# Operating System

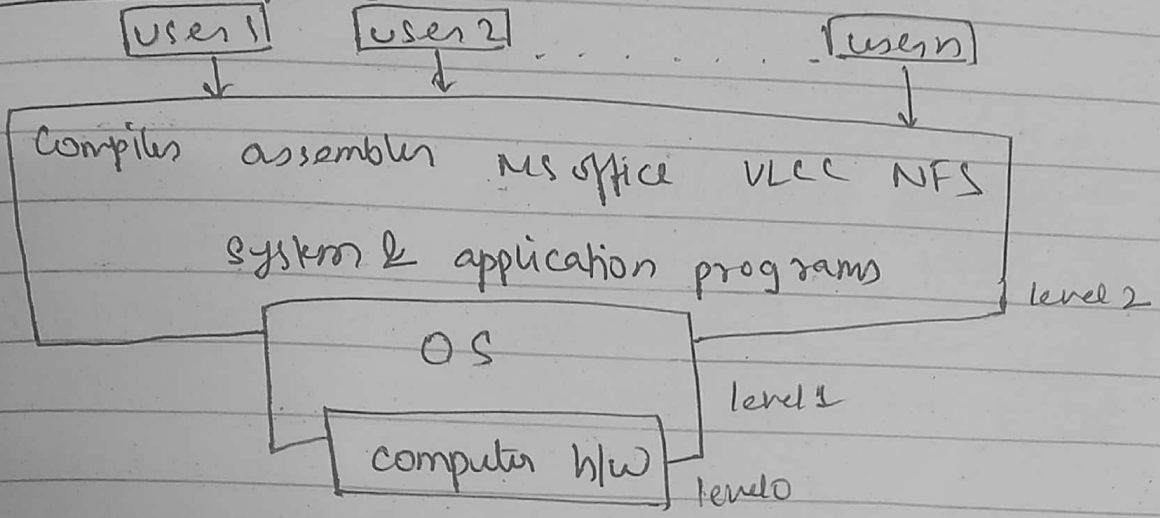
OS is a system software (to maintain co-ordination).

→ It acts as an intermediatory between h/w and user.

→ Resource manager - manage system resources in an unbiased fashion both h/w and s/w.

→ provides a platform on which other application program are installed.

Abstract view of system



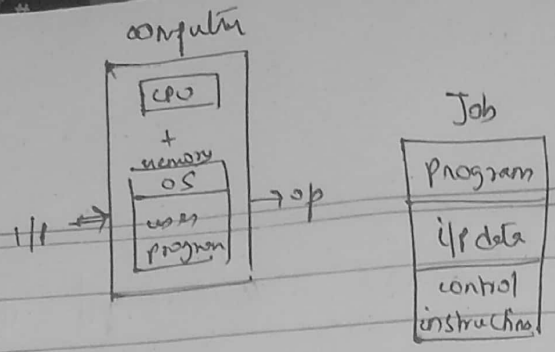

## Goals of OS

→ Primary goal
⇒ convenience/user friendly
OS shud be

→ secondary goal
⇒ Efficiency

## Functions of OS

1. Process Mangement
2. Memory "
3. I/o device "
4. File "
5. Network "
6. Security & Protection

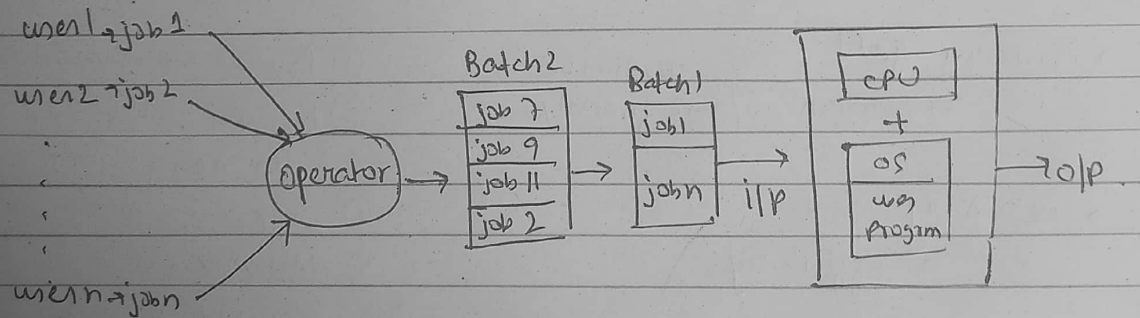



## Batch OS

In starting mainframe computers
→ common i/p and o/p devices were card readers and tape drivers
→ user prepare a job which consisted of the program, i/p data and control instructions.
→ i/p job is given in the form of punch cards and o/p appear in the form of punch card after processing.
→ so OS was very simple, always present in memory, major task is to transfer the control from one job to another.

Problem: ① Memory is limited
② Speed mismatch between input device and processor hence CPU remains idle most tym.
③ system is slow.

## Batch Processing




→ jobs with similar needs are batched together and executed through the processor as a group.
→ operator sort jobs as a deck of punch cards into batch with similar needs (say on basis of language)
→ eg: FORTRAN batch, COBOL batch etc.

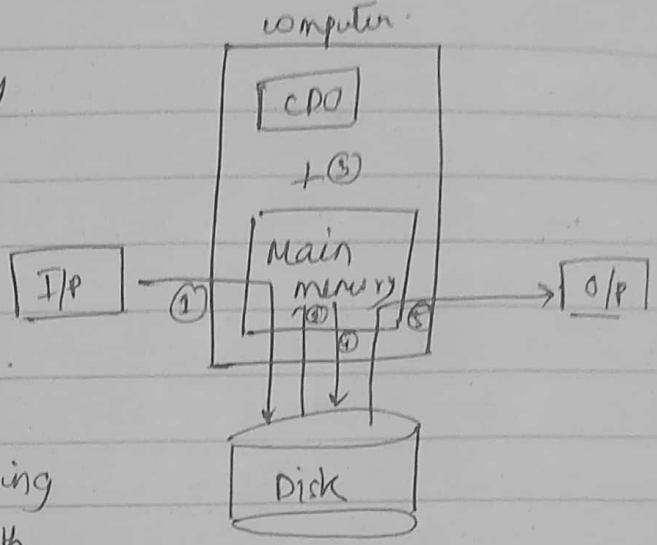
### Advantage

① in a batch job execute one after another saving time from activities like loading compiler
② during a batch execution, no manual intervention is needed.

## Disadvantage

① memory is limited
② interaction of i/p and o/p directly with CPU.
(CPU stays idle during loading and unloading).

## Spooling :- Simultaneous peripheral operations online




→ i/p & o/p devices are relatively slow compare to CPU (digital)
→ In spooling data is stored first onto the disk and the CPU interact with disk (digital) via MM.
→ Keyboard, mouse, printer etc
→ Spooling is capable of overlapping i/o operations for one job with CPU operations of other jobs.
→ many input devices can simultaneously store data onto the disk.

### Advantage

→ no interaction of i/p & o/p device with CPU.
→ CPU utilization is more as CPU is ~~always~~ busy most of the time.

### Disadv

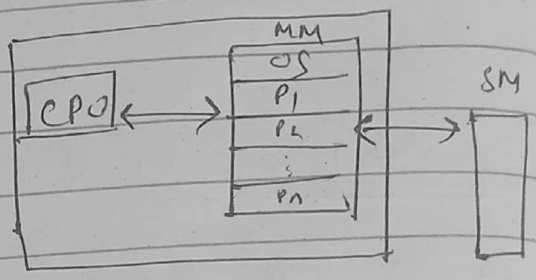
-In starting spooling was uniprogramming (i.e, CPU starts one process and den take another process only after finishing first

→ only one process is only getting executed in processor.

## (#) Multiprogramming OS

(Here we have only one processor)

→ Maximize CPU utilization
→ Multiprogramming means more than one process in main memory which are ready to execute. (Roti example)
→ Process generally require CPU time and I/O time. So if running process perform I/O or some other event which do not require CPU then instead of sitting idle, CPU make a context switch and picks some other process and this idea will continue.




→ CPU never idle unless, there is not process ready to execute or context switch.
→ CPU does not switch process on his/her own basis.
→ switch to another process only when previous process requires time for I/O operations or something else.

### Advantage

→ High CPU utilization
→ less waiting time and response time for different process
→ May be extended to multiple users.
→ Now-a-days useful when load is more.

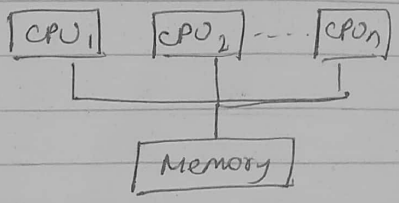
### Disadvantage

→ Different scheduling
→ Main memory management is required.
→ Memory fragmentation
→ Paging (Non-contigous memory allocation)
→ No user interaction.

## (#) Multitasking OS / Time sharing / Fair share / Multiprogramming with Round-Robin

→ Multitasking is multiprogramming with time-sharing
→ Only one CPU but switches between process so quickly that it gives an illusion that all executing at same time
→ the task in multitasking may refer to multiple threads of the same program.
→ Main idea is better response time and executing multiple process together.
→ here we have only one processor.
→ User friendly.

## (#) Multiprocessing OS (more than one processor) (Parallel execution)




→ Two or more CPU with in a single computer, in close communication sharing the system bus, memory and other I/O devices.
→ Different process may run at different CPU, true parallel execution.
→ Symmetric - one OS control all CPU, each CPU has equal rights.
→ Asymmetric - Master slave architecture, system task on one processor and application on other or one CPU will handle all h/w interrupt or I/O device, they are easy to design but less efficient.

| Advantage | Disadvantage |
|---|---|
| → Increased throughput | → more complex |
| → Increased reliability (graceful disgradation) | → Large main memory |
| → Cost saving | → overhead or coupling reduce throughput. |
| → battery efficient | |
| → true parallel processing | |

Neso

## (#) Basics of OS

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

### Important terms

(1) Bootstrap Program: → The initial program that runs when a computer is powered up or rebooted.
→ It is stored in ROM
→ It must know how to load the OS and start executing that system.
→ It must locate and load into memory the OS Kernel.

