distributed among the processors accordingly.

Real time operating system:

It give maximum time for each of the critical operations that it performs, like os calls and interrupt handling. the real time os which guarantees the maximum time for critical operations and complete them on time.

Eg of os: Mrcrosoft windows
Mac os
Linux - ubuntu, mint, fedora [opensource]
for mobiles Apple ios, Android

6

UNIT- V

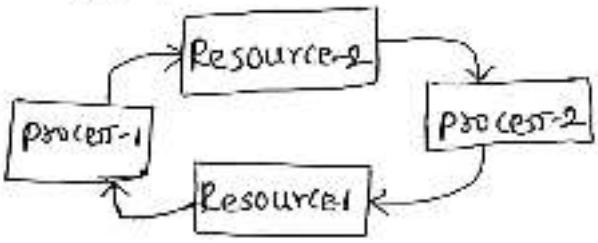
Deadlocks - System model, Deadlock characterization, methods for Handling deadlocks, deadlock prevention, deadlock avoidance Deadlock Dection, and Recovery from deadlock.

Protection - system protection, goals of protection, principles of protection, Domain of protection, Access matrix, implementation of Access matrix, Access control, Revocation of access Rights capability - based systems, Language - based protection.

Deadlock:

Deadlock is a situation where the execution of two or more processes is blocked because each process holds some resource and waits for another resource held by some other process.

Eg:-



⇒ here process P_1 holds resource R_1 and waits for resource R_2 which is held by process P_2.

⇒ process P_2 holds resource R_2 and waits for resource R_1 which is held by process P_1

⇒ None of the two processes can complete and release their resource.

⇒ Thus, both the processes keep waiting infinitely, this situation we can call it as deadlock.

In multiprogramming system, process get completed for a finite number of resource. any process requests resources,

and that resources are not available at that time, the process goes into a waiting state

System model:

→ A system model or structure consists of a fixed number of resources to be circulated among some processes The resources are then partitioned into numerous types, each partition consisting of some specific quantity of identical instances.

→ Memory space, cpu cycles, directories and files, I/O devices like keyboards, printers and CD-DVD drives are examples of resource types.

Under the standard mode of operation, any process may use a resource in only the below mentioned sequence.

Request: When the request can't be approved immediately, then the requesting job must remain waited until it can obtain the resource

use: The process can run on the resource like printer etc.

Release: The process release the resource

Deadlock characterization

There are following 4 necessary conditions for the occurrence of deadlock

- Mutual Exclusion
- Hold and wait
- No pre-emption
- Circular wait

Mutual Exclusion: There must exist at least one resource in the system which can be used by only one process at a time. if there exists no such resource, then deadlock will never occur.

printer is an example of a resource that can be used by only one process at a time.

Hold and wait: There must exist a process which holds some resource and waits for another resource held by some other process.

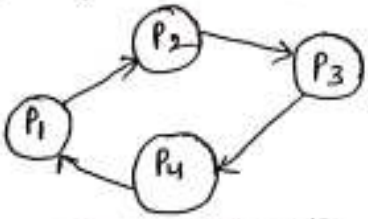
No pre-emption: By this condition, once the resource has been allocated to the process, it can not be preempted.

→ it means resource can not be snatched forefully from one process and given to the other process.

→ The process must release the resource voluntarily by itself.

Circular wait:

In this conditon all the processes must wait for the resource in a cyclic manner where the last process waits for the resource held by the first process.



→ process P_1 waits for a resource held by process P_2.
→ process P_2 waits for a resource held by process P_3
→ process P_3 waits for a resource held by process P_4
→ process P_4 waits for a resource held by process P_1.

All these four conditions must hold simultaneously for the occurrence of deadlock. If any of these conditions fail, then the system can be ensured deadlock free.

Resource Allocation graph (RAG):

→ Deadlocks can be described in terms of a directed graph called a system resource allocation graph.

→ This graph consists of a set of vertices V and a set of edges E

→ The set of vertices V is partitioned into two different types of nodes $P = \{P_1, P_2 \cdots P_n\}$, $R = \{R_1, R_2, \cdots R_m\}$

→ A directed edge from process P_i to resource type R_j is denoted by $P_i \rightarrow R_j$ & Resource to process $R_j \rightarrow P_i$.

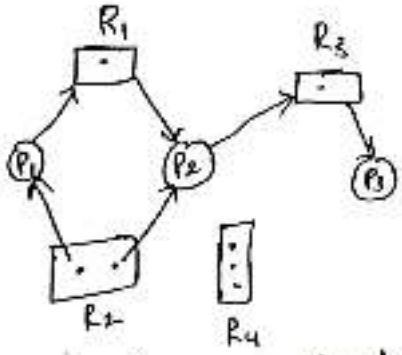
→ A directed edge $P_i \rightarrow R_j$ is called a request edge, a directed edge $R_j \rightarrow P_i$ is called an assignment edge.

The sets P, R and E

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

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

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

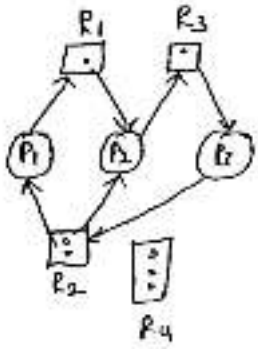


Resource allocation graph

→ In above graph, there is no cycles then no process in the system is deadlocked. if the graph does contain a cycle, then a deadlock may exist.

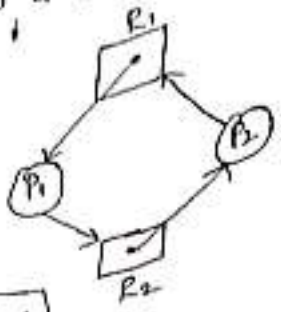
→ process P_1, P_2 and P_3 are deadlocked. Process P_2 is waiting for resource R_3, which is held by process P_3. & P_3 is waiting for P_1 or P_2 to release R_2. P_1 is waiting for P_2 to release R_1 and we also have a cycle.

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



RAG with deadlock

Eg: consider the RAG and find if the system is in a deadlock state. Otherwise find a safe sequence.



→ The given graph is single instance with a cycle. thus, the system is definitely in a deadlock state.

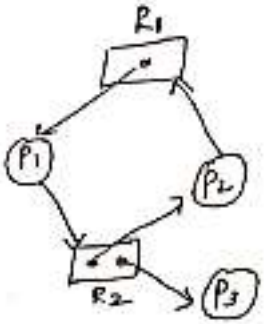
→ There are no instances available currently and both the processes require a resource to execute.

→ none of the process can be executed and both keeps waiting infinitely so system in a deadlock state.

process	Allocation		Need	
	R_1	R_2	R_1	R_2
P_1	1	0	0	1
P_2	0	1	1	0

available - $[R_1 R_2] = [0\ 0]$

Eg2:



→ The given graph is multi instance with a cycle contained in it. so, the system may or may not be in deadlock state

→ process P_3 does not need any resource so it executes, after execution, process P_3 release its resources.

Then available = $[0\ 0] + [0\ 1]$
= $[0\ 1]$

process	Allocation		Need	
	R_1	R_2	R_1	R_2
P_1	1	0	0	1
P_2	0	01	1	0
P_3	0	1	0	0

available $[R_1\ R_2] = [0\ 0]$

→ P_1 is allocated the resource, then P_1 completes its execution & release the resource

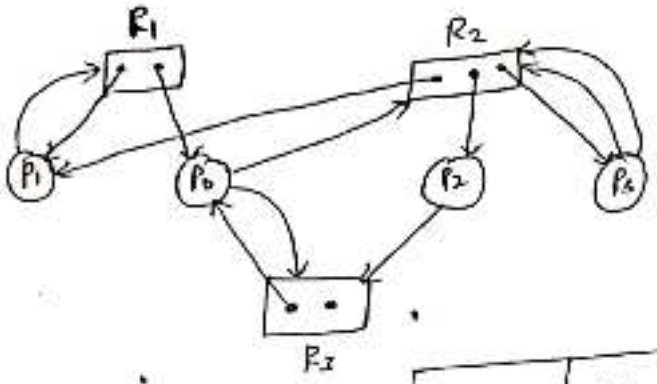
available = $[0\ 1] + [1\ 0]$
= $[1\ 1]$

→ Next P_2 is allocated the requested resource & completes its execution and release the resource.

available = $[1\ 1] + [0\ 1]$
= $[1\ 2]$

safe state is $P_3 \rightarrow P_1 \rightarrow P_2$

Eg:



Process	Allocation			Need		
	R_1	R_2	R_3	R_1	R_2	R_3
P_0	1	0	1	0	1	1
P_1	1	1	0	1	0	0
P_2	0	1	0	0	0	1
P_3	0	1	0	0	2	0

⇒ The given RAG is multi instance with a cycle. so the system may or may not be in a deadlock state.

⇒ only process P_2 completes its execution, and release the resource.

available = [0 0 1] + [0 1 0]

= [0 1 1]

available $[R_1\ R_2\ R_3]$ = [0 0 1]

⇒ with current available resources P_0 can satisfied, so P_0 complete the execution and Release the resource.

available = [0 1 1] + [1 0 1]

= [1 1 2]

⇒ with current available resource P_1 can satisfied, so P_1 complete the execution and Release the resource.

available = [1 1 2] + [1 1 0]

= [2 2 2]

⇒ next P_3 satisfied the available resource

available = [2 2 2] + [0 1 0]

= [2 3 2]

so the system is in safe state, & safe state is

$P_2 \Rightarrow P_0 \rightarrow P_1 \rightarrow P_3$

Methods for Handling Deadlocks:

Normally you can deal with the deadlock issues and situations in one of the three ways mentioned below.

→ you can use a protocal to prevend or avoid deadlocks, ensuring that the system will never enter a deadlock state.

→ We can allow the system to enter a deadlocked state, detect it and recover.

→ we can ignore the problem altogether and pretend that deadlocks never occur in the system.

Deadlock handling strategies

- Deadlock prevention
- Deadlock avoidance
- Deadlock Dection & Recovery
- Deadlock ignorance (ostrich method)

Deadlock prevention:

→ It involves designing a system that violates one of the four necessary conditions required for the occurrence of deadlock.

→ This ensures that the system remains free from the deadlock.

The various conditions of deadlock occurrence may be violated as

Mutual Exclusion, Hold and wait, No pre-emption & circular wai

Mutual Exclusion:

→ The mutual-exclusion condition must hold for nonsharable resources. eg:- printer cannot be simultaneously shared by several processes.

→ sharable resources cannot require mutual exclusion and thus cannot be involved in a deadlock.

Eg: If several processes attempt to open a read only file at the same time, they can be granted simultaneously access the file. and a process never need to wait for a sharable resources.

Hold and wait:

This condition can be violated in the following ways

case1:

→ A process has to first request for all the resources it requires for execution. once it has acquired all the resources, only then it can start its execution.

→ In this case, it ensures that the process does not hold some resources and wait for other resources.

→ The main drawback is, it is less efficient, and it is not implementable since it is not possible to predict in advance which resource will be required being execution.

case2:

→ A process is allowed to acquire the resources it desire at the current moment. after acquiring the resources, it starts execution.

→ Now before making any new request, it has to compulsorily release all the resources that it holds currently.

→ This case is efficient & implementable.

case3:

→ A timer is set after the process acquires any resource.

→ after the timer expires, a process has to compulsorily release the resource.

No preemption:

→ This condition can be violated by forceful preemption

→ A process is holding some resources and request other resources that can not be immediately allocated it.

Circular wait:

→ This condition can be violated by not allowing the processes to wait for resource in a cyclic manner.
→ To violate this condition, the following steps considered.
→ A natural number is assigned to every resource.
→ Each process is allowed to request for the resources either in only increasing or only decreasing order of the resource number.
→ In case increasing order is followed, if a process requires a lesser number resource, then it must release all the resources having larger number and vice versa.
→ this approach is most practical & implementable.
→ this approach may cause starvation but will never lead to deadlock.

Deadlock avoidance:

In deadlock avoidance each resource is carefully analyzed to see whether it could be safely fulfilled without causing deadlock. the drawback of this method is its requirement of information in advance about how resources are to be requested.

Safe state:

A state is safe if the system can allocate resources to each process in some order and still avoid a deadlock. a system is safe then there exists a safe sequence.

Sequence of process $\langle P_1, P_2 \cdots P_n \rangle$ is a safe sequence for the current allocation state. a deadlocked state is unsafe state. in unsafe state there is no safe sequence.

Banker's Algorithm:

→ Banker's algorithm is a deadlock avoidance strategy.

→ Whenever a new process is created, it specifies the maximum number of instances of each resource type that it exactly needs.

→ To implement banker's algorithm, following four data structures are used.

- available
- Max
- Allocation
- Need

available: it is a single dimensional array that specifies the number of instances of each resource type currently available.

Eg:- available [R1] = k

Max: it is a two dimensional array that specifies the maximum number of instances of each resource type that a process can request.

Eg:- max [P1] [R1] = k

Allocation: it is a two dimensional array that specifies the number of instances of each resource type that has been allocated to the process.

Eg:- Allocation [P1] [R1] = k

Need: it is a two dimensional array that specifies the number of instances of each resource type that a process requires for execution.

Eg:- Need [P1] [R1] = k

→ Banker's algorithm is executed whenever any process puts forward the request for allocating the resources.

→ It checks whether the request made by the process is valid or not

Eg: consider the following snapshot with Resource types A, B, C. Resource type A has ten instances, resource type B has five instances and resource type C has seven instances. then find Need matrix and safe sequence.

	Allocation			max			available			Need		
	A	B	C	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	5	3	3	3	2	7	4	3
P_1	2	0	0	3	2	2	5	3	2	1	2	2
P_2	3	0	2	9	0	2	7	4	3	6	0	0
P_3	2	1	1	2	2	2	7	4	5	2	1	1
P_4	0	0	2	4	3	3	7	5	5	5	3	1
	7	2	5				10	5	7			

step1: find the allocated Resources 7 2 5 (A B C).

subtract the allocated Resources from total Resources.

A B C Total Resources
10 5 7
7 2 5 allocated Resources

3 3 2 - available resources.

Need = max - available

if Need $\leq$ available then execute the particular process Otherwise move to next process.

and safe sequence is $P_1 \rightarrow P_2 \rightarrow P_4 \rightarrow P_0 \rightarrow P_2$

Eg2:

	Allocation			max			available			Need		
	A	B	C	A	B	C	A	B	C	A	B	C
P_0	1	0	1	4	3	1	3	3	0	3	3	0
P_1	1	1	2	2	1	4	4	3	1	1	0	2
P_2	1	0	3	1	3	3	5	3	4	0	3	0
P_3	2	0	0	5	4	1	6	4	6	3	4	1
							8	4	6			

Safe sequence is $P_0 \rightarrow P_2 \rightarrow P_1 \rightarrow P_3$

→ A request is valid if and only if the number of requested instances of each resource type is less than the need declared by the process in the beginning. If the request is invalid, it aborts the request.

→ if the request is valid, it checks if the number of requested instances of each resource type is less than the number of available instances of each type.

→ if the sufficient number of instances are not available, it asks the process to wait longer.

→ if the sufficient number of instances are available, the requested resources have been allocated to the process.

available = available - Request[i]

Allocation[i] = Allocation[i] + Request[i]

Need[i] = Need[i] - Request[i]

→ Now, banker's Algorithm follows the safety algorithm to check whether the resulting state it has entered in is a safe state or not

→ if it is a safe state, then it allocates the requested resources to the process in actual.

→ if it is an unsafe state, it asks the process to wait longer.

Safety Algorithm:

→ 1. Let work and finish be vectors of length m & n
initialize work = available & finish[i] = false for i = 0, 1, ... n-1.

→ 2. finish[i] = false ; Needi ≤ work go to step 4

→ 3. work = work + Allocation;
finish[i] = true go to step 2

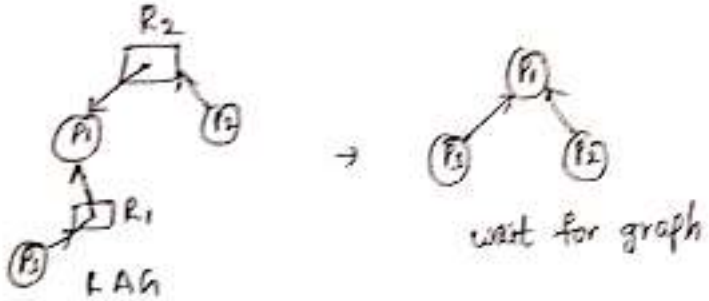
→ 4. if finish[i] = = true for all i, then the system is in safe sate.

→ it requires $m \times n^2$ operations to determine whether a state is safe

Deadlock Dection:

→ Deadlock detection can be done by using the following methods wait for graph, Banker's Algorithm for detection of deadlock.

→ wait for graph is an enhanced version of Resource allocation graph in which we do not show the resources.



wait for graph

→ if all the resources have single instances and cycle is being formed, then the system is in deadlock. if cycle is there, but resources have more than instance, then the deadlock may or may not be present.

Recovery from Deadlock:

When a deadlock has been detected in the system by deadlock detection algorithm, then it has to be recovered by using some recovery mechanism. they are

Process termination
Resource preemption

process termination:

→ one or more processes are terminated to eliminate deadlock.

→ Terminate all deadlocked processes which will break deadlock immediately, but it is a bit expensive because there may be some processes which have been executing for a long time.

→ Abort one process at a time until the deadlock cycle is eliminated. However, it has some overhead since, after terminating each process, detection algorithm has to be executed for deciding to further terminate the process or not.

Resource preemption:

Here, resources are deallocated or preempted from some processes and the same are allocated to others until deadlock is resolved we have three important issues to implement this scheme they are

selecting a victim:

we need to decide which process or resource are to be preempted, the decision is based on cost factor which include the number of resources, a deadlocked process is holding and CPU time consumed by it.

Rollback: The process which was preempted cant continue normal execution, because its resources are taken back.

starvation:

we should ensure that a particular process should not starve every time preemption is done.

Protection:

System protection:

→ protection refers to a mechanism for controlling the access of progra processes, or users to the resources defined by a computer system.

→ protection ensures that the resources of the computer are used in a proper way.

→ It ensure that each object accessed correctly and only by those processes that are allowed to do.

→ OS designer faces challenge of creating a protection scheme that cannot be by passed by any software that may be created in the future.

Goals of protection:

→ We need to provide protection for several reasons. The most obvious is the need to prevent the bad, intentional violation of an access restriction by user.

→ an unprotected resource cannot defined against misuse by c unauthorized user. A protection oriented system provides means to distinguish between authorized and unauthorized usage.

→ The role of protection in a computer system is to provide a Mechanism for the implementation of the policies governi resource use.

→ These policies can be established in a variety of ways. Some are fixed in the design of the system. while other are formulated by the management of a system. still others are defined by the individual users to protect their own files and programs.

Principle of protection:

→ The time tested guiding principle for protection is the principle of least privilege. it dictates that programs, users and even systems be given just enough privileges to perform their tasks.

→ The principle of least privilege can help produce a more secure computing Environment.

Access control:

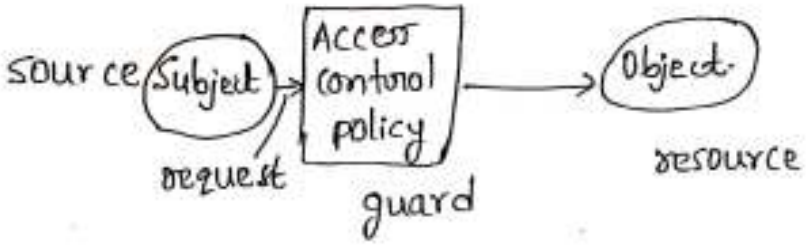
→ It allows the activities of valid users, mediating every attempt by a user to access a resource in the system.

→ Object: An entity that contains or receives information access to an object potentially implies access to the information it contains.

Eg:- file, programs, printer, disk etc.

→ Access rights: The permission granted to a user to perform an operation.

Eg:- Read, write, execute etc.



Domain of protection:

→ Domain is a collection of objects and a set of access rights for each of the objects

→ A process operates within a protection domain that specifies the resources that the process may access.

→ Each domain defines a set of objects and the types of operations that may be invoked on each object.
→ The ability to execute an operation on an object is an access right.
→ The system will consists of such multiple domains each having certain predefined access right on different object.
→ During execution of the process it can change the domain this is called domain switching.
→ A domain can be realized in a variety of ways

Each user may be a domain, in this case the set of objects that can be accessed depends on the identity of the user.

Each process may be a domain, in this case, the set of objects that can be accessed depends on the identity of the process.

Access Control Matrix:

→ protection model can be viewed abstractly as a matrix, called an access matrix.
→ rows represent domains, columns represent objects.
→ Each entry in the matrix consists of a set of access rights.
→ The entry access (i, j) defines the set of operations that a process executing in Domain, can invoke on object.

Object / domain	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

→ If a process in domain D_i tries to do operation on object O_j then operation must be in the access matrix.

use of Access matrix:

Access matrix design separates mechanism from policy mechanism: operating system provides access matrix + rules it ensures that the matrix is only manipulated by authorized agents and that rules are strictly enforced.

policy: user dictates policy. who can access what object and in what mode.

object / domain	F_1	f_2	f_3	laserprinter	P_1	P_2	P_3	P_4
D_1	read		read			switch		
D_2				print			switch	switch
D_3		read	execute					
D_4	read write		read write		switch			

Access matrix with domains as objects

object / domain	f_1	f_2	f_3
D_1	execute		write
D_2	execute	read*	execute
D_3	execute		

object / domain	f_1	f_2	f_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute	read	

access matrix with copy rights

object / domain	f_1	f_2	f_3
D_1	owner execute		write
D_2		read owner	read* owner write
D_3	execute		

object / domain	f_1	f_2	f_3
D_1	owner execute		write
D_2		owner read* write*	read* owner write
D_3		write	write

Access matrix with owner rights

Implementation of Access Matrix:

Each column = Access control list for one object

Domain D_1 = Read, write
Domain D_2 = Read
Domain D_3 = Read

Each Row = capability List
for each domain, what operations allowed on what objects.

Object 1 – Read
Object 4 – Read, Write, Execute
Object 5 – Read, Write, Delete, copy

There are several methods to implement access matrix.

Global table:

→ The simplest implementation of the access matrix
→ matrix table consists of set of ordered triples $\langle$ Domain, Object, right set $\rangle$.
→ whenever an operation m is executed on object O_j with domain D_i then a global table is searched for triple $\langle D_i, O_j, R_k \rangle$. if triple is found, operation is allowed to continue otherwise it deny access.

Disadvantages:

- usually large thus cannot be kept in main memory
  Additional I/O is needed.
- It must have separate entry in domain

Access Lists for Objects:

Access control list (ACL) focus on the object

ACL $\equiv$ column of the access control matrix

$O_j \langle D_i, R_k \rangle$

→ ACL defines all domain with non empty set of access rights for that object.
→ Access rights are often defined for groups of users because individual subjects may create a huge list.
→ ACL is stored in the directory entry of the file

Capability List:

Capability list focus on the object
capability list = rows of the access control matrix.

→ capability is pointer to the object, contain address of the object.
→ Each domain has its capability list which contain list of capability together with operation allowed.
→ capability list is itself a protected object
- maintained by operating system
- Accessed by user only indirectly.

Capability Based system:

Hydra:

fixed set of access rights known to and interpreted by the system. Analysis of user defined rights performed only by user's program. System provides access protection for use of these rights.

Cambridge CAP system:

Data capability provides standard read, write, e of individual storage segments associated with object so capability interpretation left to the subsystem, through its protected procedures.

Language Based protection:

Specification of protection in a programming Lang allows the high-level description of policies for the allocation and use of resources.

Language implementation can provide software for protection enforcement when automatic hardware supported checking is unavailable.

Revocation of access rights:

In a dynamic protection system, we may sometimes need to revoke access rights to objects shared by different users.

- Immediate versus delayed:
- Selective versus general
- partial versus total
- Temporary versus permanent

Revocation capabilities include following

- Reacquisition
- Back pointers
- indirection
- keys

UNIT-II

Process and CPU scheduling - process concepts - The process, process state, Process control Block, Threads, Process scheduling - scheduling queues, schedulers, context switch, operations on process, system calls - fork(), exec(), wait(), exit(), interprocess communication - ordinary pipes and named pipes in unix.

Process scheduling - Basic concepts, scheduling criteria, scheduling Algorithms, multiple processor scheduling, real-time scheduling, Thread scheduling, Linux scheduling and windows scheduling.

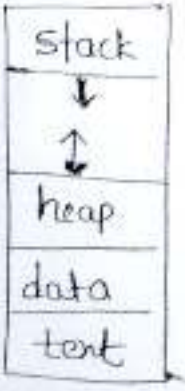
Process synchronization, Background, The critical section problem, Peterson's solution, Synchronization Hardware, semaphores, classic problems of synchronization, monitors, synchronization in Linux & windows.

Process:

A process is a program in execution. process execution must progress in sequential order. process is an active entity that requires a set of resources including program counters, processor.

Each process has its own address space. it is divided into three regions.

- Text region: It stores the code that the processor executes.
- data region: it stores the static and global variables and dynamically allocated memory that the process used during execution.
- stack region: it include temporary data like functions parameters, return address, local variables



memory in process

program: program is a set of instructions where as process is a set of executable instructions and program is a passive entity and process is a active entity.

Process states:

→ If a process executes, it changes state of the process. Each process having following states

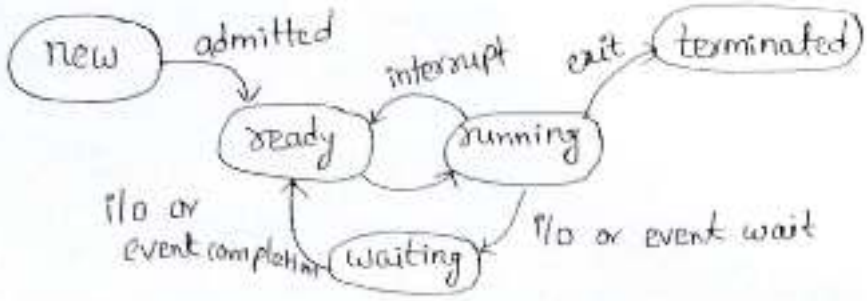
New: OS creates a new process to which resources are not allowed.

Ready: The process is waiting for the CPU to reach the header of the queue.

Running: The process is allocated to CPU for its execution.

Waiting or Block: The process is waiting for some event to occur such as I/O completion.

Termination: The process completes all its operations and release the resources which are allocated to it while it is running. It deletes PCB contents.



Process States

Process Scheduling:

In multiprogramming system, there are several programs or processes running in the system, but all the processors are not executed by cpu. scheduling is choose a particular process for execution these complete process can be done by using process scheduling

- The main intension of process scheduling is to keep the cpu busy at all times.
- It ensures maximum utilization of the CPU because a process is always running at the specific instance of time.
- For a single processor system, there will be only one process running
- If there are more processes, the rest will have to wait until the CPU is free and can be rescheduled by the process manager/process scheduler.

Scheduling queues:

OS maintains different queues for different purposes in order To help process scheduling we are going to use scheduling queues

Scheduling queues are two types

- Job queue
- Ready queue
- Device queue

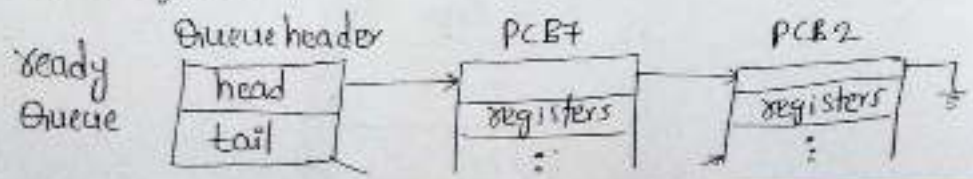
Job queue: it contains the newly created process which are usually present on the harddisk. job queue contains all processes in the system.

Ready queue: The processes that are residing in main memory & are ready and waiting to execute are kept on a list called the ready queue.

(or)

Ready queue contains the processes which are waiting for the CPU

- this queue is generally stored as a linked list
- the ready queue header contains pointer to the 1st and final PCB's in the list.
- Each PCB includes a pointer field which points to the next PCB in the ready queue.



Process control block (PCB):

When a process is created by OS its corresponding PCB is created which describes about the process. and we can also call task control block. PCB contains following information.

Process id: Which is used for uniquely identifying a process

process state: process may be in any one of the state new, ready, running, waiting and terminated.

Program counter: it stores the address of next instruction to be executed by the process.

