Operating Systems

"I ka GUNDA" of the System. ~~is~~ OS

1.1 What is OS?

Whatever used as an **interface** between the user and the core machine is OS.
(hardware)

Eg. Steering of car, switch of fan.

Why need a OS?

To enable everyone to use h/w in a **CONVENIENT & EFFICIENT** manner

→ Teacher bhai ka rey chalta

now we will interact w/ the H/W or machine only thru OS (sys ka ek hi gunda hai, OS)

Definition

"A program or system software"

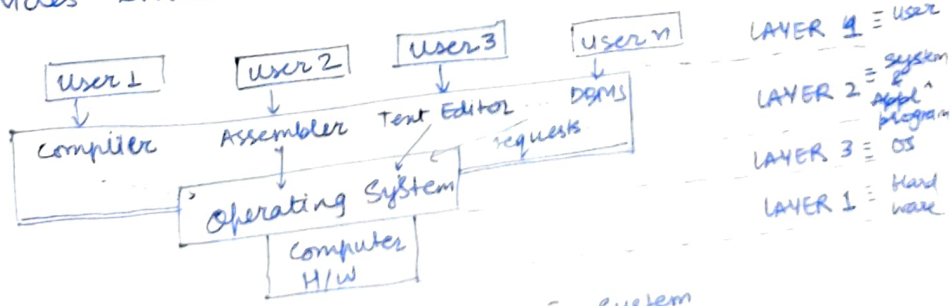
- which acts as an **INTERMEDIARY** between user & h/w
- RESOURCE MANAGER / ALLOCATOR

  Manage **resources** in an UNBIASED fashion b/w

  h/w (CPU time, memory, sys buses) & s/w (access, authorization, semaphores) & provides FUNCTIONALITY to apps
- provides a **platform**

  on which other app^ programs can be installed,

  provides environment for their execution



Abstract View of a Computer System (4 layers)

eg. Linux, MAC or IOS, Windows, Android, Ubuntu, Tizen, Debian
~1.5% 12-13% 82% 87%

1.2 Goals of OS & Funcs

Goals are the ultimate destination, but we follow functions to implement them.

GOALS

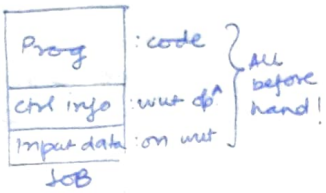
- Primary goal: CONVENIENCE (user friendly)
- Secondary goal: EFFICIENCY, reliability, maintainability

FUNCTIONS

- *Process manag^
- *Memory manag^
- I/O device m.
- File m.
- Network m.
- Security & Protection

Evolution of OS

1. > Early comp° were not interactive device, their user used to prepare a JOB (→ now process or transaction) which had 3 parts:
   - Program
   - Control Information
   - Input data

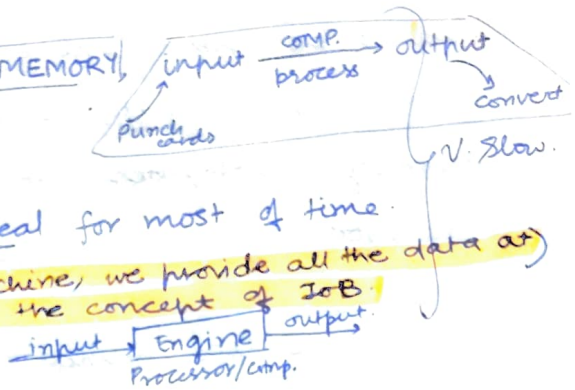


> Only one job input at a time, NO MEMORY

> Examples: Punch cards, tape drives (these devices were v slow)

> So, slow devices, processor was ideal for most of time.

(Since there is no interaction with machine, we provide all the data at once & that is the concept of JOB)



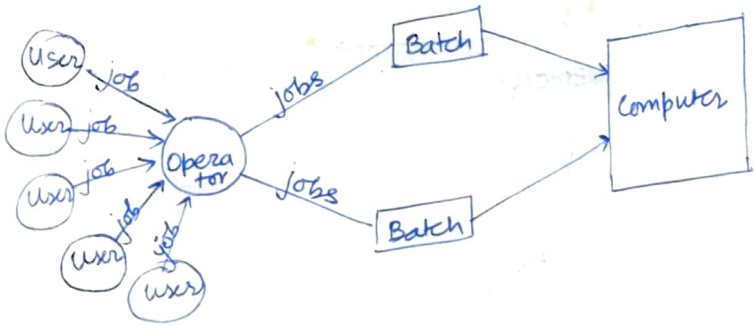
2 Batch OS

> One of the first kind of OS.

> To speed up the processing, jobs WITH SIMILAR TYPES (programming language) were batched together & were run thru processor as a group (batch)

> Done by operator / 'Batch Monitor'

> eg: FORTRAN jobs, COBOL jobs etc.



Disadvantages
1. MEMORY LIMITATION - becz of which interactive process / multiprogramming is not possible.
2. Direct interaction of I/O devices w/ CPU.
3. CPU idle.

Advantages
1. Loading the compiler once ⇒ System utilization due to reduced turn around time.
2. Increased performance (as a new job gets executed asa previous job is finished, NO HUMAN INTERVENTION MANUAL)

Introducing the concept of memory!

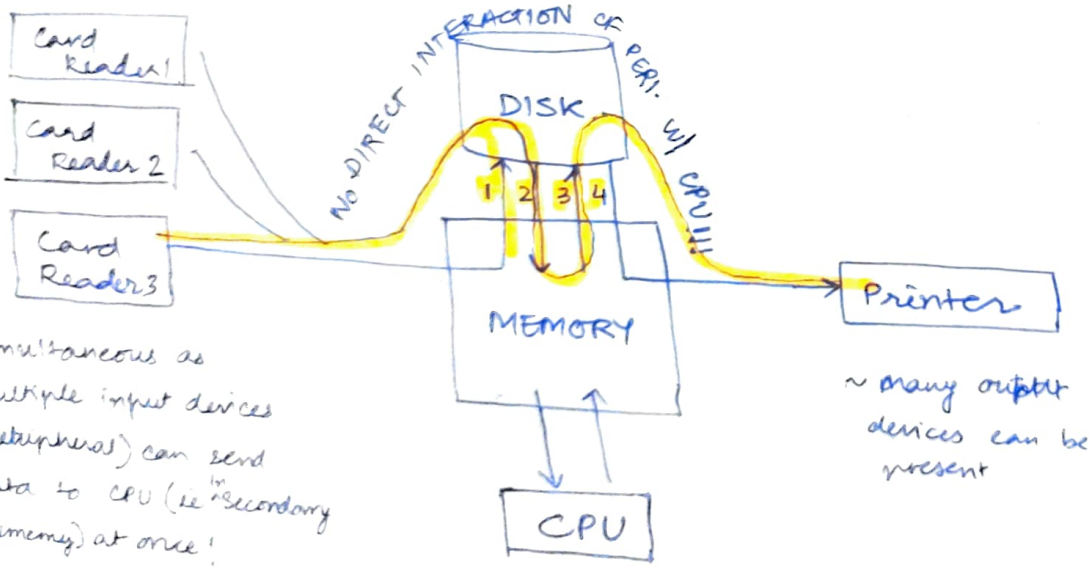
(Imp) Spooling

Spool ≡ send data to an intermediate store

↳ (Simultaneous Peripheral Operations Online)
Full form.
↳ more than 1 device can send input at once (simul.) to the CPU as more is connected directly to the CPU.

Spooling is a process in which data is temporarily held to be used & executed by a device, program or the system. Data is sent to and stored in memory or other volatile storage until the program or computer requests it for execution.

⇒ I/O devices like punch card readers, printers are (physical nature) slow hence we dont want them to interact with the CPU directly. For that we introduced memory.



Simultaneous as multiple input devices (peripherals) can send data to CPU (ie in secondary memory) at once!

~ many output devices can be present

Example: Printer (Read KG notes)

- There is no multiprogramming [CPU waits for processes to come back]

Why spooling?

machine → to → machine interaction [disk → CPU]

FASTER!

Multiprogramming OS → 1 PROCESS IN CPU AT A TIME

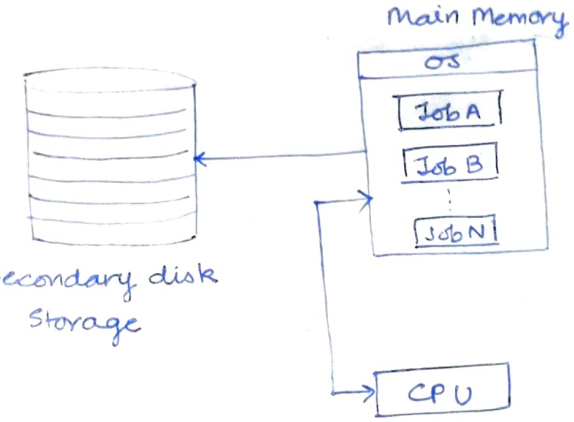
(Is NON-PREMPTIVE IN NATURE)

The main idea is to load multiple processes in the main memory & execute a process until it is executable, then pick the next process & so on so that the CPU/processor is NEVER IDLE.