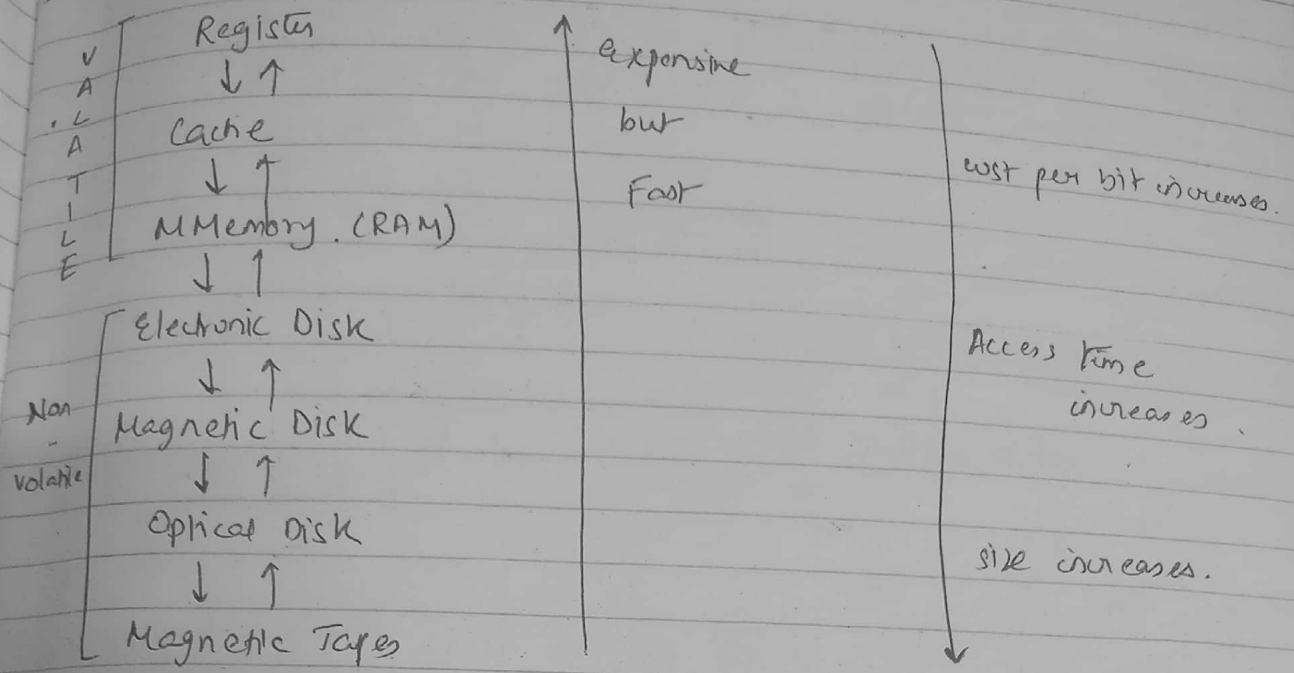
(2) Interrupt: → The occurence of an event is usually signalled by an Interrupt from Hardware or Software.
→ Hardware may trigger an interrupt at any time by sending a signal to the CPU, usually by the way of the system bus.

(3) System call (Monitor call) :- Software 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 transfers execution to a fixed location.
↓
The fixed location usually contains the starting address where the Service Routine of the interrupt is located.

The interrupt Service routine executes.
On completion, the CPU resumes the interrupted computation.

## Storage Structure




Volatile - looses it contents when power is removed.
Non " - does not loose content " " " ".

## I/O structure

→ Storage is only one of many types of I/O devices with a computer.
→ A large portion of OS code is dedicated to managing I/O, both because of its importance to the reliability and performance of a system and because of varying nature of the devices.
→ A general-purpose computer system consist of CPUs and multiple device controllers that are connected through a common bus.
→ Each device controller is in charge of a specific type of device

maintains

- Local Buffer Storage
- Set of Special Purpose Registers.

→ Typically, OS have a device driver for each device controller.

→ This device driver understands the device controller and presents a uniform interface to the device to the rest of the OS.

## #Working of I/O operation - Video 4.

## #Computer System Architecture

Types of computer systems based on number of general purpose processors:

### (1) Single Processor system

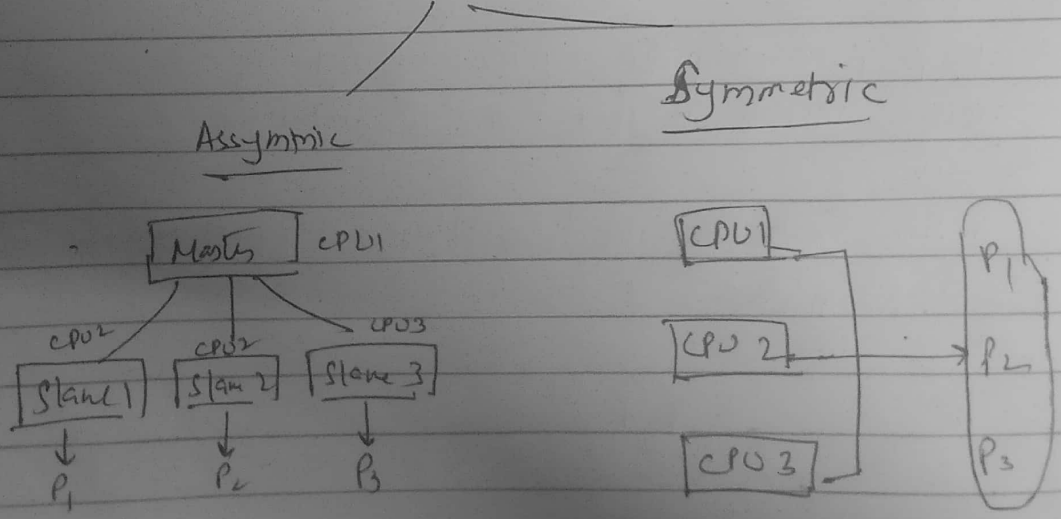
→ One main CPU capable of executing a general purpose instruction set including instructions from user processes.

→ Other special purpose processors are also present which performs device specific tasks.

### (2) Multiprocessor System.

→ Also known as parallel systems or tightly coupled systems.

→ Has two or more processors in close communication, sharing the computer bus and sometime the clock, memory, and peripheral devices.



③ Clustered System

→ Like multiprocessor systems, clustered systems gather together multiple CPUs to accomplish computational work.
→ They are composed of two or more individual systems coupled together.
→ Provides high availability.
→ Can be structured asymmetrically or symmetrically.

↓

- One machine in Hot-standby mode
- Others run applications.

- Two or more hosts run applications
- Monitors each other.

## ④ Operating System Structure

① Operating Systems vary greatly in their makeup internally.
② Commonalities.

/ \

Multiprograming  Time sharing.

### Multiprograming

→ A single user cannot, in general, keep either the CPU or the I/O devices busy at all times.
→ Multiprograming increases CPU utilization by organizing jobs (code and data) so that the CPU always has one to execute.
→ Multiprogrammed systems provide an environment in which the various system resources (for example, CPU, memory, and peripheral devices) are utilized effectively, but they do not provide for user interaction with the computer system.

## Time sharing (Multitasking)

→ CPU executes multiple jobs by switching among them.
→ Switches occur so frequently that the users can interact with each program while it is running.
→ Time sharing requires an interactive computer system, which provides direct communication between the user and the system.
→ A time-shared OS allows many users to share the computer simultaneously.

→ Uses CPU scheduling and multiprograming to provide each user with a small portion of a time-shared computer.
→ Each user has at least one seperate program in memory.
→ A program loaded in memory and execution is called a PROCESS.

## OS Services

Services provided by OS are:-

1) User Interface. — CUI
                    — GUI
2) Program execution. (Run program)
3) I/O operations.
4) File System Manipulation / Management
5) Communications. (b/w process) (over network).
6) Error detection
7) Resource Allocation.
8) Accounting - which user use how much resource.
9) Protection & Security.

~~4) Protection & Security~~

5) User Operating System Interface

CLI GUI

or command Interpreter.

1) Provide a CLI, that allows a users to directly enter commands that are to be performed by the OS.
2) Allows the user to interface with the OS via a GUI.

~~Command Interpreter~~

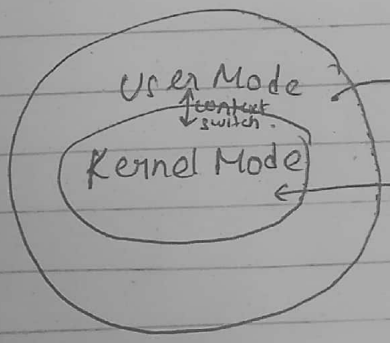
Command Interpreter — heart of OS

→ Some OS 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.

System Calls - call made by program to access resource.

System calls provide an interface to the services made available by an OS.

6) Two modes of program execution - ① User Mode
② Kernel Mode




does not ~~does~~ have direct access to memory and hardware. (safemode)

Priviledged mode (direct access) (may system crash)

The call made by program to become kernel mode ie, to access memory or hardware directly is called system call.

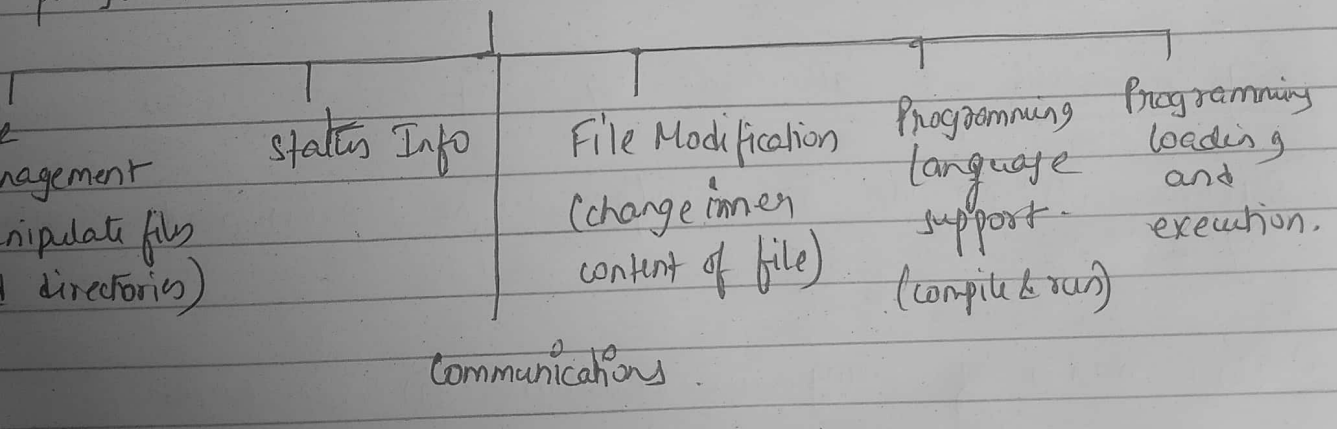
- System call is the programatic way in which a computer program requests a service from the kernel of the OS.
- These calls are generally available as routine written in C or C++.

## # Types of System Calls

1. Process Control - end, abort, load, execute, wait, create process.
2. File Manipulation - open, close, create, del, read, write.
3. Device Management - request, release, read, write, attach or detach.
4. Information Maintanance - date, time, get data, set data/process.
5. Communications - create connection, receive or send messages, attach or detach remote device.

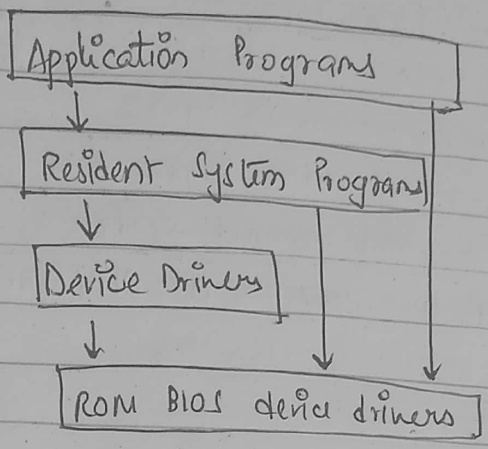
## # System Programs

→ System programs provide a convenient environment for program development and execution.