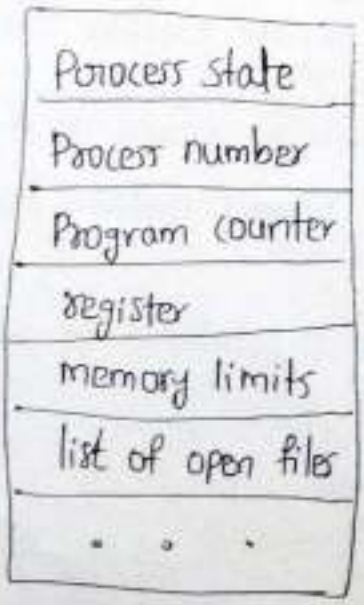
CPU registers: it include accumulator, index registers, stack pointers, general purpose registers etc which allows the process to be continued after the interrupt occur.

priority number on CPU-scheduling information: OS allocates the priority number to each process according to the priority no it allocate the resources.

Memory-Management information: it include the value of base registers, limit registers and page tables depending on the memory system used by OS.

Accounting information: it includes the amount of CPU and real time used, time limits, account numbers and so on.

I/O status information: it contains list of I/O devices allocated to the process.

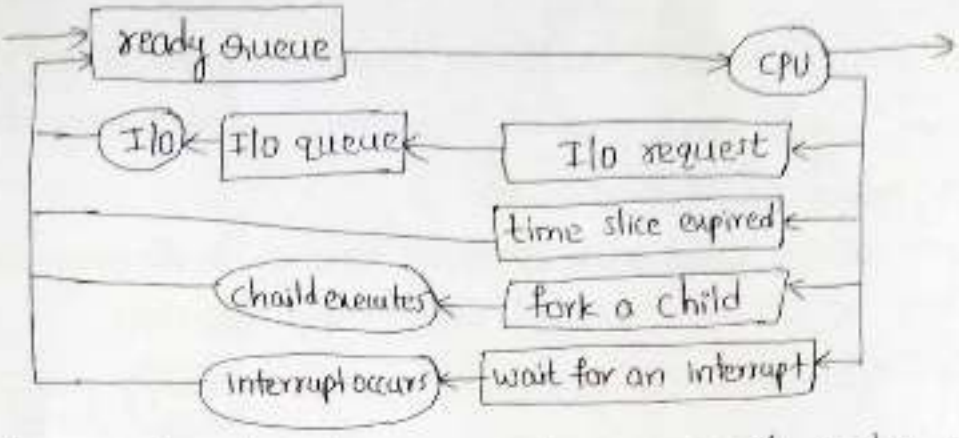


PCB

Device Queue:

Device Queue contains list of processes waiting for a Particular I/O device.

→ process scheduling can be represented using queuing diagram.



→ Here rectangular box represents a queue & circles represents the resources that serve the queue. and arrows indicate flow of processes.

→ A new process is initially placed on the ready queue & wait until it is selected for execution.

→ Once the process is assigned to the CPU several events may occur while running

→ The process may require some I/O device to perfor operations. so it is placed in the I/O queue.

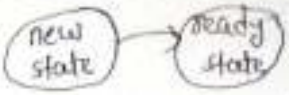
→ The process may create new sub process and wait for its termination

→ The process could be removed from the CPU as a result of interrupt and placed in the ready queue.

Schedulers:

The selection process can be carried out by schedulers their are three types of schedulers

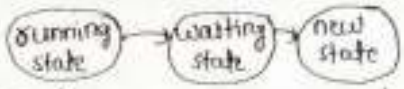
- long term schedulers
- short term schedulers
- medium term schedulers

Long term schedulers (or) Job scheduler: 

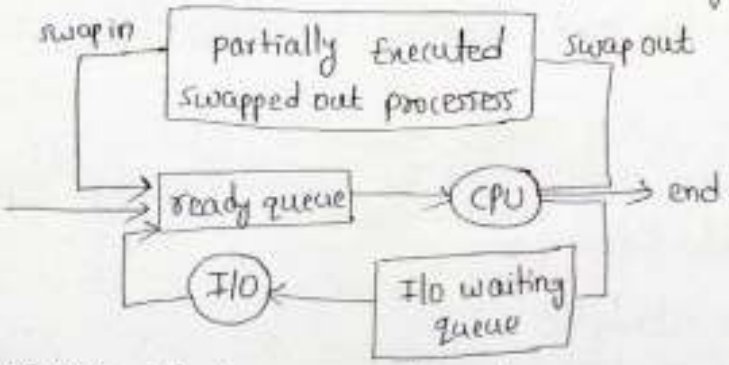
→ It selects the process from the pool (secondary memory) and keeps them in the ready queue for its execution.
→ It control the degree of multiprogramming (no. of processes present in memory).
→ It select a good process mix of I/O bound and CPU bound processes.
→ The process which spent most of their time in performing I/O operations then it is called I/O bound process.
→ The process which spent most of their time in performing some computations on CPU it is called CPU bound processes.

Short term schedulers (or) CPU schedulers: 

→ it selects one of the process from ready queue and allocate it to the CPU for the execution.
→ it select process whose CPU burst time is very high then all the jobs after that, will have to wait in the ready queue for a long time this problem is called starvation.

Medium term scheduler: 

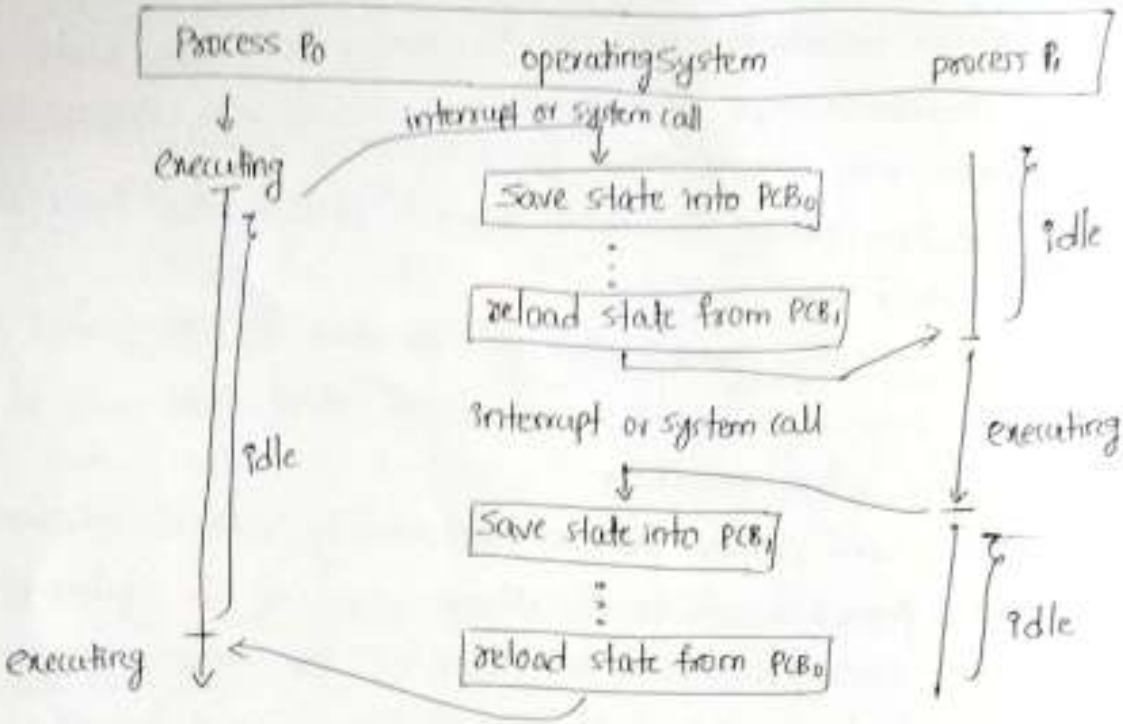
→ It removes the processes from memory, when an interrupt occurs and later the process is reintroduced into the memory for its execution and the process is continued when it is left.
→ it reduces the degree of multiprogramming.



→ It takes care of the swapped out processes. if the running state processes needs some I/O time for the completion then there is a need to change its state from running to waiting.

Context switching:

switching the cpu from one process to another process



→ It is a mechanism to store and restore the state or context of the CPU in PCB.

→ Switching the cpu to another process requires saving the state of the old process and loading the saved state for the new process. This task is known as a context switch.

→ The context switching is time taken to store previous process details and to load next process

A context switching can be described as follows

→ suspense the execution of current process and storing the cpu state for that process in memory.

→ Retrieving the context of the next process from memory and restoring it in the cpu registers

→ Returning to the location indicated by the program counter (PC) in order to resume the process.

→ context switch time is pure overhead because the system is not doing any useful work while switching.

Operations on processes:

- Process creation
- Process termination

Process creation:

→ A process may create new processes using 'create' process system call.
→ The creating process is called parent process, the new process is called child.
→ In unix, a new process can be created by using fork system call.
→ When a process creates a new process then their will be two possibilities while executing
  i. The parent process execute concurrently with its children.
  ii. The parent waits until all (or) some of its children have terminated
→ There are two possibilities in terms of the address space of new process
  i. The child process is duplicate of the parent process.
  ii. The child process has a new program loaded into it.

Process termination:

→ A process terminates when it finishes the execution of final statement and request the OS to delete it by using 'exit' system call.
→ When the OS receives the exit system call it deletes (or) deallocates the resources and delete the processes.
→ A parent may terminate the execution of its children for following reasons
  i. The task assign to the child is no longer required.
  ii. The child exit the time allocated to it.
  iii. The parent is exiting and the OS does not allow the child to continue.
→ The parent process may wait for a termination of a child process using wait system call.

System calls :

fork():
A parent process uses fork to create a new child process. The child process is a copy of the parent. After fork both parent and child executes the same program but in separate processes.

syntax: #include<unistd.h>

pid_t fork(void);

where pid- is child process
0- return in the child, "-1"- return to parent

⇒ If fork is called for n times, the number of child processes or new processes created will be $2^n - 1$.

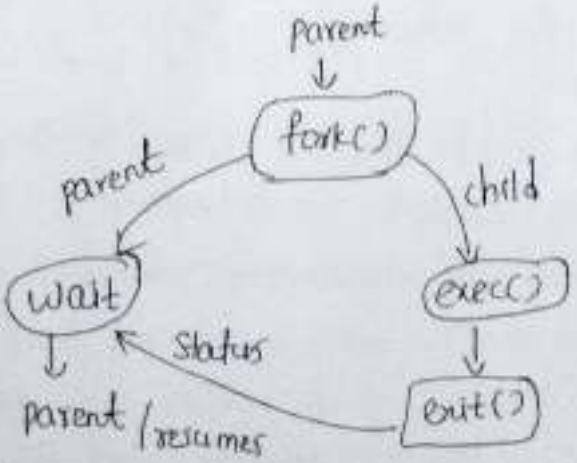
Wait(): The wait system call blocks the caller until one of its child process terminates.

syntax: pid_t wait(int *status);

status is not null, wait store the exit status of the terminated child on success, wait returns the pid of the terminated child on failure it return -1.

exec(): it replaces the program executed by a process the child may use exec after a fork to replace the process memory space with a new program executable making the child execute a different program that the parent.

exit(): Terminates the process with an exit status.



Interprocess communication:

If a process executing concurrently in the os may be either independent process (or) co-operative process.

Independent process:

A process is said to be independent if its execution does not depend on other process. any process that does not share data with any other process is said to be independent process.

co-operative process:

A process is said to be co-operative if its execution depends on any other process. any process share data with any other process is said to be co-operative process.

reasons for co-operation

- Information sharing
- computation speed up.
- modularity
- convenience

→ Inter process communication is a mechanism which allows the co-operating processes to exchange the data and information.

→ IPC can be implemented in two ways

i. shared memory

ii. message-passing

Shared memory: A region of shared memory is created by a process and the other process which wants to communicate must attach its address space to the shared memory

→ The processes exchange the information by reading and writing data to the shared area.

→ These processes are also responsible for ensuring that they are not writing to the same location simultaneously.

→ It allows maximum speed and convenience of communication.

Message passing:

- In message passing processes communicates with each other without using any kind of shared memory if two processes P_1 and P_2 want to communicate with each other, they proceed as follow
  - Establish a communication link (if already exist, no need to establish).
  - Start exchanging messages using basic primitives
    - Send (message, destination) or send (message)
    - receive (message, host) or receive (message)

→ The message size can be fixed or variable size if it is fixed size, it is for OS designer but complicate for programmer and if it is variable size, it opposite to fixed size.

→ several methods are available for logically implementing the link and send, receive operations.

i. Disect or indirect communication
ii. Synchronous or Asynchronous communication
iii. Automatic or explicit buffering

Direct or indirect communication:

The processor involved in ~~this~~ direct communication, it must mention the name of the sender or recipient. The primitives are defined as

- Send (P_i, message) - Send a message to process P
- receive (P_i, message) - Receive a message from a process P_i

In indirect communication, the messages are sent to and received from mailboxes or ports. the primitives are defined as

- Send (A, message) - Send a message to mailbox A
- receive (A, message) - Receive a message from mailbox A.

→ in direct communication only one link established between 2 processes. but Indirect communication link anociated with more than 2 processes

→ In each pair of processes, there exists exactly one link in direct communication, where as in indirect communication there exists different links between processes.

Synchronous or Asynchronous communication:

→ The message passing can be done by using Primitives like blocking, non-blocking

→ blocking send: the sending proces is blocked until the message is received by the receiving process or by the mailbox.

→ nonblocking send: The sending process sends the message and resumes operation.

→ Blocking receive: The receiver blocks until a message is available.

→ non blocking receive: The receiver retrieves either avatld msg or null.

Automatic or explicit buffering:

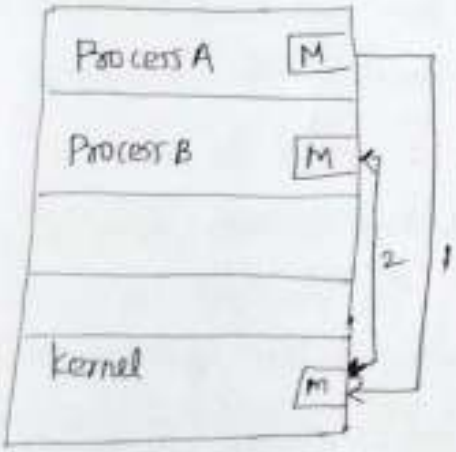
Whether communication is direct or indirect, messages exchanged by communicating processes reside in a temporary queue. this queues can be implemented in 3 ways.

Zero Capacity: The queue has a maximum length of zero.

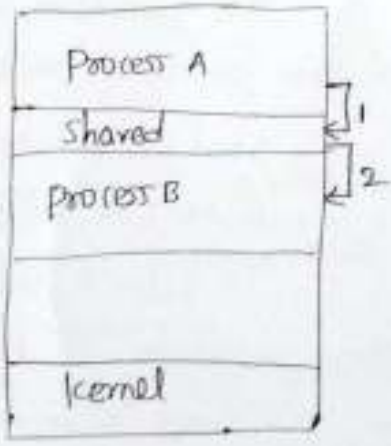
Bounded capacity: The queue has finite length.

unbounded capacity: The queue has infinite length.

Pipes.



Message passing



Shared memory

Communicatim models

Pipes:

It is one of the first IPC mechanism. A pipe is one-way communication only, a pipe passes a parameter such as the output of one process to another process which accepts it as input. the system temporarily holds the piped information until it is read by the receiving process.

In implementing pipes four issues must be considered

→ whether pipe allow unidirectional or bidirectional communication
→ if two-way communication is allowed it is half duplex or full duplex
→ Relationship between the processes.
→ pipes are communicate over a network (or) same machine.

Generally pipes are two types

i) Ordinary pipes
ii) Named pipes

Ordinary pipes:

→ It allows only one-way communication i.e, one process writes on the one end and other process reads on the other end.
→ ordinary pipes are constructed using the function pipe (int fd[])

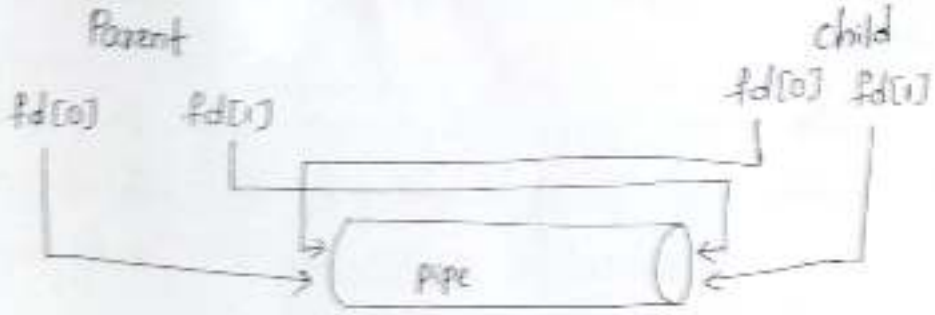
int fd[] - file descriptors
fd[0] - is the read end of pipe
fd[1] - is the write end of pipe

→ unix treats pipes as a special type of file thus, pipes can be accessed using ordinary read() and write() system calls.
→ Ordinary pipes cannot be accessed from outside the process that creates it. parent process creates a pipe and uses it to communicate with a child process.

i.e, child inherits the pipe from its parents

→ The parent writes to the pipe and the child reads from it. both the parent & child processes close their unused ends of pipes.

→ A process reading from the pipe can detect end of file when the writes has closed its ends of the pipe.
→ once the processes have finished communicating and terminated.
→ ordinary pipes on windows systems are termed anonymous pipes.



Named pipes:

Named pipes provides bidirectional communication. their is no parent child relationship. only half duplex transmission is permitted

→ Once a named pipe is established several processes can use it for communication.
→ named pipes continue to exist after communicating processes have finished. named pipes are refered as (FIFO's) in unix systems.
→ A FIFO is created with the mkFIFO() system call and manipulated with the ordinary open(), read(), write() and close() system calls

Scheduling criteria:

Many criteria have been suggested for comparing the CPU scheduling algorithms they are

(i) Throughput: Number of processes executed per unit of time

(ii) Turn around time:

It is the amount of time spent in the system (or) The time interval between submission of a process to its completion.

(iii) waiting time:

The sum of the periods spend waiting in the ready queue, amount of time a process has been waiting in the ready queue to acquire get control to the CPU.

(iv) Response time:

It is the amount of time taken to start responding.

(v) CPU utilization:

The main purpose of it is to make the CPU as busy as possible.

(vi) Brust time:

Amount of time taken to complete its execution.

Scheduling Algorithms:

To decide which process to execute first and which process to execute last to achieve maximum CPU utilization.

1. First come First Serve (FCFS) scheduling
2. Shortest - Job - First (SJF) "
3. priority scheduling
4. Round Robin (RR) scheduling
5. multilevel queue scheduling
6. multilevel feedback queue scheduling

CPU Scheduling Algorithm:

- CPU scheduling is a process which allows one process to use the CPU while the execution of other process is on hold due to unavailability of any resources like I/O.
- Whenever CPU becomes idle, the OS must select one of the processes in the ready queue to be executed. The selection process is carried out by the short-term scheduler.
- The aim of CPU scheduling is to make the system efficient, fast

Dispatcher:

The dispatcher is the module that gives control of the CPU to the process selected by short-term scheduler. The dispatcher function involves the following

- switching context
- switching to user mode
- jumping to the proper location in the user program to restart that program

Processes can be scheduled in two ways

i) Non-preemptive
ii) preemptive

Non-preemptive:

Once the CPU has been allocated to a process the process keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state.

Preemptive: When the OS decides to favour another process it release the concurrently executing process.

- A low priority process has to suspend its execution if the high priority process is waiting in a queue

First come first serve scheduling (FCFS):

→ The process which requests the CPU first, gets the CPU allocated First

→ Eg: buying tickets at ticket counter

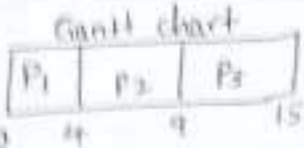
→ FCFS is non-preemptive (process continues until burst cycles complete)

Example: 1 with out arrival time

process no	burst time
1	4
2	5
3	6

P.no	BT	WT	TAT
1	4	0	4
2	5	4	9
3	6	9	15

Gantt chart



Waiting time: $P_1 - 0$
$P_2 - 4$
$P_3 - 9$

Total turn around time: $P_1 - 4$
$P_2 - 9$
$P_3 - 15$

average waiting time = $(0+4+9)/3 = \frac{13}{3} = 4.3$ ms

Note:

→ Arrival time: Time at which the process arrives in the ready queue

→ completion time: Time at which process completes its execution

→ Burst time: Time required by a process for CPU execution

→ Turn Around time: Time difference between completion time and arrival t

$$TAT = CT - AT \quad (or) \quad TAT = WT + BT$$

→ waiting time (WT): Time difference between turn around time & burst ti

$$WT = TAT - BT$$

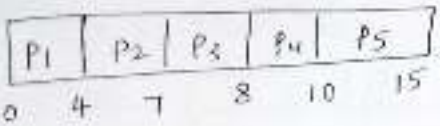
→ gantt chart: it is a barchart which illustrates the particular schedulag including the start and finish time of each process

Eg 2:

P NO	AT	BT
1	0	4
2	1	3
3	2	1
4	3	2
5	4	5

P NO	AT	BT	CT	TAT	WT
1	0	4	4	4	0
2	1	3	7	7-1=6	3
3	2	1	8	8-2=6	5
4	3	2	10	10-3=7	5
5	4	5	15	15-4=11	6

Gantt chart:



Completion time P_1 - 4
P_2 - 7
P_3 - 8
P_4 - 10
P_5 - 15

TAT = CT - AT ⇒ 4 - 0 = 4 - P_1

WT = TAT - BT ⇒ 4 - 4 = 0 - P_1, P_2 = 6-3 = 3, P_3 = 6-1 = 5

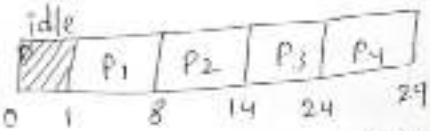
AWT = (0+3+5+5+6)/5 = 19/5 = 3.8 ms

Eg 3:

P NO	AT	BT
1	1	7
2	2	6
3	3	10
4	4	5

P NO	AT	BT	CT	TAT	WT
1	1	7	8	7	0
2	2	6	14	12	6
3	3	10	24	21	11
4	4	5	29	25	20

Gantt chart:



AWT = (0+6+11+20)/4 = 37/4 = 9.2 ms

Advantages:
- it is simple and easy to understand
- it can be easily implemented using eq data structure
- It does not lead to starvation.

Disadvantages: It suffers from convoy effect

Shortest Job first Scheduling (SJF):

→ In SJF the process with less CPU burst time will be processed first, before other processes if two processes have same burst time then they will be scheduled by using FCFS scheduling.

→ SJF is of two types

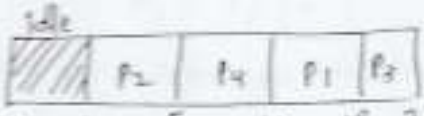
1. Non-preemptive – simply call it as SJF
2. pre-emptive – shortest-remaining time first scheduling (SRTF)

Eg1: non-preemptive SJF

P.no	AT	BT
1	0	8
2	1	4
3	2	9
4	3	5

In SJF lowest BT execute 1st

Gantt chart



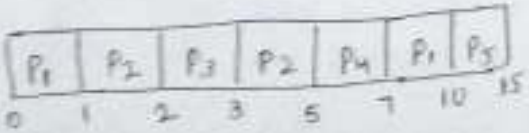
because lowest BT execute first, but it contain AT-1, so CPU idle for 1 sec

P.NO	AT	BT	CT	TAT	WT
1	0	8	18	18	10
2	1	4	5	4	0
3	2	9	27	25	16
4	3	5	10	7	2

Eg2: pre-emptive SJF (or) SRTF

P.NO	AT	BT
1	0	4 3 2 1 0
2	1	3 2 1 0
3	2	1 0
4	3	2 1 0
5	4	5 4 3 2 1 0

P.NO	AT	BT	CT	TAT	WT
1	0	4	10	10	6
2	1	3	5	4	1
3	2	1	3	1	0
4	3	2	7	4	2
5	4	5	15	11	6



AWT = (6+1+0+2+6)/5 = 15/5 = 3 ms

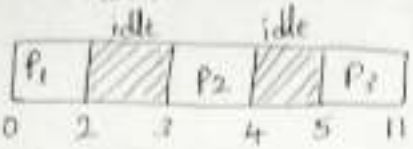
ATAT = (10+4+1+4+11)/5 = 30/5 = 6 ms

Eg3: SRTF

P.NO	AT	BT
1	0	2
2	3	1
3	5	6

P.NO	AT	BT	CT	TAT	WT
1	0	2	2	2	0
2	3	1	4	1	0
3	5	6	11	6	0

Gantt chart:



AWT = 0

ATAT = (2+1+6)/3 = 9/3 = 3 ms

SRTF: When a process is running state and a new process arrives whose CPU burst time is shorter than the active process.

Advantages:

→ SRTF is optimal and guarantees the minimum average waiting time.

Disadvantages:

→ it leads to starvation for processes with larger burst time.

→ processes with larger burst time have poor response time.

Note:

Convoy effect: in non-preemptive algorithms, once CPU time has been allocated to a process, other processes can get CPU time after the current process has finished. this situation called convoy effect in which the whole operating system slows down due to few slow processes and convoy effect for a system.

Starvation problem: a process ready to run for CPU can wait indefinitely because of low priority compare to other process. this can lead indefinitely waiting for the process for CPU which is having low-priority, this leads to starvation problem.

solution to starvation is aging: Aging is a technique of gradually increase the priority of processes that wait in the system for a long time

Priority Scheduling Algorithm: [preemptive / non-preemptive]

→ In priority scheduling each process is assigned a priority, in this process with a high priority is to be executed first and so on.

→ process with same priority are executed on FCFS scheduling

→ priority can be decided based on memory requirements, time requirement (or) any other resource requirements.

→ priority numbers are 0 to 7 (or) 0 to 4095 fixed numbers.

Eg1: non-preemptive

P no	BT	priority
1	5	6
2	8	4
3	2	2
4	6	0

P	P3	P2	P1

0 6 8 16 21

P NO	BT	priority	CT	TAT	WT
1	5	6	21	21	16
2	8	4	16	16	8
3	2	2	8	8	6
4	6	0	6	6	0

Average TAT = (21+16+8+6)/4 = 51/4 = 12 ms

Average WT = (16+8+6+0)/4 = 30/4 = 7.2 ms

Eg 2: non-preemptive

P no	BT	AT	priority
1	4	0	2 (high)
2	2	1	4
3	3	2	6
4	5	3	10 (low)
5	1	4	8

P1	P2	P3	P5	P4

0 4 6 9 10 15

P NO	BT	AT	priority	CT	TAT	WT
1	4	0	2	4	4	0
2	2	1	4	6	5	3
3	3	2	6	9	7	4
4	5	3	10	15	12	7
5	1	4	8	10	6	5

ATAT = (4+5+7+12+6)/5 = 34/5 = 6.8 ms

AWT = (3+4+7+5)/5 = 19/5 = 3.8 ms

Advantages:
- It consider the priority of the processes and allows the important pro... to run first.
- priority scheduling in preemptive mode is best suited for real time os.

Disadvantages:
Processes with lesser priority may starve for CPU.
There is no idea of response time and waiting time.

Round Robin scheduling:
- Round Robin algorithm takes a particular process for execution.
- it will execute the process only for time quantum or time slice. after the completion of time slice it will be preempted and next process will be allocated to the CPU.
- Every process will execute only in its time quantum so that no process will be in waiting state.

Eg: TQ/TS = 2

Pno	BT	CT
1	4	
2	2	
3	3	
4	5	

Pno	BT	CT	TAT	WT
1	4	10	10	6
2	2	4	4	2
3	3	11	11	8
4	5	14	14	9

P_1	P_2	P_3	P_4	P_1	P_3	P_4	P_4

0 2 4 6 8 10 11 15 14

ATAT = (10+4+11+14)/4 = 24/4 = 9.7 ms

AWT = (6+2+8+9)/4 = 24/4 = 6.25 ms

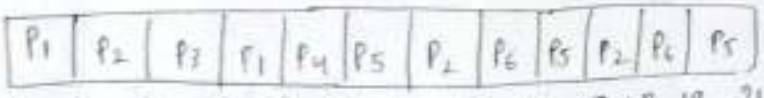
Note:
- If decreasing the value of time quantum, is better in terms of response time.
- If increase the value of time quantum, is better in terms of number of context switch.
- The value of time quantum should be such that it is neither too big nor too small.