Processor कहीं भी wait नहीं करेगा

1) Jobs are initially kept on the disk in Job pool. These processes in the pool await for allocation of main memory

2) For the jobs present in the main memory, CPU starts executing them one-by-one. If the job in CPU has to collect some data / perform any other operation, CPU switches to another job and so on.



Note:
It isn't like multiple jobs are being processed in CPU at once. At once, CPU works on one job only, multiple jobs are ready to be executed in the main memory.

Eg: CPU starts w/ Job A. After some execution Job A needs to perform some I/O operation, hence it leaves. So CPU starts executing Job B. Later Job A can come back complete its execution.

⇒ As long as at least one job needs to be executed, the CPU will be IDLE. THE SHOW MUST GO ON !!!

Advantages (1) High efficiency & CPU utilization (2) less response time or waiting time or turnaround time (3) Multiple tasks run in most apps, and multiprogramming sys help/handle these better. (4) Several processes share CPU time

Disadv.: (1) COMPLICATED SCHEDULING ALGO (2) COMPLEX MAIN MEMORY MANAGEMENT

FYI

Disk/Secondary memory loads data into (Main) Memory which then directly interacts w/ CPU

RAM

Why?

Secondary memory is V. LARGE. If CPU interacts w/ such large memory, it will slow down. Hence, we have MAIN memory which is v. small in size & small amt of data which is to be used now is only loaded into it. And CPU interacts directly w/ main memory.

more processes in MM ⇒ less chances of CPU being idle

(152) Multitasking OS → STILL ONE PROCESS AT A TIME

~ extended idea of Multiprocessing

- Also called Time sharing / Multiprogramming with Round Robin / Fair Share

Eg The execution/running of various programs like MP3 music, chrome, excel at the same time.

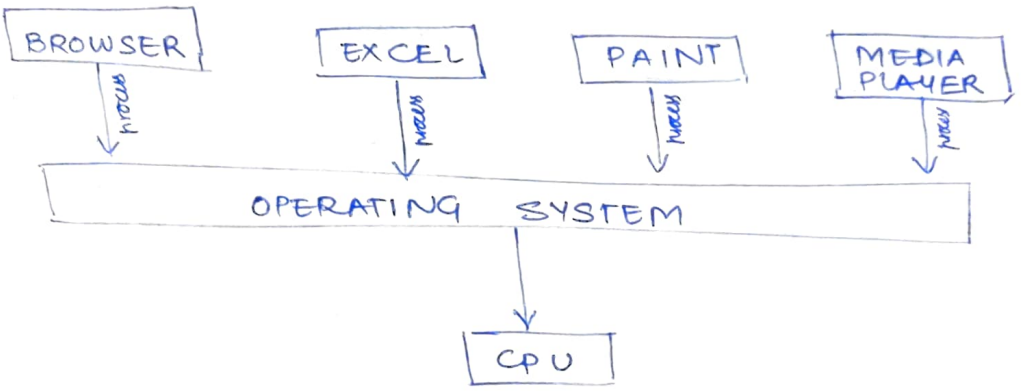
- Achieved by context switching, illusion of parallelism.

MAIN IDEA:

Time sharing (or multasking) is a logical extension of multiprogramming, it allows many users to share the computer simultaneously.

> The CPU executes multiple jobs by switching among them SO FREQUENTLY, that every user is given the impression that the entire sys is dedicated to his/her use.

Advantage over multiprogramming: BETTER RESPONSE TIME
(PRE-EMPTIVE IN NATURE)



Multitasking

For multitasking to take place, firstly there should be multiprogramming ie multiple programs ready for execution. And secondly the concept of time sharing.

1.5.2 Multitasking OS → STILL ONE PROCESS IN CPU AT A TIME

~extended idea of Multiprocessing

- Also called Time sharing / Multiprogramming with Round Robin / Fair Share

Eg The execution/running of various programs like MP3 music, chrome, excel at the same time.

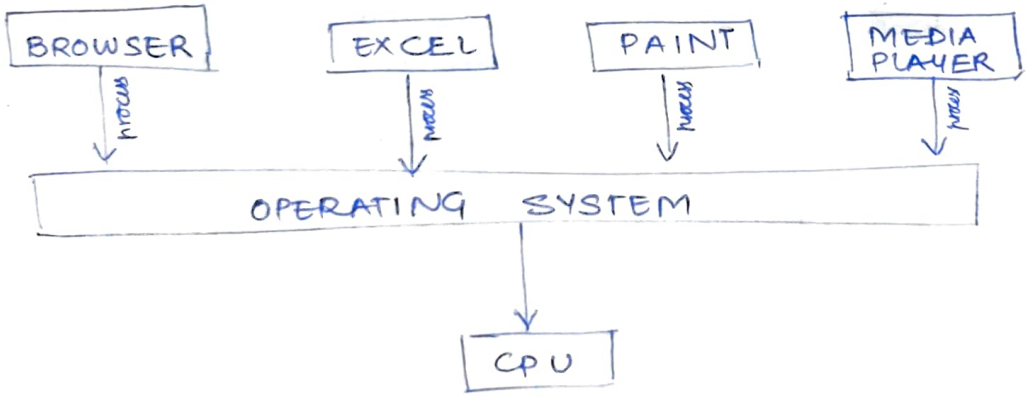
- Achieved by context switching, illusion of parallelism.

MAIN IDEA:

Time sharing (or multasking) is a logical extension of multiprogramming, it allows many users to share the computer simultaneously.

> The CPU executes multiple jobs by switching among them SO FREQUENTLY, that every user is given the impression that the entire sys is dedicated to his/her use.

Advantage over multiprogramming: BETTER RESPONSE TIME (PRE-EMPTIVE IN NATURE)



Multitasking

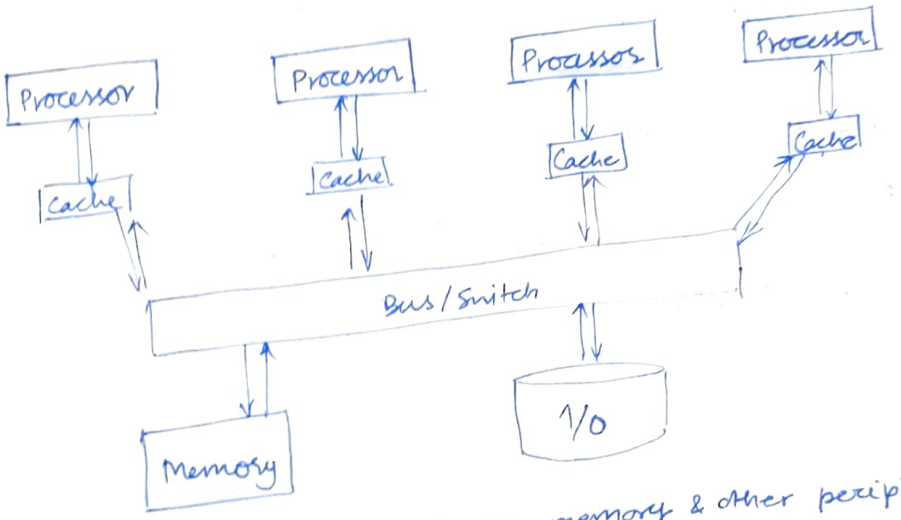
For multitasking to take place, firstly there should be multiprogramming ie multiple programs ready for execution. And secondly the concept of time sharing.

1.5.3 Multiprocessing OS ~ reverse of multitasking → MULTIPLE PROCESS AT A TIME IN CPU

- Also called tightly coupled system.
- Multiprocessing OS refers to two or more CPU within a single computer system.
- Multiprocessing in reverse fashion.
- So in multiprocessing OS, multiple processes each run on seperate CPU, we achieve a true parallel execution of processes.

↓

Parallel processing is ability of CPU to simul. process incoming jobs.



- Shares sys resources like system buses, memory & other peripheral devices
- Used when complexity of job is high & CPU divides & conquers.
  Eg. artificial intelligence, weather forecasting, image processing etc. ([Complex systems])

1.5.3.1 Types of multiprocessing OS

(1) Symmetric multiprocessing: No-boss-worker relationship, all processors are =
All copies commun. w/ each other → single shared memory
Same OS instance

- Used in most motherboards these days.

(2) Asymmetric mp: Master-slave relationship

- Eg. A particular CPU will respond to all hardware interrupts, whereas other work will be ÷ed among the CPUs ÷ly. Or execution of kernel-mode code may be restricted to 1 CPU, other ÷ed in user-mode.
- LESS EFFICIENT! bcz of restrictions but easy design.

Advantages of Multiprocessing OS

1) Increased throughput :

Throughput is units of info a system can process in a given amt of time.

2) Economy of Scale : 4 computers price >>> 1 computer w/ 4 processors. price >> 4 seperate Computers

3) Increased Realiability (fault tolerance)

1 Processor Shut down <<< 1 of 4 processors shut down.

4). less battery consumption & heat generation

Small task → Run where 1 big processor >>> much heat & power used than Small task → Run 1 of 4 small processors.