## Operating System Structures

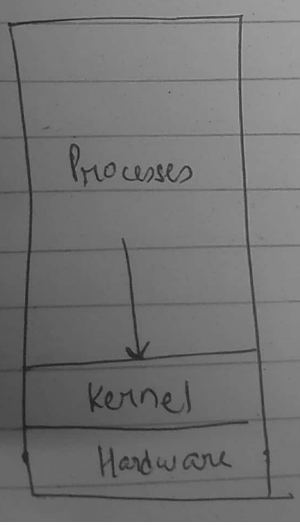
1. Simple Structure

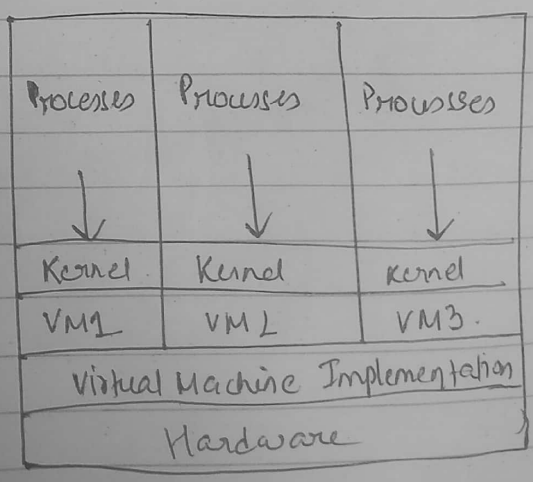



2. Monolithic Structure
3. Layered Structure - OS divided into layers. (semi best structure)
4. Micro Kernels.
5. Modules (Best)

## Virtual Machines

The fundamental idea behind a virtual machine is to abstract the hardware of a single computer (the CPU, memory, disk drives, network interface cards and so forth) into several different execution environments, thereby creating the illusion that each seperate execution environment is running its own private computer.



Implementation

Virtual Machine Software - Runs in Kernel mode } Physical.
Virtual " itself - Runs in User Mode. }

Just as the physical machine has two modes, however, so must the virtual machine.

Consequently, we must have:

- A virtual user mode and
- A virtual kernel mode.

BOTH OF WHICH RUN IN A PHYSICAL USER MODE.

# PROCESS MANAGEMENT

Process : A process can be thought of as a program in execution.
Thread : is the unit of execution within a process. A process can have anywhere from just one thread to many threads.

## Process State

- State of a process is defined in part by the current activity of that process.

States are :-
1. New - The process is being created
2. Running - Instructions are being executed
3. Waiting - waiting for some event to occur.
4. Ready - waiting to be assigned to a processor.
5. Terminated - finished execution.

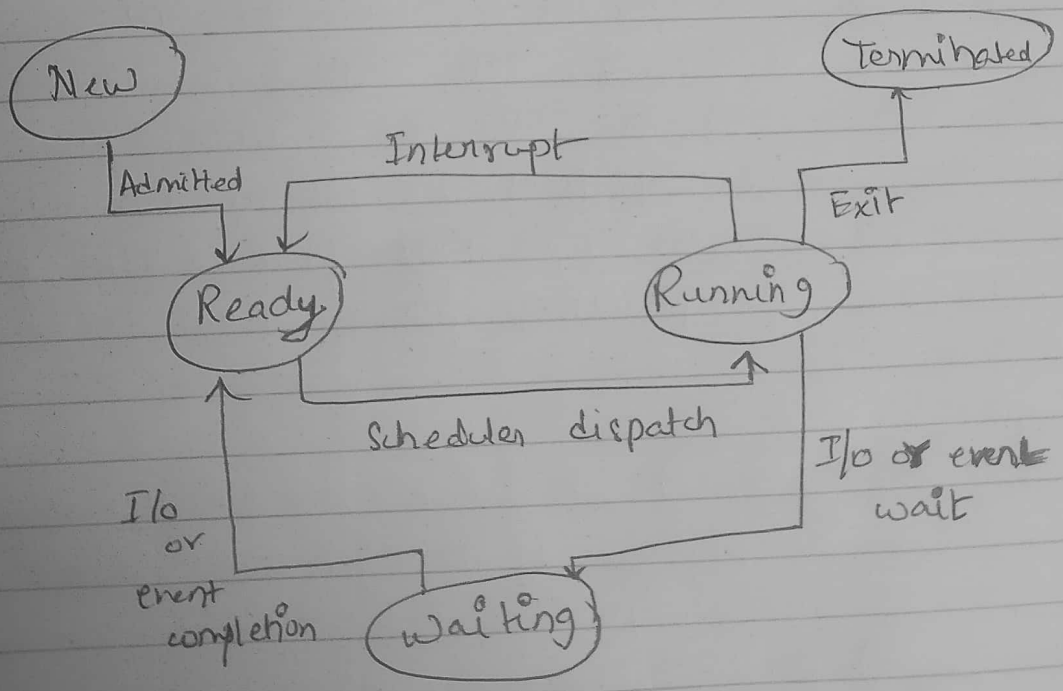
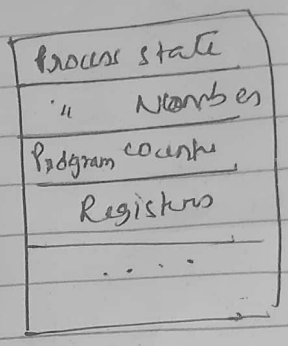



Diagram of process state

## Process Control Block (PCB).

Each process is represented in the OS by a PCB, also called a task control block.




Contains :-
1. Process ID
2. " State
3. Program counter - tells address of next line to be executed
4. CPU Registers used.
5. CPU scheduling info
6. Memory management info.
7. Accounting info
8. I/O status info

## Process Scheduling

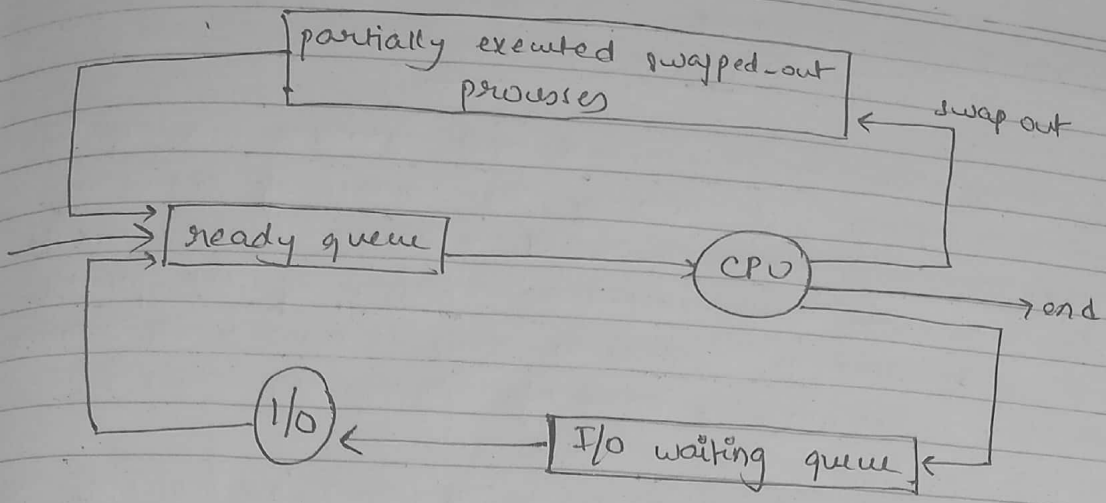
→ Objective of multiprogramming is to have some process running at all times, to maximize CPU utilization.

→ Objective of time sharing is to switch the CPU among processes so frequently that users can interact with each program while it is running.

→ To meet those objectives, the process scheduler selects an available process (possibly from a set of several available processes) for program execution on the CPU.

Scheduling queues - (1) Job queue - As processes enter the system, they are put into a job queue, which consist of all processes in the system.

(2) Ready queue - The processes that are residing in main memory and are ready and waiting to execute are kept on a list called the ready queue.




## Context Switch

→ Interrupts cause the OS to change a CPU from its current task and to run a kernel routine.

→ Such operations happen frequently on general-purpose systems.

→ When an interrupt occurs, the system needs to save the current CONTEXT of the process currently running on the CPU so that it can restore the context when its processing is done, essentially suspending the process and then resuming it.

→ The context is represented in the PCB of the process.

Switching the CPU to another process requires performing a state save of the current process and a state restore of a different process. This task is known as context switch.

→ Context-switching time is pure overhead, because the system does no useful work while switching.

→ Its speed varies from machine to machine.

## Operations on Processes

### (I) (Process Creation)

- A process may create several new processes, via a create-process system call, during the course of execution.
- The creating process is called a parent process and the new processes are called the children of that process.
- Each of these new processes may in turn create another processes, forming a tree of processes.

When a process creates a new process, two possibilities exist in terms of execution:

1. The parent continues to execute concurrently with its children.
2. The parent waits until some or all of its children have terminated.

There are also two possibilities in terms of the address space of the new process:

1. The child process is a duplicate of the parent process (it has the same program and data as the parent).
2. The child process has a new program loaded into it.

### (II) (Process Termination)

- A process terminates when it finishes executing its final statement and asks the OS to delete it by using the exit() system call.
- At that point, the process may return a status value (typically an integer) to its parent process (via the wait() system call).
- All the resources of the process - including physical and virtual memory, open files and I/O buffers - are deallocated by the OS.

Termination can occur in other circumstances as well:

- A process can cause the termination of another process via an appropriate system call.
- Usually, such a system call can be invoked only by the parent of the process that is to be terminated.
- Otherwise, users could arbitrarily kill each other's jobs.

A parent may terminate the execution of one of its children for a variety of reasons, such as these:

1. The child has exceeded its usage of some of the resources that it has been allocated.
2. The task assigned to the child is no longer required.
3. The parent is exiting, and the OS does not allow a child to continue if its parent terminates.

## Interprocess Communication

- Processes executing concurrently in the OS may be either independent processes or co-operating processes.

- Independent processes - They cannot affect or be affected by the other processes executing in the system.

- Co-operating processes - They can affect or be affected by the other processes executing in the system.

- Any process that shares data with other processes is a co-operating process.

- There are several reasons for process co-operation
  - Information sharing
  - Computation speedup
  - Modularity
  - Convenience

Co-operating processes an interprocess communication (IPC) mechanism that will allow them to exchange data and information.

There are two fundamental models of interprocess communication:

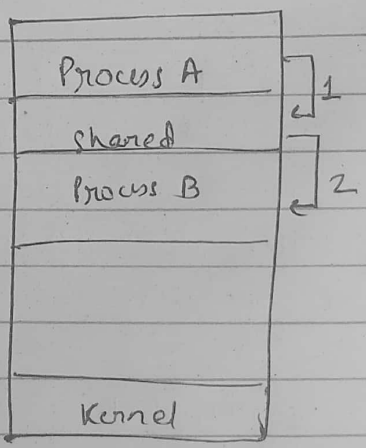
(1) Shared memory
(2) Message passing.