Eg 2: TQ/TS = 2

P NO	AT	BT
1	0	4
2	1	5
3	2	2
4	3	1
5	4	6
6	5	3

P NO	AT	BT	CT	TAT	WT
1	0	4	8	8-0=8	8-4=4
2	1	5	18	18-1=17	17-5=12
3	2	2	6	6-2=4	4-2=2
4	3	1	9	9-3=6	6-1=5
5	4	6	21	21-4=17	17-6=11
6	5	3	19	19-5=13	13-3=10

Gantt chart for RR



Ready queue:

$P_5, P_6, P_2, P_5, P_6, P_2, P_5, P_4, P_1, P_3, P_2, P_1$ (or) $P_1\ P_2\ P_3\ P_1\ P_4\ P_5\ P_2\ P_6\ P_5\ P_2\ P_6\ P_5$

Average turn around time = (8+17+4+6+17+13)/6 = 65/6 = 10.8

Average waiting time = (4+12+2+5+11+10)/6 = 44/6 = 7.3

Advantages:
- it gives the best performence in terms of average response time.
→ it is best suited for time sharing system, client server architecture.

Disadvantages:
→ it leads to starvation for processes with larger burst time as they have to repeat the cycle many times.
→ its performence heavily depends on time quantum.
→ priorities can not be set for the processes.

Eg:

Multilevel Queue scheduling:

Various types of processes present in the system and we cannot place them in single ready queue and we cannot place them in single ready queue and we cannot apply same scheduling algorithm

→ The concept of multilevel queue scheduling is introduced.

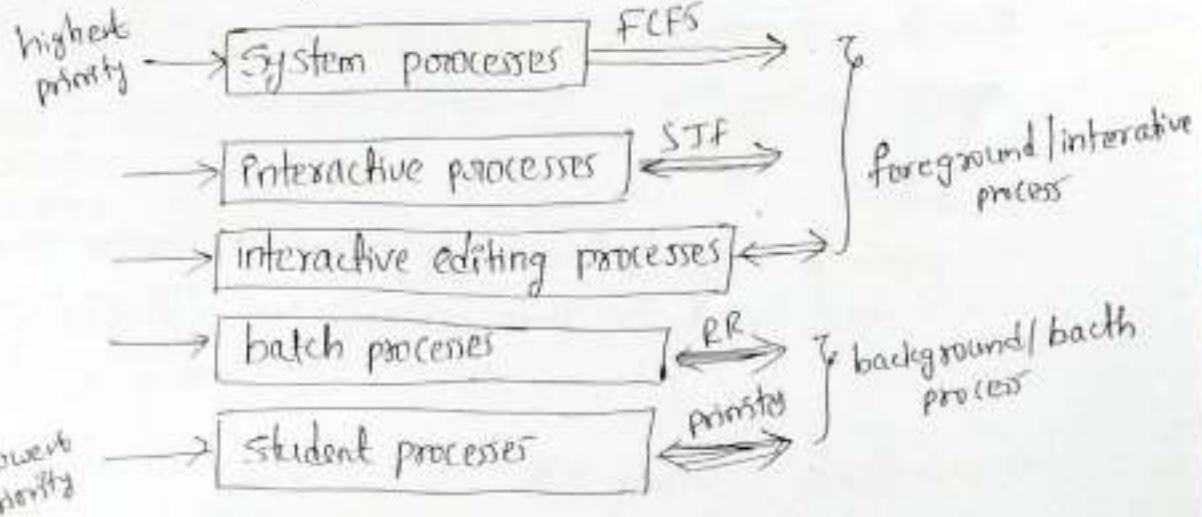
→ This algorithm divides the ready queue into several separate queue.

→ These processes are placed on the ready queues based on their time. Processes are permanently assigned to one queue based on properties like process priority, memory size, process type etc.

→ Each queue has its own scheduling algorithm.

→ The high priority processes will be placed in the top level ready queue and low priority processes will be placed in bottom level ready queue.

→ after completion of all the processes in the top level ready queue the further level ready queue processes will be executed.



Multilevel Queue scheduling.

→ In this the processes are divided into two groups i.e, foreground process or interative (high priority) and background process or batch process (low priority).

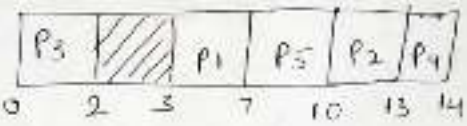
→ Advantage: We can apply different algorithm for each process

→ Disadvantage: lower level processes will face the starvation problem.

Eg:1 FCFS

P.NO	AT	BT
1	3	4
2	5	3
3	0	2
4	5	1
5	4	3

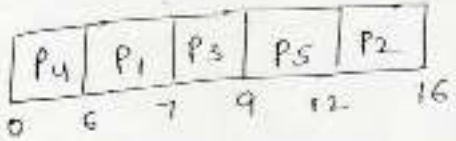
gantt chart:



Eg2: SJF (non-pre-emptive)

P.NO	AT	BT
1	3	1
2	1	4
3	4	2
4	0	6
5	2	3

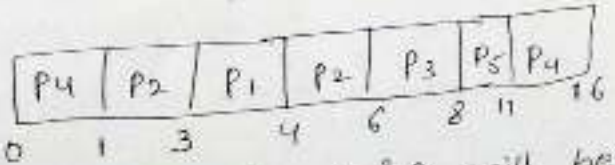
gantt chart:



Eg3: SRTF (pre-emptive)

P NO	AT	BT
1	3	1
2	1	4
3	4	2
4	0	6
5	2	3

gantt chart:

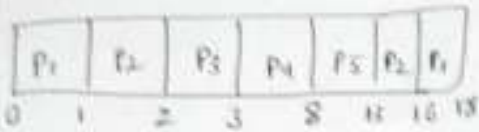


Note: if P_2 execute 1 sec then P_5 & P_2 will become same BT (3)

Eg4: priority scheduling (pre-emptive)

P.NO	AT	BT	priority
1	0	9	9 (low)
2	1	3	3
3	2	1	4
4	3	5	5
5	4	2	5 (high)

gantt chart:



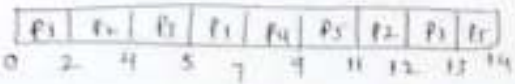
Note: In priority scheduling non-preemptive algorithm consider the lowest number is high priority & high number is low priority. where as in preemptive priority scheduling high number number consider high priority and low number consido low priority.

Eg5: Roundrobin

P.NO	AT	BT
1	0	5
2	1	3
3	2	1
4	3	2
5	4	3

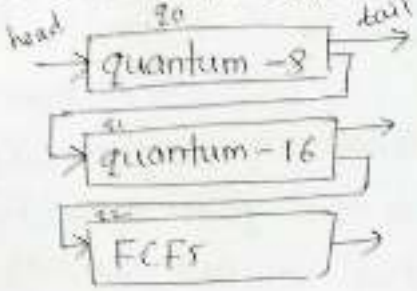
Ready queue: P1 P2 P3 P1 P4 P5 P2 P1 P5

Gantt chart:



Multilevel feedback queue scheduling:

→ It allows the processes to move between the queues the processes are placed on the queue based on CPU burst time.
→ If a process uses more CPU time then it will be moved to low priority queue a process that waits too long in a low priority queue may be moved to high priority queue
→ Consider a multilevel feedback queue schedulers with 3 ready queues
→ A process enters in the ready queue is placed in q_0. A process in q_0 is given a time quantum of 8 millisec. if it doesnot finish with in its time then it will be moved to tail of the q_1.
→ if q_0 is empty then the processes present in q_1 will be executed which have a time quantum of 16 sec.
→ if the process execution is not completed in q_1 then it will be placed in q_2 which are executed in FCFS.



Multilevel feedback queue

Multilevel feedback queue scheduler is defined by the following terms.

- The number of queues
- The scheduling algorithm for each queue
- The method used to determine when to upgrade a process to a higher priority queue.
- The method used to determine when to denote a process to a lower priority queue.
- The method used to determine which queue a process will enter when that process needs service.

Multiple processor scheduling:

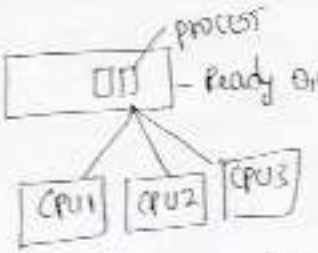
→ In multiple processor scheduling it include multiple processes and multiple Processors (CPU) with in a single system, then thir is a issue which process is going to execute first in which CPU.

→ In this case one dedicated CPU decide which process is execute first and in which CPU.

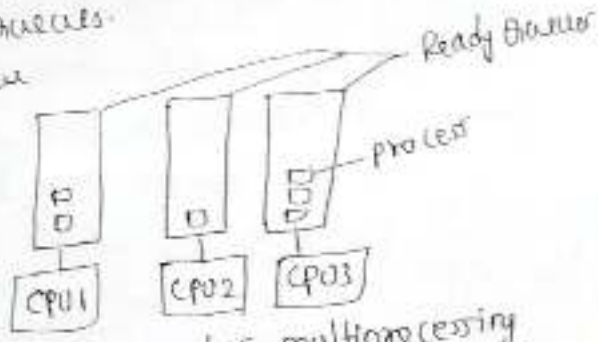
→ generally there are two types of multiple processor scheduling

1. ASymmetric multiprocessing: it maintain only one single queue for all the processes.

2. Symmetric multiprocessing (SMP): In each processes it maintain separate Ready queues.



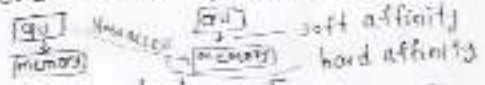
Asymetric multiprocessing



Symmetric multiprocessing.

→ In Asymmetric, at particular time only one CPU access the ready queue no other CPU's can access the global queue. this mechanism access the process by serially, we can call Locking mechanism.

→ processor Affinity: To choose which process to execute only one CPU not other CPU's.



→ Disadvantage of smp is Load imbalance [some CPU having the queue with lot of process in ready state while other CPU having less no of process in queue]

→ Load balancing: workload balanced among all processors to fully utilize the benfits of processor.

→ general approaches to load balancing 1. push migration
2. pull migration

→ push migration: specific task checks the load of all processors.

→ pull migration: idle processor pull awaiting task from a busy processor

Process Synchronization:

→ When multiple processes execute concurrently sharing same system resources, some times there may be leads to inconsistency results i.e, Changes made by one process may not reflect on the other process accessing the shared data.

→ In order to avoid this type of inconsistency processes should be synchronised with each other.

→ process synchronization is a mechanism that deals with the synchronization of processes.

→ It controls the execution of processes running concurrently to ensure that consistent results are produced.

```
Eg:  function ( )
     {
        i = 10;
        R(i);
        i = i+1;
        w(i);
     }
```

When processes P_1 and P_2 are executing independently they may produce the output as follows

P_1	P_2
i = 10	i = 11
i = 11	i = 12

When processes P_1 and P_2 executing concurrently they may produce inconsistent results

P_1	P_2
i = 10	i = 10
i = 11	i = 11

Race condition: The final output produced depends on the execution order of instructions of different process in above example we get different outputs based on instruction execution.

Critical Section problem: (CS):

Critical section is a segment of code in which a process may perform specific task such as writing to a file, updating a table etc.

The critical section problem arises when two or more processes accessing the shared data at the same time.

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

A solution to the critical section problem must satisfy the following requirements

1. Mutual Exclusion
2. progress [When no process in the CS, any process that request entry into the CS must be permitted without any delay].
3. Bounded waiting / No starvation

Mutual Exclusion: only one process is present inside the critical section at any time. No other process can enter the critical section until the process already present inside it completes.

progress: If number of process is executing in CS and some process wants to enter into the CS. then only those processes which are not executing in remainder section are allowed to enter into the CS.

Bounded waiting:

The wait of a process to enter the critical section is bounded a process gets to enter the critical section before its wait gets over.

Mutual Exclusion is mandatory criteria for CS.

Peterson's solution:

It is the software based solution to the critical section problem. Peterson's algorithm is used for mutual exclusion and allows two processes to share a single use resource without conflict it uses only shared memory for communication. and it works with two processes

The two conditions or variables used in peterson solution

- turn / int turn;
- flag / boolean flag[2];

→ Turn indicates the next process to enter its critical section

→ Flag is used to indicate the process which is ready to enter its critical section

```
do {
    flag[i] = TRUE;
    turn = j;
    while ( flag[j] && turn == j);
        critical section
    flag[i] = false;
        remainder section
} while(TRUE);
```

we prove that peterson solution is correct we need to show following

- Mutual exclusion is preserved
- The progress requirement is satisfied
- The bounded waiting requirement is met.

Synchronization Hardware:

→ To solve the critical section problems we introduced software based solution such as peterson solution but it is not guaranteed to work on modern computer architectures instead of that we use simple tool i.e. Locks

Locks:

→ Locks is one of the solution to the critical section problems.

→ A process must acquire a lock before entering the critical section and release the lock when it exits the critical section.

```
do {
        acquire lock
                critical section
        release lock
                remainder section
} while (true);
```

→ In an unique processor environment critical section problem can be solved by disabling the interrupts while a shared resource or variable is being modified. so, that the instructions in critical section are executed without any preemption.

→ But this solution is not feasible for multiprocessor system because, disabling interrupts in a multiprocessor environment can be time consuming, as the message is passed to all the processors. for this "test" and "set" instructions is introduced

```
lock = false
boolean TestAndSet (boolean *lock)
{   boolean temp = *lock;
    lock = true;
    return temp;
}
```

→ mutual exclusion implementation with TestAndSet instructions.

```
do { while (TestAndSet (&lock));
        // do nothing
        // critical section
lock = false;
        // remainder section
} while (true);
```

Semaphores:

It is one of the synchronization tool and provides a solution for critical section problem. it is a non-negative integer variable which takes only one integer value and it can be accessed through two operations. wait() and signal().

→ wait() (P): a process calls the wait operation when it wants to enter into the critical section it is represented by 'P'

→ signal() (V): a process calls the signal operation when it exits the critical section. it is represented by 'V'.

```
Wait(s)
  {
    while(s<=0); // no-operation
     s--;
  }
signal(s)
  {
    s++;
  }
```

→ A semaphore operation is automic i.e, when one process modifies the semaphore value then no other process can simultaneously modify the same semaphore value.

Types of semaphores:

1. Binary semaphore
2. Counting semaphore

Binary semaphore:

It is also known as mutex. it deals with the critical section problem for multiple process.

It ranges only between 0 and 1

0- no process

1 - representing any process.

Counting semaphore:

It can be used to control access to a given resource consisting of finite number of instances. the semaphore is initialized to number of resource available.

→ if a process wants to use a resource then it performs wait operation on the semaphore. i.e, it is decrementing the semaphore value.

→ when the semaphore value goes to 'zero' it means all the resources are used by the processes.

Limitations of semaphores:

1. Busy waiting: When a process is in critical section any other process that tries to enter its critical section must look continuously in the entry code. this looping creates a problem. i.e, cpu doesn't perform any operations to overcome the busy waiting the signal semaphore operation must be modified.

```
do {
    wait(s);
    entry code;
    critical section;
    exit code;
    signal(s);
    remainder section;
} while(true);
```

→ In wait() operation, when process reads semaphore value and it is not a positive value then that process will be blocked for some time then that blocked process will be placed in the waiting queue by the semaphores.

→ In signal operation, the wake up operation is used to move the process from waiting state to the ready state

2. Deadlock and Starvation:

Two or more process are waiting indefinitely for an event that can be caused only by one of waiting process is called Deadlock. consider 2 processes P_0 & P_1 each wants to access semaphores 'S' and 'Q'. when P_0 executes wait(S) and P_1 executes wait(Q) in order to complete their execution P_0 is requesting for semaphore Q and P_1 is requesting for semaphore S and which is held by P_0.

as a result, both processes are waiting indefinitely with in the semaphore this situation is called Deadlock.

P_0	P_1
wait(S);	wait(Q);
wait(Q);	wait(S);
signal(S);	signal(Q);
signal(Q);	signal(S);

3. Priority inversion:

When a high priority process is waiting for the low priority process in order to complete its execution

Classic problems of Synchronization:

1. Bounded-Buffer problem
2. Reader-writer problem
3. Dining-philosophers problem

Bounded-Buffer problem (or) producer and consumer problem:

→ A producer can produce an item and place in the buffer with fixed size. a consumer can pick items and can consume them.

→ if producer produce an item at the same time consumer should not consume any item.

→ To solve this problem, we need two counting semaphores full & empty

mutex = 1 –
full = 0 // initially, all slots are empty
empty = n // All slots are empty initially

→ solution for producer

```
do {
    // produce an item
    wait (empty);
    wait (mutex);
    // add to buffer
    signal (mutex);
    signal (full);
} while (true);
```

1. overflow (empty) underflow (full)
2. atleast one cell empty atleast one slot full
3. at a time can do both operations (producing, consuming)

→ solution for consumer

```
do {
    wait (full);
    wait (mutex);
    // remove an item from buffer
    signal (mutex);
    signal (empty);
    // consume item
} while (true);
```

buffer size = 3

empty	full	mutex	
3	0	1	
2	1	1	
1	2	1	} producer
0	3	1	
1	2	0	
2	1	0	
3	0	0	
2	1	0	
1	2	0	
0	3	0	
1	2	1	

Reader- Writer problem:

→ if one process is reading at the same time no other process cannot perform the write operation, otherwise the changes will not effect that file.

→ if one process is reading the file, then others may read it at the same time

→ once a writer is ready, it performs its write. only the one writer may write at a time if atleast one reader is reading, no other process can write.

→ Readers may not write and only read

→ solution for reader writer problem has the priority over writer. the reader process share following data structures:

Semaphore mutex, wrt;
Int readcount → it tells no. of process performing read in CS and initial value is '0'

→ Writer process:

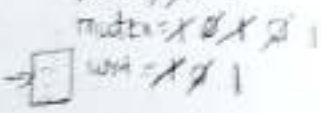

```
do {
// writer request for critical section
wait(wrt);
    // performs the write & leave CS
signal (wrt);
}while(true);
```

→ Reader process:

```
do {
// Reader wants to enter critical section
wait (mutex); // The no. of readers has incremented by 1
readcount++;
// atleast one reader in critical section, i.e, no writer enter
if(readcount == 1)
    wait (wrt);
// other reader can enter critical section
signal (mutex); // Read
wait (mutex);
readcount--;
if(readcount == 0)
    signal (wrt); // writer can enter
signal (mutex); // reader leaves
}while(true);
```

Entry section

Exit section



Dining - philosophers problem:

→ In this problem states that 5 philosophers seated around a circular table with one chopstick between each pair of philosophers.

→ There is one chopstick between each philosopher. a philosopher may eat if he can pickup the two chopsticks adjacent (left & right) to him.

→ one chopstick may be picked up by any one of its adjacent followers but not both.

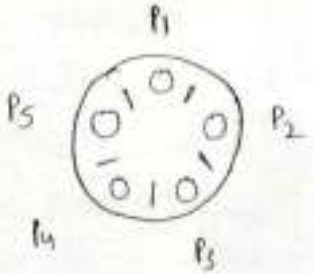
Solution:

→ allow at most four philosophers to be sitting simultaneously at the table.

→ allow a philosopher to pick up her chopstick only if both chopsticks are available.

→ an odd philosopher picks up 1st her left chopstick and then her right chopstick, where as an even philosopher picks up her right chopstick and then left chopstick.

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



Monitors:

→ Suppose a process interchanges the order in which the wait() and signal() operations on semaphore mutex are executed.

```
signal(mutex);
    critical section
wait(mutex);
```

In this case several processes may be executing in critical section simultaneously, & it violate the mutual exclusion condition

→ In case 2: thwo processes replaces signal(mutex) with wait(mutex)

```
wait(mutex);
    critical section
wait(mutex);
```

In this case deadlock will occur.

→ To deal with such errors, researchers have developed high level language synchronization construct that is monitor.

monitor syntax:

```
monitor monitorName
{
// shared variable declarations
procedure P1()
{ . . .
}
procedure P2()
{ . . .
}
procedure Pn()
{ . . .
}
}
```

→ Monitor is a module and it includes data and procedures. process calls the procedure and procedure will give access permission to process to access the data and one process can enter onto monitor at atime. monitor not allow to access the multiple process at atime.

→ To provide synchronization mode monitor allow the condition variables. ex: condition a,b;

→ if a process wants to sleep inside the monitor or it allows a waiting process to continue in that case conditional variables are used in monitors.

→ The only operations that can be invoked on a condition variable are wait() and signal() operation.

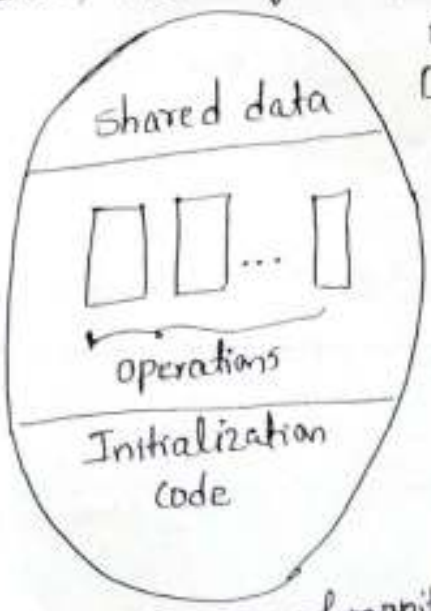
a.signal()

a.wait();

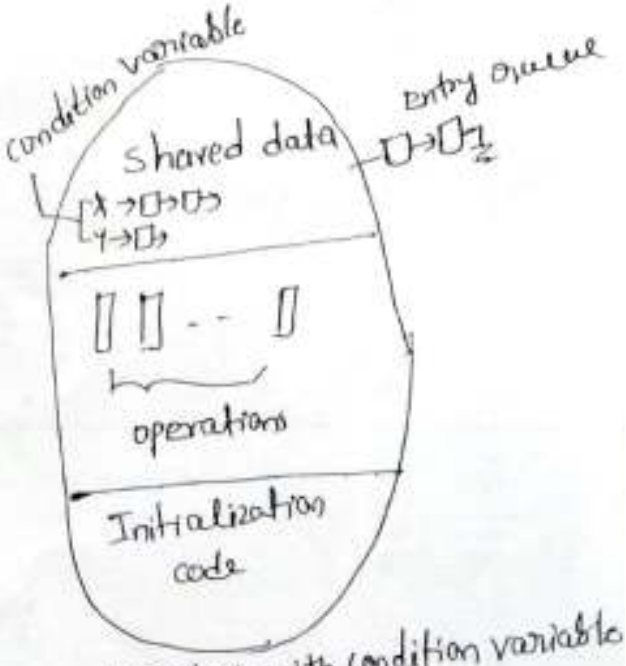
Two operations can be performed on condition variable.

Signal and wait: if resource is currently not available current Process put to sleep, it release the lock for monitors.

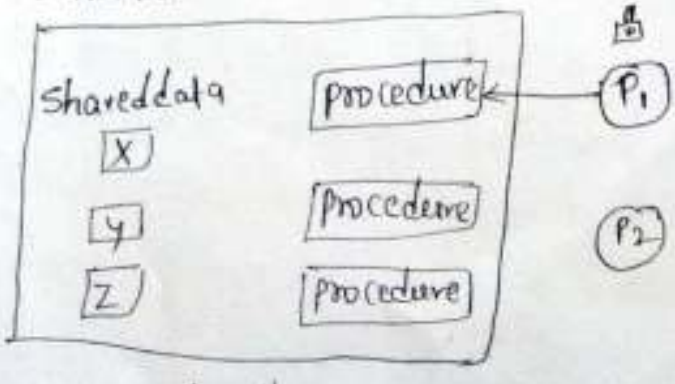
Signal and continue: it wakes up one process which are sleeping as a result of call to wait this cause a waiting process to resume immediately. the lock is automatically passed to the waiter, the original process blocks.



Schematic view of monitor



monitor with condition variable



monitor

LINUX Scheduling:

Scheduling can be defined as a way of allocating the time of CPU to various task that are present within the OS.

⇒ in Linux, processes are scheduled two types

- Real time processes
- Normal processes

Real time processes: deadline that have to met, and it should never be blocked by allow priority task.

Normal process: it may be act as a iterative or batch process.

→ it execute the process O(1) and O(n) time complexity
→ O(n) - where n is the number of runnable processors
→ O(1) - constant time required to pick the next process to execute.
→ real value range 0-99 & nice value priority lies 100 to 140

Windows scheduling:

→ It is one of preemptive scheduling. high priority thread execute first and low priority thread execute last.
→ the priority divided into two classes 1. variable class
   1. variable class [priority 1 to 15]
   2. Real time class [priority 16 to 31]
→ The dispatcher schedule the queue.
→ if no thread is found is found the dispatcher will execute a special thread called idle thread.

Realtime scheduling:

In scheduling Algorithms we can consider the TAT, WT throughput and response time, but where as in realtime scheduling different metrics are used.

→ A realtime system is one whose correctness depends on timing as well as functionality.
→ timelines: → soft realtime
→ feasibility
→ hard realtime

Thread scheduling:

When there are several processes and each process have multiple threads, then we have two levels

user level threads

kernel level threads

userlevel threads: it managed by a thread library and the kernel is unaware of them. To run on a CPU, user level threads must ultimately be mapped to an associated kernel level thread.

→ user level threads run on lightweight process (LWP) is known' process contention scope(PCS)

→ The gap between userlevel and kernel level threads, lies in how are scheduled many to one and many to many models.

Synchronization in Windows and Linux:

Synchronization in windows:

Windows XP operating system is a multithreaded kernel that provides support for real time applications and multiple processors.

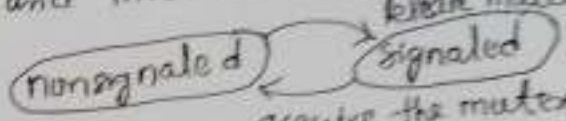
→ To synchronize threads according to several several different mechanisms, including mutexes, semaphores, events and timers.

→ Events are similar to condition variables; that is, they may notify a waiting thread when a desired condition occurs.

→ provide synchronization we use dispatcher objects

→ Dispatcher objects may be in either a signaled state or a nonsignaled state.

→ signal state indicate an object is available & thread will not block when acquiring the object. a nonsignal state indicate that an object is not available and thread will block when attempting to acquire the object.



Mutex dispatcher object

Synchronization in Linux:

→ Linux was a nonpreemptive kernel.

→ it provides spinlocks and semaphores for locking in the kernel.

Single processor	multiple processors
Disable kernel preemption	Acquire spin lock
Enable kernel preemption	Release spin lock

→ it provides two system calls

preempt - disable() } for disabling & enabling kernel preemption
preempt - enable()

→

UNIT- III

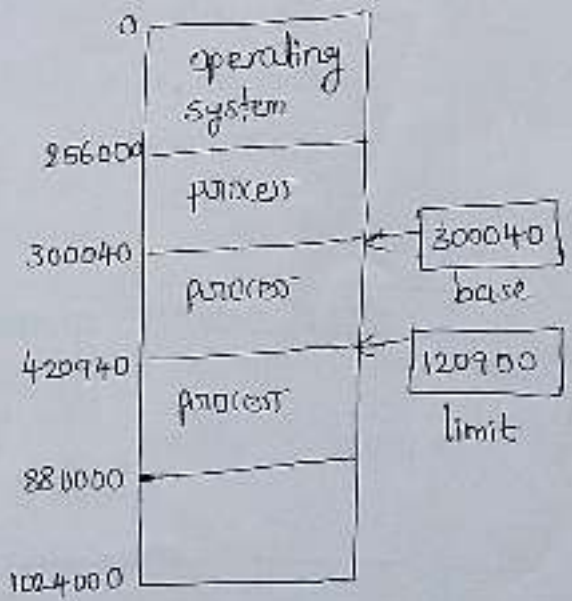
Memory Management and virtual memory - memory management strategies Background, swapping, contiguous memory allocation, segmentation, paging structure of page table, IA-32 segmentation, IA-32 paging.
Virtual memory management background, Demand paging, copy-on-write Page Replacement, page Replacement algorithms, Allocation of frames, Thrashing, virtual memory in windows.

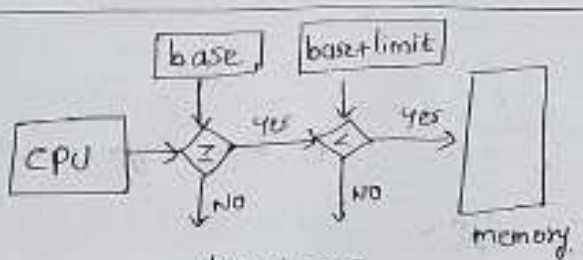
Introduction & Background:

→ Memory consists of a large array of words or bytes, each word has its own address.
→ CPU fetches the instruction from memory and execute these instructions

Basic Hardware:

→ Main memory and registers are only storage, CPU can access directly
→ Register access in one CPU clock, but main memory can take many cycles
→ cache sists between main memory and CPU registers.
→ A pair of base and limit registers define the logical address space.
→ CPU hardware compares every address generated in user mode with the registers.
→ A program executing in user mode attempt to access OS memory or other user's memory results in a trap to the OS, which treats the attempt as a fatal error.
→ This prevents a user program from accidentally modifying the code or data structure of either the OS or other users.





trap to OS
monitor - addressing error.