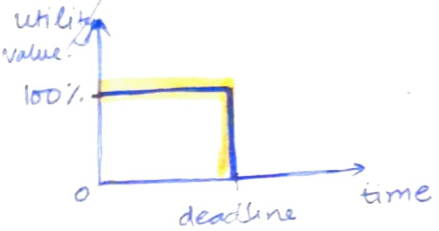
Disadvantages of MP OS

1) More complex
2) Requires content switching which may impact performance
3) If any one processor fails, it will affect performance
4) Large main memory reqd.

1.6 **Real-time OS**

(Time sharing OS)

It is a time-bound system which needs to process tasks within defined time constraints otherwise system will fail or task will fail.



Very quick & fast respondent sys.

> Used in environments where large no. of events must be processed in SHORT TIME, w/o buffer delays.

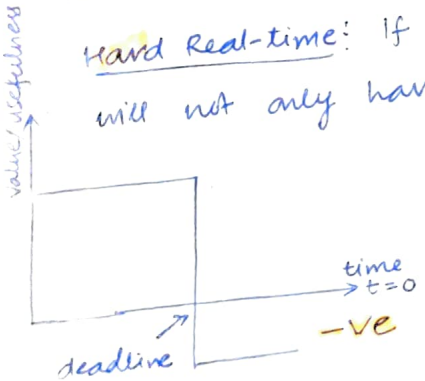
> Example: Air traffic contrl, stock market, defense sys like RADAR, (ATC) hospital sys.

> Most tasks remain in main memory for quick execution hence less complex memory maga^ is reqd.

1.6.2 **Types of Real-time OS**

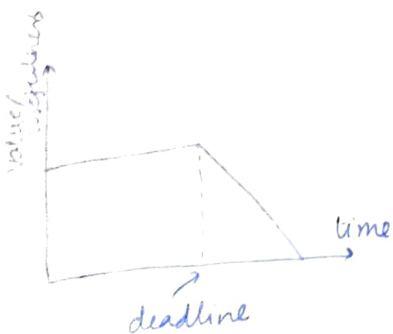
Hard Real-time: If the task isnt completed within deadline, it will not only have no use, but might have -ve effects too.

Example: air bag ctrl, anti-lock brake



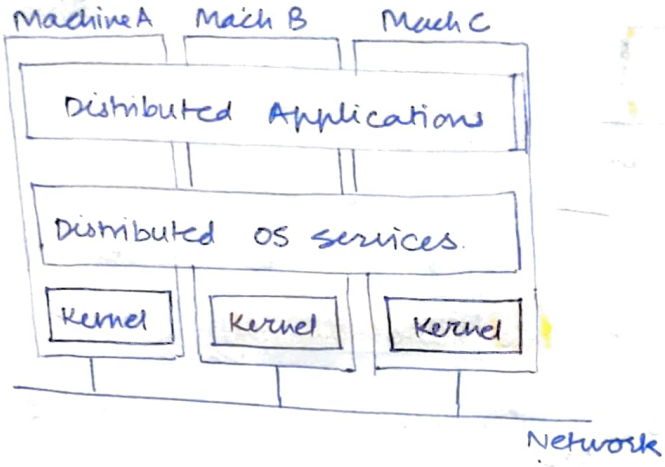
Soft Real-time: A system which is not contrained to extreme rules. The critical time of these systems can be delayed to some extent.

Example: Digital camera, online games like PUBG, mobile phones etc.



1.7 Distributed OS

It is a software (used as a single resource) over a collection of independent, networked, communicating, loosely coupled nodes & physically seperated computational nodes.



> User feels they are working on single computer, monolithic operating system.

> The computer distributes the extra load over other systems.

> Four major reasons for building DOS:

(i) Resources sharing
(ii) computation speed up
(iii) reliability
(iv) communication

Eg. Inferno ~~from~~, Plan 9 Bell Labs.

INTERFACE B/W USER & OS (interface b/w use & HARDWARE)

1.8 CLI V/S GUI ; Interface b/w User & OS

CLI → Command Line Interface

eg: In Unix or Linux user may choose among several shells: Bourne shell, C shell, Bourne-Again shell, Korn shell ... etc

→ exact action

GUI → Graphical User Interface

Touch / Mouse
- coordinate (x, y)
- mic → action

→ Easy

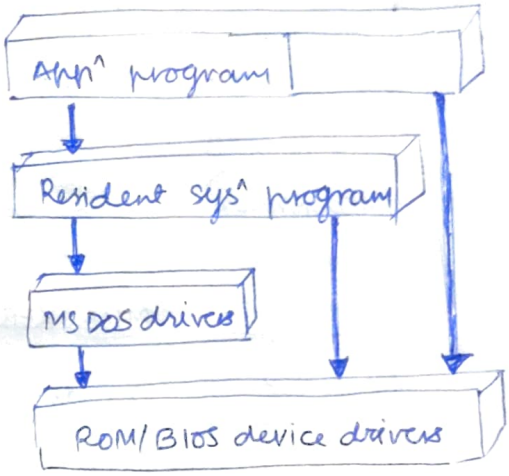
Better based on personal preferance

19 Structure of OS

→ Modularity : Every sys/task is ±ed into modules rather than 1 monolithic system. Each module is well defined.
coupling cohesion

(A) Simple structure : Don't have a well-defined structure.
Started small & grow beyond their orignd scope.

Eg MS-DOS.

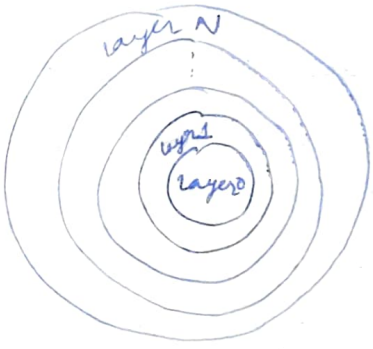


(B) Layered structure : Proper hardware support, OS broken into pieces. Much greater control

1) Top-down approach
2) Has LAYERS
3) Info hiding.

Disadv : Complex ! Becz of layer
Have to follow approach to do anything

Eg : OS/2 (not popular)

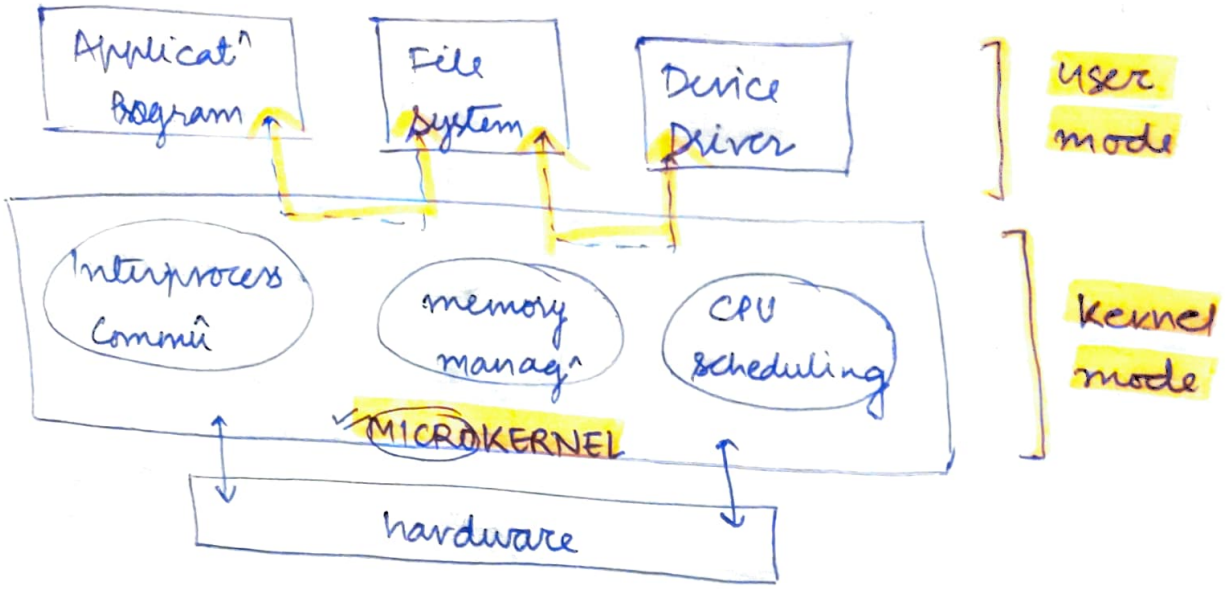


A layered OS

1.10 Micro-Kernel Approach

→ the size of OS starts incr with the no. of its responsibilities Hence we ÷e

The method in which all non-essential components of OS are removed from the Kernel & implemented as a system and user-level programs.



- First used in 1980's by Carnegie Mellon University. Developed an OS called Mach.