→ In the shared-memory model, a region of memory that is shared by co-operating process is established.
Processes can exchange information by reading and writing data to the shared region.

→ In the message passing model, communication takes place by means of messages exchanged between the co-operating processes.
Process 1 → Message → Kernel → Message → Process 2.

## Shared Memory Systems

- Interprocess communication using shared memory requires communicating processes to establish a region of shared memory.
- Typically, a shared-memory region resides in the address space of the process creating the shared-memory segment.
- Other processes that wish to communicate using this shared memory segment must attach it to their address space.
- Normally, the OS tries to prevent one process from accessing another process's memory.
- Shared memory requires that two or more processes agree to remove this restriction.



## Producer Consumer Problem

A producer process produces information that is consumed by a consumer process.

- One soln to the problem is shared memory.
- To allow producer and consumer processes to run concurrently, we must have available a buffer of items that can be filled by the producer and emptied by the consumer.
- This buffer will reside in a region of memory that is shared by the producer and consumer process.
- A producer can produce one item while the consumer is consuming another item.
- The producer and consumer must be synchronized, so that the consumer does not try to consume an item that has not yet been produced.

### Buffers

- Unbounded - no practical limit of size, producer need not wait
- bounded. - fixed buffer size, both have to wait if size is full.

## Message Passing

Message passing provides a mechanism to allow processes to communicate and to synchronize their activities without sharing the same address space and is particularly useful in a distributed environment, where the communicating processes may reside on different computers connected by a network.

A message-passing facility provide at least two operations.

1. Send
2. Receive.

Message sent by a process can be of either fixed or variable size.

If two processes want to communicate with each other, there must be communication link between them.

- Direct or indirect (Issue - Naming)
- Synchronous or asynchronous
- Automic or explicit buffering.

(#) Next video.

# CPU SCHEDULING

- scheduler
- Scheduling algorithm
- Dispatcher

→ A process execution consist of a cycle of CPU execution and I/o execution.

→ Normally every process begins with CPU burst that may be followed by I/o burst, then another CPU burst and then I/o burst and so on, eventually in the last will end up on CPU burst.

CPU bound → These are those processes which require most of time on CPU.

I/o bound → These are those process which requires most of the time on I/O devices and peripherals.

Conclusion → A good CPU scheduling idea shud choose the mixture of both so that both I/o devices and CPU can be utilised efficiently.

CPU scheduling

| Non pre-emptive (Non-forceful) | Pre-emptive (forcefull) |
|---|---|
| ① When a process complete its execution | ① If a process enters in the ready state either from new or waiting state and it is a high priority process. |
| ② When a process leaves CPU voluntarily to perform some I/o operation or to wait for an event | ② If a process switches from running state to ready state because time quantum expires. |

## CPU scheduling terminology:-

(1) Burst time / execution time / running time :- is the time process require for running on CPU

(2) Waiting time :- time spend by a process in ready state waiting for CPU.

(3) Arrival time :- when a process enters ready state

(4) Exit time :- when process completes execution and exit from system.

(5) Turnaround time :- total time spend by a process in the system.

$$T.AT = ET - AT = BT + WT.$$

(6) Response time :- time between a process enters ready queue and get scheduled on the CPU for the first time.

## Criteria for CPU scheduling Algorithm.

→ Average waiting time / or Turnaround time. ↓
→ Average response time ↓
→ CPU utilisation ↑
→ Through put ↑ - no. of process executing per unit time

# FIRST COME FIRST SERVE (FCFS)

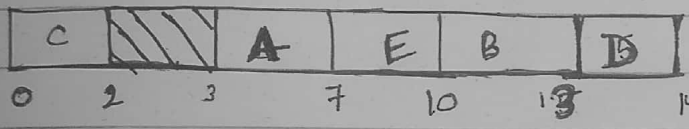
→ Simplest scheduling algo, it assign CPU to the process which arrives first
→ easy to understand and can easily be implemented using queue data structure.
→ Always non-preemptive in nature.
→ Avg. WT is generally quite long.

ex

| PID | BT | AT |
|---|---|---|
| A | 4 | 3 |
| B | 3 | 5 |
| C | 2 | 0 |
| D | 1 | 5 |
| E | 3 | 4 |

| TAT = ET-AT | WT = TAT - BT. |
|---|---|
| 7 - 3 = 4 | 4 - 4 = 0 |
| 13 - 5 = 8 | 8 - 3 = 5 |
| 2 - 0 = 2 | 2 - 2 = 0 |
| 14 - 5 = 9 | 9 - 1 = 8. |
| 10 - 4 = 6. | 6 - 3 = 3 |

gantt chart




Convoy effect :- smaller process have to wait for long time for bigger process to release CPU.

Advantage of FCFS :- simple, easy to use, easy to understand, easy to implement, must be used for background process where execution is not urgent.

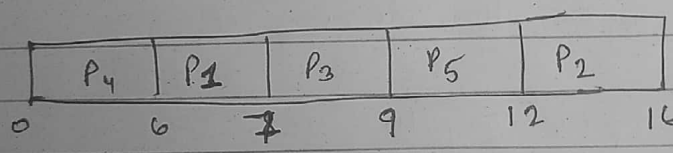
Disadvantage :- suffer from convoy effect, normally higher average waiting time, no consideration of priority or burst time, should not be used for interactive systems. (No starvation - seen in biased CPU systems).

## Shortest Job First (Non Preemptive) / Shortest Remaining time First (SRTF) (Preempting)

→ Out of all available process, CPU is assigned to the process having smallest burst time requirement (no priority, no seniority).
→ If there is a tie, FCFS is used to break tie.
→ Can be used both with preemptive and non-preemptive approach.
→ SRTF is also called as optimal as it guarantee minimal average waiting time.

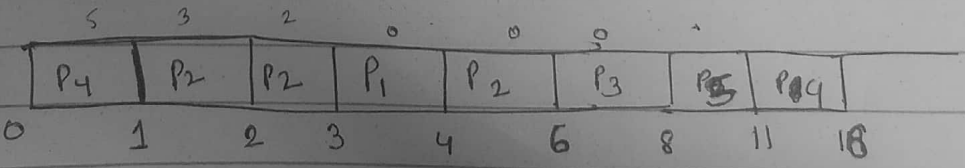
| Pid | AT | B.T |
|---|---|---|
| P1 | 3 | 1 |
| P2 | 1 | 4 |
| P3 | 4 | 2 |
| P4 | 0 | 6 |
| P5 | 2 | 3 |

Non preemp




| | TAT | WT |
|---|---|---|
| P1 | 7 - 3 = 4 | 4 - 1 = 3 |
| P2 | 16 - 1 = 15 | 15 - 4 = 11 |
| P3 | 9 - 4 = 5 | 5 - 2 = 3 |
| P4 | 6 - 0 = 6 | 6 - 6 = 0 |
| P5 | 12 - 2 = 10 | 10 - 3 = 7 |

preemp




TAT of $P_1 = 4 - 3 = 1$
$P_2 = 6 - 1 = 5$
$P_3 = 8 - 4 = 4$
$P_4 = 16 - 0 = 16$
$P_5 = 11 - 2 = 9$.

WT of $P_1 = 1 - 1 = 0$
$P_2 = 5 - 4 = 1$
$P_3 = 4 - 2 = 2$
$P_4 = 16 - 6 = 10$
$P_5 = 9 - 3 = 6$

Advantage:- ① SRTF guarantees minimal average waiting time
② Provide a standard for other algo in terms of average waiting time
③ Better average Response time compare to FCFS.
④ Increases throughput.

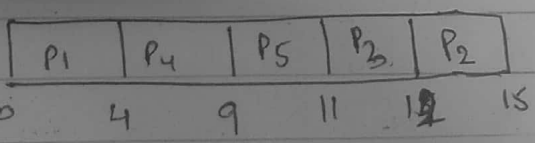
Disadvantage:- ① Algo cannot be implemented as there is no way to know burst time of a process.
② Process with longer CPU burst time required will go into starvation.
③ No idea of priority, process with larger burst time have poor response time.

## PRIORITY ALGO

→ Here a priority is associated with each process.
→ At any instance of time out of all available process, CPU is allocated to the process which possess the highest priority (number may be higher or lower).
→ Tie is broken using FCFS order
→ No importance given to AT or BT.
→ Supports both non-preemptive and preemptive version.

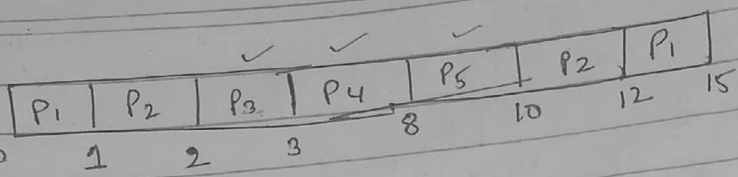
| PID | AT | BT | Priority |
|---|---|---|---|
| $P_1$ | 0 | 4 | 2 |
| $P_2$ | 1 | 3 | 3 |
| $P_3$ | 2 | 1 | 4 |
| $P_4$ | 3 | 5 | 5 |
| $P_5$ | 4 | 2 | 5 |

Non-P




| PID | TAT | WT |
|---|---|---|
| $P_1$ | 4 - 0 = 4 | 0 |
| $P_2$ | 15 - 1 = 14 | 11 |
| $P_3$ | 12 - 2 = 10 | 9 |
| $P_4$ | 9 - 3 = 6 | 1 |
| $P_5$ | 11 - 4 = 7 | 5 |

Preempting




| PID | TAT | WT |
|---|---|---|
| $P_1$ | 15 | 11 |
| $P_2$ | 11 | 8 |
| $P_3$ | 1 | 0 |
| $P_4$ | 5 | 0 |
| $P_5$ | 6 | 4. |

Advantage :- ① Provides a facility of priority specially for system process.
② allows to run important process first even if it is a user process.

Disadvantage :- ① Here process with the smaller priority may starve for the CPU
② No idea of response time or waiting time.
Note :- Ageing is a technique of gradually increasing the priority of process that wait in the system for long time.