Address Binding:

→ A user program will go through several steps, before being executed.

→ address binding of instructions and data to memory addresses can happen at three different stages.

⇒ Compile time: if memory location known a priori, absolute code can be generated, must recompile code if starting location changes eg: DOS programs.

⇒ Load time: must generate relocatable code if memory location is not known at compile time.

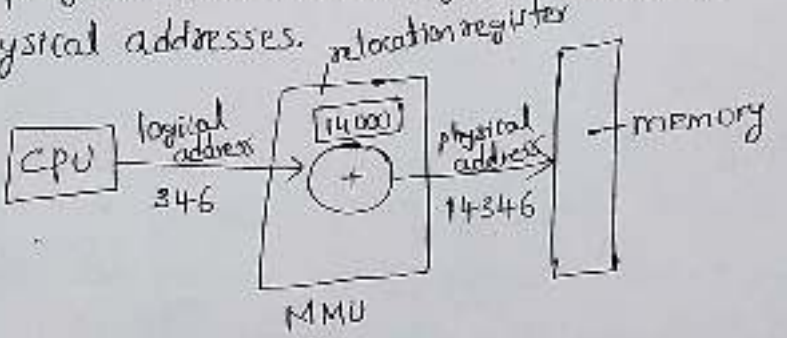
⇒ Execution time: Binding delayed until run time, if the process can be moved during its execution from one memory segment to another.

Logical versus physical address space

Logical address: generated by the CPU, also refered as virtual address.

physical address: address seen by the memory unit i.e, loaded to memory address register.

→ logical and physical addresses are the same in compile-time and load-time address-binding. logical and physical addresses differ in execution time address binding schema.
→ The run-time mapping from virtual to physical addresses is done by a hardware device called the memory management unit (mmu)
→ In mmu, the value in the relocation register is added to every address generated by a user process at the time it is sent to memory.
→ The user program deals with logical addresses, it never sees the real physical addresses.



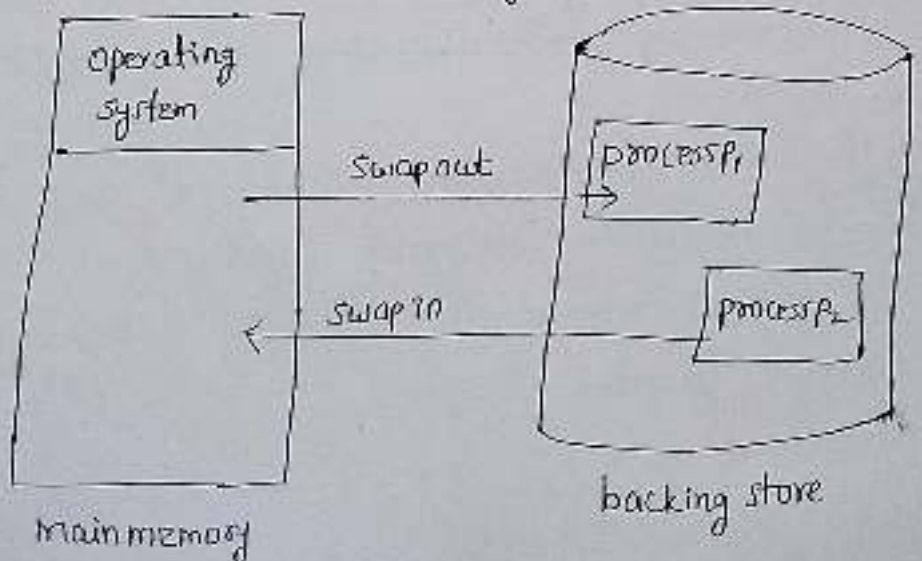
Dynamic loading:
→ All routines are kept on disk in a relocatable load formed and main program is loaded into memory and is executed.
→ Routine is not loaded until it is called
→ The relocatable linking loader is called to load the desired routine.
→ Better memory space utilization since unused routine is never loaded.
→ it is useful when large amounts of code are needed to handle frequently occurring cases.
→ No special support from the OS is required implemented through program design.

Dynamic Linking and Shared Libraries:

→ Linking postponed until execution time
→ Small piece of code, stub used to locate the appropriate memory resident library routine.
→ Stub replaces itself with the address of the routine, and executes the routine.
→ Dynamic linking is particularly useful for libraries. this system also known as Shared libraries.

Swapping:

→ A process needs to be in memory for execution but sometimes there is not enough main memory to hold all the currently active processes in a timesharing system.
→ A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution.
→ Similar to RR CPU scheduling algorithm, when a TQ expires the memory manager will swap out that process to swap another process into the memory space that has been freed.



→ backing store: it is a disk to store data that is present in memory

→ swapping can be done by priority based scheduling algorithm lower priority process is swapped out so higher priority process can be loaded and executed.

→ The swapped out process will be swapped back into the same memory space it occupied previously due to the restriction by the method of address binding.

→ The dispatcher swaps out a process in memory if there is no free memory region and swaps in the desired process from a ready queue.

→ The major part of swap time is transfer time. total transfer time is directly proportional to the amount of memory swapped.

Eg: user process is 100MB

Transfer rate of 50 MB per sec

Transfer rate = 100/50 = 2 sec [2 sec = 2*1000 m.c = 2000 ms]

swap time = transfer time + seek time (latency time)

seek time = 8 sec

swap time = 2000 + 8

= 2008 millisec

Total swap time = swapout + swap in

= 2008 + 2008

= 4016 millisec

Contiguous memory Allocation:

→ The memory is usually divided into two partitions: one for the resident operating system and one for the user processes.

→ contiguous memory allocation is a memory allocation technique

→ it allows to store the process only in a contiguous fashion. thus entire process has to be stored as a single entity at one place inside the memory. it is of two types

contiguous memory allocation Technique

- Static partitioning (or) fixed size partitioning
- Dynamic partitioning (or) variable size partitioning

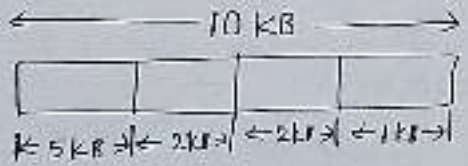
Static partitioning:

→ In static partitioning, main memory is pre divided into fixed size partitions.

→ The size of each partition is fixed and can not be changed

→ each partition is allowed to store only one process

Eg: 10 kB memory divided into fixed size partitions



→ These partitions are allocated to the processes as they arrive. The partition allocated to the arrived process depends on the algorithm followed.

Algorithm for partition allocation
- First fit Algorithm
- Best fit
- Worst fit

First fit:

→ It starts the searching the partitions serially from the starting.
→ When an empty partition that is big enough to store the process is found, it is allocated to the process.
→ obviously, the partition size has to be greater than or at least equal to the process size.

Best fit:

→ It first scans all the empty partitions.
→ next it allocate the smallest size partition to the process.
→ Best fit works best because space left after the allocation inside the partition is of very small size. internal fragmentation also least and search time is more

Worst fit:

→ it first scans all the empty partitions.
→ next it allocates the largest size partition to the process.
→ worst fit works worst, because space left after the allocation inside the partition is of very large size thus internal fragmentation is maximum.

Translating logical address into physical address:

→ cpu always generates a logical address. a physical address is needed to access the main memory.
→ The translation scheme uses two registers
  - Relocation Register
  - Limit Register

→ Relocation Register stores the base address or starting address of the process in the main memory.

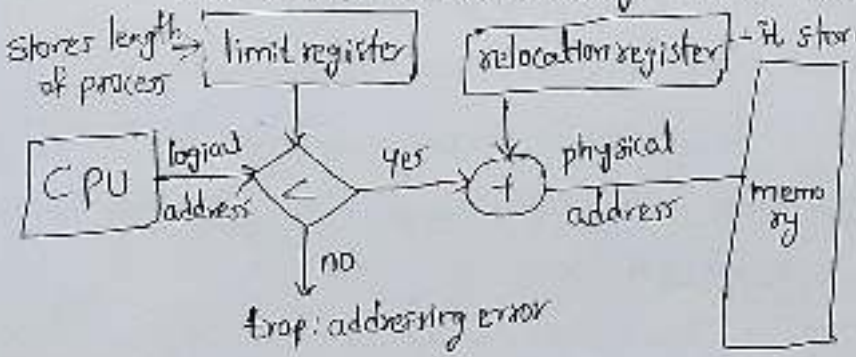
→ Limit register stores the size of or length of the process.

→ case1: [generated address >= Limit]

If address is found to be greater than or equal to the limit, a trap is generated.

→ case 2: [generated address < Limit]

The address must always lie in the range [0, limit -1]



Disadvantages of static partition:

→ It suffers from both internal fragmentation and external fragmentation.

→ It utilizes memory inefficiently.

→ There is a limitation on the size of process since process with size greater than the size of largest partition can't be stored and executed.

Fragmentation:

It may be of two types

1. internal fragmentation
2. External fragmentation

Internal fragmentation:

→ It occurs when the space is left inside the partition after allocating the partition to a process.
→ This space is called as internally fragmented space.
→ and this space cant allocated to any other process
→ This is because only static partitioning allows to store only one process in each partition.
→

External fragmentation:

→ it occurs when the total amount of empty space required to store the process is available in the main memory.
→ But because the space is not contiguous so the process can not be stored.

Segmentation: [non-contiguous Memory allocation]

→ Segmentation is a memory management technique in which, the memory is divided into the variable size parts.
→ Each part is known as segment which can be allocated to a process.
→ The details about each segment are stored in a table called as segment table. Segment table is stored in one of the segments
→ and segment table contains following information.

Base: it is base address of the segment
Limit: it is the length of the segment

→ it supports user view of memory. a logical address space is a collection of segments.

10

Translation of Logical address into physical address by segment table:

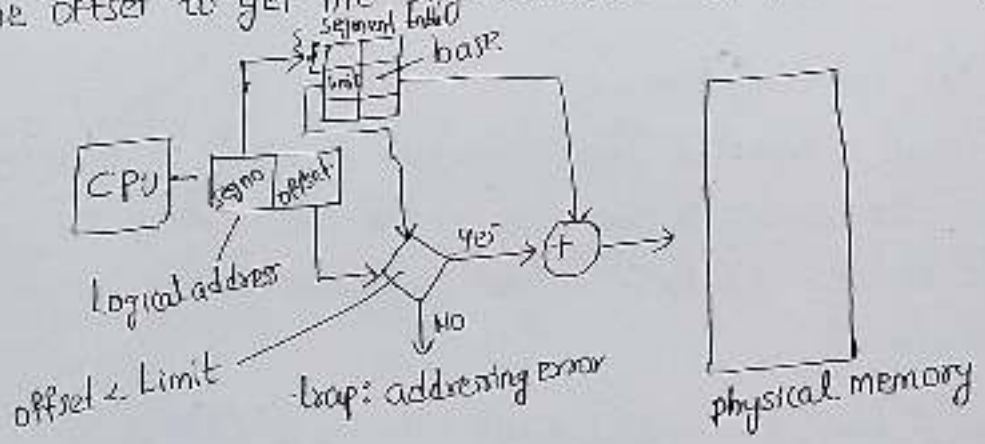
→ CPU generates a logical address which contains two parts

< segment-number, offset >

→ The segment number is mapped to the segment table. the limit of segment compared with the offset

→ If the offset is less than the limit then the address is valid, otherwise it throws an error as the address is invalid

→ In case of valid address, the base address of the segment is added to the offset to get the physical address of main memory.



Eg:

seg.no	Base	length
1	2300	14
2	90	100
3	1327	580

which of the following logical address will produce trap addressing error.

1. 1, 11 [segment no, offset]
2. 2, 100
3. 3, 425

→ segment offset must always lie in the range [0, limit-1]

→ for option 1: offset = 11, segment no = 1

The segment must lie [0, 14-1] = [0, 13]

so offset address lies between 0 to 13, no trap will be produc

physical address = 2300 + 11 [Base + offset]

= 2311

→ for option 2: [0, 99], but offset address is 100. so trap will be occur.

Paging:

→ Paging is a non-contiguous memory allocation technique.

→ To avoid external fragmentation, it allows to store parts of a single process in a non-contiguous fashion. thus, different parts of the same process can be stored at different places in the main memory.

→ Paging is a fixed size partitioning scheme. in paging, secondary memory and main memory are divided into equal fixed size partitions.

→ The partitions of secondary memory or logical memory are called as pages. the partitions of main memory or physical memory are called as frames.

→ Each process is divided into parts where size of each part is same as page size.

→ The page size is defined by the hardware. the size of the page is power of '2', varying between 512 byter and 16MB per page.

→ CPU generates a logical address consisting of two parts

1. Page Number
2. page offset

→ Page Number specifies the specific page of the process from which CPU wants to read the data.

→ Page offset specifies the specific word on the page that CPU wants to read.

Note: Logical address generated by the CPU is represented in bits

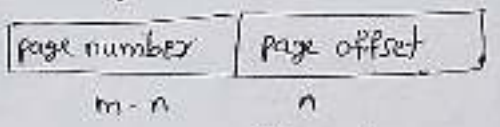
(LAS) Logical address space generated by a program is represented in words or bytes.

(PA) physical address available in main memory is represented in bits

(PAS) physical address space: represented in words or bytes. the set of all physical addresses corresponing to the logical addresses.

→ The size of the logical address space is 2^m, and page size is 2^n, then the high-order bits m-n represent the page number, and the n low-order bits represent page offset.

Eg:



$1K = 2^{10}$
$1M = 2^{20}$
$\because 1G = 2^{30}$

→ If LA (Logical address) = 31 bit, LAS = 2^{31} words
$= 2^{30} \cdot 2^1 = 2 \times 1G$
$= 2G$ words

→ LAS = 128 M words
$2^7 \times 2^{20} = 2^{27}$ words, the LA = 27 bits

→ PA = 32 bit, the PAS = 2^{32} words, PAS = $2^{20} \cdot 2^2$
$= 4 \times 1G = 4G$ words

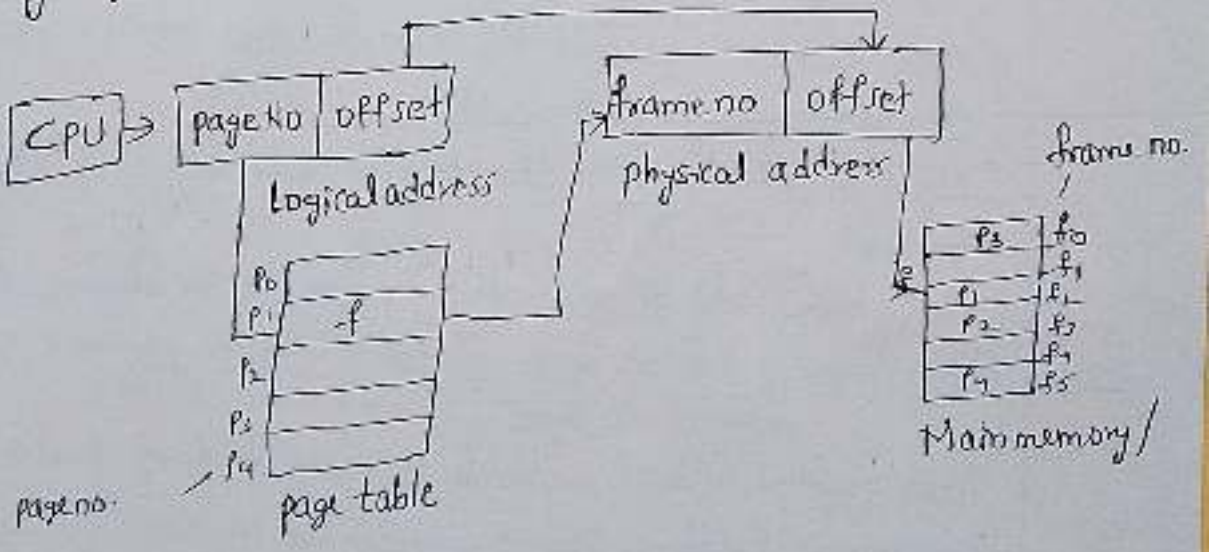
→ PAS = 16M words
$2^4 \times 2^{20}$ words, then PA = 24 bits

→ no of frames = Physical address space / frame size

→ no of pages = Logical address space / page size

→ frame offset = page offset.

Translating logical address into physical address

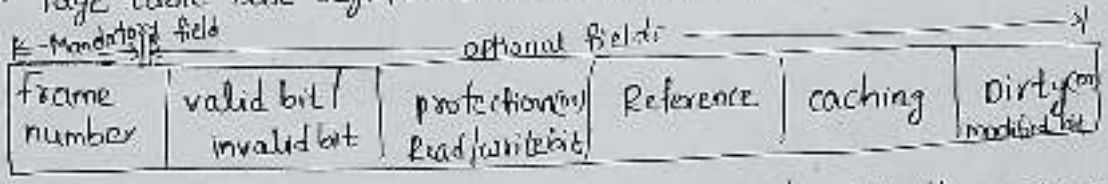


Page table:

→ page table maps the page number referenced by the cpu to the frame number where that page is stored.

→ Number of entries in page table = Number of pages in which the process is divided.

→ Page table base register (PTBR) contains the base address of page table.



Frame no: specifies the frame where the page is stored in the main memory

valid/invalid bit: if bit is present in main memory it specify by 1 otherwise set to 0

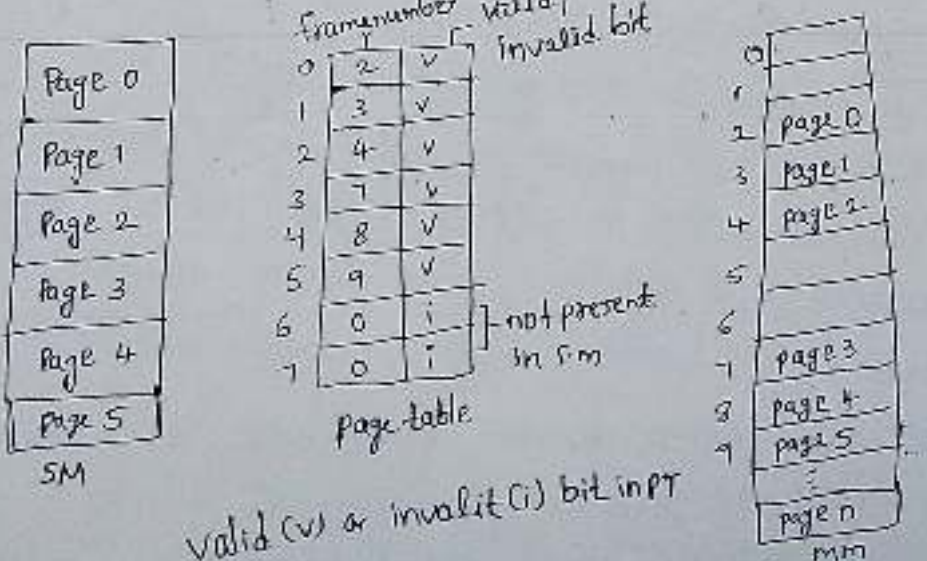
protection: To perform only read operation set bit as 0, if bit is set to 1 then both read and write operations are allowed

Reference: if the page has been referenced recently, then this bit is set to 1 otherwise set to 0. [used in LRU page replacement policy]

caching: whenever fresh or new data is required cache is disabled by setting bit as 1 otherwise set to 0

Dirty bit: if the page has been modified, then this bit is set to 1 otherwise set to 0. Dirty bit helps to avoid unnecessary writes.

Eg:



Valid (v) or invalid (i) bit in PT

→ In above example frame number 0 - indicate the page 6 and page 7 but those pages are not present in secondary memory

Page fault:

→ When a page referenced by the CPU is not found in the main memory it is called as a page fault.

→ When a page fault occurs, the required page has to be fetched from the secondary memory into the main memory.

→ Disadvantage of paging is, it increase the effective access time due to increased number of memory accesses i.e one memory access is get the frame number from the page table, another memory access is get the word from the page.

→ To reduce the effective access time we use TLB

Translation Lookaside Buffer:

→ Being a hardware, the access time of TLB is very less as compared to the main memory.

→ TLB consist of page Number and frame number.

In paging schema using TLB, the logical address generated by the CPU is translated into the physical address using following steps

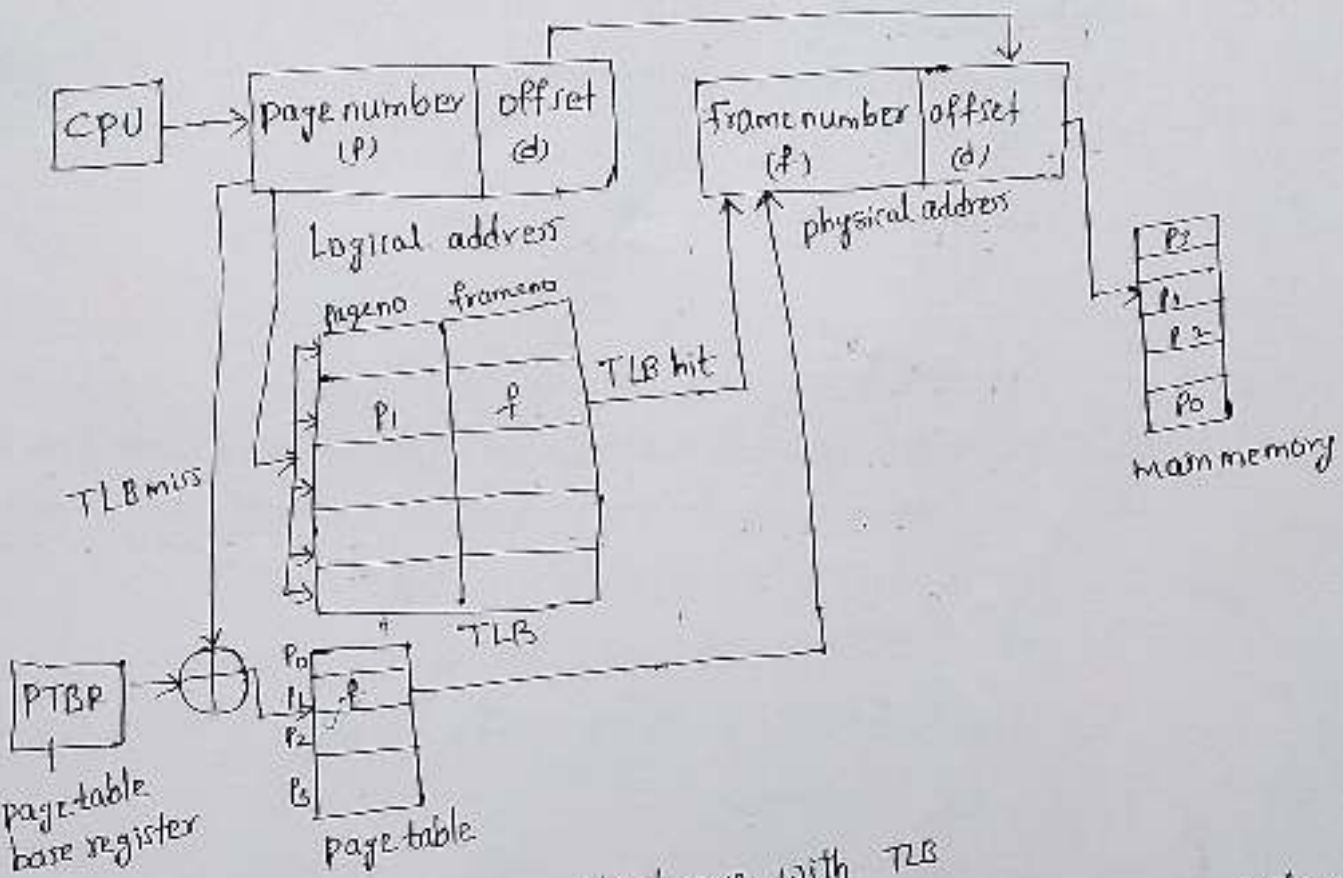
→ TLB is checked to see if it contains an entry for the reference page number. the referenced page number is compared with the TLB entries all at once.

→ if there is a TLB hit, TLB contains an entry for the referenced page number. in this case, TLB entry is used to get the corresponding frame number for the referenced page number.

→ if TLB does not contain an entry for the referenced page number, A TLB miss occurs. in this case, page table is used to get the referenced page number. Then, TLB is updated with the page number and frame number for future references.

→ After the frame number is obtained, it is combined with the page offset to generate the physical address.

→ Then, physical address is used to read the required word from the main memory.



Paging Hardware with TLB

→ To identify process TLB stors (ASIDs) address space identifiers in each TLB entry

→ Unlike page table, there exists only one TLB in the system

→ advantages of TLB is, it reduces the effective access time, only one memory access is required when TLB hit occurs.

→ The percentage of times that a particular page number is found in the TLB is called the hit ratio.

Effective Access Time:- Hit ratio of TLB × (Access time of TLB + Access time of mainmemory) + miss ratio of TLB × (Access time of TLB + 2 × Access time of main memory).

Eg: A paging scheme uses a TLB. the effective memory access takes 160 ns and a main memory access takes 100ns. what is the TLB Access time if the TLB hit ratio is 60% and there is no page fault.

16

Sol:

Effective access time = 160 ns

Main memory access time = 100 ns

TLB hit ratio 60% = 0.6

TLB miss ratio = 1 − TLB hit ratio

= 1 − 0.6

= 0.4

Let TLB access time = T ns

EAT = Hit ratio of TLB × (Access time of TLB + Access time of MM) +
Miss ratio of TLB × (Access time of TLB + 2 × Access time of

$160 = 0.6 * (T + 100) + 0.4 \times (T + 2 \times 100)$

$160 = 0.6 \times T + 0.6 \times 100 + 0.4 \times T + 0.4 \times 200$

$160 = 0.6 \times T + 60 + 0.4 \times T + 80$

$160 = T + 140$

$160 - 140 = T$

$T = 20$ ns //

Eg2: A paging scheme uses a TLB. A TLB access takes 10 ns and a main memory access takes 50 ns. What is EAT if the TLB hit ratio is 90% and there is no page fault.

A:

TLB access time = 10 ns

Main memory access time = 50 ns

TLB hit ratio = 90% = 0.9

TLB miss ratio = 1 − TLB hit ratio ⇒ 1 − 0.9 = 0.1

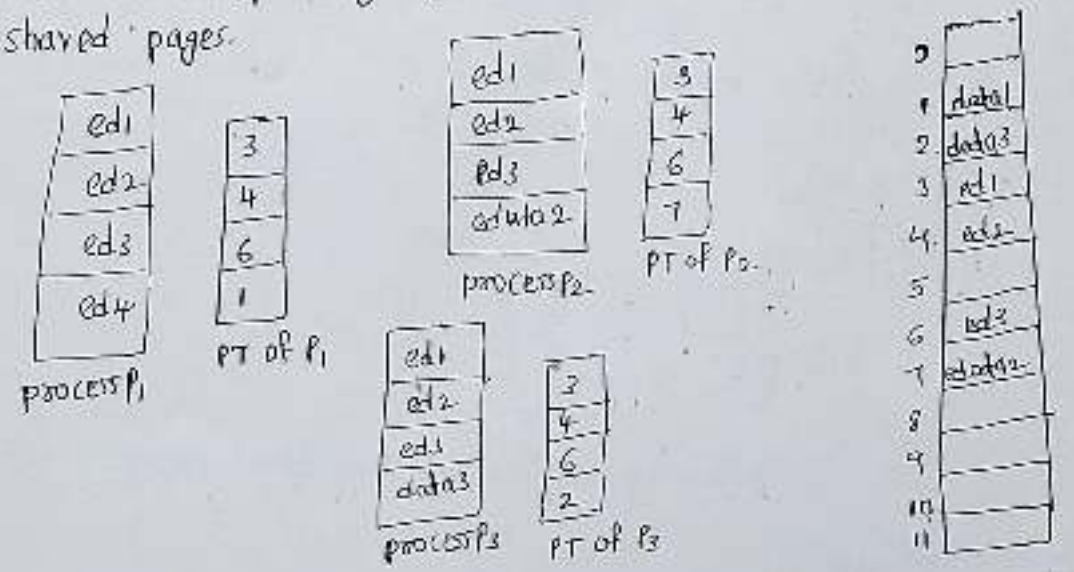
EAT = $0.9 \times (10 + 50) + 0.1 (10 + 2 \times 50)$

$= 0.9 \times 60 + 0.1 \times 110$

$= 54 + 11$

EAT = 65 ns //

→ The main advantage of paging is the possibility of sharing common code. some operating systems implement shared memory using shared pages.