Advantages

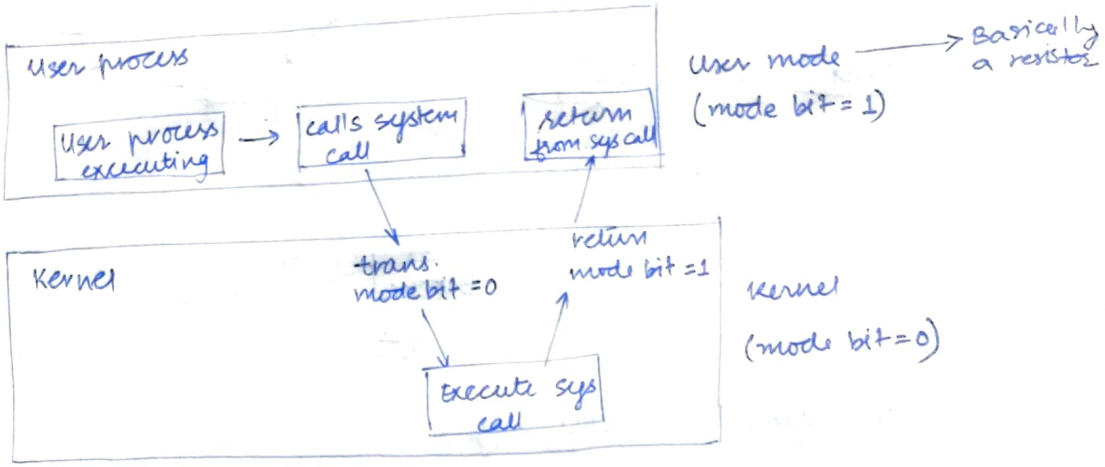
1. It is easy to make changes in the kernel becz of modulation of microkernel (microkernel is a smaller kernel)

eg. MINIX 3 microkernel is written in only 12000 lines of code by Andrew S. Tanenbaum.

1.11 Mode bit

Modes of operation : User-mode & kernel mode
(also called supervisor mode or system mode/privileged mode)

A BIT CALLED MODE BIT is used to indicate the current mode: Kernel (0) or User (1).



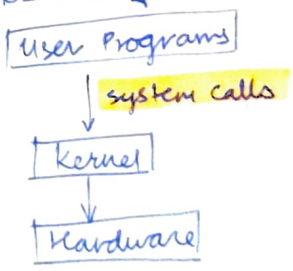
Since control of Hardware devices is with OS, we use mode bit. User prog. requests OS to execute tasks on its behalf (sensitive tasks)

When the computer system is executed on behalf of a user application, the sys is in user mode. However when a user application requests a service from the OS (via a system call), the system must transition from user mode to kernel mode to fulfil the request.

1.12 System Call

System calls provide the means for user program to ask (request) the OS to perform tasks reserved for the OS on the BEHALF of USER PROGRAM.

{ user programs cant be directly related to the hardware hence OS performs tasks on its behalf }



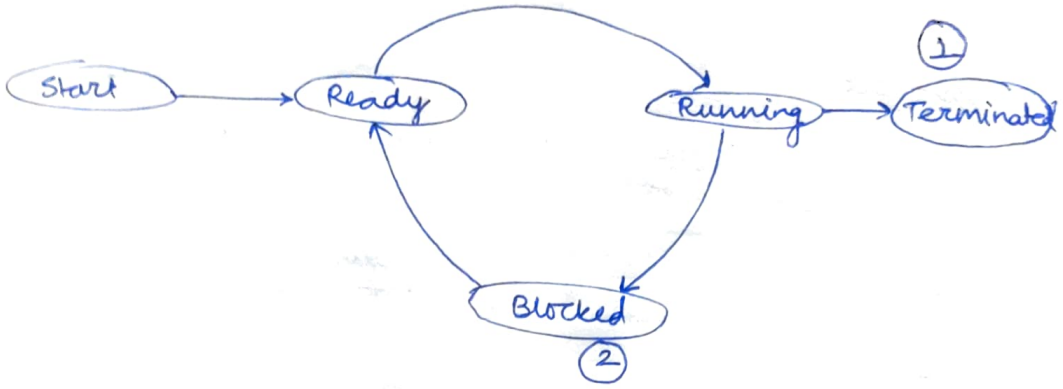
> OS provide certain kind of "menu" for the system calls that can be made thru APIs

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

- generally written in C or C++.

(3.1) CPU scheduling : Pre-emptive V/S Non-pre-emptive.

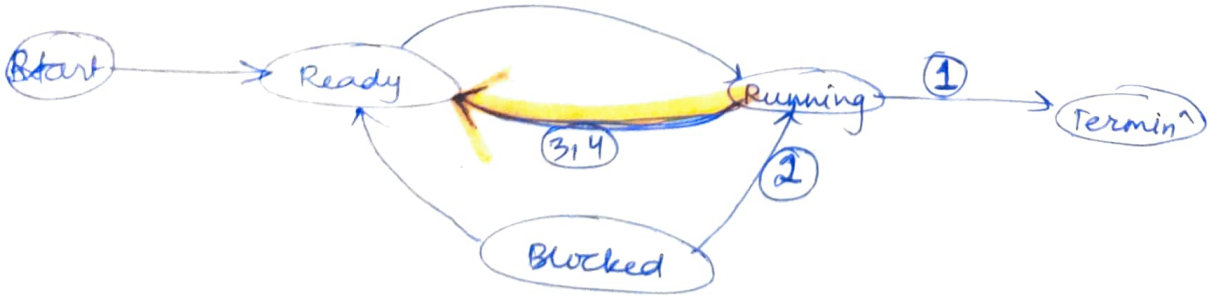
Non-pre-emptive : once a process has been allocated the CPU, the process keeps running until it releases the CPU willingly.



A process leave the CPU only

- when a process completes its execution (Termination state)
- or a process wants to perform i/o oper^n (Blocked state)

Pre-emptive : A process can leave the CPU willing or can be forced out for a higher priority process.



So a process leaves the CPU if :

(1) it completes its execution (Terminated)

(2) leave CPU voluntarily to perform i/o oper (Blocked)

(3) A higher priority process enters ready state, then the process will be forced in ready state

(4) When a process switches from running to ready state bec of time quantum expire

3.2 Scheduling criteria

(1) **CPU utilization**:

CPU should be as busy as possible no matter the output.

(2) **Throughput**:

Doesn't matter how long the CPU works/ how busy it is, the output should be high.

Throughput is the measure of work = $\frac{\text{No. of processes completed}}{\text{unit time}}$

(3) **Waiting Time**

Based on the time a process has to wait.

(4) **Response Time**

Based on the time CPU takes to respond to you, like in Round Robin.

Note:

CPU scheduling algo doesn't affect the amt of time during which a process executes I/O.

IT ONLY AFFECTS THE AMOUNT OF TIME THAT A PROCESS SPENDS WAITING IN THE READY QUEUE.

- It is desired to maximize CPU utilization & throughput & to minimize turnaround time, waiting time & response time.

3.3 Terminology

(1) **Arrival Time (AT)**: Time at which process enters a ready state

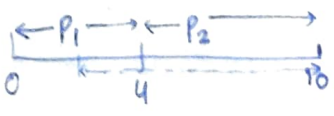
(2) **Burst Time (BT)**: Amt of CPU time a process needs for execution

(3) **Completion Time (CT)**: Time at which process finishes execution

(4) **Turn Around Time (TAT)**: (CT − AT) or (WT + BT) // The time a process stayed in the CPU
(WT → waiting time)

(5) **Waiting Time**: (TAT − BT) // The amt of time the process stayed in the sys − the actual time needed for execution

Process	AT	BT	CT	TAT	WT
P_1	0	4	4	4−0=4	4−4=0
P_2	2	6	10	10−2=8	8−6=2

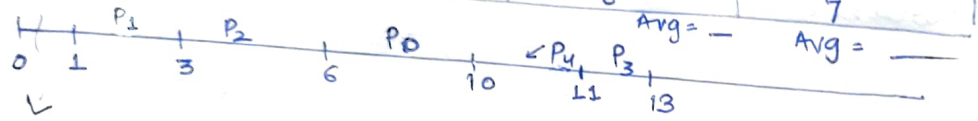


(3.4) First Come First Serve (FCFS)

- Simplest Scheduling algo
- FIFO ~ Queue : process that requests CPU first is allocated the CPU first
- It is NON-PRE-EMPTIVE in nature.

Q.

P.No	AT	BT	CT	TAT = CT - AT	WT = TAT - BT
P_0	2	4	10	8	4
P_1	1	2	3	2	0
P_2	1	3	6	5	2
P_3	4	2	13	9	7
P_4	3	1	11	8	7
				Avg = —	Avg = —



No process

Advantages of FCFS

1) Easy to understand & can be easily implemented using Queue DSa.
2) Can be used for Background processes, where the execution is not urgent.

Disadvantage : (1)

	AT	BT	TAT	WT
P_0	0	100	100	0
P_1	1	2	101	101 - 2 = 99

Avg = 49.5s

If Small process is executed first, large process won't be affected much but the vice-versa is not true.

	AT	BT	TAT	WT
P_0	1	100	101	1
P_1	0	2	2	0

Avg = 0.5s

Sol^n: Process smaller process first !!

CONVOY EFFECT

→ When smaller effect has to wait bec of larger process
⇒ More avg waiting time.