ROUND ROBIN (~~non~~ pre-emptive)

→ This algo is designed for time sharing system where it is not necessary to complete one process and then start another, but to be responsive and divide time of the CPU among the process in ready state.
→ Here, ready queue will be a treated as circular queue.
→ We have time quantum up to which a process can hold the CPU in one go, with in which either a process terminate or process must release the CPU and re-enter

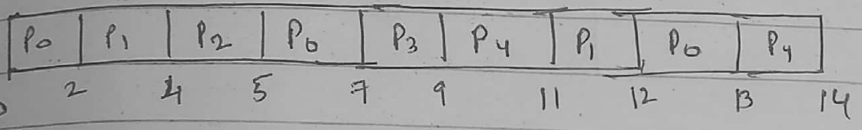
in the circular queue and wait for the next chance.
→ RR is always pre-emptive in nature

ex

| Pid | AT | BT | TAT | WT |
|---|---|---|---|---|
| × $P_0$ | 0 | 5 | 13 | 8 |
| × $P_1$ | 1 | 3 | 11 | 8 |
| × $P_2$ | 2 | 1 | 3 | 2 |
| × $P_3$ | 3 | 2 | 6 | 4 |
| $P_4$ | 4 | 3 | 7 | 4 |

tq = 2

$P_0$ $P_1$ $P_2$ $P_0$ $P_3$ $P_4$ $P_1$ $P_0$ $P_4$




Advantage :- ① Perform best in terms of average response time
② Works will in case of time sharing system, ~~direct~~ client server architecture and interactive system.
③ Kind of SJF implementation.

Disadvantage :- ① Longer process may starve.
② Performance depends heavily on time quantum.
③ no idea of priority.
④ Average WT under RR policy is often long.

| Long-term scheduler | Short-term scheduler |
| --- | --- |
| (1) It picks up the process from jobpool and puts it into the ready queue. | (1) It picks up the process from the ready queue and put it into the CPU. |
| (2) LTS selects the process less frequently. | (2) selects the process more frequently. |
| (3) It controls the degree of multiprogramming. | (3) less control over the degree of multiprogramming. |
| (4) LTS is there in batch systems but it may or may not be present in time sharing system. | (4) STS is there in batch system and is minimally present in the time sharing system also. |

(#) Medium-term scheduler is a part of swapping. It removes the process from the memory. It reduces the degree of multiprogra... The medium-term scheduler is in charge of handling the swapped out-processes.

A running process may become suspended if it makes an I/O request. A suspended processes cannot make any progress towards completion. In this condition, to remove the process from memory and make space for other processes, the suspended process is moved to the secondary storage. This process is called swapping. and the process is said to be swapped out or rolled out. Swapping may be necessary to improve the process mix.

→ In a ... for ... no... wa... → ... be... a...

Sy...

F...

(1)

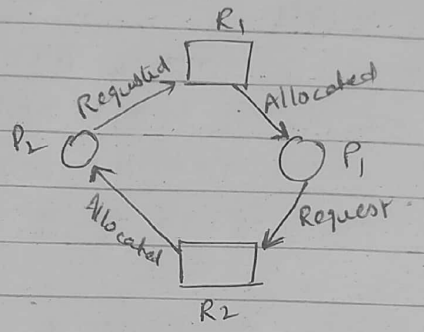
(2)

Waiting
finite (starvation) | infinite (deadlock)

# DEADLOCKS

→ In a multiprogramming system, a number of process compete for limited number of resources and if a resource is not available at that instance, then process enters into waiting state.

→ If a process unable to change its waiting state indefinitely because the resources requested by it are held by another wait process, then system is said to be in deadlock.




Deadlock is defined as a permanent blocking of a set processes that compete for system resources

## System model

→ Every process will request for resource (Reusable / consumable).
→ If entertained, then process will use the resource
→ Process must release the resource after use.

Preemptable (CPU, memory) | Non-preemptable (Printer).

## Four conditions of deadlock

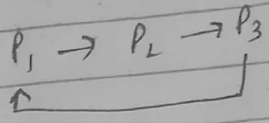
① Mutual exclusion :- At least one resource type in the system which can be used in non-sharable mode, i.e, mutual exclusion (one at a time / one-by-one), eg:- printer.

② Hold & wait :- A process is currently holding at least one resource and requesting additional resource which are being held by other processes.

③ No pre-emption :- A resource can not be pre-empted from a process by any other process. Resource can be released only

voluntarily by the process holding it.

④ Circular wait : Each process must be waiting for a resource which is being held by another process, which in turn is waiting for the first process to release the resource.




## Deadlock Handling Methods

① Prevention (→ side effects are low device utilization and reduced system throughput) - means design such a system which violate at least one of four necessary conditions of deadlock and ensure independence from deadlock.

② Avoidance - system maintains a set of data using which it takes a decision whether to entertain a new request or not, to be in safe state. (Banker's Algo).
[safe, unsafe.

③ Detection and recovery - Here we wait until deadlock occurs and once we detect it and recover (kill the process) from it.
- complex process.

④ (widely used.) Ignorance - We ignore the problem as if it does not exist. (ostrich method) - Used by Linux & windows.
- restart PC when deadlock occurs
- ignore bcoz we can't affect speed by OS by writing code for it.
↓ occurs rarely.

## Deadlock Prevention

- used where deadlock cost is severe
- cost of prevention is more
- used in real time systems.

- Objective is to remove any one necessary condition of deadlock.

- Mutual exclusion can't be violated as sharing property of a resource depends on its hardware and we can't change the hardware of any resource.
ex. printer.

## Violation of Hold & Wait

→ Conservative approach :- Process is allowed to start execution if and only if it has acquired all the resources (less efficient, not implementable, easy, deadlock independent).

↓

a process must release the resources acquired if it cannot acquire all the resources simultaneously.

→ Do not hold :- Process will acquire only desired resources, but before making any fresh request it must release all the resources that it currently hold. (efficient, implementable).

→ wait timeout :- we place a maximum time upto which a process can wait. After which process must release all the holding resources.

## Violation of No Pre-emption

Forceful preemption :- We allow a process to forcefully preempt the resource holding by other processes.

→ This method may be used by high priority process or system process.

→ The process which are in waiting state must be selected as a victim instead of process in the running state

↓

a process can't take a resource from a process which is currently running.

## Violation of Circular wait

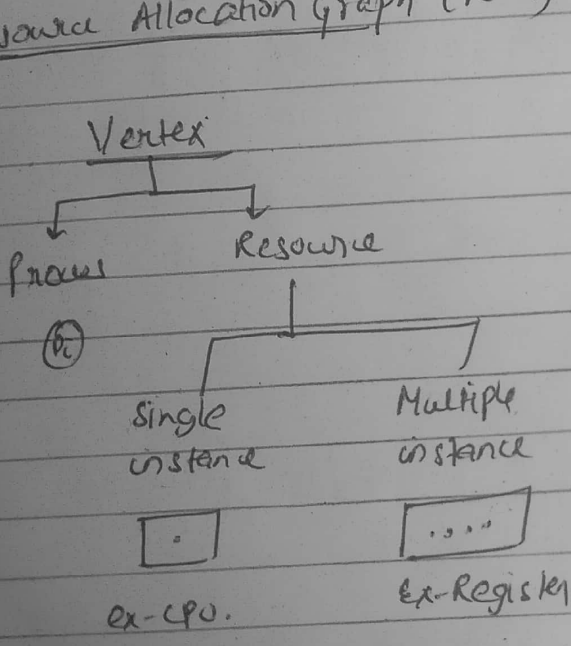
→ circular wait can be eliminated by first giving a natural number of every resource.

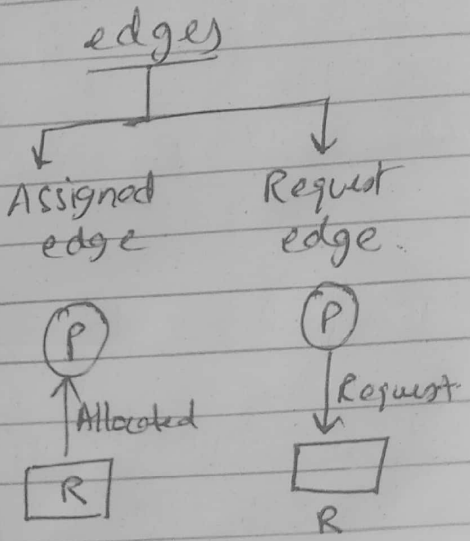
$f: N \to R$

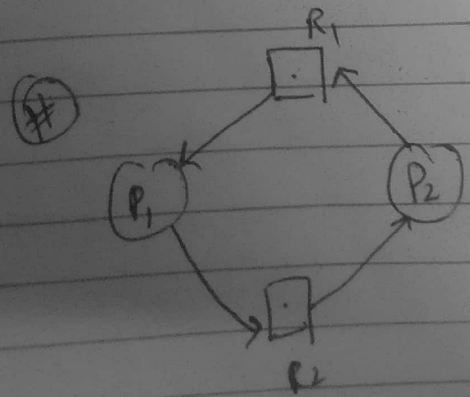
→ allow every process to either access only in the increasing or decreasing order of the resource number.

→ If a process require a lesser number (in case of increasing order), that it must first release all the resources larger than required number.

## Resource Allocation Graph (RAG)




| | Allocate | | Request | |
|---|---|---|---|---|
| | $R_1$ | $R_2$ | $R_1$ | $R_2$ |
| $P_1$ | 1 | 0 | 0 | 1 |
| $P_2$ | 0 | 1 | 1 | 0 |

Availability = ($R_1$ $R_2$) = (0, 0)

↓

not available

hence deadlock

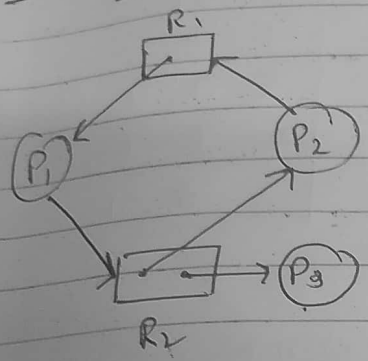
For single instance Resource

(#) if RAG has circular wait (cycle)
⇒ Always deadlock.

(#) if RAG has no cycle ⇒ no deadlock.

} always true.

For multiple instance Resource




| | Allocation | | Request | |
|---|---|---|---|---|
| | $R_1$ | $R_2$ | $R_1$ | $R_2$ |
| $P_1$ | 1 | 0 | 0 | 1 |
| $P_2$ | 0 | 1 | 1 | 0 |
| $P_3$ | 0 | 1 | 0 | 0 |

current availability = $(R_1, R_2) = (0, 0)$

↓ $P_3$ executes.

$(0, 1)$

↓ $P_1$ executes.

$(1, 1)$

↓ $P_2$ executes.

$(1, 2)$.

Hence no deadlock.

(#) For multi instance,

(#) circular wait in RAG does not always lead to deadlock.

safe - no deadlock
unsafe - deadlock.

**Banker's Algo** (Deadlock avoidance).
- also used for deadlock detection.

Total A = 10, B = 5, C = 7.

| Process | Allocation | | | Max Need | | | current Available | | | Remaining Need | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | A | B | C | A | B | C | A | B | C | A | B | C |
| $P_1$ | 0 | 1 | 0 | 7 | 5 | 3 | 3 | 3 | 2 | 7 | 4 | 3 |
| $P_2$ | 2 | 0 | 0 | 3 | 2 | 2 | ↓$P_2$ 5 | 3 | 2 | 1 | 2 | 2 |
| $P_3$ | 3 | 0 | 2 | 9 | 0 | 2 | ↓$P_4$ 7 | 4 | 3 | 6 | 0 | 0 |
| $P_4$ | 2 | 1 | 1 | 4 | 2 | 2 | ↓$P_5$ 7 | 4 | 5 | 2 | 1 | 1 |
| $P_5$ | 0 | 0 | 2 | 5 | 3 | 3 | ↓$P_1$ 7 | 5 | 5 | 5 | 3 | 1 |
| Total | 7 | 2 | 5 | | | | ↓$P_3$ 10, 5, 7. | | | | | |

$P_2 \rightarrow P_4 \rightarrow P_5 \rightarrow P_1 \rightarrow P_3 \Rightarrow$ safe sequence.
↳ not unique seq. → can vary.