Sharing of code in a paging environment

Structure of page table:

The most common techniques used for structuring the page table are

- Hierarchical paging (or multilevel paging
- Hashed page tables
- Inverted page tables

Hierarchical paging / multilevel paging:

→ If the page table might be too big to fit in a contiguous space, so we may have a hierarchy with several levels.

→ so, we break up the logical address space into multiple page tables.

In this technique we use

1. Two level page table
2. Three level page table.

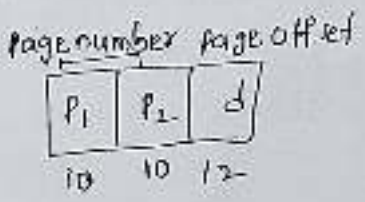
Two level page table:

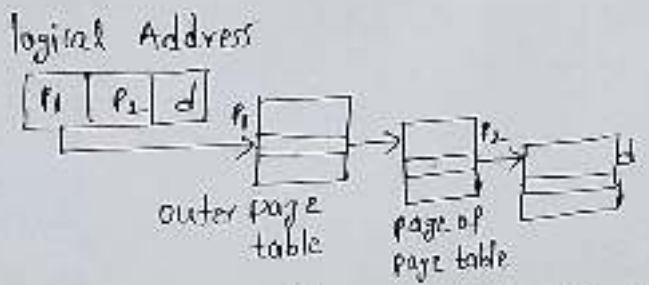
→ A logical Address (on 32-bit machine with 4K page size) is divided into a page number consisting of 20 bits
a page offset consisting of 12 bits

[∵ $4K = 2^2 \times 2^{10}$
$= 2^{12}$
= 12 bits of
32 - 12 = 20]

→ Since the page table is paged, the page number is further divided into
- A 10-bit page number
- A 10-bit page offset

Then, logical Address follows

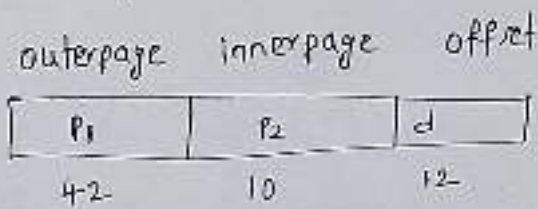


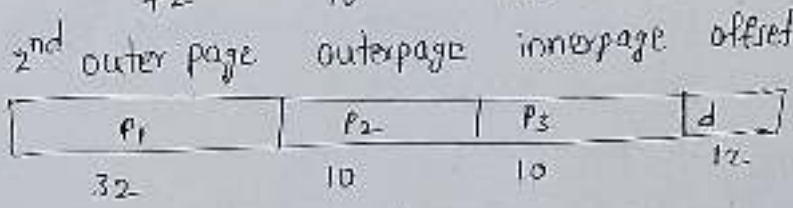


Address translation for a two level paging

Three level paging:

→ A logical address (on 64-bit machine with 4k page size) is divided into



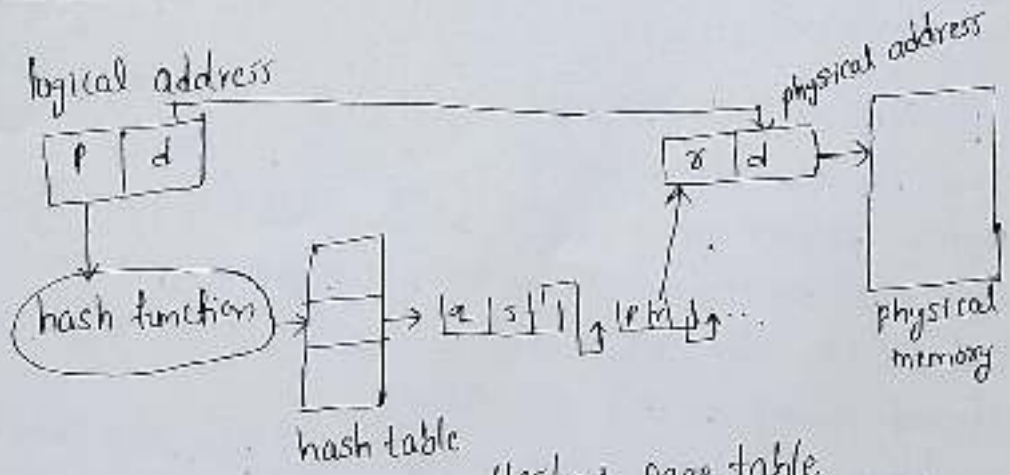


Hashed page tables:

→ it is common approach used when address space is > 32 bits

→ The virtual page number is hashed into a page table this page table contains a chain of elements or linked list elements hashing to the same location.

→ Each element consists of three fields
1. virtual page number
2. The value of the mapped page frame
3. A pointer to the next element in the linked list



Hashed page table

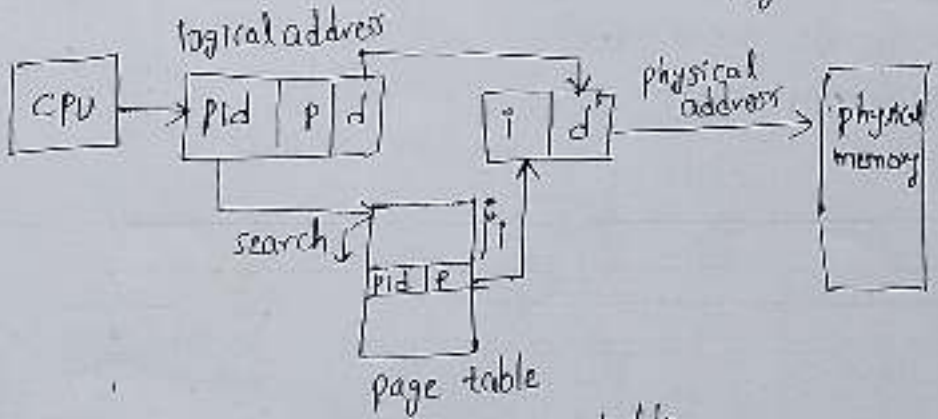
Inverted page tables:

→ The inverted page table combines a page table and a frame table into one data structure

→ One entry for each virtual page number & real page of memory.

→ Entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page.

→ Decreases memory needed to store each page table, but increases time needed to search the table when a page reference occurs.



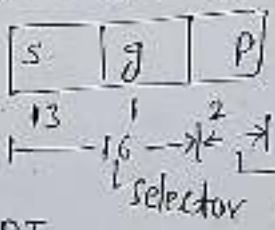
Inverted page table

→ < process-id, page-number, offset >

IA-32 Segmentation:

→ Intel pentium architecture allows a segment to be 4 GB large, and maximum number of segments per process is 16K.
→ The logical address of a process is divided into two partitions
→ the first partition consist of up to 8k segments that are private to that process. the second partition consist of 8k segments that are shared among all the processes.
→ First partition information kept in the local descriptor table (LDT)
→ Second partition information kept in the global descriptor table (GDT)
→ Each entry in LDT or GDT consist of 8-byte segment descriptor.

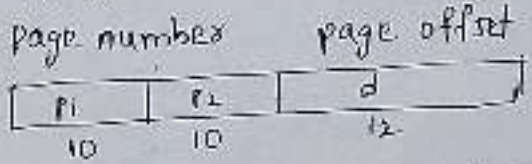
logical address represented as

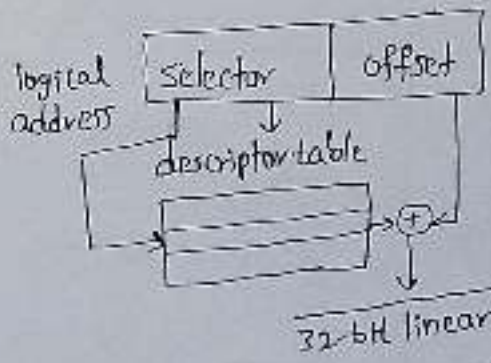


s- segment number
g - segment in the GDT or LDT
p - protection

IA-32 paging:

→ The pentium architecture allows a page size of either 4KB or 4 MB for 4-KB pages pentium uses a two-level paging scheme in which the division of the 32-bit linear address follows





Intel pentium segmentati

Copy - on - Write (COW):

→ fork() system call used for creating child process, again if you call the fork() system call it will create the duplicate of its parents and creating parent address space for child but it is unnecessary.

→ Instead of this we use a technique COW (copy-on-write)

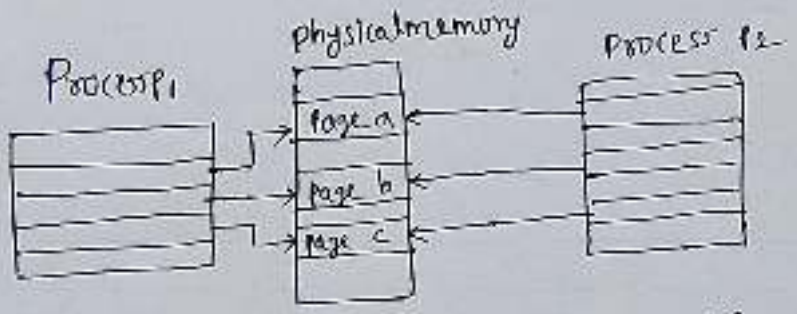
→ COW allows the parent and child processes initially to share the same pages these pages marked as COW.

→ By using these technique, only modified pages are copied and all unmodified pages can be shared by the parent and child processes.

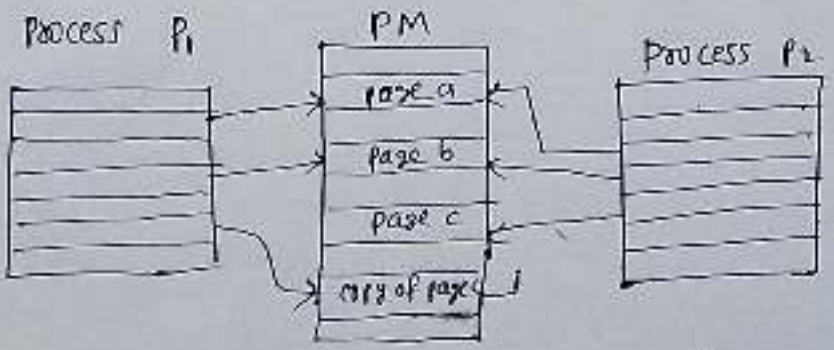
→ This technique is common in operating systems, including windows XP, Linux, & Solaris.

→ copied page can be allocated to free page

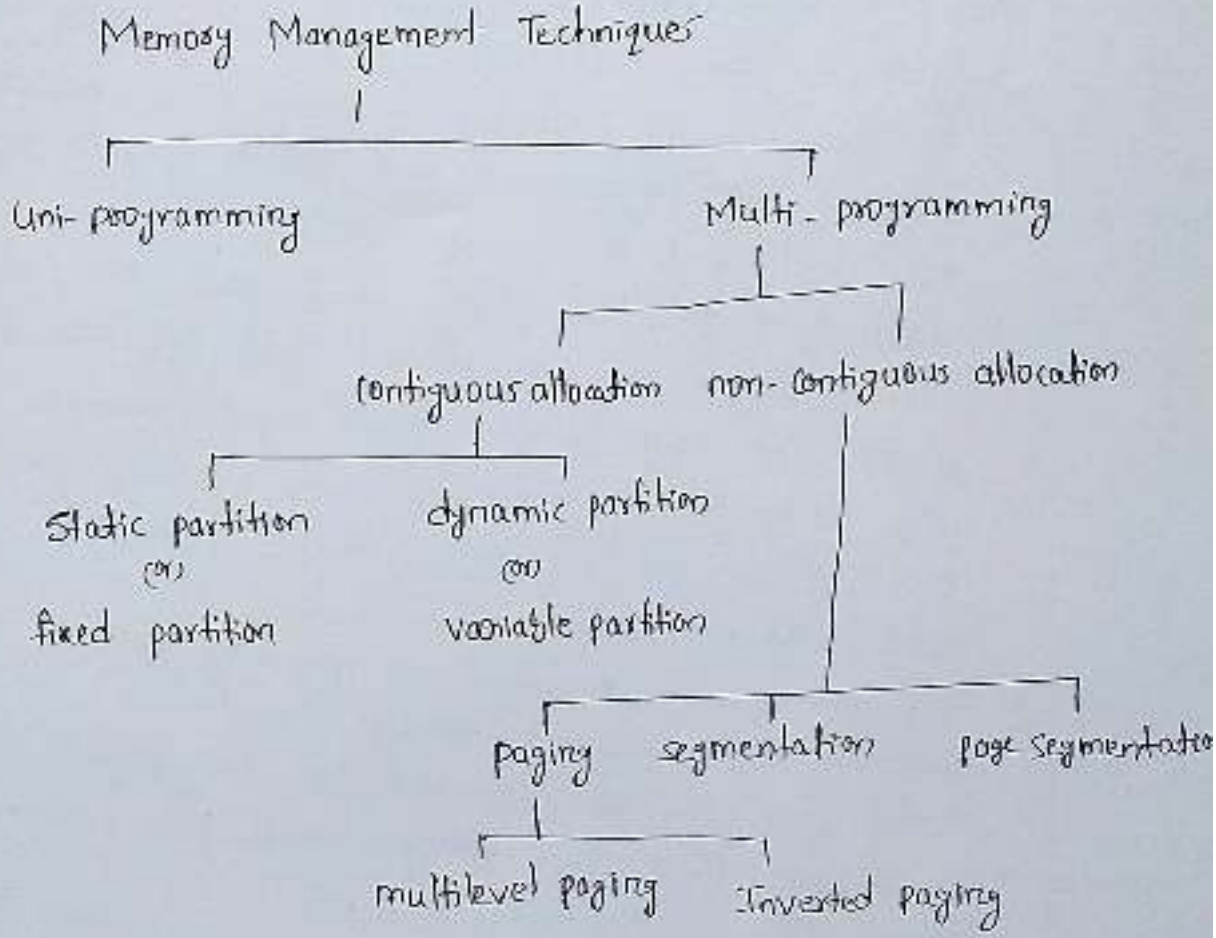
→ OS allocate these pages using a technique known as 'zero-fill-on-demand'



Before process P_1 modifies page c



After process P_1 modifies page c



Virtual Memory Management:

→ Virtual Memory is a storage schema that provides user an illusion of having a very big main memory.

→ In this case user can load the bigger size processes than the available main memory.

→ Instead of loading one big process in the main memory the OS loads the different parts of more than one process in the main memory.

→ By that, the degree of multiprogramming will be increased.

→ It is useful for users where physical memory is small.

In real time, most processes never need all their pages at once, for following reasons:

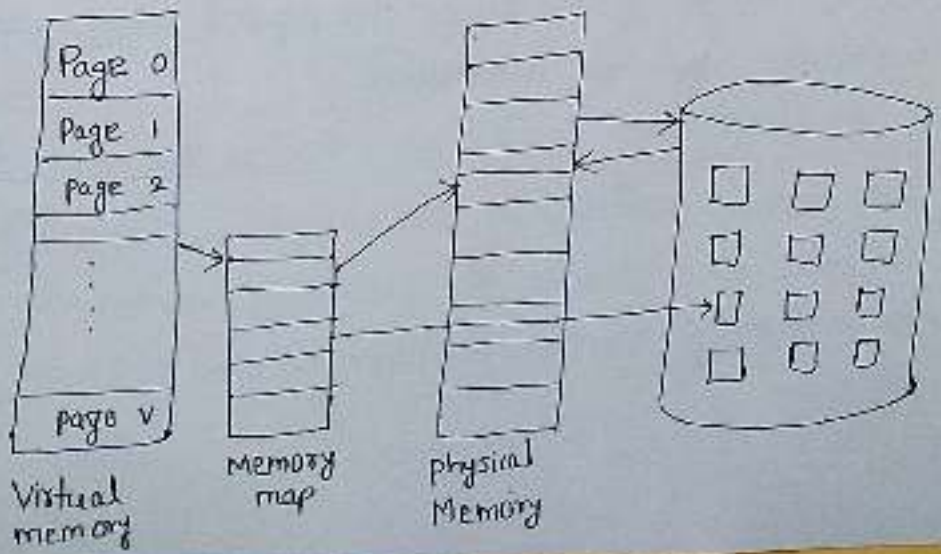
- Error handling code is not needed unless that specific error occurs some of which are quite rare.
- Arrays are often over-sized for worst-case, only a small fraction of the arrays are used in real time.

Advantages:

→ Large programs can be written, as virtual space available is huge compared to physical memory.

→ Less I/O required, leads to faster and easy swapping of processes.

→ More physical memory available, as programs are stored on virtual memory, so they occupy very less space on actual physical memory.



Demand paging:

→ Demand paging is a type of swapping done in virtual memory systems. In Demand paging, the data is not copied from the disk to the RAM until they are needed or being demanded by some program.

→ The data will not be copied when the data is already available on the memory this is called lazy swapper because only the demanded pages of memory are being swapped from the secondary storage (disk space) to the main memory.

→ In contrast during pure swapping, all the memory for a process is swapped from secondary storage to main memory during the process startup.

Basic concepts or working of Demand paging:

→ The demand paging working is based on a page table implementation. The page table maps logical memory to physical memory. The page table uses a bitwise operator to mark if a page is valid or invalid.

→ A valid page is one that currently resides in main memory an invalid page can be defined as the one that currently resides in secondary memory.

When a process tries to access a page, the following will happen.

i) attempt to access the page, the page is valid, page processing instruction continues as normal.

ii) if the page is an invalid one, then a page fault trap occurs.

iii) The memory reference is checked to determine if it is a valid reference to a location on secondary memory or not if not, the process is terminated. Otherwise, the required page is paged in.

Procedure for handling page fault

→ We check an internal table for this process to determine whether the reference was a valid or an invalid memory access
→ if the reference was invalid, we terminate the process. If it was valid, but we have not yet brought in that page, we now page it in.
→ We find a free frame, schedule a disk operation to read the desired page into the newly allocated frame
→ When the disk read is complete, we modify the internal table kept with the process and the page table to indicate that the page is now in memory
→ We restart the instruction that was interrupted by the trap The process can now access the page.

Performence of Demand pagiry:

effective access time with page fault is

EAT = (1-P) xma * + P x page fault time

probability of page fault p: $0 \leq p \leq 1$

Eg: page fault service time of 8 ms, memory access time 200 ns find EAT, in nanoseconds

EAT = (1-P) x ma + P x page fault time

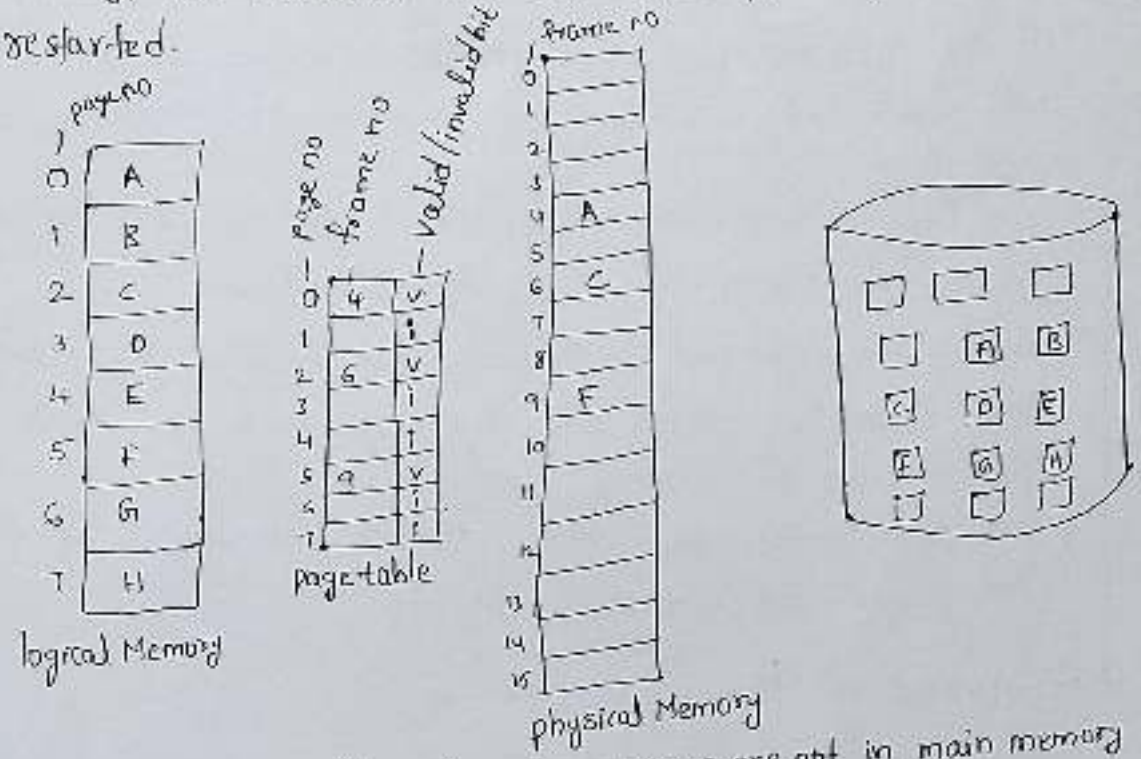
= (1-P) x 200 + P x 8,000,000 (∵ 1ms = 1,000,000 ns)

= 200 - 200P + P x 8,000,000

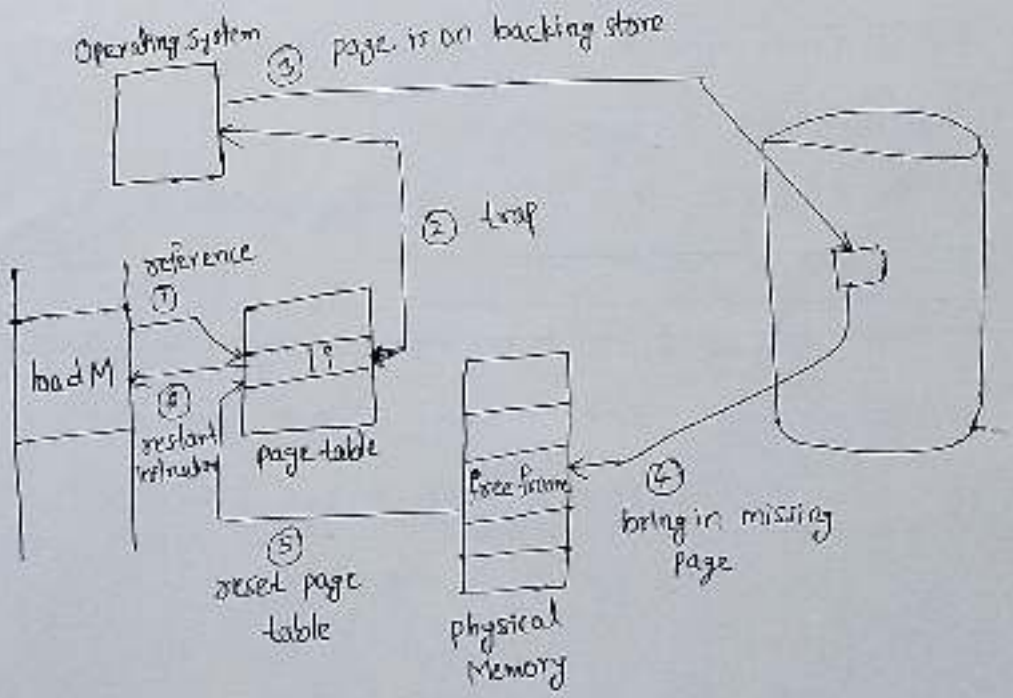
= 200 + 7,999,800 P

(iv) Now the disk operation to read the desired page into main memory is scheduled.

(v) Finally, the instruction that was interrupted by the OS trap is restarted.



Page table when some pages are not in main memory



Steps in Handling a page fault

Page Replacement:

Page replacement is a process of swapping out an existing page from the frame of a main memory and replacing it with an the required page.

Page replacement is required when, all the frames of main memory are already occupied. Thus, a page has to be replaced to create a hole or room for the required page.

Page Replacement Algorithms:

Page Replacement algorithms help to decide which page must be swapped out from the main memory to create a room for the incoming page.

1. FIFO Page Replacement Algorithm
2. LIFO Page Replacement Algorithm
3. LRU Page Replacement Algorithm
4. Optimal page Replacement Algorithm
5. Random page Replacement Algorithm
6. LFU
7. MFU

→ A good page replacement algorithm is one that minimizes the number of page faults.

Page replacement takes the following approach

→ If no frame is free, find the location of the desired page on the disk.

→ Find a free frame: if there is a free frame, use it; if there is no free frame, use a page replacement algorithm to select a victim frame. Write the victim frame to the disk. Change the page and frame tables accordingly.

→ Read the desired page into the newly freed frame, change the page and frame tables.

→ Restart the user process.

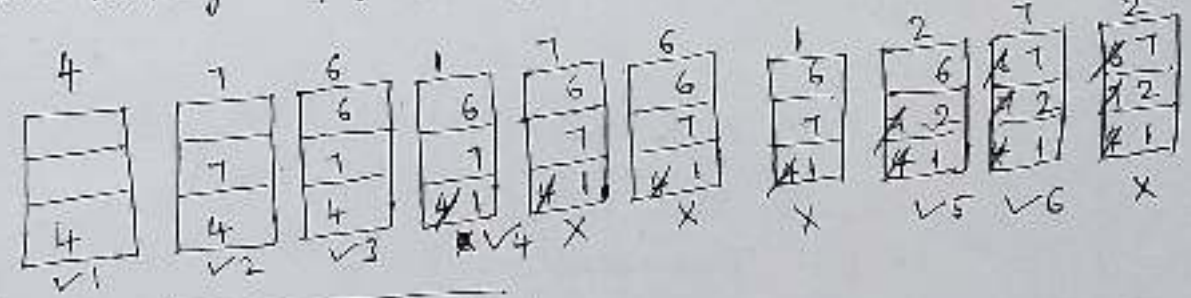
FIFO Page Replacement Algorithm:

→ It works based on principle of first in first out

→ it replaces the oldest page that has been present in the main memory for the longest time.

→ it is implemented by keeping track of all the pages in a queue. we replace the page at the head of the queue when a page is brought into memory, we insert it at the tail of the queue.

Eg1: reference string = 4, 7, 6, 1, 7, 6, 1, 2, 7, 2, find total no. of page faults, hit ratio, miss ratio. Assume that all the frames are initially empty and system uses 3 page frames for storing process



Queue: 7 2 1 6 7 4

Total number of page faults = 6

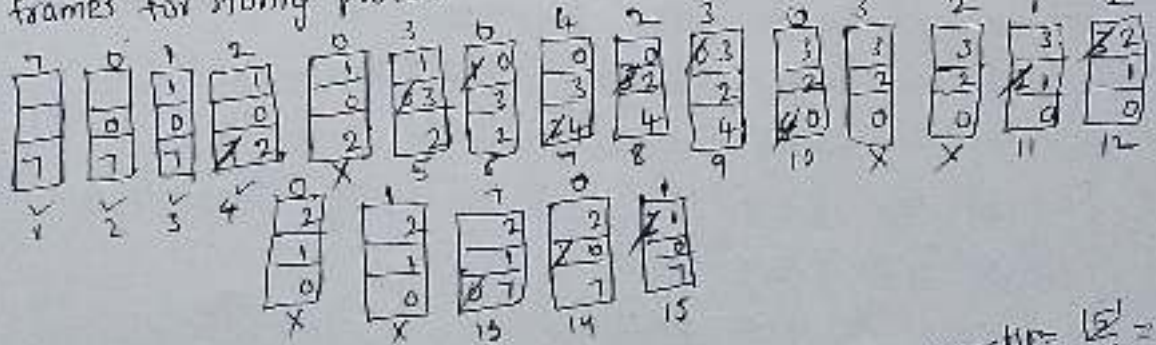
hit ratio = 1 - miss ratio

= 1 - 0.6 [page fault or page misses = total no. of page misses / total no. of reference = 6/10 = 0.6]

= 0.4

∴ hit ratio = 0.4, miss ratio = 0.6

Eg2: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1 = reference string, 3 page frames for storing process.



miss ratio = 15/20 = 0.75

no. of page faults = 15

hit ratio = 0.8

⇒ disadvantage of FIFO Page replacement algorithm is, it suffer from belady's Anomaly.