Dis(2): Trouble-some w/ time-sharing systems
(3) Higher avg waiting time & TAT compared to other algo.

(3.5) Shortest Job First (SJF) [Non-pre-emptive]

Decision for selecting the next process for CPU:

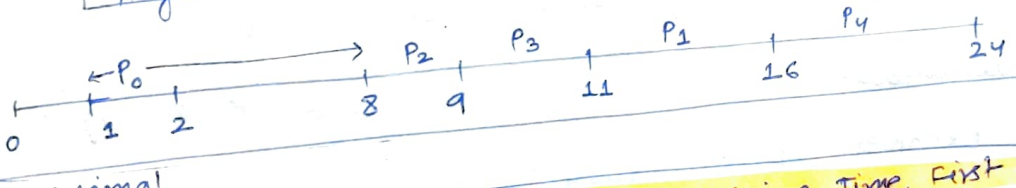
Pick the **SMALLEST BURST TIME**.

[If there's a tie it is solved using FCFS]

Non-pre-emptive approach: Once a job is in the CPU, next job (WITH SMALLEST BURST TIME among all AVAILABLE processes) will only be executed after its complete.

Q

P.No.	AT	BT	CT	TAT	WT
P_0	1	7	8	7	0
P_1	2	5	16	14	9
P_2	3	1	9	6	5
P_3	4	2	11	7	5
P_4	5	8	24	19	11
Avg.					6



most optimal

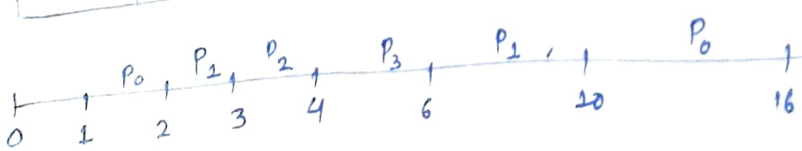
Pre-emptive Approach: **Shortest Remaining Time First (SRTF)** or Shortest Next CPU Burst.

Can contact miss to execute a process which has smaller CPU Burst time

P	AT	BT	CT	TAT	WT
P_0	1	~~7~~ 6	16	15	9
P_1	2	~~5~~ 4	10	8	4
P_2	3	1	4	1	0
P_3	4	2	6	2	0
P_4	5	8	24	19	11
					24/5 ≈ 5 or (4.8)

Of all pree. algos

most optimal



Same as Non-pre-emp. as BT is same

SRTF Advantage & Disadvantage

Adv

1. Pre-emptive gives OPTIMAL AVG WAITING TIME
   Hence called OPTIMAL ALGO.
   → Purely greedy.
   ∴ This provides standard for other algos wt
2. Provides better response time compared to FCFS. (Best Round robin)

Dis

1. THIS ALGO CANT BE IMPLEMENTED as there is no way to know the length of next CPU burst
2. Here process with longer CPU burst time will go into STARVATION.
3. No idea of priority, longer process has poor response time
   → can be system operation, imp process

SRTF Implementation

Formula: EXPONENTIAL AVERGING TECHNIQUE.

$$T_{n+1} = T_n \alpha + (1-\alpha) t_n \qquad \alpha \simeq 0.5$$

(prev estima" T_n, Actual Score t_n)

P	ST	t
P_1	10	20
P_2	12	15
P_3	14	13.5
P_4	x	13.75

This way we are trying to predict next burst estimate time.

Estimating time for 1 Process

∴ Tho this idea is theoretical

6. Non-pre-emptive Priority Scheduling

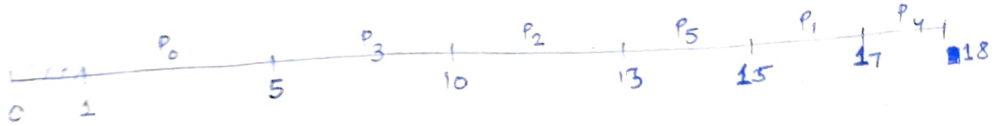
CPU Scheduling based on PRIORITY of processes of all the AVAILABLE processes.

⇒ Tie breaker: FCFS

No importance to burst time.

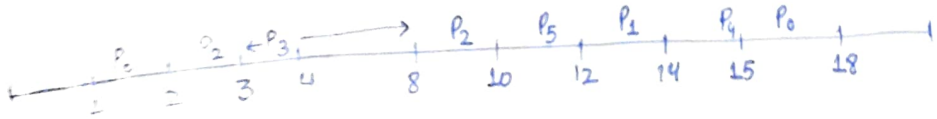
P	AT	BT	Priority	CT	TAT	WT
P_0	1	4	4	5	4	0
P_1	2	2	5	17	15	13
P_2	2	3	7	13	11	8
P_3	3	5	8(H)	10	7	2
P_4	3	1	5	18	15	14
P_5	4	2	6	15	11	9
						46/5 ≈ 9.2

For this Ques:
8(H)
Higher the no. higher the priority.



[Pre-emptive]: Based on priority.
Just pull out the running process incase a process w/ higher priority is encountered.

P.no	AT	BT	Prio.	CT	TAT	WT
P_0	1	4 3	4	18	17	14
P_1	2	2	5	14	12	10
P_2	2	3 2	7	10	8	6
P_3	3	5	8(H)	8	5	0
P_4	3	1	5	15	12	11
P_5	4	2	6	12	8	6
						47/5 = 9.4



Advantages of Priority Scheduling

1) Gives a facility specially to SYSTEM PROCESS.
   (Bcz of which CPU/System works nicely)
2) Allows us to run important process even if it is a user process.

Disadvantages

1) Here process with smaller priority may Starve for CPU.
2) No idea of response time / waiting time.

Note:

To prevent **Starvation**:

Specially for SYSTEM PROCESS or important user process

Ageing: a technique for gradually increasing the priority of processes that wait in system for long time. Eg. priority will be after every x mins.

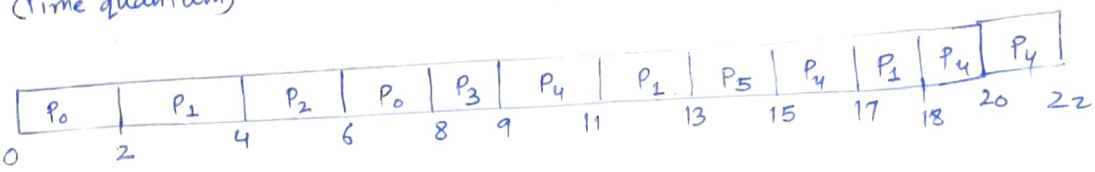
Round Robin Scheduling

1) Imp for **TIME SHARING** systems, where it is imp to be responsive & divide CPU time.
2) Uses a ready queue in FIFO fashion.
3) ALWAYS PRE-EMPTIVE!
4) CPU is allocated according to Time quantum (T.q) → Fixed time for context switch. (TQ can be any value)

Q

P.No	AT	BT	CT	TAT	RT (Response Time)	WT
P_0	0	~~4~~ ~~2~~	8	8	0	4
P_1	1	~~5~~ ~~3~~ ~~2~~	18	17	1	12
P_2	2	~~2~~	6	4	2	2
P_3	3	~~1~~	9	6	5	5
P_4	4	~~6~~ ~~4~~ ~~2~~	22	18	5	12
P_5	6	~~3~~ ~~1~~	15	19	7	16
					= 20/8	= 43/6

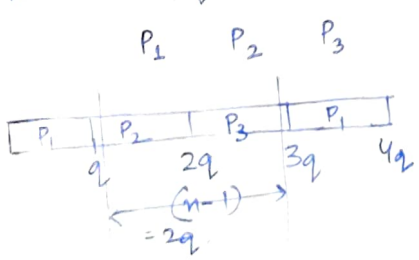
T.q = 2 (Time quantum)

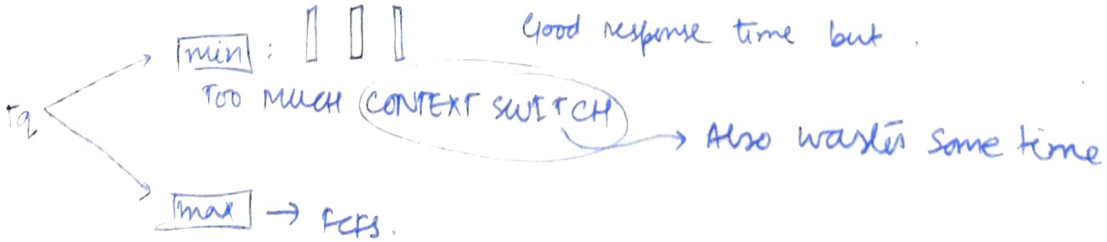


Ready Queue : $P_0\ P_1\ P_2\ P_0\ P_3\ P_4\ P_1\ P_5\ P_4\ P_1\ P_4$

Note:

- If there are n processes in the ready queue, q ≡ time quantum
  Each process gets 1/n of CPU Time in chunks of at most q time units.
- Each process must wait NO LONGER THAN (n-1) × q time units until next time quantum.





- Need to choose the apt time quantum!