Here all processes can be executed hence safe state and no deadlock.

→ In real life, banker's algo can't be implemented as a process can't tell in advance how many resources it needs during execution. Resource allocation is a dynamic process and not static process.

example

| | Allocation | | | Max Need | | | current available | | | Remaining need | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | E | F | G | E | F | G | E | F | G | E | F | G |
| $P_0$ | 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 |
| Total | 5 | 1 | 6 | | | | 8 | 4 | 6 | | | |

$P_0 \rightarrow P_2 \rightarrow P_1 \rightarrow P_3 \rightarrow$ safe sequence.
No deadlock.

Safe state – A state is safe if the system can allocate resources to each process in some order and still avoid a deadlock. (only if there exist a safe sequence).

Unsafe state – If no safe sequence exist.

- A safe state is not a deadlocked state. Conversely, a deadlocked state is an unsafe state.
- Not all unsafe states are deadlocks.

→ pdf notes

## Deadlock Detection

For single instance resource
- make wait-for process graph.
  (if cycle exist - deadlock)
  (no cycle – no deadlock).

For multi-instance resource
- Banker's Algo – $O(m \times n^2)$.

## Deadlock Recovery

- Process Termination
- Resource Pre-emption.

Banker's Algo example

| | 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 | 0 1 1 |
| $P_4$ | 0 0 2 | 4 3 3 | 7 5 5 | 4 3 1 |

$P_1 \rightarrow P_3 \rightarrow P_4 \rightarrow P_0 \rightarrow P_2$ → safe state

If $P_1$ request (1,0,2) — $1,0,2 \leq (3\,3\,2)$ Available. → grant

| | Allocation | Max | Available | Need. |
|---|---|---|---|---|
| | A B C | A B C | A B C | A B C |
| $P_0$ | 0 1 0 | 7 5 3 | 2 3 0 | 7 4 3 |
| $P_1$ | 3 0 2 | 3 2 2 | 5 3 2 | 0 2 0 |
| $P_2$ | 3 0 2 | 9 0 2 | 7 4 3 | 6 0 0 |
| $P_3$ | 2 1 1 | 2 2 2 | 7 4 5 | 0 1 1 |
| $P_4$ | 0 0 2 | 4 3 3 | 7 5 5 | 4 3 1 |

$P_1 \rightarrow P_3 \rightarrow P_4 \rightarrow P_0 \rightarrow P_2$. → safe state

Hence ~~grant~~ request shud be allowed.

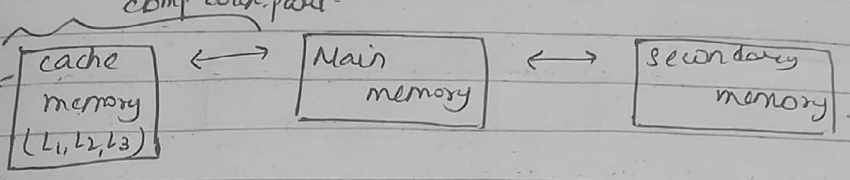
If $P_4$ request (3,3,0) → $(3\,3\,0) \leq (3\,3\,2)$.

| | Allocated | Max | Available | Need. |
|---|---|---|---|---|
| | A B C | A B C | A B C | A B C |
| $P_0$ | 0 1 0 | 7 5 3 | 0 0 2 | 7 4 3 |
| $P_1$ | 2 0 0 | 3 2 2 | | 1 2 2 |
| $P_2$ | 3 0 2 | 9 0 2 | | 6 0 0 |
| $P_3$ | 2 1 1 | 2 2 2 | | 0 1 1 |
| $P_4$ | 3 3 2 | 4 3 3 | | 1 0 1 |

Unsafe state → hence shud not be granted.

# MEMORY MANAGEMENT

Memory
- size (↑)
- Access time (↓)
- per unit cost (↓)




① locality of reference - prefetch related data together and data found is mm.
② we ignore cache in OS.
③ CPU generates logical address for SM.
④ MM have physical address.

LA → MMU → PA. ⇒ Address translation } does two main questions

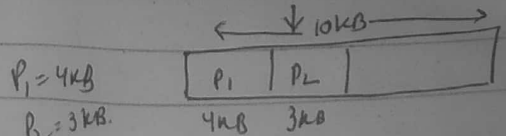
⑤ How process is translated from SM to MM.?

## Process Allocation in MM.

- contigous - easy to access → fast. process. ex-array
  - (problem - external fragmentation).
    - ↳ total space needed by a P is der but not in contigous manner and hence P cant be allocated
- non-contigous.
  - linked list
  - accessing process is difficult. (slow).
  - no external fragmentation.

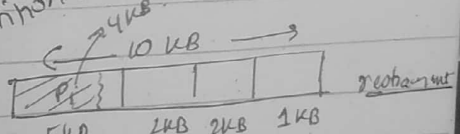
## Contigous Memory Allocation

- Fixed size Partitioning
- variable size "




∞ process arrive, memory is partitioned.
- no internal fragmentation.

partition size may not be same.




rearrangement

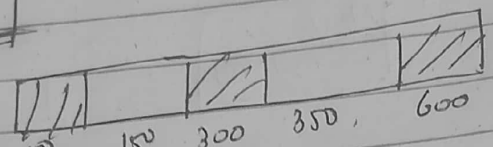
- process take one full fixed size partition.
- per ~~process~~ partition can have only one process
- no reuse of partition.
- internal fragmentation.

## Contigous Memory Allocation Algorithms

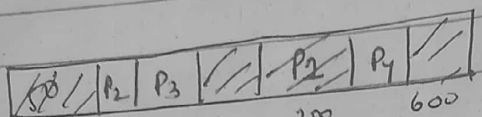
- Best fit - search all memory and then allocate smallest block which can
- Worst fit - largest block is allocated.
- First fit - start from starting and put a process den where it can fit completely.

**variable size**

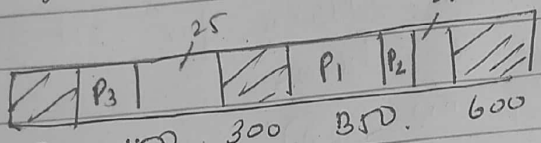
$P_1 = 300$
$P_2 = 25$
$P_3 = 125$
$P_4 = 50$




first

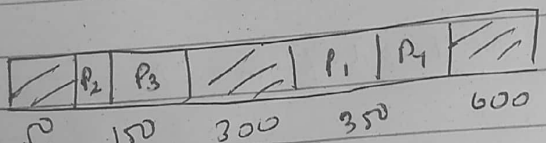



best




→ $P_4$ cant fit
→ external fragmentation
→ best performed worst.

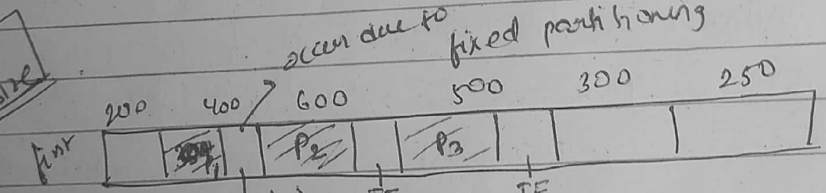
worst




- performed best as compared to best fit.

① Worst fit perform better in variable size partitioning.

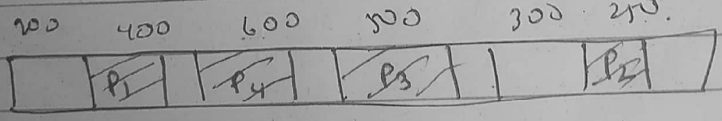
**fixed size**



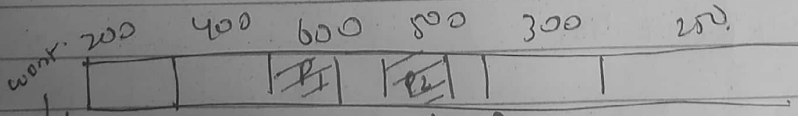

first — occur due to fixed partitioning

→ $P_4$ cant be allocated
→ external frag is also there

$P_1 = 357$
$P_2 = 210$
$P_3 = 468$
$P_4 = 491$




best → no ext. frag




→ $P_3$, $P_4$ cant be allocated due to ext. frag.

↳ (performed ~~best~~ worst in fixed)

① external fragmentation — occurs only when the space needed by a process is available but not in contigous manner.

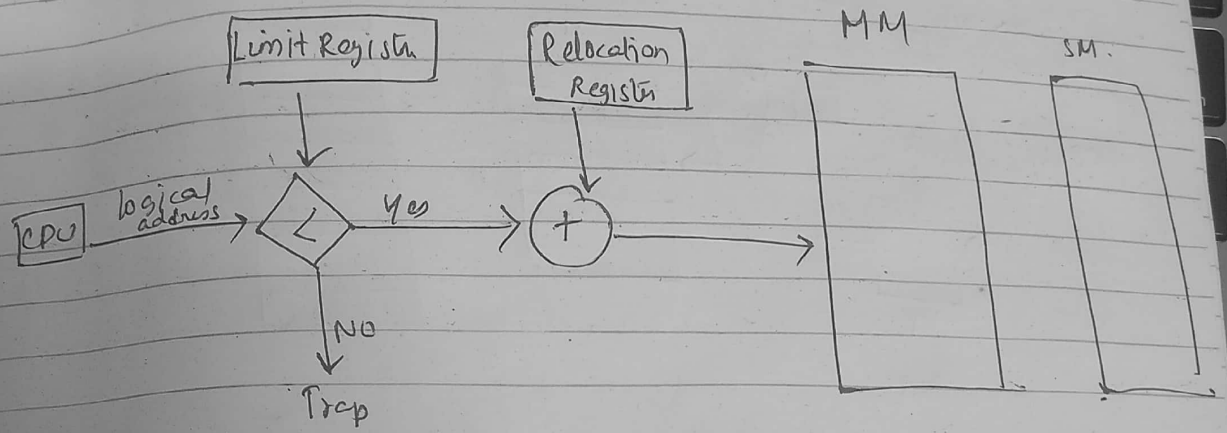
If process size is only bigger than total space of available, then it is not ext - fragmentation.

size of ext frag = size of process which cannot fit

external fragmentation soln
① compaction – brings all free memory together.
② Non-contiguous.

## Address Translation in Contigous Memory Allocation

① cpu always generates address for secondary memory
② as then converts this logical address to physical address
↓
address of process stored in MM.