Belady's Anomaly:

Belady's Anomaly is the phenomenon of increasing the number of page faults on increasing the number of frames in main memory.

⇒ an algorithm suffers from belady's anomaly if and only if does not follow stack property.

⇒ Algorithm that follow stack property are called as stack based Algorithms. these algorithms do not suffer from belady's anomaly.

Note:
1. FIFO, Random page Replacement and second chance algorithms suffer from belady's Anomaly.
2. LRU, optimal page Replacement Algorithms follows the stack based algorithms, hence they do not suffer from belady's Anomaly.

Optimal page Replacement Algorithm:

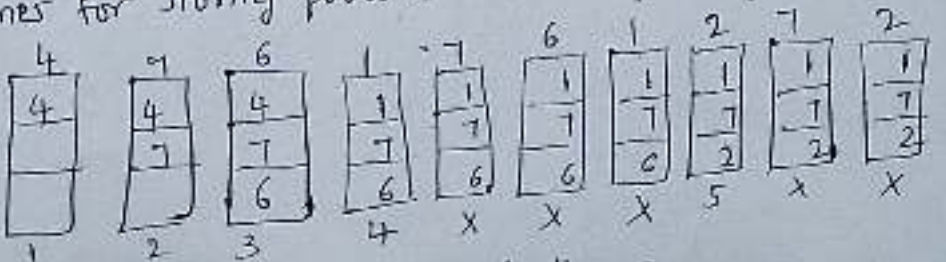
⇒ it replaces the page that will not be refered by the cpu in future for the longest time.

⇒ it is practically impossible to implement this algorithm

⇒ This is because the pages that will not be used in future for the longest time cannot be predicted.

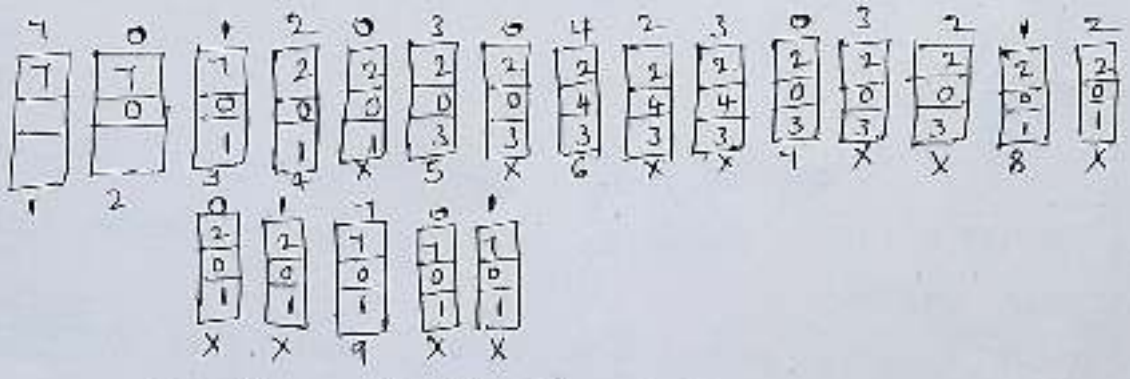
⇒ However, it is the best known algorithm and gives the least number of page faults.

Eg:- Reference string: 4, 7, 6, 1, 7, 6, 1, 2, 7, 2, it uses 3 page frames for storing process.



page faults = 5

Eg2: Reference string: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1



no. of page faults = 9

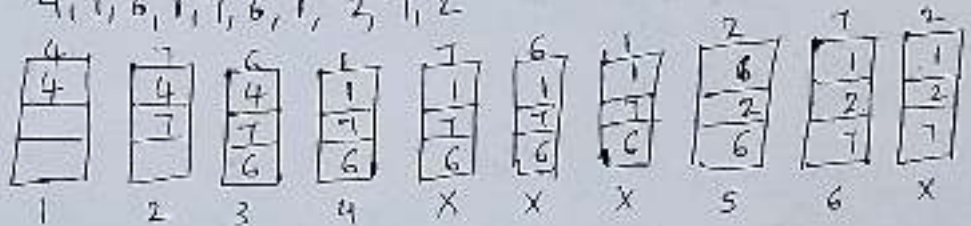
hit ratio = 1 - 0.55 = 0.45

miss ratio = $\frac{11}{20}$ = 0.55

LRU Page Replacement:

→ it works based on principle of Least Recently used
→ it replaces the page that has not been refered by the CPU for the longest time.

Eg: 4, 7, 6, 1, 7, 6, 1, 2, 7, 2

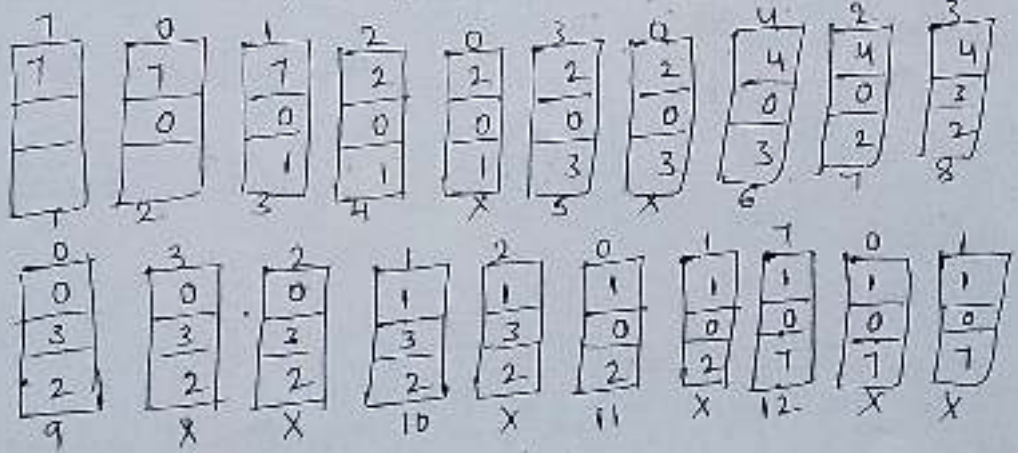


no. of page faults = 6

hit ratio = 0.4

miss ratio = 0.6

Eg2: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1



no. of page faults = 12

hit ratio = 0.40

miss ratio = $\frac{12}{20}$ = 0.51

16

Counting based page Replacement:

1. Least frequently used (LFU) page replacement Algorithm
2. Most frequently used (MFU) page replacement Algorithm

→ LFU requires that the page with the smallest count be placed
→ MFU is based on the argument that the page with the smallest count was probably just brought in and has yet to be used.
→ implementation of LFU & MFU is expensive.

Allocation of frames:

→ Frame allocation algorithms are used if you have multiple processes it helps decide how many frames to allocate to each process.

There are various constraints to the strategies for the allocation of frames.

→ We cannot allocate more than the total number of available frames.
→ At least a minimum number of frames should be allocated to each process. this constraint is supported by two reasons. the first reason is, the number of allocated frames are less then it increase page fault ratio, decreasing the performance of the execution of the process.
→ second reason is, there should be enough frames to hold all the different pages that any single instruction can reference.

Frame allocation algorithms:

two algorithms are commonly used to allocate frames to a process.

- Equal Allocation
- proportional allocation

Equal allocation:

→ In a system with x frames and y processes, each process gets equal number of frames. i.e. x/y

Eg: if the system has 48 frames and 9 processes, each process will get 5 frames. i.e, $^{5}48/_{9}$, 3 frames which are not allocated to any process can be used as a free-frame buffer pool.

→ disadvantage is allocation of a large number of frames to a small process will eventually lead to the wastage of a large number of allocated unused frames

→

Proportional allocation:

→ frames are allocated to each process according to the process size.

→ for a process P_i of size S_i, the number of allocated frames is

$$a_i = (S_i / S) * m$$

m - number of frames in the system

S - sum of the sizes of all the processes

Eg:- A system with 62 frames, if there is a process of 10KB and another process of 127 KB then 1st process will be allocated

$$\left(\frac{10}{137}\right) * 62 = 4 \text{ frames} \qquad [\because S = 10 + 127 = 137]$$

2nd process $\left(\frac{127}{137}\right) * 62 = 57$ frames

→ advantage is all the processes share the available frames according to their needs, rather than equally.

Global Vs Local Allocation:

→ The number of frames allocated to a process can also dynamically change depending on whether we have used global replacement or local replacement for replacing pages in case of page fault

17

Local replacement:

→ When a process needs a page which is not in the memory, it can bring in the new page and allocate it a frame from its own set of allocated frames only.

→ advantage: The pages in memory for a particular process and the page fault ratio is affected by the paging behavior of only that process

→ Disadvantage: A low priority process may hinder a high priority process by not making available to the high priority process its frames.

Global replacement:

→ When a process needs a page which is not in the memory, it can bring in the new page and allocate it a frame from the set of all frames, even if that frame is currently allocated to some other process i.e, one process can take a frame from another.

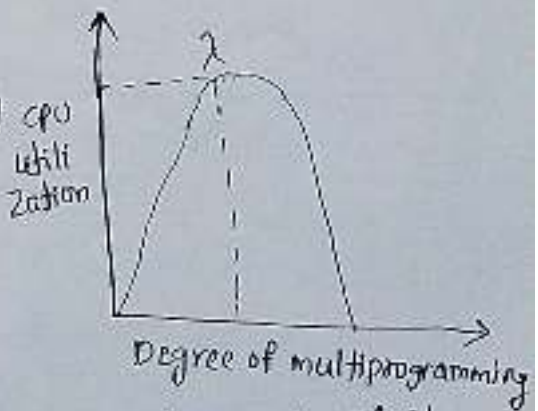
→ advantage: increase throughput

→ Disadvantage: The page fault ratio of a process can not be controlled by the process itself.

Thrashing:

→ At any given time, only few pages of any process are in main memory and therefore more processes can be maintained in memory.

→ Unused pages are not swapped in and out of memory.



→ When the OS brings one page in, it must throw another out. If it throws out a page just before it is used, then it will just have to get that page again almost immediately. Too much of this leads to a condition called Thrashing.

→ The system spends most of its time swapping pages rather than executing instructions.

In above diagram, initial degree of multi programming upto some extent of point, the cpu utilization is very high and the system resources of are utilized 100%. But if we further increase the degree of multi programming the cpu utilization will drastically fall down and the system will spent more time only in the page replacement and the time taken to complete the execution of the process will increase. this situation in the system is called as thrashing.

Causes of Thrashing:

1. High degree of multiprogramming
2. Lacks of frames.

If a process has less number of frames then less pages of that process will be able to reside in memory this leads to thrashing so sufficient frames are allocated to each process to prevent thrashing.

Recovery of Thrashing:

→ 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 scheduler to suspend some of the processes so that we can recover the system from thrashing.

Virtual Memory in windows:

Windows XP implements virtual memory using demand paging with clustering. clustering handles page faults by bringing in not only the faulting page but also several pages following the faulting page. It works working-set maximum if sufficient memory is available. If not it works working set minimum.

UNIT-IV

Storage Management - file system- concept of a file, system calls for file operations - open(), read(), write(), close(), seek() unlink(), Access methods - Directory and Disk structure, file system Mounting, file sharing, Protection

File system implementation - File System Structure, file system implementation, Directory file system implementation, Allocation methods, free-space management, efficiency and performence mass storage structure - overview of mass storage structure, Disk structure, Disk attachment, Disk scheduling, Disk management, swap space management

File System:

→ Computers can store information on various storage media, such as magnetic disks, magnetic tapes, and optical disks. so that the computer system will be convenient to use.

→ File is a collection of related information that is recorded on secondary storage. or file is a collection of logically related entities. from user's perspective a file is the smallest allotment of logical secondary storage.

→ A file is a sequence of bits, bytes, lines or records.

→ different types of information stored in a file. source programs, object programs, executable programs, numeric data, text, payroll records, graphic images, sound recordings, and so on.

File Attributes:

Name: The symbolic file name is the only information kept in human readable form.

identifier: identifies the file within the file system, it is the non-human-readable name for the file.

Type: This information is needed for systems that support different types of files.

Location: This information is pointer to a device and to the location of the file on that device.

Size: The current size of the file and possibly the maximum allowed size are included in this attribute.

protection: It determines who can do reading, writing, executing and soon

Time, date, and user identification: This information may be kept for Creation, last modification, and last use. These data can be useful for protection, security, and usage monitoring.

File operations:

File is an abstract data type. operating system can provide basic six operations.

- Creating a file:
- Writing a file
- Reading a file
- Repositioning within a file
- Deleting a file.
- Truncating a file

System calls for file operations:

Basically there are total 5 types of I/O system calls

1. create()
2. open()
3. close()
4. read()
5. write()

Create(): used to create a new empty file.

indicate permission of new file

↑

→ it return -1 when an error occur otherwise it return first unused file descriptor.

open(): used to open the file for reading, writing or both.

syntax:
int open(const char* path, int flags [, int mode]);

where path - path to file which you want to use, if absolute path begin with '/' [same dir of file], use relative path i.e only file name with extension [same dir of file].

flags - O_RDONLY - read only
O_WRONLY - write only
O_RDWR - read and write
O_CREAT - Create file if doesn't exist

close(): To close file which pointed by file descriptor

syntax: int close(int fd); ← file descriptor

it return - 0 - on success
it return - '-1' - on error.

read(): From the file indicated by the file descriptor fd, the read() function reads count bytes of input into the memory area indicated by buffer.

syntax: Size_t read(int fd, void* buf, size_t cnt); ← length of buffer
↑ buffer to read data

Returns - how many bytes were actually read
return - 0 on reaching end of file
return - '+1' on error or signal interrupt
returns

write():

Syntax: size_t write (int fd, void * buf, size_t cnt);

fd: file descriptor
buf: buffer to write data
cnt: length of buffer

Returns - how many bytes were actually written.
Return - 0 on reaching end of file
return - 1 on error or signal interrupt

lseek(): lseek is a system call that is used to change the location of the read/write pointer of a file descriptor. The location can be set either in absolute or relative terms.

Syntax: lseek (int fieldes, off_t offset, int whence);

the fd of the pointer that is going to be moved
The offset of the pointer
The method in which offset is interpreted

it returns the offset of the pointer from the beginning of the file.

unlink(): deletes a name from the filesystem. if that name was last link to a file and no processes have the file open the file is deleted and the space it was using is made available for reuse.

It return zero on success, otherwise -1.

Syntax: #include <unistd.h>
int unlink (const char * pathname);

Note: link() - Creates a new link to an existing file.

Access Methods:

When file is used, information is read and entered into computer memory and there are several ways to access this information of the file. Some system provide only one access method for files. other system support many access methods and choosing the right one for a particular application is a major design problem.

There are three way to access a file into computer system

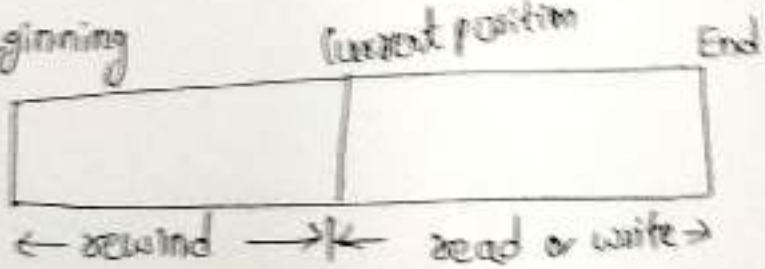
- Sequential Access
- Direct Access
- Index Sequential Access

Sequential Access:

→ It is Simplest access method. Data is accessed one record right after another record in an order. For example, editor and compiler usually access the file one after another.

→ When we use read command, it move ahead pointer by one using readnext.

→ When we use write command, it will allocate memory and move the pointer to the end of the file by using writenext



→ The Disadvantage it provide poor performences.

Direct Access(or) relative access:

→ In DA, a file is made up of fixed. length logical records that allow program to read and write records in no particular order

→ It based on the disk model of a file since disk allows random access to any file block.

→ In DA, the file is viewed as numbered sequence of block or record. thus, we may read block 14 then block 59 and then we can write block 17. There is no restriction on the order of reading and writing for direct access file.

→ A block number provided by the user to the operating system is normally a relative block number. the 1st relative block number is 0 and then 1 and so on.

Index Sequential Access:

→ It is built on top of sequential access.

→ it control the pointer by using index.

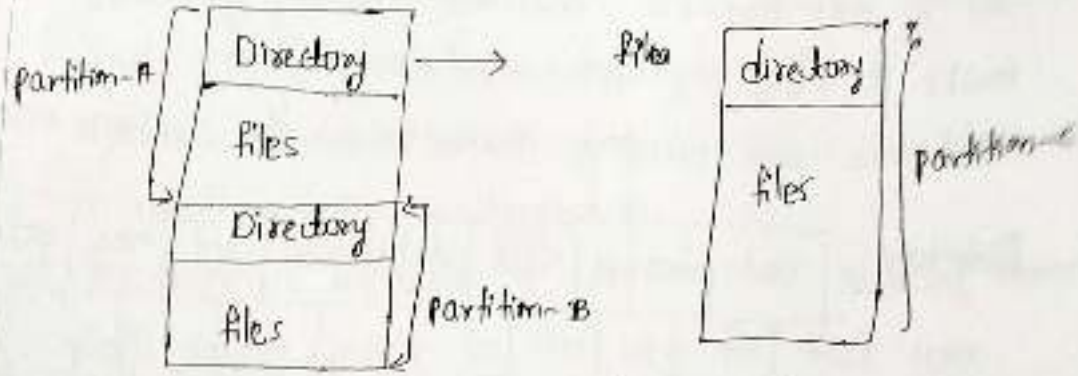
→ To find a record in the file, we first search the index and then by the help of pointer we access the file directly.

Directory and disk structure:

Directory can be defined as the listing of the related files on the disk. The directory may store some or the entire file attributes.

To get the benfit of different file systems on the different operating systems, A hard disk can be divided into the number of partitions of different sizes. The partitions are also called volumes or mini disks.

Each partition must have at least one directory in which all the files of the partition can be listed.



file system organization

Every directory supports a number of common operations on the file.

- file creation
- search for a file
- Delete a file
- List a Directory
- Rename a file
- Traverse the file system

There are several logical structures of directory, these are given below

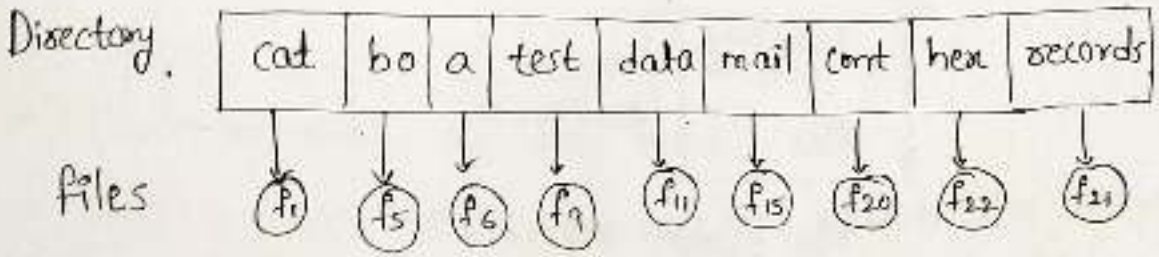
1. single level directory
2. Two level directory
3. Tree structured directory
4. Acyclic graph directory
5. General graph directory

Single - level directory:

→ It is a simplest directory structure.

→ In it all files are contained in same directory which make it easy to support and understand.

→ Each file and directory must have the unique name.



Single - level directory

Advantages:

→ implementation is very easy because of single level

→ if files are smaller in size, searching will faster

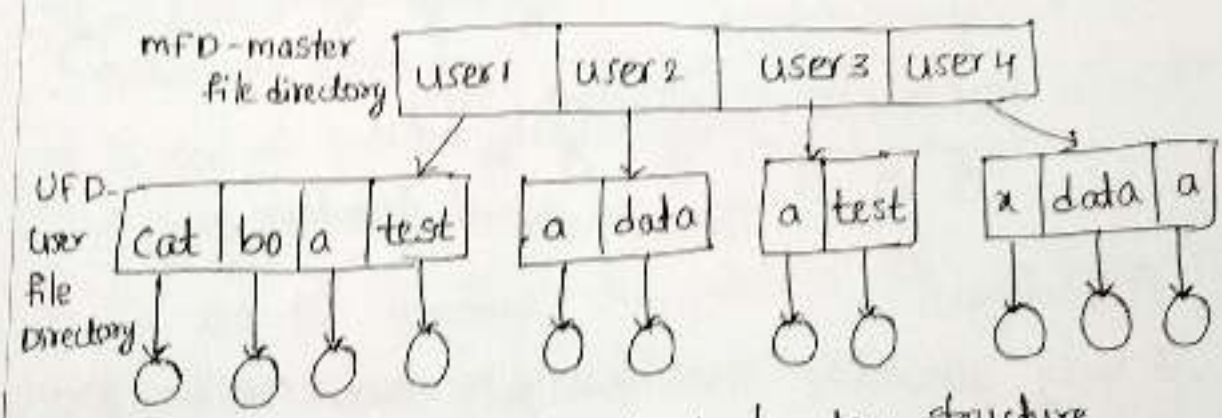
→ The file operations like file creation, searching, deletion, updating are very easy in single - level.

Disadvantages:

- If two files having same name collision will occur
- Searching will become time taking if directory will large
- We can not group same type of files.

Two-level Directory:

- In single-level directory, it leads to confusion of files names among different users the solution to this problem is to create a separate directory for each user.
- In two-level, each user has there own user file directory (UFD). The UFD's has similar structures, but each list only the files of single user.
- System's master files directory (MFD) is searches whenever a new user id = s logged in.
- The MFD is indexed by username or account number, and each entry points to the UFD for that user.



Two-level directory structure

advantages: → we can give full path like username/dir/file

→ Different users can have same directory as well as filenam

→ searching of files is very easy because of path.

Disadvantage:

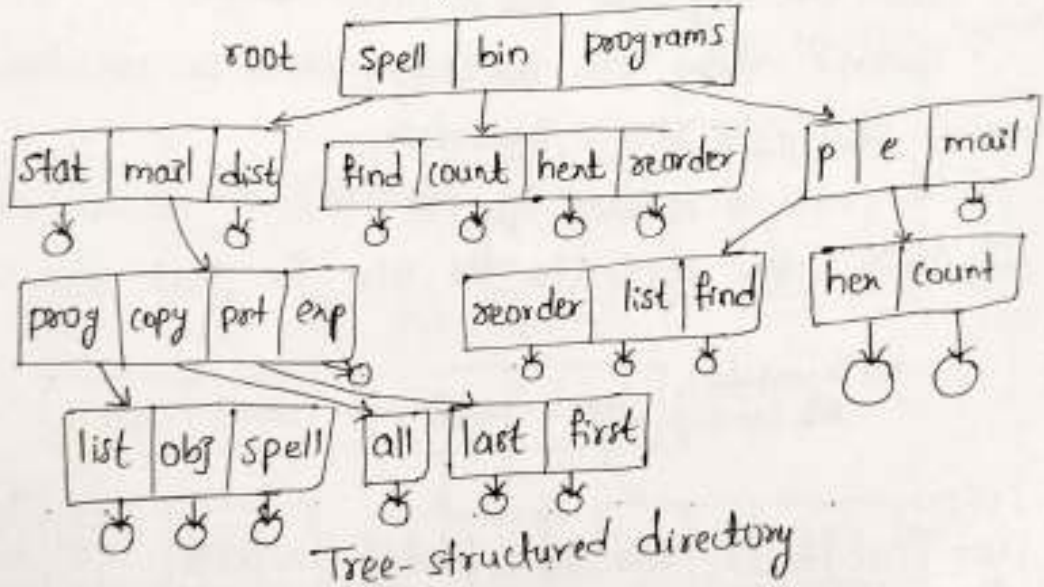
→ A user is not allowed to share files with other users.

→ Still it not very scalable, two files of the same type cannot be grouped together in the same user.

Tree- structured directory:

→ A tree structure is the most common directory structure The tree has a root directory, and every file in the system have a unique path.

→ The natural generalization is to extend the directory structure to a tree of arbitary height. this generalization allows the user to create there own subdirectories and to organize on their files accordingly.



Tree- structured directory

Advantages:

→ very generalize, since full path name can be given

→ very scalable, the probability of name collision is less

→ Searching becomes very easy, we can use both absolute path as well as relative.

Disadvantages:

→ We cannot share files.
→ It is inefficient, because accessing a file may go under multiple directories.
→ Every file does not fit into the hierarchical model, files may be saved into multiple directories.

Acyclic graph directory:

→ It is a graph with no cycle and allows to share subdirectories and files the same file or sub directory may in two different directories.
→ It is a natural generalization of the tree structured directory.
→ it is used in the situation like when two programmers are working on a joint project and they need to access files.

Advantages:
→ We can share files
→ Searching is easy due to different paths

Disadvantages:

→ We share the files via linking, in case of deleting it may create the problem.
→ In case of softlink, after deleting the file we left with a dangling pointer
→ In case of hardlink, to delete a file we have to delete all the reference associated with it.

General graph directory:

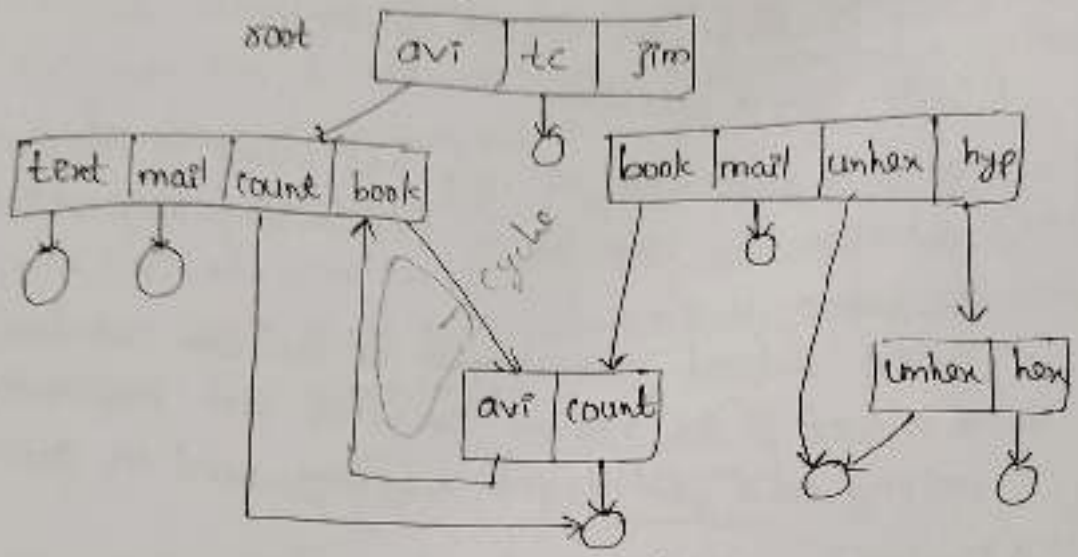
→ It allow the cycles within a directory structure where multiple directories can be derived from more than one parent directory.
→ To calculate the total size or space complex in this directory.

Advantages:

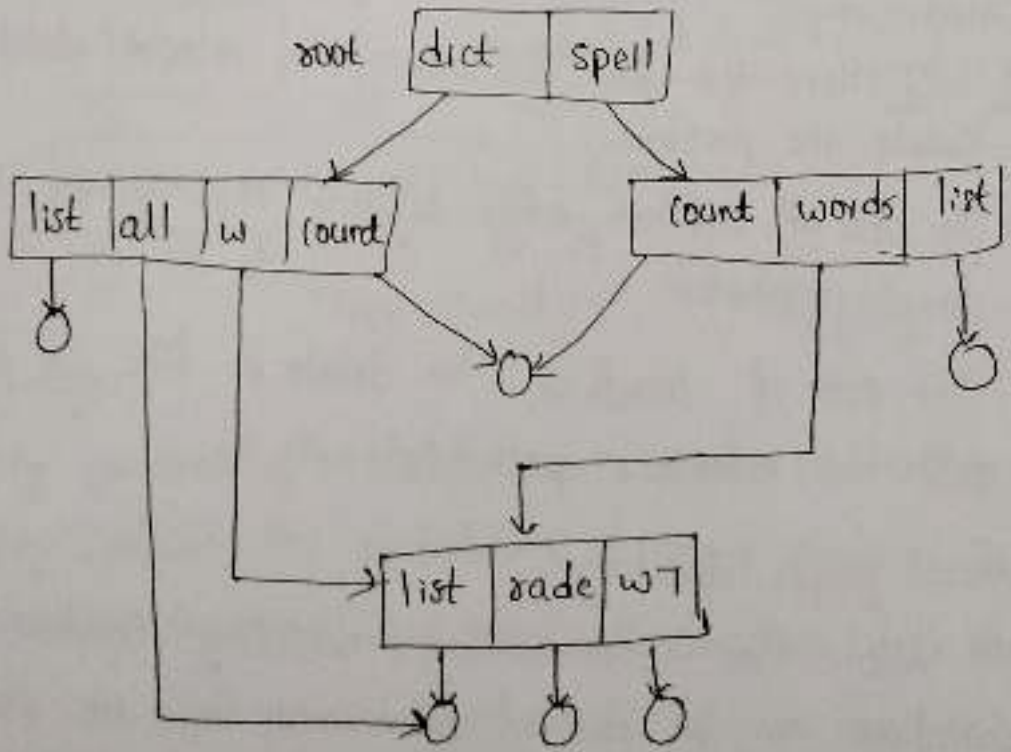
- It allows cycles
- it is more flexible than other directories structure.