Advantage

1) Perform best in terms of avg RESPONSE TIME.

2) Works in case of TIME-SHARING systems, CLIENT SERVER ARCHITECTURE & interactive system.

3) Kind of SJF IMPLEMENTATION

- Ultimately the shorter tasks will get executed early.
- lesser response time due to content switch leading to lesser waiting time for smaller processes.

Disadvantage

1) Longer processes will STARVE
2) Performance based on TIME QUANTAM
3) No idea of priority.

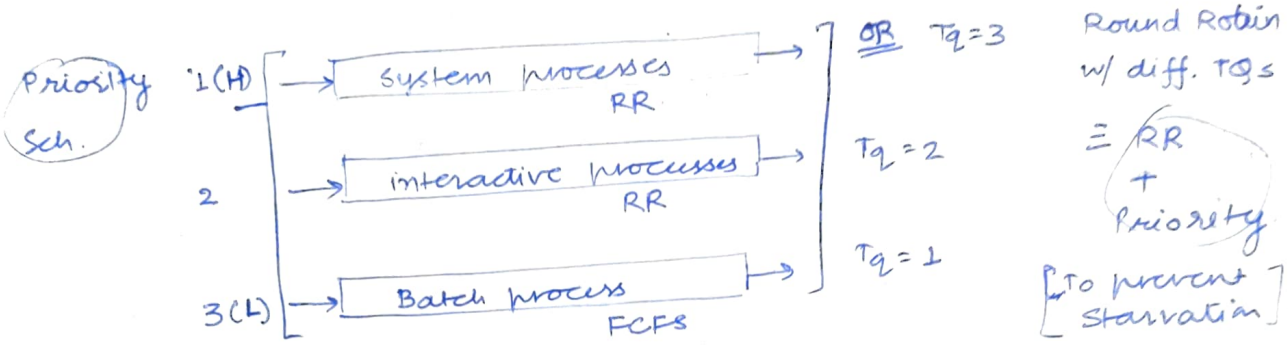
(3.8) Multi-level queue Scheduling

- For processes that can be easily ÷ed into groups

  Example: Foreground (interactive) processes & background (batch) processes && System processes.

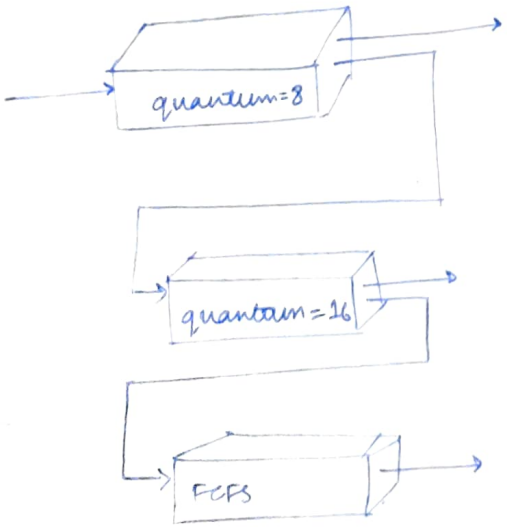
- A multilevel queue sch. algo partition can be made for ready queues.

  - processes are permanently assigned to one queue based on some property of process; memory size, priority, type etc.



Different Ready Queues.

(3.9) Multi-level Feedback queue Scheduling



Complete here (8tq) (EXIT) else ↓

Complete here (24tq) (EXIT) else ↓

Ultimately longer processes will go into FCFS.

Adv

- No segregation needs to done. (segregation can be stubborn)
- SCFS nearly

Here changing queue is possible !!

Note: Can use round robin or ageing (allows changing queues) b/w the queues so that the processes in FCFS dont starve

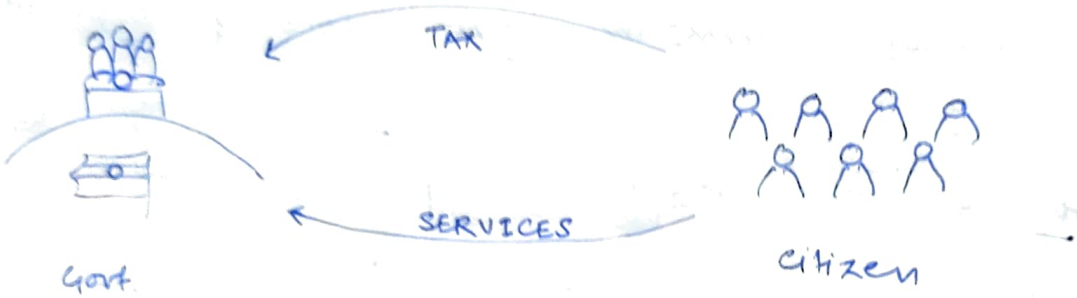
5.1 Basics of Deadlock

In multiprogramming environment, several processes may compete for FINITE NUMBER OF RESOURCES.

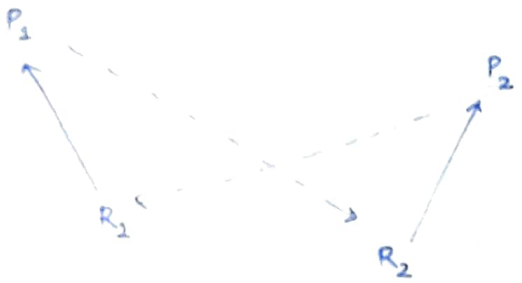
→ In this scenario, there is a possibility of Deadlock

Examples:

1)



2)



P_1 is waiting for P_2 to complete & P_2 is waiting for P_1 to complete

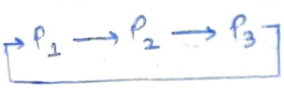
When a resource is not available, process enters waiting state. Sometimes, a waiting process is never again able to change bec the resource it has req. is held by another waiting process. This sit. is called a DEADLOCK.

⇒ STARVATION is LONG waiting but DEADLOCK is ∞ waiting.

5.3 Necessary Conditions for Deadlock

A deadlock can occur if all these 4 conditions occur in system simultaneously

- **MUTUAL EXCLUSION** → at least one resource held in non-sharable mode eg. one printer in library
- **HOLD & WAIT** → a process must be holding at least one resource & waiting to acquire additional resources held by other processes
- **Non pre-emption** → a resource can only be released VOLUNTARILY, after it completes its execution. Every process has same rights
- **Circular wait**



5.4 Deadlock Handling methods

1) Prevention: Design protocols that there is no possibility of deadlock

2) Avoidance: Try to avoid deadlock in run time, so ensuring that the system will never enter a dealock.

3) Detection: We allow the system to enter a deadlock state, then detect it & recover. (when the problem is less severe & frequent)

4) Ignorance: We can ignorance the problem altogether & pretend that deadlocks never occur in the system.

5.4.1 System Model

Usually, a process utilizes a resource in the following sequence:

Request: The process requests the resource.

→ If the resource cant be granted (is in use) the requesting process must wait until it can acquire the resource.

Use: The process can use resource

→ eg. a process requested a printer, the process can print on the printer now

Release: The process releases the resource immediately.

5.4.1 Prevention — NO POSSIBILY of DEADLOCK.

if we resolve even one of the four condns of Deadlock, the problem will be solved :

* Mutual Exclusion : No solx for Mutual Exclusion in prevention as resource cant be made shavable, as it is hardware property (& process also cant be convinced to do some tasks).

In general, we cant resolve deadlocks by denying mutual exclusion condn, becz some resource are NON-SHARABLE.
eg. Printer is not sharable.

✓ Hold & Wait.

can be solved by various methods like these :-

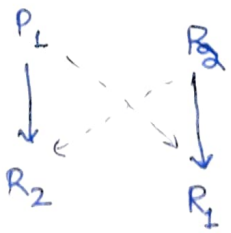
(1) In conservative approach, process is allowed to run if & only if it has acquired all the resources.

(2) A process may request some resources & use them. Before it can request any additional resources, it must release all the resources that it is currently allocated.

Jitna resource se kam nora hai pehle vah karo jiss aur resource chahiye to pehle jiss wali release karo.

Here P_1 may use R_2 tills it can, but asa it wants R_1 too, it will first have to release R_2.

Result: It might get both R_1 & R_2 or it may loose both.



Note: Here a process can starve but.

Starvation >> Deadlock.

(3) Wait time outs, we place a max time out up to which a process can wait. After which process must release all the holding resources & exit.

✓ No Pre-emption

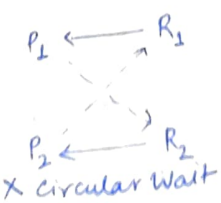
We can pre-empt resources to break deadlock. But we can acquire resources iff:

(1) The process from which we are taking away the resources is in waiting stage.
(Can't / should not acquire resources from processes in running state)

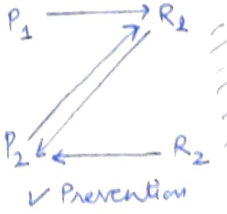
(2) Set a priority list from the process which can pre-empt resources from other resources.
eg. process of OS should be able to run first & acquire resources.