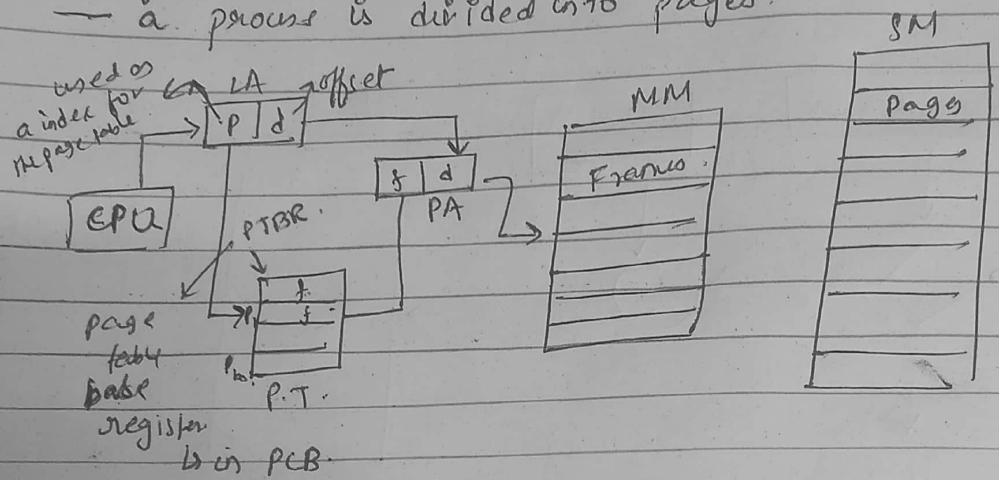



Relocation register have the addresses of all processes. It contains there base address. Any LA is added to base address of P and then accessed data.
Limit Register system helps to ensure that no process can access data out of it bounds.

① Now contigous memory allocations have two problems of external fragmentation and internal fragmentation. In ext. fragmentation, a lot of memory is wasted, and hence to overcome this, we move to non-contigous memory allocation.
- Paging
- Segmentation.

## Paging

- is fixed size partioning scheme (hence internal frag).
- divide SM into equal size partitions called pages.
- size of each page is same
- MM is also divided into equal size partition called frame
- frame size = page size.
- a process is divided into pages.




- CPU generates LA for SM, now as SM is divided into pages, LA contains both page number and offset (instead of saying instruction number) offset says which instruction to refer in that specific page.

- Conversion of LA to PA.

① maintains a page table - a data structure.
② every process have an individual page table
③ no. of entries in page table is equal to no. of pages into which a particular process is divided.
④ page table contains the address of frame nos - corresponding to each page of a process. ie, where page 1 is kept in MM or frame.

advantage - access is very fast. at the cost of maintaining a database of page table.

disvantage of paging - ① need to access MM twice, once for page table and 2nd for data.
② Internal fragmentation.

Sums

① If n digits, then possible combinations $= 2^n$.

② ←n→
n bit address.

with n bit address we can generate $2^n$ different address and therefore memory had $2^n$ different location.
∴ size of memory = $2^n$ × size of each location
(generally 1B)

③ $2^{10} = 1K$
$2^{20} = 1M$
$2^{30} = 1G$
$2^{40} = 1T$
$2^{50} = 1P$.

④ Memory = 64 KB. Find no. of bits in address.

Memory size = Total no. of locations × size of each location.
∴ no. of loc = $\frac{M.S}{\text{size of each loc}}$

here no. of loc = $\frac{64KB}{1B}$
$= 64K$ = 64000 locations.
$= 2^6 \cdot 2^{10}$
$= 2^{16}$.
∴ no. of bits = 16.

Q. ←LA→ : |←22B→|←11→| ; ←33→

←PA→ : |←13→|←d 11→| ; ←24→

LA = 33 bit
PA = 24 bit
PS = 2KB
↓
No. of instructions in a page = $\frac{2KB}{1B}$
$= 2K = 2^{11}$

SM: Size = $2^{33} \times 1B$
$= 2^3 \times 2^{30} \times 1B$
$= 8GB$

MM: Size = $2^{24} \times 1B$
$= 2^2 \times 2^{20} \times 1B$
$= 4MB$

## TLB → to reduce time penalty.

① As we know that for paging we need to access main memory twice which reduces the fastness of the system. Moreover suppose a page contains 100 instructions and as most instructions are executed sequentially, for every instruction we need to access the page table, i.e. say $P_1$ contains first 100 instructions, ∴ for 100 instruction we need to access Page table every time to get its frame number, which is quite hectic as we are doing same task again and again. To solve this we have a hardware called TLB i.e. translation look Aside buffer.

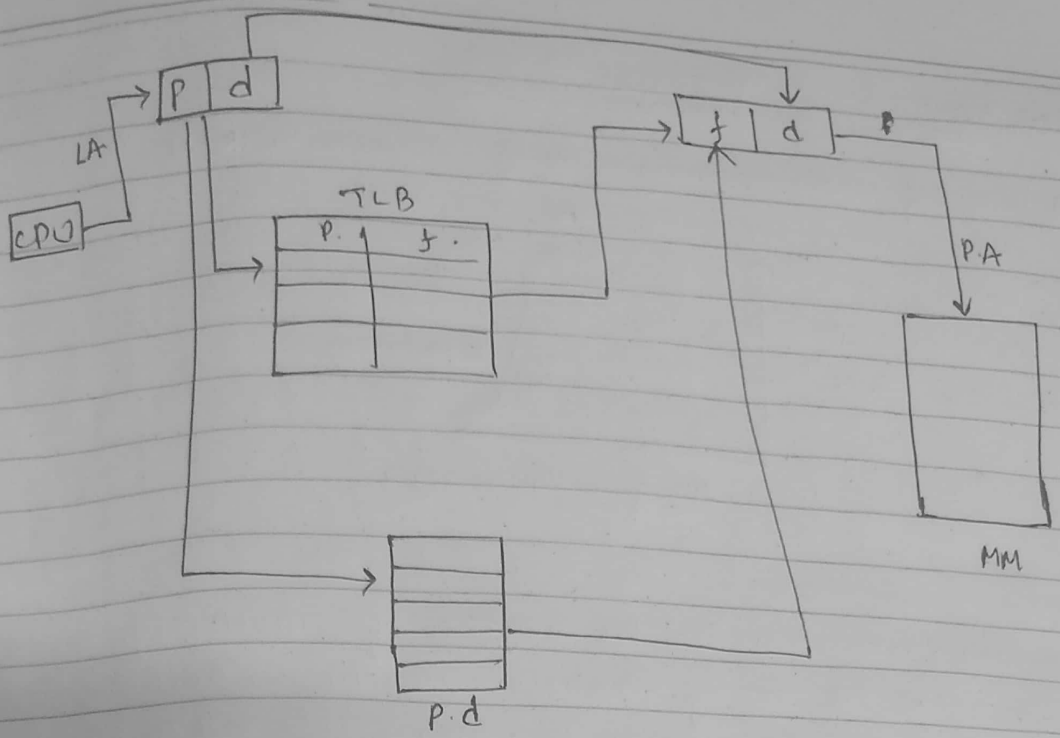
② TLB has 2 colums - page no. & frame no.

TLB.

| P. | f |
|---|---|
| $P_1$ | $f_{32}$ |

③ whenever a page table is accessed for a particular page, that particular page and frame is updated to TLB. Next tym whenever that page is again accessed, it first lok into TLB and hence process becomes faster as buffer is very small and easy to access.

④ Now TLB is not different for different processes. Every tym there is a context switch, the TLB needs to cleared so that it can update the page number for the next process.




Q. MM = 400 μs

TLB = 50 μs

h = 90% of TLB.

Find total access time.

→ when no TLB, den access time = 2 × 400

= 800 μs.

When TLB, access time = .9 [50 + 400] + .1 [50 + 400 + 400]

= .9 (450) + .1 (850)

= 490 μs.

## Disadvantages of TLB

① can hold data of one process at a tym.

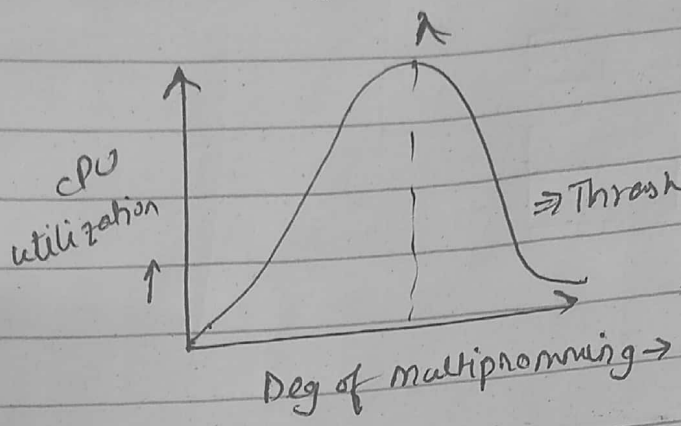
② when context switches are more, TLB performance ↓↓s.

soln

- multiple copies of TLB
- some part of TLB can be reserved for OS.

# Thrashing

① Degree of multipromming - no. of process in RAM




We do paging to increase the degree of multiprogramming, i.e, increase the no. of process present in RAM. We do so, by bringing only few pages of different process to RAM. Now when CPU tries to access any page which is not present in RAM, it is called page fault. With increase in degree of multiprogramming, page fault also increases. Whenever page fault occurs, the system becomes busy to solve that and CPU utilization decreases. and this is called Thrashing