Disadvantages:

- It is more costly than others
- it needs garbage collection.



General graph directory.



Acyclic-graph directory

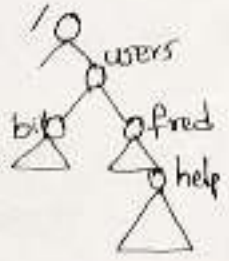
File system mounting:

→ Mounting is a process by which the operating system makes files and directories on a storage device such as hard drive, CD-ROM, or network share that make availability for users to access through the computer's file system.

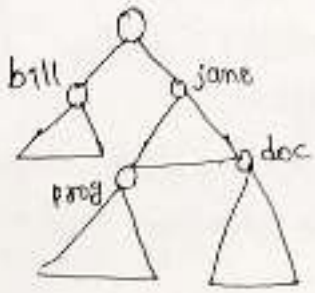
→ A file system must be mounted before it can be available to processes on the system.

→ The mount procedure is straight forward The operating system is given the name of the device and the mount point.

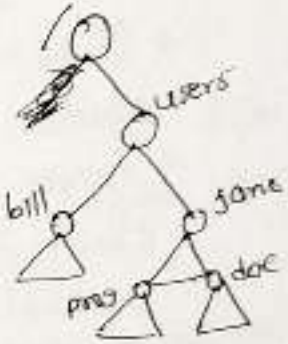
→ The operating system verifies that the device contains a valid file system.



Existing system



Unmounted volume



mount point

→ Windows operating systems automatically discover all device and mount all located file system at boot time.

→ In unix mount commands are explicit.

File sharing:

→ file sharing is very desirable for users who want to collaborate and to reduce the effort required to archieve a computing goal.

⇒ on distributed systems, files may be shared across a network

→ Network file system (NFS) is a common distributed file sharing method. sharing may be done through a protection scheme.

→ Multiple users, user IDs identify users, allowing permissions and protections to be per user. Group IDs allow users to be in groups, permitting group access rights.

→ In Remote file systems, Networking allows the sharing of resources spread across all the world the following method that are use to share file.

- manually via programs like FTP
- automatically, using distributed file systems
- Semi automatically via the world wide web.

→ client-server model allows clients to mount remote file systems from servers.

→ server can serve multiple clients. NFS is standard UNIX client-server file sharing protocal. CIFS is standard windows protocal.

→ Standard operating system file calls are translated into remote calls.

→ In failure modes, Local system can fail for a variety of reasons, including failure of the disk containing the file system, corruption of the directory structure or other disk management information.

→ Remote file systems have more failure modes, due to network failure, server failure.

→ Recovery from failure can involve state information about status of each remote request.

Protection:

→ When information is stored in a computer system, we want to keep it safe from physical damage and improper access.

→ protection can be provided in many ways: for a single user laptop systems, we might provide protection by locking the computer in a desk drawer or file cabinet.

Protection mechanism:

→ It provide controlled access by limiting the types of file access that can be made. Access is permitted or denied depending on several factors.

Read: Read from the file

Write: write or rewrite the file

Execute: Load the file into memory and execute it

append: Write new information at the end of the file.

Delete: Delete the file and free its space for possible reuse.

List: List the name and attributes of the file.

→ The most common approach to the protection problem is to make access dependent on the identity of the user.

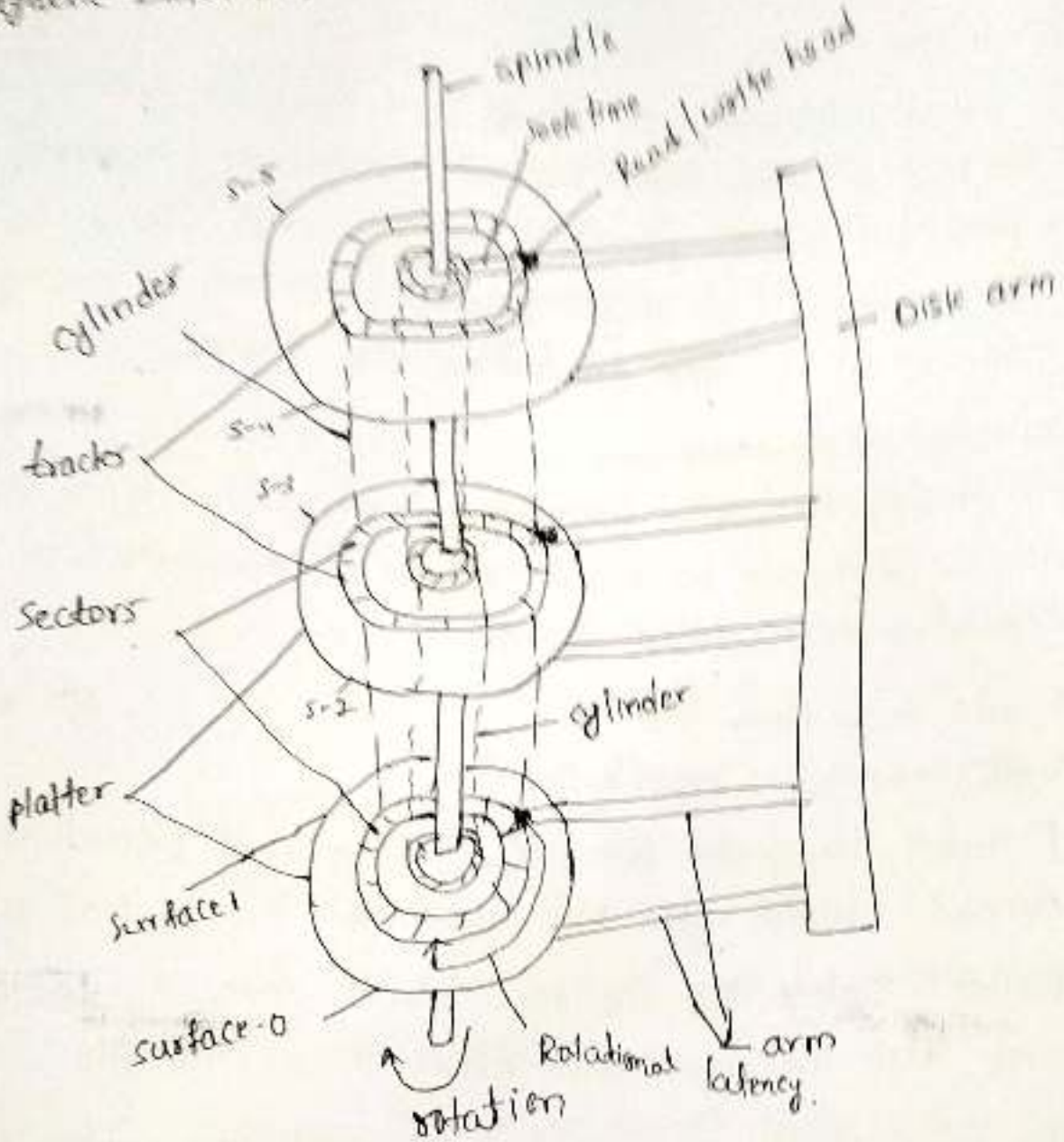
the main problem with access lists is their length if we want to allow everyone to read a file, we must list all users with read access. To overcome this problem system recognize user connection with each file.

Owner: The user who created the file is the owner.

Group: A set of users who are sharing the file and need similar access is a group or work group.

Universe: All other users in the system constitute the univers

Magnetic Disk : Diagram



Moving Head disk mechanism

File system structure:

Ex Hard disks have two important properties that make them suitable for secondary storage of files in file systems.

→ Blocks of data can be rewritten in place, and they are directly accessing any block of data ✓

Disks offer the massive amount of secondary storage where a file system can be maintained. They have two characteristics which make them a suitable medium for storing various files

→ A disk can be used to rewrite in place, it is possible to read a chunk from the disk modify the chunk and write it back there in the same place.

→ A disk can access directly any given block of data it contains. Hence, it is easy to access any file either in sequence or at random and switching from one single file to another need only to move the read write head and wait for the disk to rotate to that specific location.

• Disks are usually accessed in physical blocks, rather than a byte at a time. Block sizes may range from 512 bytes to 4K or larger.

→ File system organize storage on disk drivers, and can be viewed as a layered design

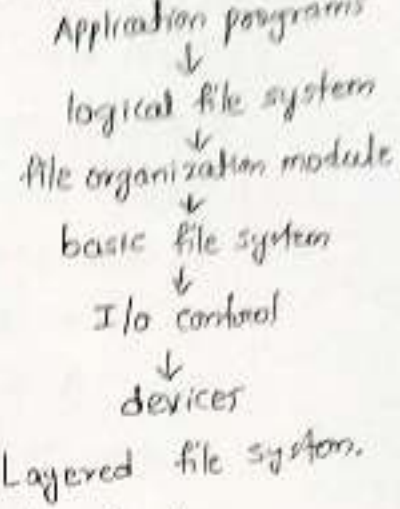
At the lower layer are the physical devices, consisting of the magnetic media, motors & controls, and the electronics connected to them and controlling them.

I/o control consist of device drivers, special software programs which communicate with the devices by reading and writing special codes directly to and from memory addresses corresponding to the controller cards register.

The basic file system level works directly with the device drivers in terms of retrieving and storing raw blocks of data without any consideration for what is in each block. Depending on the system, blocks may be referred to with a single block number, or with head sector cylinder combinations.

The file organization module knows about files and their logical blocks, and how they map to physical blocks on the disk. The logical file system deals with all of the meta data associated with a file.

The layered approach to file systems means that much of the code can be used in different file systems, and only certain layers need to be filesystem specific.



Layered file system.

File system implementation:

File systems store several important data structures on the disk.

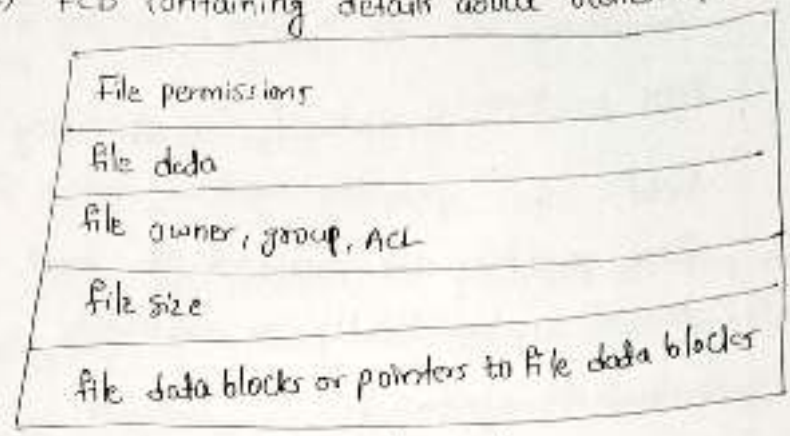
→ A boot control block can contain information needed by the system to boot an operating system from that volume. if the disk does not contain an operating system, this block can be empty. in UFS, it is called boot block, in NTFS, it is the partition boot sector.

→ A volume control block contains volume details, such as the number of blocks in the partition, the size of the blocks, a free block count and free-block pointers. In UFS, this is called a superblock, in NTFS, it is stored in the master file table.

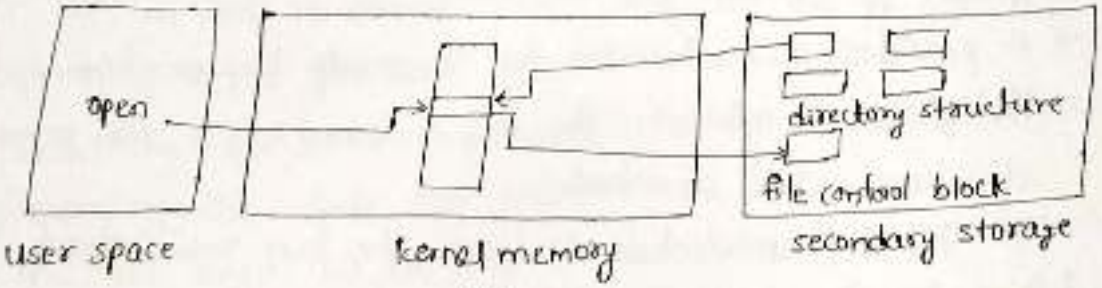
[UFS- Unix file system, NTFS - new Technology file system]

→ A directory structure is used to organize the files, in UFS this includes file names and associated inode numbers. In NTFS, it is stored in the master file table.

File control Block(FCB) FCB containing details about ownership, size, permissions, dates etc.

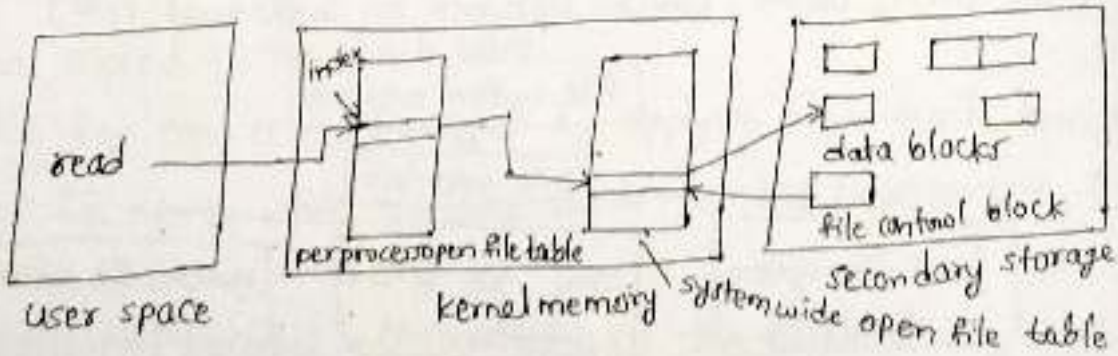


File control block



file open

→ above figure referes to opening a file plus buffer hold data blocks from secondary storage.



file read

→ above figure referes to reading a file as per process open file tables containing a pointer to the system open file table as well as some other information.

Partitioning and mounting:

→ A disk can be divided into multiple partitions, or a partition can span multiple disks. each partition can either is raw, containing no file system, containg a file system

Root partition: It contains the operating system, other partitions can hold other operating systems, other file system, or be raw, Root partition are mounted at boot time. other partitions can mount automatically or manually.

virtual file system (VFS):

→ It separates the file system generic operations from their implementatio by defining a clean VFS interface.

→ It provides a mechanism for uniquely representing a file throughout a network. the VFS is based on a file-representation structure called a vnode.

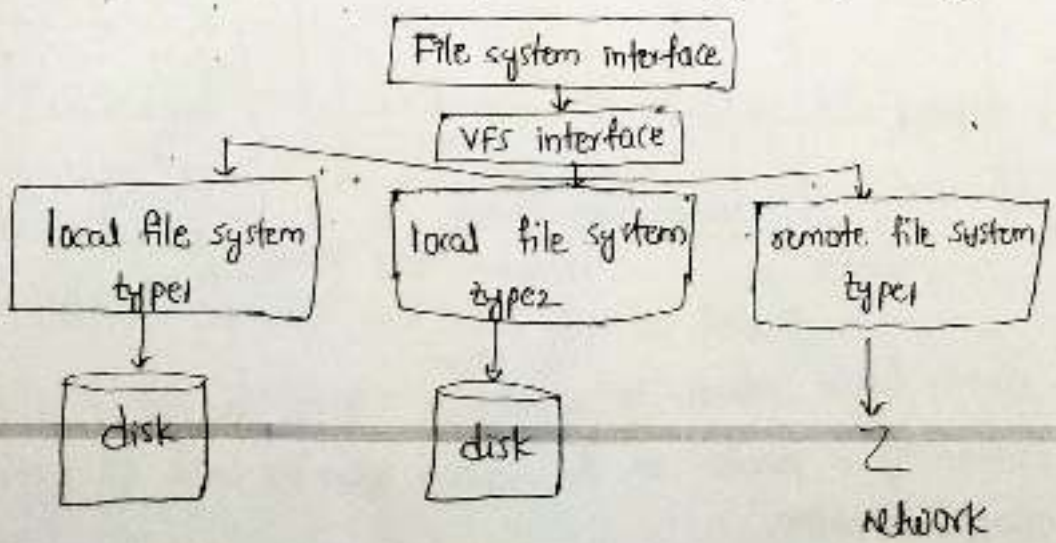
The VFS architecture in Linux the four main object types defined by the linux VFS are

→ The inode object, which represents an individual file

→ The file object, which represents an open file

→ The superblock object, which represents an entire file system.

→ The dentry object, which represents an individual entry.



virtual file system.

Directory implementation:

The Directory implementation algorithms are classified according to the data structure they are using. the selection of an appropriate directory implementation algorithm may significantly affect the performence of the system.

1. Linear List 2. hash table

Linear List:

→ In Lineart List, each file contains the pointers to the data blocks which are assigned to it and the next file in the directory.

→ When a new file is created, then the entire list is checked whether the new file name is matching to a existing file name or not in case, it doesn't exist, the file can be created at the beginning or at the end.

→ Searching for a unique name is a big concern because traversing the whole list takes time.

→ The list needs to be traversed in case of every operation (creation, del, updating etc) on the files, so the file system becomes inefficient.

Hash table:

→ A key-value pair for each file in the directory gets generated and stored in the hash table.

→ The key can be determined by applying the hash function on the file name while the key points to the corresponding file stored in the directory.

→ searching becomes efficient due to the fact i.e, only hash table entries are checked using the key.

Allocation methods / File Allocation methods:

The allocation methods define how the files are stored in the disk blocks. there are 3 main disk space or allocation methods.

- Contiguous Allocation
- Linked Allocation
- Indexed Allocation

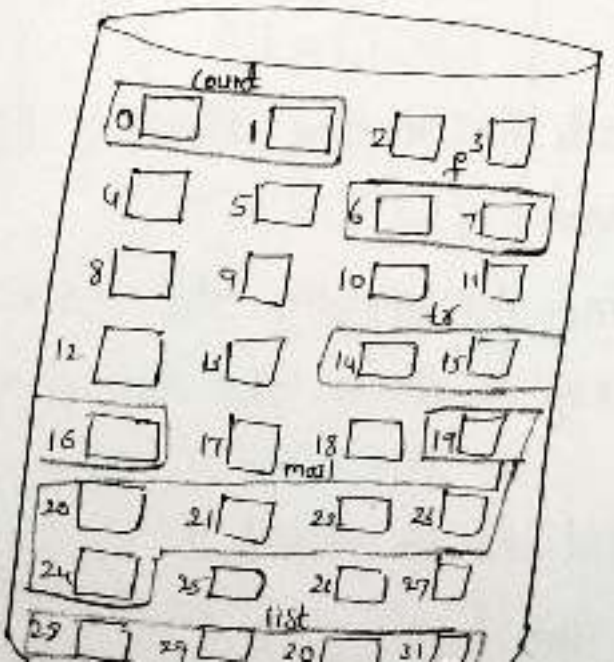
The main aim of file allocation methods is to provide efficient disk space utilization, fast access to the file blocks.

Contiguous Allocation:

→ Each file occupies a contiguous set of blocks on the disk eg: if a file requires 'n' blocks and starting location 'b', then it occupies blocks b, b+1, b+2 ... b+n-1.

→ so, it required address of starting block and length of the allocated portion.

→ starting block - 19, length - 6 so it occupies 19, 20, 21, 22, 23, 24 blocks.



Directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Advantage:

→ There is flexible to increase the file size.

→ It doesn't suffer from external fragmentation, better utilization in terms of memory.

Disadvantage:

→ To access the each block it need time, it makes linked allocation slower

→ it doesn't support random or direct access

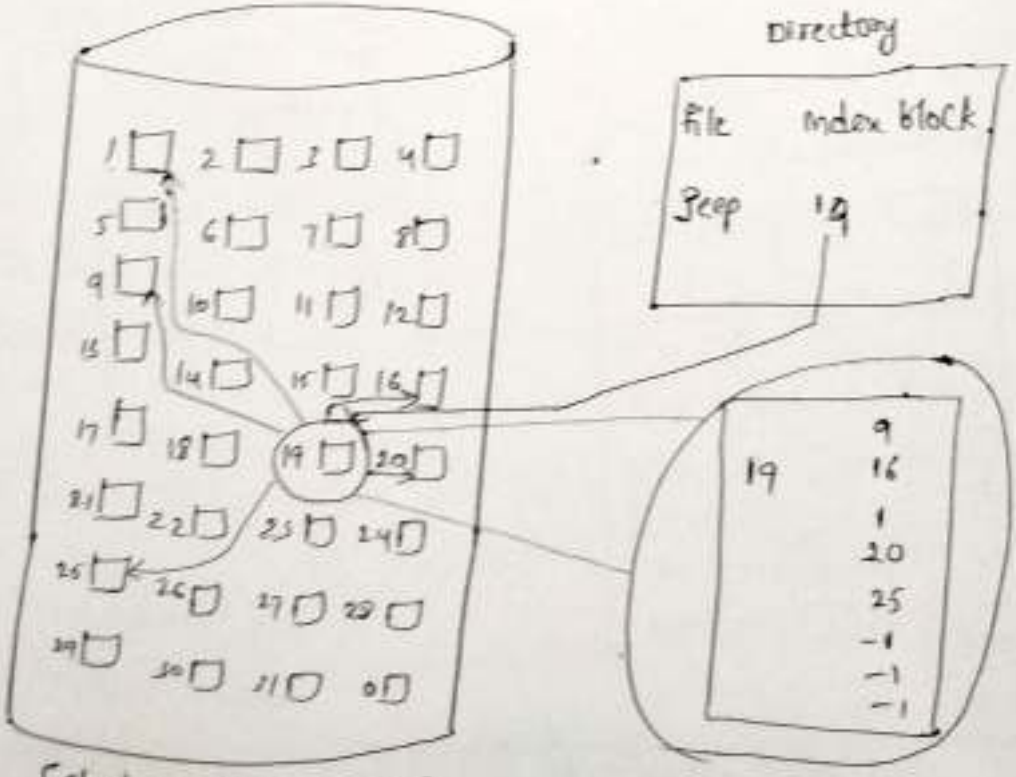
→ pointers required in LLA

Indexed Allocation:

→ It contains a special block known as index block. and index block contains the pointer to all the blocks occupied by a file

→ Each file has its own index block

→ the ith entry in the index block contains the disk address of of the ith file block.



Indexed Allocation of disk space

Advantages:

→ Both Direct and sequential allocation supported by this. for direct access, the address of kth block of the file which starts at block b can easily obtained as (b+k).

→ It extremely fast.

Disadvantages:

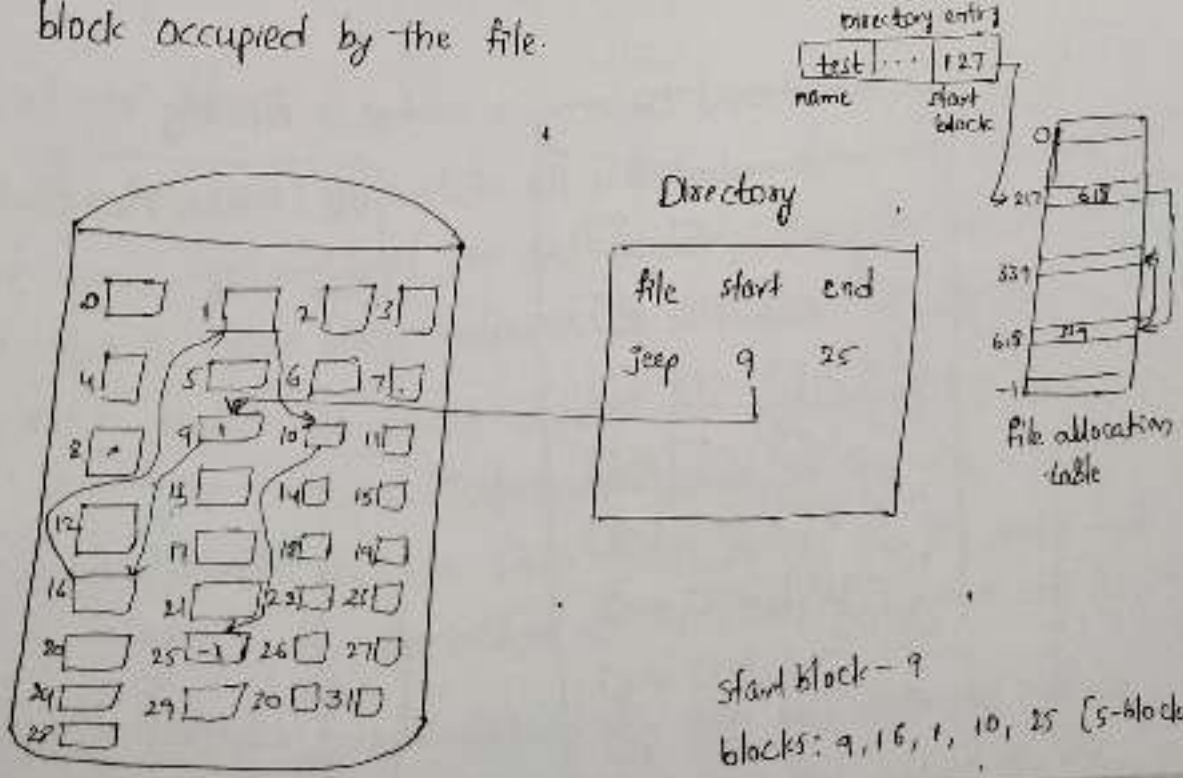
→ It suffers from both internal and external fragmentation. this makes it inefficient in terms of memory utilization.

→ Increasing file size is difficult because it depends on the availability of contiguous memory at a particular instance.

Linked List Allocation:

→ In LLA, each file is a linked list of disk blocks which need not be contiguous.

→ The disk entry contains a pointer to the starting and the ending file block. each block contains a pointer to the next block occupied by the file.



Linked Allocation of diskspace

Start block - 9

blocks: 9, 16, 1, 10, 25 (5-blocks)

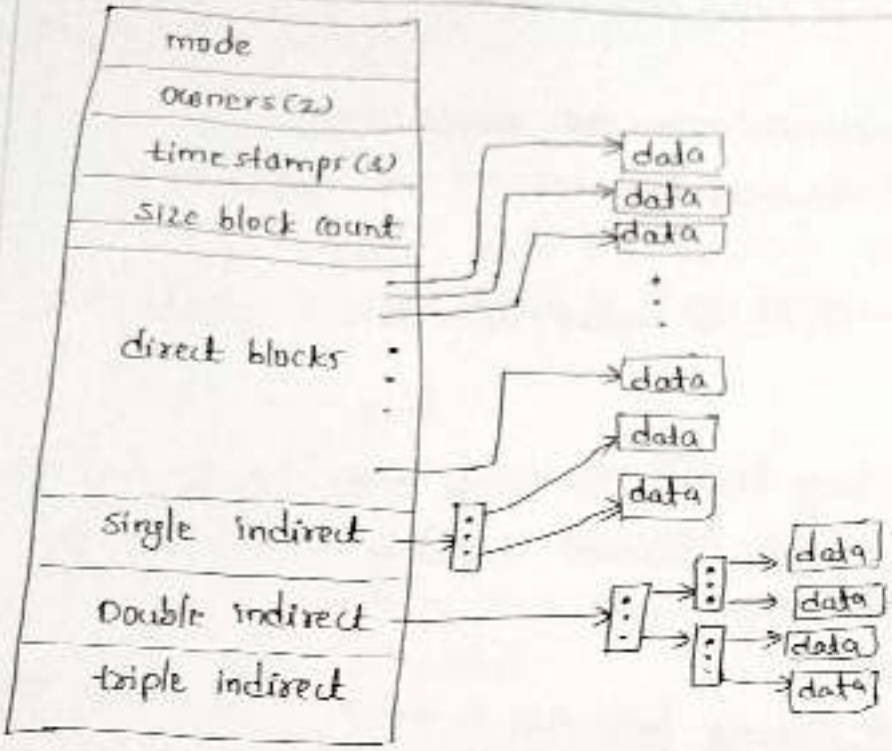
starting block: 1

ending block: -1

Advantages:
→ It support direct access of blocks
→ no external fragmentation
Disadvantages:
→ The pointer overhead for indexed allocation is greater than linked allocation.