✓ Circular Wait

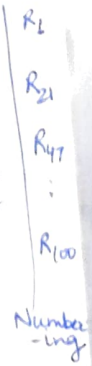
In order to prevent circular wait to cause deadlock:
We can give a ① natural number mapping to every resource & then any ② process can request only in ↑ing order
• If a process wants lower number, the process must first release all the resources larger than that number & give a fresh request.



X circular Wait



✓ Prevention



• Problem w/ Prevention

Prevention techniques put some kind of restrictions / condns on the processes & resources.
Because of this the system becomes slow & resource utilization & system throughput aren't optimized.

While deadlock is not common & even if it happens it isn't deadly. Hence, prevention becomes expensive. We can rather use liberal approaches like avoidance.

5.4.2 Avoidance

To avoid deadlocks we require additional information about how resources are to be requested.

With this additional information, the OS can decide for each request whether process should wait/not.

BANKER'S ALGO

In order to avoid deadlock in run time, system must try to maintain some books like a banker, whenever someone asks for a loan (resource), it is granted only when the books allow.

MAX NEED	E	F	G
P_0	4	3	1
P_1	2	1	4
P_2	1	3	3
P_3	5	4	1

−

ALLOCATION	E	F	G
P_0	1	0	1
P_1	1	1	2
P_2	1	0	3
P_3	2	0	0

=

CURRENT NEED	E	F	G
P_0	3	3	0
P_1	1	0	2
P_2	0	3	0
P_3	3	4	1

SYSTEM MAX		
E	F	G
8	4	6

AVAILABLE		
E	F	G
3	3	0

5 3 4

[ ] : Given data

[ ] : Can be given data to be calc

Consider a situation: All processes demand "current Need" to run further

1. $[P_0$ & $P_2]$ are two such processes whose "current Need" can be fulfilled using "Available" resources.

   Note: The sequence doesn't matter in deadlock

2. Suppose we run P_0 first.

   Now, if we satisfy the "current Need" of P_0, P_0 will have to run, P_0 will have to execute. (Even the more need is satisfied).

   "Execute" implies it will have to give up all its acquired resources

Hence, if we believe we can run P_0 with the "Available" resources, we can simply add the "Allocation" to "Available".

We results in :-

Available		
E	F	G
3	3	0

$+ \quad 1 \quad 0 \quad 1$

$= \quad 4 \quad 3 \quad 1$

3. We the current situation, we check if we can run P_1.

= NOT POSSIBLE

4. We try P_2 (RUN).

~ P_0

Available		
E	F	G
4	3	1

$+ \quad 1 \quad 0 \quad 3$

$= \quad 5 \quad 3 \quad 4$

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

∴ System is safe

5. check & run P_1

Available		
E	F	G
5	3	4

$+ \quad 1 \quad 0 \quad 2$

$6 \quad 4 \quad 6$

There exist a seq. in which we can run & exit the CPU with the available resources.

Such a seq is known as safe sequence.

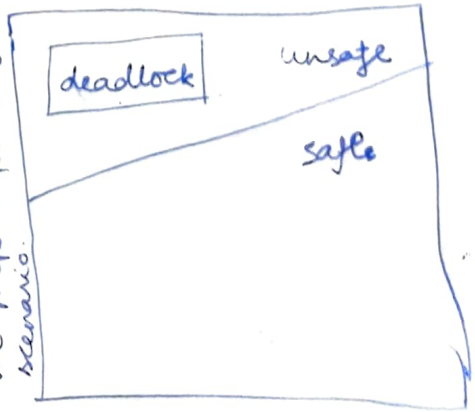
6. Now run P_3

Available		
E	F	G
6	4	6

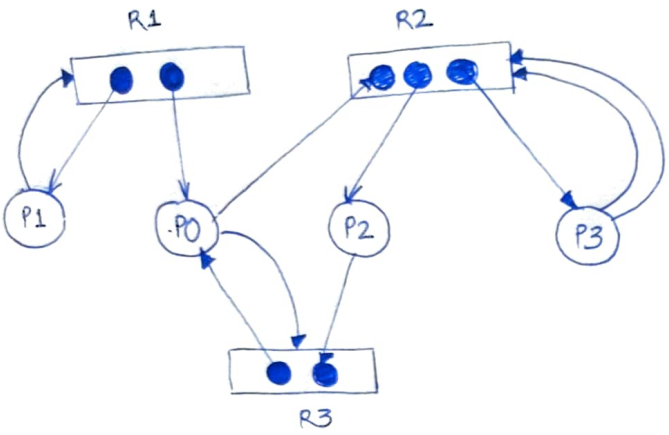
$+ \quad 2 \quad 0 \quad 0$

$8 \quad 4 \quad 6$

Even if there exist one safe seq., the system is safe In unsafe too, there is a possibility of deadlock, we prepare for worst scenario.



Resource Allocation Graph



: a box repⁿ a TYPE of resource

: Number of that resource

: Allocation

: Request

: Process

- No cycle in resource allocation graph is necessary but not sufficient condn for detection of deadlock.
- If every resource have only one resource in the resource allocation graph then detection of cycle is necessary & sufficient condition for deadlock detection.

5.4.3 Detection & Recovery

- Beca not every problem is worth solving
- we dont check safety and resources are imm allocated to the processes which request them, if available.
- If there is a possibility of deadlock, it is detected using 2 different approaches :-
  - (1) Active approach → "regular checkups". eg. every hour, when CPU utiliz is below 40 per.
  - (2) Lazy approach → when CPU utilization is below 40 per or some unusual performance is there, we go for detection

Methods of Recovery

(1) Process Termination

(i) Terminate all processes : Prev. work is lost 'disadv'

(ii) Terminate one by one

(iii) Preempt resources one by one. (Partial Kill)

Pick one process → Selection based on workdone so far or no. of resources etc...

(2) Resource Pre-emption

5.4.4 Ignorance / Ostrich Algo

Here we believe there is no such concept as "Deadlock".

Hence if the sys hangs we restart it etc.

⇒ We are ignoring the problem, beca we feel it is not worth solving.

~ To an ostrich held by hunter: Head in sand

PROCESS MANAGEMENT

2.1 Basics of a Process

A program in execution is known as process

- Program: is a set of instructions.
- Process: When OS permits a program to run/execute. Hence, a process is a program in execution. (executing sequence)
- PCB (Process ~~out~~ control Block): All info of a process is stored in its PCB.
  - → PCB = confirmation that OS has accepted the request from the program.
  - → No PCB = No Process

Program	Process
1) Program is a passive entity. ie a file containing list of instructions stored on disk (often called executable file)	1) Process is a active entity. which requires resources like main memory, CPU time, registers, system buses etc.

2) A program becomes a process when an executable file is loaded into main memory & when its PCB is created.

3)

P_i = Process

Even if two processes may be associated with same program, they will be considered as two seperate execution seq. & are totally different process.

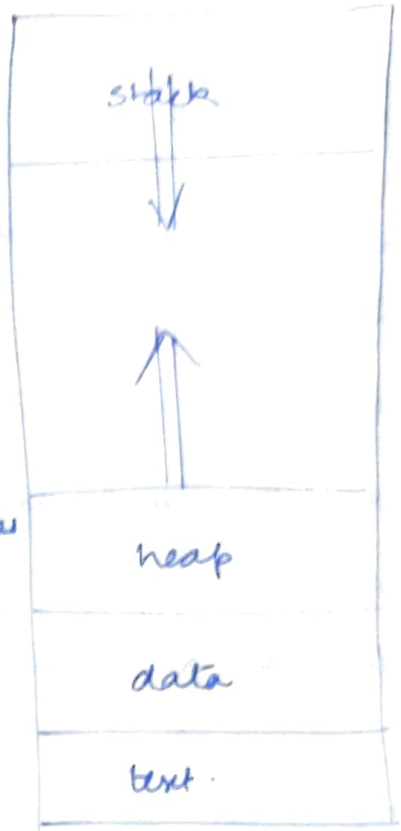
A process consists of the following sections :-

(1) Text Section :- also known as Program Code / Section.

(2) Stack : which contains the temporary data (Function Parameters, return addresses & local variables).

(3) Data Section : containing global variables

(4) Heap : which is memory dynamically allocated during runtime.



Upar se niche ki taraf grow karta hai !!

Both grow towards each other in Reverse Direction

So that memory utilisation is 100%.

eg. You start writing in a notebook from back & from front (2 subjects) so that no page goes wasted!

-> When two processes are associated w/ the same program, each copy is treated as seperate process. Even though the text section is same, but the data, heap & Stack sections can vary.

(2.2) Process Control Block (PCB)

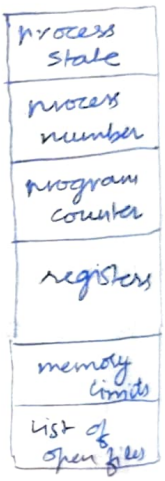
also called task control Block

- PCB serves as a repository for any info that may vary from process to process.

Parts of PCB:

(1) Process State: The state may be ready, new, running, waiting, halted, and so on.

(2) Process Number: A unique no given to every process.