↓

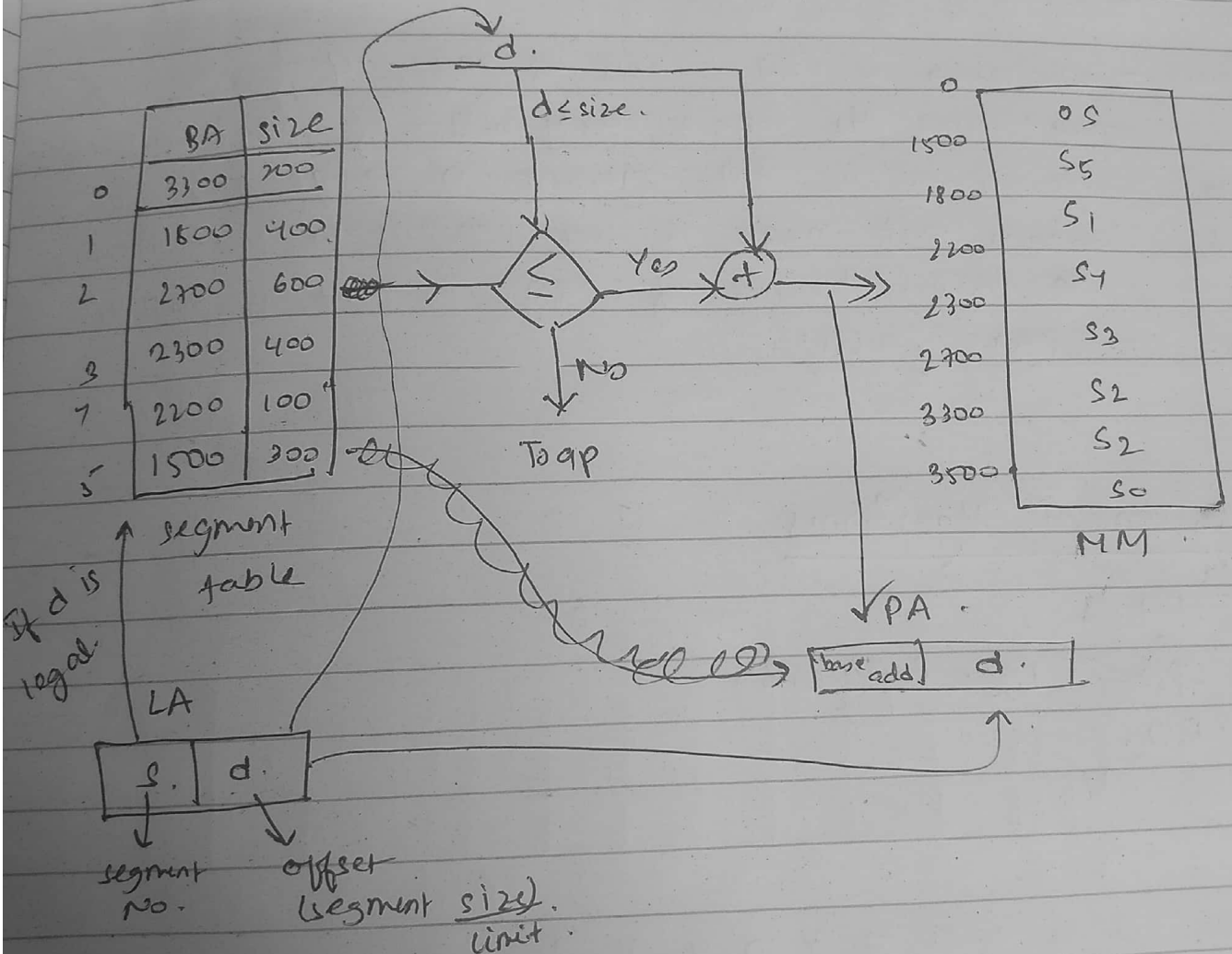
when excessive page fault occurs.

Soln

- MM size ↑↑
- long term scheduler.

# Segmentation

- A process is divided into segments of variable size.
- segment size may differ.
- Now these segments are moved to MM.
- have segment table for conversion of LA to PA.
  - 2 columns
    - ① base address
    - ② size.



# VIRTUAL MEMORY

- an illusion that a process whose size is greater than the size of MM can also be executed.
- we divide process into pages and bring only required pages only to MM. by swap in / Roll in process.
- advantage - ① an illusion of bigger memory
② many process can be brought to MM.

Page fault - whenever a CPU access a page, it first goto page table to check dirty bits. If page's corresponding frame no. is not present is page table, it means that page is not present in MM. And this absence of page in MM is called page fault.

↓
leadings to thrashing
if page faults is frequent.

## Page Replacement Algorithms

- ① FIFO
- ② Optimal Page Replacement
- ③ LRU.
- ④ Most RU.

## FIFO

string: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 1, 2, 0.
No. of frames = 3.

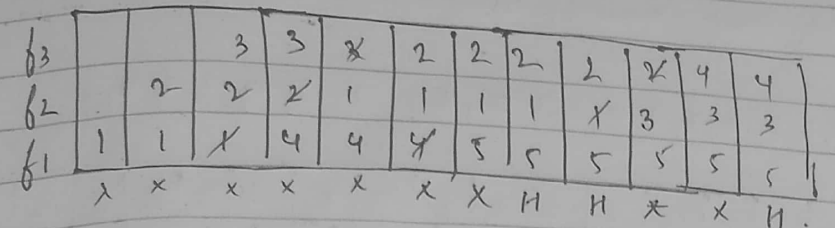
| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| f3 | | | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 3 | 3 | 3 | 3 | 2 | 2 |
| f2 | | 0 | 0 | 0 | 0 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 1 | 1 | 1 |
| f1 | 7 | 7 | 7 | 2 | 2 | 2 | 2 | 4 | 4 | 4 | 0 | 0 | 0 | 0 | 0 |
| | x | x | x | x | Hit | x | x | x | x | x | x | Hit | x | x | Hit |

Fifo - jo sbse pehle aya, usko replace krdo.

No. of hits = 3
No. of miss = 12

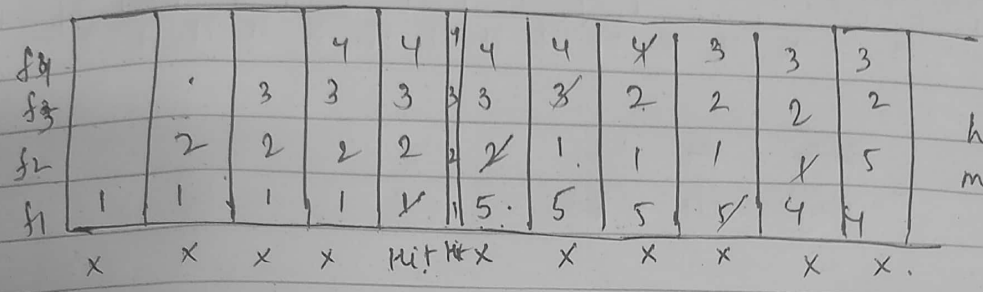
hit ratio = $\frac{3}{15} \times 100$.

## Belady's Anomaly




hit = 3
miss = 9.

Ref - 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5




hit = 2
miss = 10.

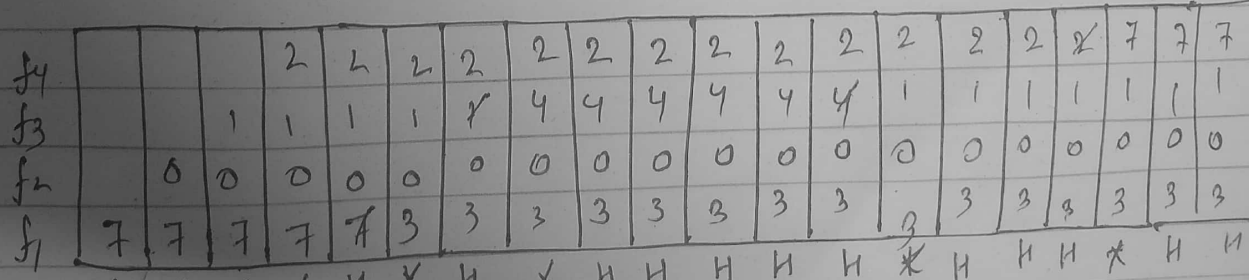
In fifo, general concept/logic says if we increase the number of frames, page fault must also reduce, but this does not happen. Infact increasing the number of frames, increases page fault and this is called belady's anomaly.

(*) FIFO suffers from belady's anomaly.

## Optimal Page Replacement

L Replace the page which is not used in longest dimension of time in future

Ref:- 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

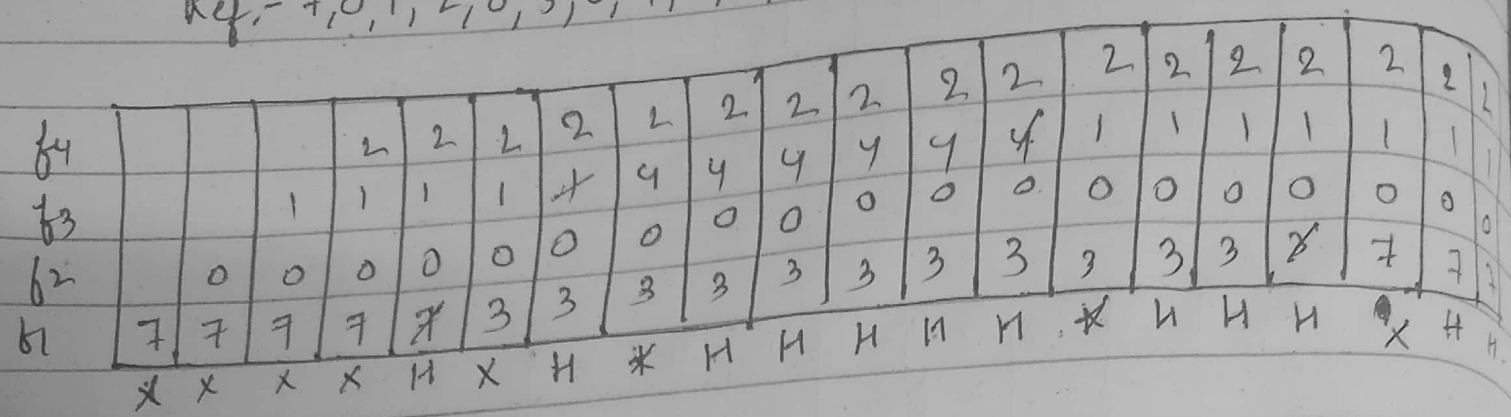



hits = 12
pagefault = 8.

## LRU

L Replace the least recently used page in past.

Ref:- 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1



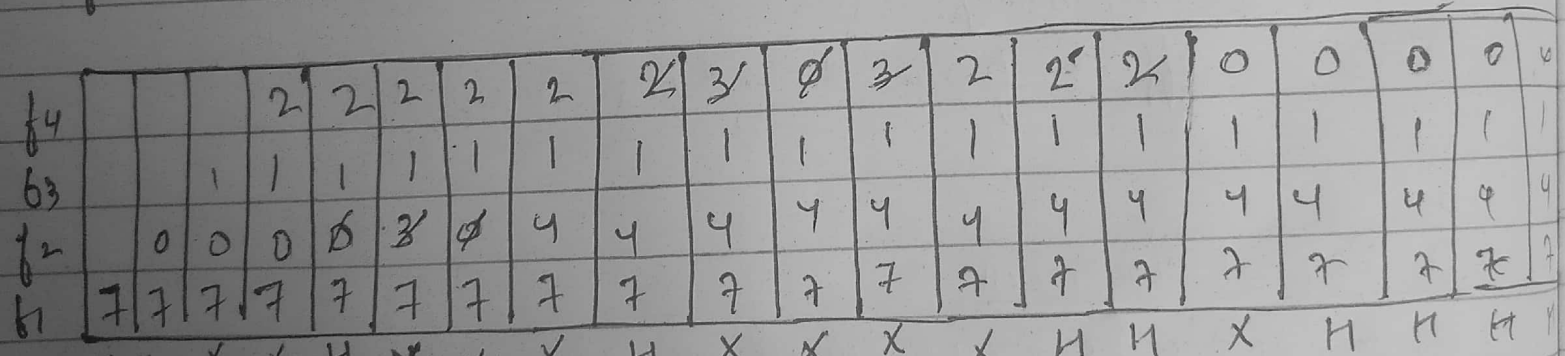
Hits = 12

page fault = 8.

## Most Recently Used

L Replace the most recently used page in past.

Ref: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1



Hit = 8

page fault = 12.