For very large files, single index block may not be able to hold all the pointers. following mechanisms can be used to resolve this

1. Linked scheme:
This scheme links two or more index blocks together for holding the pointers. Every index block would then contain a pointer or the address to the next index block
2. Multilevel index: First level index block is used to point to the second level index blocks which points to the disk blocks occupied by the file.
3. Combined Schema: it contain a special block called the inode (Information node), it include all information about the file such a the name, size, authority, etc and remaing space of inode is used to store the Disk block addresses which contains actual file
- direct blocks: the pointer contains the address of the disk blocks that contains data of the file.
- Single indirect: The disk block does not contain the file data bu the disk address of the blocks that contain the file data.
- Double indirect: do not contain file data but the disk address of the blocks that contain the address of the blocks containtaing the file data.



Inode

Free space Management:

Most of the systems disk space is limited, we need to reduce the space from deleted files for new files. To keep track of free disk space, the system maintains a free-space list. the free space list records all free disk blocks - those are not allocated to some file or directory. if we create a new file it allocate space to new file. and that space is removed from the free space list.

The free space list can be implemented mainly as

- Bitmap or Bit vector
- linked list
- Grouping
- Counting

Bitmap or BitVector:

It is a collection or series of bits where each bit corresponds to a disk block the bit can take two values 0 and 1.

0 - indicates that the block is allocated

1 - indicates block is free block

it can be represented as 16 bitmap 0000111000000110

Advantages:

- simple to understand
- finding the 1st free block is efficient. it requires scanning the words in a bitmap for a non-zero word (zero valued word has all bits '0') The 1st free block is then found by scanning for the first 1 bit in the non-zero word.

The block number can be calculated as

(number of bits per word) * (number of 0-value words) + offset of first 1 bit.

In above diagram we get 16 bit map 0000111 0000 0110

1-word = 8 bits

2-word

The block number = 8 X 0 + 5 — 1st word — non zero bit is 5

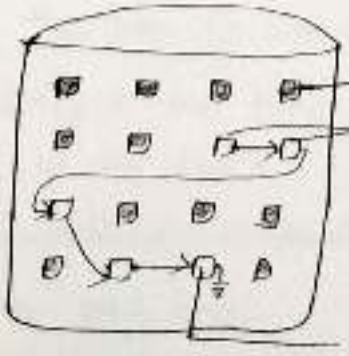
No of bits in word — all bits are not zero in above words

Linked List:

In bitmap if we increase the disk size, it is not easy to maintain free disks blocks

In Linked List free disks blocks contains a pointer to the next free block. The block number of every 1st disk is stored at separate location on disk.

drawback of this method is the I/o required for free space list traversal.



Grouping: it stores the address of free blocks in the first free block If n free blocks, out of these n blocks, the first n-1 blocks are actually free and last block contains the address of the next free n blocks. advantage is free disk blocks can be found easily.

counting:

This approach stores the address of the first free disk block and a number 'n' of free contiguous disk blocks that follow the first block. Every entry in list contain address of 1st free block and number 'n'.

Efficiency and performance:

efficiency dependent on disk allocation and directory algorithms types of data kept in file's directory entry.

Performance:

→ disk cache - separe section of main memory for frequently used blocks.

→ free behind and read-ahead - techniques to optimize sequential access.

→ improve pc performence by dedicating section of memory as vir disk, or RAM disk.

Mass storage structure :

Secondary storage devices are non-volatile. means the stored data will be intact even if the system is turned off.

→ secondary storage is also called auxilary storage.
→ secondary storage is less expensive when compared to primary memory like RAMs.
→ The speed of the secondary storage is also lesser than that of primary storage.
→ The data which is less frequently accessed is kept in the secondary storage.

Examples are magnetic disks, magnetic tapes and removable drives

Magnetic Disk:

→ A magnetic disk contains several platters. each platter is divided into circular shaped tracks.
→ The length of the tracks near the centre is less than the length of the tracks from the centre. Each track is further divided into sectors.
→ Tracks of the same distance from centre from a cylinder A read-write head is used to read data from a sector of the magnetic disk.
→ The speed can be measured as transfer rate and random access time.
→ Transfer rate: The rate at which the data moves from disk to the computer. the sum of the seek time and rotational latency is called random access time
→ seek time: time taken by the arm to move to the required track. rock: rotational latency is the time taken by the arm to reach the required sector in the track.
→ data is logically arranged and addressed as an array of block of fixed size The size of a block is 512 or 1024 bytes.

Magnetic tapes:

→ It is a permanent secondary storage. and it holds large quantities of data. but access time slow.
→ mainly used for backup, storage of infrequently used data, transfer medium between systems.

Disk structure:

→ Disk drives are addressed as large 1-dimensional arrays of logical blocks, where the logical block is the smallest unit of transfer
→ The 1-dimensional array of logical blocks is mapped into the sectors of the disk sequentially.
→ sector '0' is the first sector of the first track on the outer most cylinder.
→ Mapping proceeds in order through that track, then the rest of the tracks in that cylinder. and then through the rest of the cylinders from outermost to innermost.

Disk Attachment:

Computers access disk storage in two ways

1. Host attached storage
2. Network attached storage

Host attached storage:

→ Local disks are accessed through I/O ports
→ The desktop pc uses an I/O bus architecture called IDE or ATA. this architecture supports a maximum of two drives per I/O bu
→ High-end workstations and servers use I/O architectures such as SCSI and fiber channel (FC)

Network attached storage: (NAS)

→ It connects storage devices to computers using a remote Procedure call (RPC).

→ NAS can be implemented using SCSI cabling, or ISCSI over Internet protocols and standard network connections, allowing long-distance remote access to shared files.

→ NAS allows computers to easily share data storage, but tends to be less efficient than standard host-attached storage.

Storage Area Network: (SAN)

→ Connects computer and storage devices in a network, using storage protocols instead of network protocols.

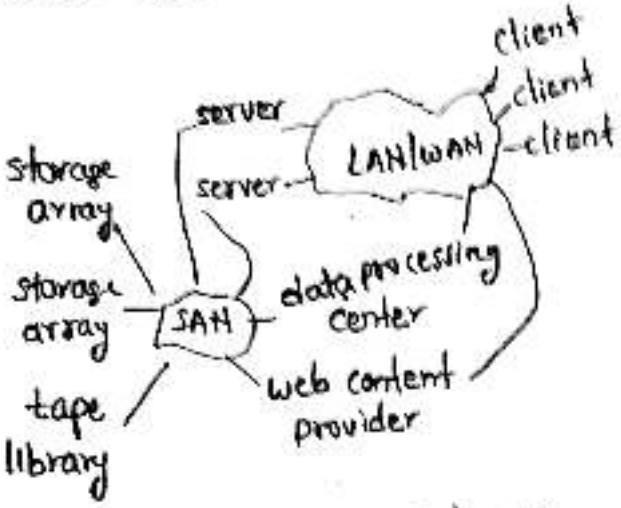
→ One advantage of this is that storage access does not tie up regular networking bandwidth.

→ it is very flexible and dynamic, allowing hosts and devices to attach and detach on the fly.

→ it is also controllable, allowing restricted access to certain hosts and devices.

Disk scheduling:

Network-attached storage



Storage area Network

Disk scheduling:

→ Generally a process needs two type of time, CPU time and I/O time, for I/O time, it requests the operating system to access the disk.

→ seek time: time taken in locating the disk arm to specified track where the read/write request will be satisfied

→ Rotational Latency: time taken by the desired sector to rotate itself to the position from where it can access the R/W heads.

→ Transfer time: time taken to transfer the data

→ Disk access time:

Disk access time = Rotational latency + seek time + Transfer time

→ Disk Response time: it is the average of time spent by each request waiting for the I/O operation.

→ The main purpose of disk scheduling algorithm is to select a disk request from the queue of IO [input output] requests and decide the schedule when this request will be processed.

→ Goal of disk scheduling Algorithm is

- fairness
- High throughout
- Minimal traveling head time

The Disk scheduling Algorithms are

- FCFS scheduling Algorithm
- SSTF (shortest seek time first) Algorithm
- SCAN
- C-SCAN
- LOOK
- C-LOOK

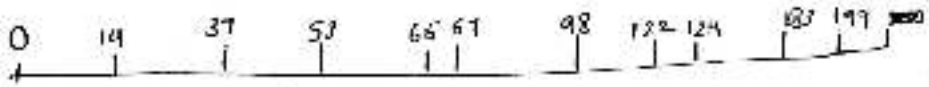
FCFS scheduling:

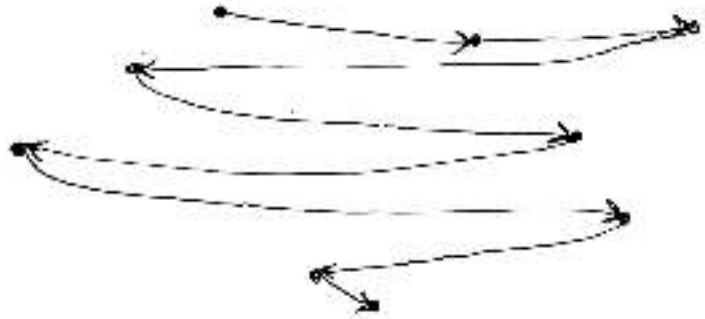
⇒ first come first served algorithm is the simplest disk scheduling algorithm. It services the IO requests in the order in which they arrive.

→ There is no starvation.

→ Disadvantage: the scheme does not optimize the seek time.
The request may come from different processes therefore there is the possibility of inappropriate main movement of the head.

Eg:- A disk queue with requests for I/o to blocks on cylinders
queue = 98, 183, 37, 122, 14, 124, 65, 67, head starts at 53 with 200 tracks.





FCFS disk scheduling

Number of cylinders moved by the head

$= (98-53) + (183-98) + (183-37) + (122-37) + (122-14) + (124-14) + (124-65) + (67-65)$

$= 45 + 85 + 146 + 85 + 108 + 110 + 59 + 2$

$= 640$

Eg:- queue = 45, 21, 67, 90, 4, 50, 89, 52, 61, 87, 25, head point = 25 find number of head movements. [376-A]

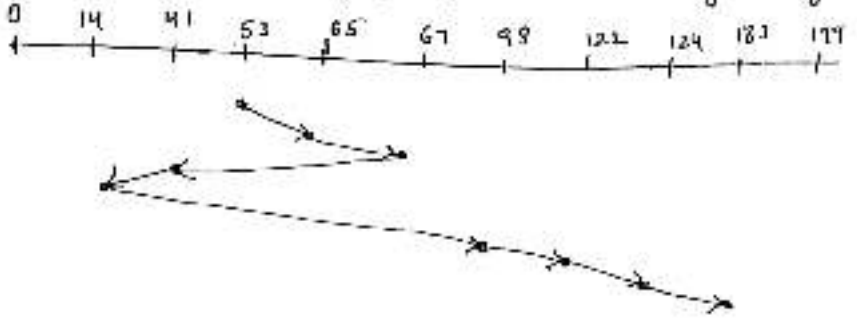
SSTF Scheduling:

⇒ SSTF selects the disk I/O request which requires the least disk arm movement from its current position regardless of the direction. It reduces the total seek time as compared to FCFS.

⇒ It allows the head to move to the closest track in the service queue.

⇒ disadvantage: it may cause starvation for some requests
- Switching direction on the frequent basis slows the working of algorithm.
- It is not the most optimal Algorithm.

Eg: Disk queue with request for I/O to blocks on cylinder 98, 183, 41, 122, 14, 124, 65, 67. Initially at cylinder number 53. The cylinder numbered from 0 to 199. find total head movement by using SSTF.



Total head movements

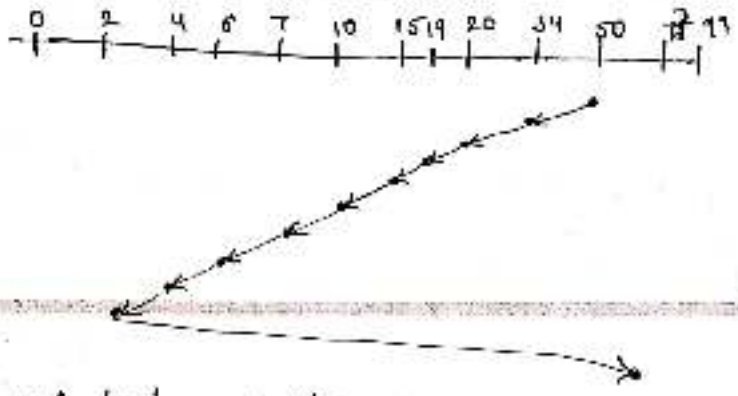
= (65-53) + (67-65) + (67-41) + (41-14) + (98-14) + (122-98) + (124-122) + (183-124)

= 12 + 2 + 26 + 27 + 84 + 24 + 2 + 59

= 236

Eg2: disk with 100 cylinders, the sequence - 4, 34, 10, 7, 19, 73, 2, 15, 6, 20

current cylinder head position = 50. if it takes 1 ms to move from the one cylinder to adjacent one.



= (50-34) + (34-20) + (20-19) + (19-15) + (15-10) + (10-7) + (7-6) + (6-4) + (4-2) + (73-2)

= 16 + 14 + 1 + 4 + 5 + 3 + 1 + 2 + 2 + 71

= 119

time taken to head movement = 1 ms

= 119 × 1

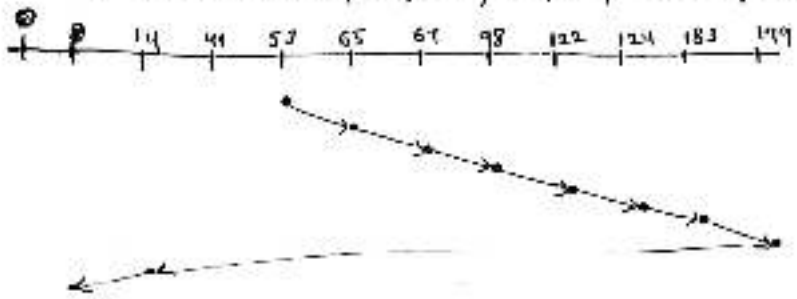
= 119 msec

Total head movements = 119

SCAN / Elevator Algorithm:

→ It scans all the cylinders of the disk back and forth
→ head starts from one end of the disk and move towards the other end servicing all the requests in between.
→ after reaching the other end, head never reverse its direction and move towards the starting end servicing all the requests in between.
→ advantage: It is simple, easy to understand and implement.
→ Disadvantage: It causes long waiting time for the cylinder just visited by the head.

Eg: 98, 183, 41, 122, 14, 124, 65, 67, head point = 53, assume head moves forward direction.



Total head movements:

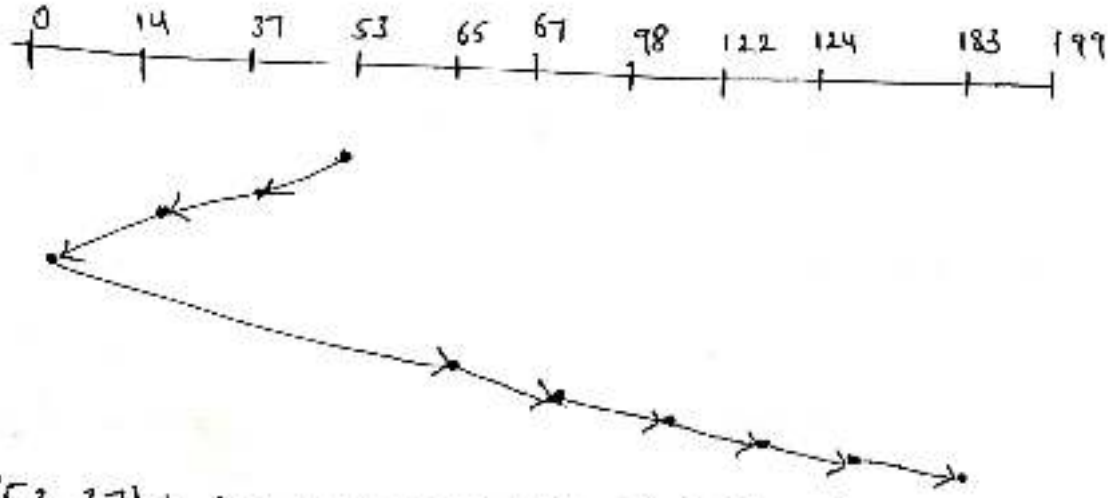
$= (199-53) + (199-14)$
$= 146 + 185$
$= 331$

(or)

Total head movements:

$= (65-53) + (67-65) + (98-67) + (122-98) + (124-122) + (183-124)$
$+ (199-183) + (199-183) + (199-41) + (41-14)$
$= 12 + 2 + 31 + 24 + 2 + 59 + 16 + 158 + 27$
$= 331$

Eg: 2: 98, 183, 37, 122, 14, 124, 65, 67, head - 53, assume disk arm moves toward 0.



$= (53-37) + (37-14) + 14 + 65 + (67-65) + (98-67) + (122-98)$
$+ (124-122) + (183-124)$
$= 16 + 23 + 14 + 65 + 2 + 31 + 24 + 2 + 59$
$= 236$

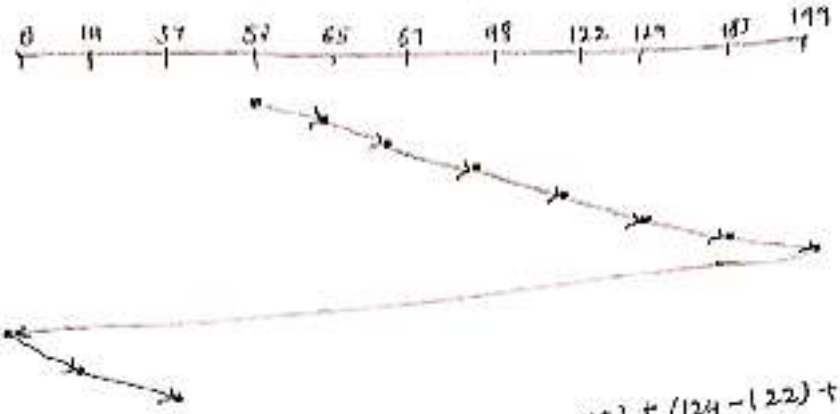
C-SCAN scheduling:

→ Circular-SCAN is an improved version of the SCAN Algorithm.

→ Head starts from one end of the disk and move towards the other end servicing all the requests in between.

→ After reaching the other end, head reverse its direction to the starting end without servicing any request in between.

Eg: 98, 183, 37, 122, 14, 124, 65, 67, head starts at 53



$= (65-53) + (67-65) + (98-67) + (122-98) + (124-122) + (183-124)$
$+ (199-183) + (199-0) + (14-0) + (37-14)$

$= 12 + 2 + 31 + 24 + 2 + 59 + 16 + 199 + 14 + 23$

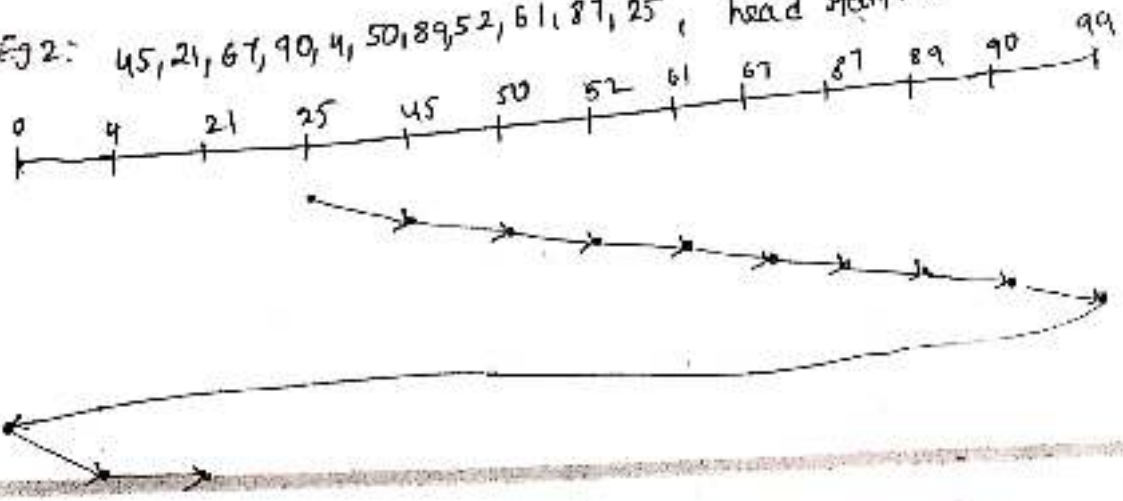
$= 382$ (or)

$= (199-53) + (199-0) + (37-0)$

$= 146 + 199 + 37$

$= 382$

Eg 2: 45, 21, 67, 90, 4, 50, 89, 52, 61, 87, 25, head start = 25



$= (99-25) + (99-0) + (21-0)$

$= 74 + 99 + 21 = 194$

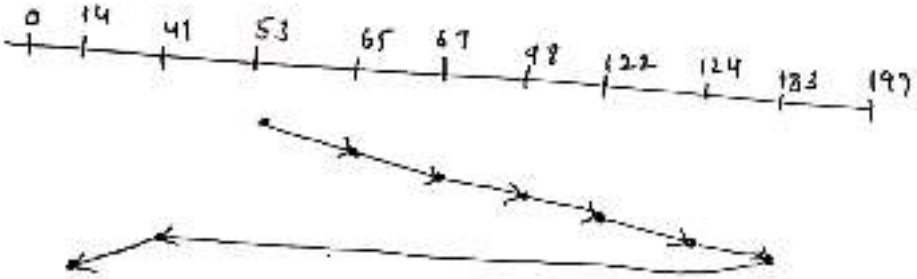
Look Scheduling:

→ It is an improved version of the SCAN Algorithm

→ Head starts from the first request at one end of the disk and moves towards the last request at the other end servicing all the requests in between.

→ after reaching the last request at the other end, head reverse its direction, to the first request at the starting end serving all the requests in between.

Eg: 98, 183, 41, 122, 14, 124, 65, 67, head starts at 53



Total head movements

$= (65-53) + (67-65) + (98-67) + (122-98) + (124-122)$
$+ (183-124) + (183-41) + (41-14)$

$= 12+2+31+24+2+59+142+27$

$= 299$

(Or)

$= (183-53) + (183-14)$

$= 130 + 169$

$= 299$

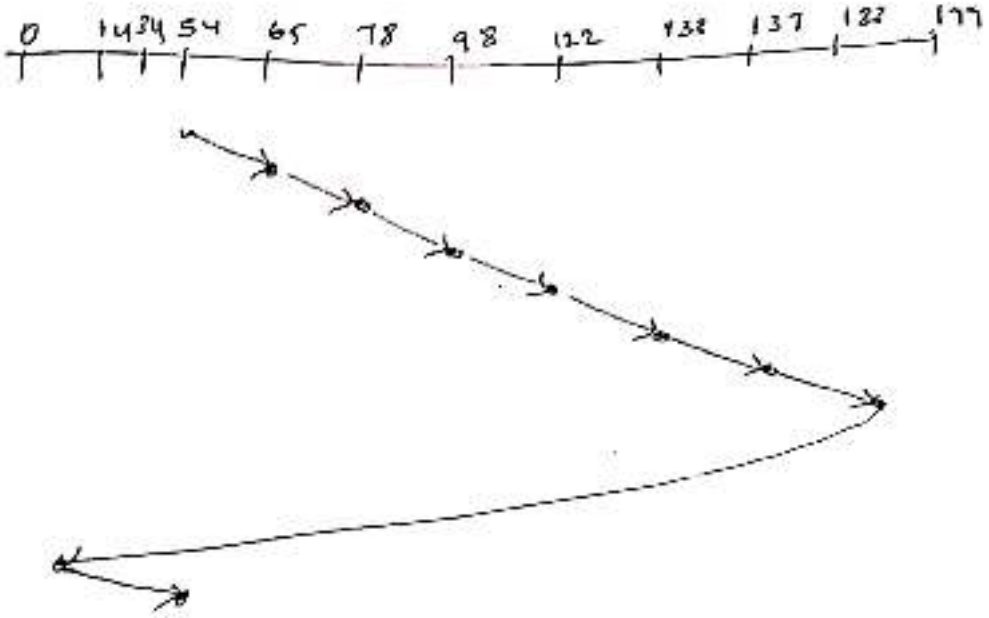
→ The difference between SCAN & Look Algorithm is, SCAN - it scan all the cylinders of the disk starting from one end to other end even if there are no requests at the ends.

→ Look - it scans all the cylinders of the disk starting from the first request at one end to the last request at the other end.

C-Look scheduling:

→ It similar to C-SCAN. the arm of the disk moves outwards serving request until it reaches the highest request cylinder, then it jump to the lowest request cylinder without serving any request then it again start moving outwards servicing the remaing requests.

Eg:- 34, 98, 137, 122, 183, 14, 133, 65, 78 with 200 tracks, head point 54 moving left direction. find no of head movements.



$= (65-54) + (78-65) + (98-78) + (122-98) + (137-122)$
$+ (137-133) + (183-137) + (183-14)$

$= 11+13+20+24+11+4+46+169$

$= 298$

Eg:- A disk contain 200 tracks (0-199). Request queue contains track nos 82, 170, 43, 140, 24, 16, 190 respectively. Current position of R/W head 50. Calculate total no of track movements by R/W head using.

Disk Management:

In Disk management we mainly focus about

- Disk formatting
- Boot block
- Bad block

Disk formatting:

→ It is a low level formatting or physical formatting

→ Dividing a disk into sectors that the disk controller can read and write.

→ A new disk is a blank state and fill each sector with a special data structure - header, data and trailer.

→ Header and Trailer contains information used by disk controller. such as a sector number and an error-correcting-code (ECC).

→ When the controller writes a sector of data, ECC is updated with a value calculated from all the bytes in the data area.

→ When the sector is read, ECC is recalculated and is compared with the stored value

→ partition the disk into one or more groups of cylinders. each partition can be treated as a separate disk.

Boot Block:

→ Bootstrap program initializes system, initialize cpu registers, device controllers, main memory and next it start the OS.

→ In PC, how a tiny bootstrap program is stored in ROM and bring in a full bootstrap program from disk to bootstrap loader.

→ full bootstrap program: it stored in boot block, at a fixed location on the disk. then it load the OS and start the OS.

Bad blocks:

- MS-DOS format performs logical formatting and it scans the disk to find bad blocks.
- It writes a special value into the corresponding FAT entry for bad blocks.
- If bad blocks found during operation, look that bad blocks

MS-DOS disk layout.

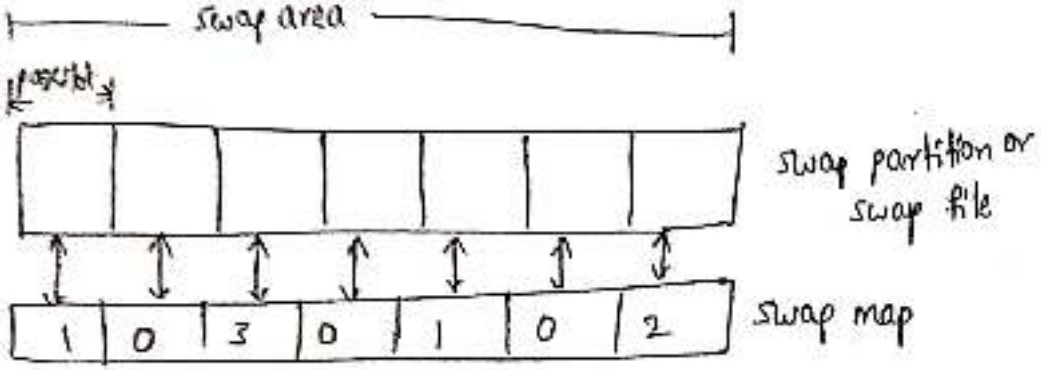
Swap-space Management:

- Swap space - virtual memory use disk space as an extension of main memory.
- The main goal of space space management is to provide the best throughput for virtual machine system.
- Space space use: it can hold entire process image, and paging store pages that have been pushed out of memory.
- Some os support multiple swap sapace space. If system will run out of swap space, some processes must be aborted or system crashed

Swap-space location:

- Swap space create a separate partition called raw partition.
- Swap space manager is responsible for allocate & deallocating blocks

Eg:



The data structures for swapping on Linux systems