(3) Program Counter: The counter indicates the address of the next instruction to be executed for this process.

(4) CPU registers: These include accumulators, index registers, stack pointers, and general purpose registers.

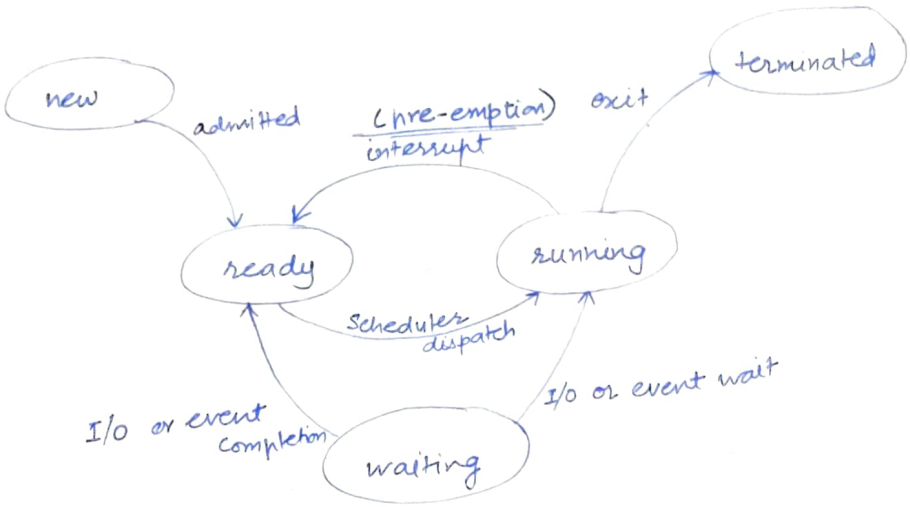
(5) CPU scheduling information: This info includes a process priority, pointers to scheduling queues and any other scheduling parameters.

(6) Memory-management info: This information may include such items as the value of base and limit registers and the page tables, or the segment tables...

(7) Accounting information: contains the amt of CPU & real-time used, time limits, account nos, job or process nos ... etc

(8) I/O status info: This info includes the list I/O devices allocated to the process, a list of open files.

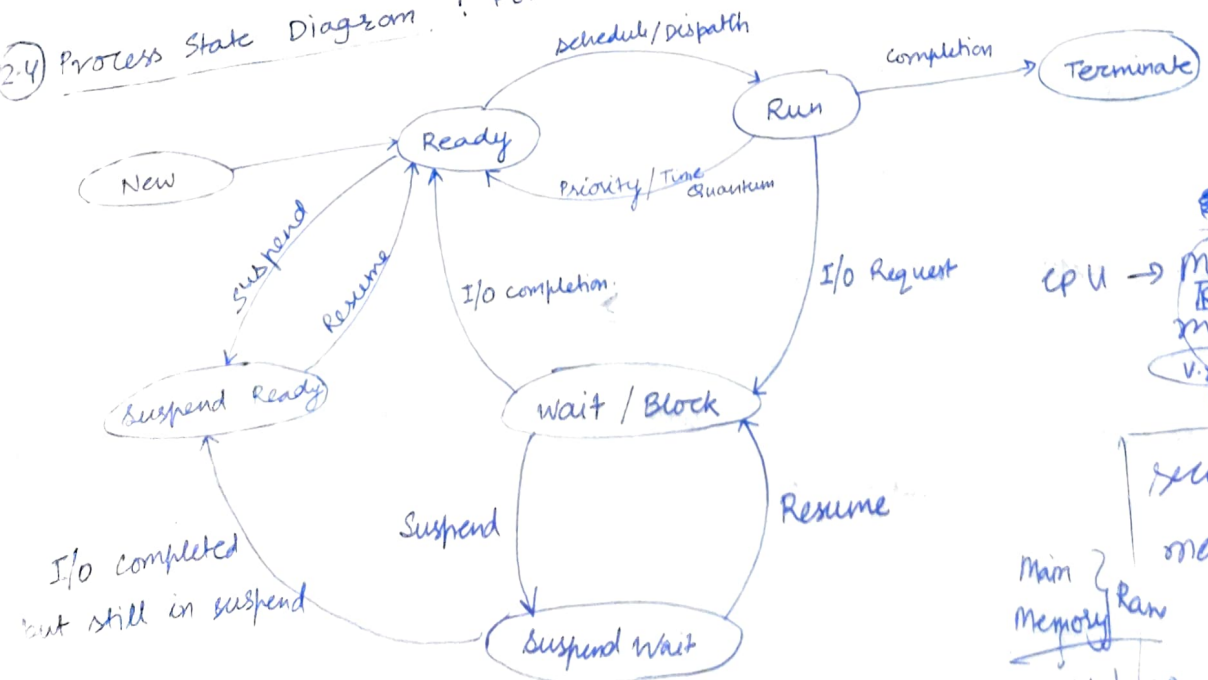
(2.3) Process life cycle



A process changes (states) as it executes. The states :-
Various STATES :

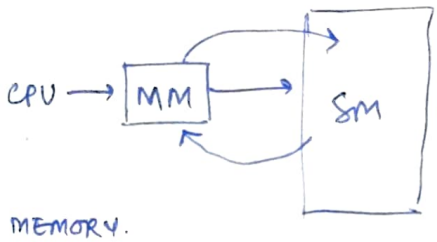
- New : The process is being created.
- Running : Instructions are being executed.
- Waiting (Blocked) : The process is waiting for some event to occur (such as an I/O completion or reception of a signal).
- Ready : The process waiting to be assigned to a processor.
- Terminated : The process has finished execution.

(2.4) Process State Diagram : For basic idea of schedulers!



CPU → main RAM memory
V. small
secondary memory
Main Memory Ram
Should have v. less load

1 Suspend Ready



Story :

1. CPU is directly connected to MAIN MEMORY.
2. Main memory is a really small memory which is supposed to have very less no. of processes.
   RAM: 4GB/8GB.
3. Secondary memory is a large memory space, like a backup ~ 1TB etc

So, when main memory is full or has a lot of load it suspends a few process which are even ready to run. Such state is known as Suspend Ready. Also known as swap-in & swap-out (when process goes back into the memory)

Suspend Ready

1. Ready to run by itself: Process
2. Suspended by OS

2. Suspend Wait

The process which are waiting or are in the waiting queue BUT TAKING UP SPACE OF THE MAIN MEMORY can also be victimised to suspension, hence suspend wait.

When they go back, they can go back in the waiting state / waiting queue.

3. Extra arrow suspend wait to Ready

When the event the process was waiting for is complete & it doesn't want to wait anymore but go into the Ready state the suspend wait → Suspend Ready

2.5 System Queues

To maintain the record of 'कौन process कौनसे state में था, कौन कहाँ से migrate किया, ..' we maintain System queues. [To keep a record / to keep things sorted]

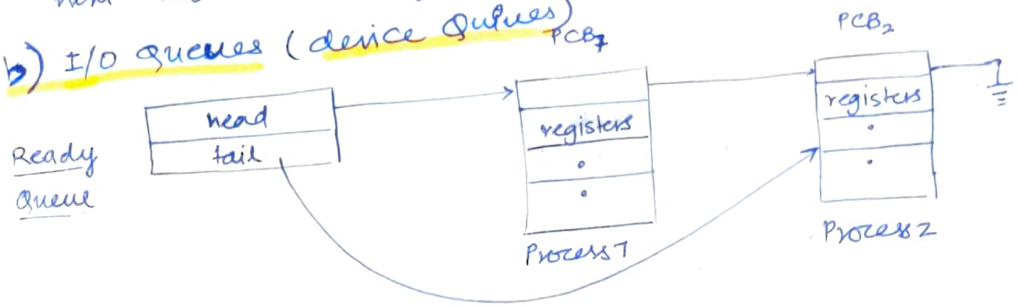
- **Scheduling Queues:** As processes enter the system, they are put into a job queue, which consists of all processes in the system.

a) **Ready-queue:** A process that is residing in the main memory and (to be in the ready state) CPU is ready to execute is kept as a list called Ready Queue.

This queue is generally stored as a ~~list~~ linked list.

⇒ A ready-queue header contains pointer to the first & final PCB in the list. Each PCB includes a pointer field that points to the next PCB in the ready queue.

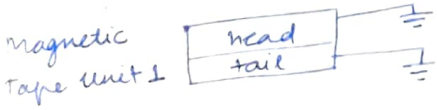
b) **I/O queues (device Queues)**

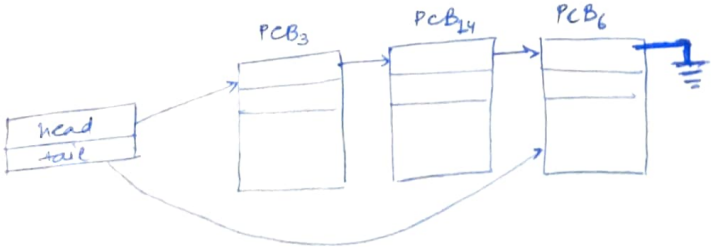


Process ↔ PCB

: No process in queue

: No process in queue.

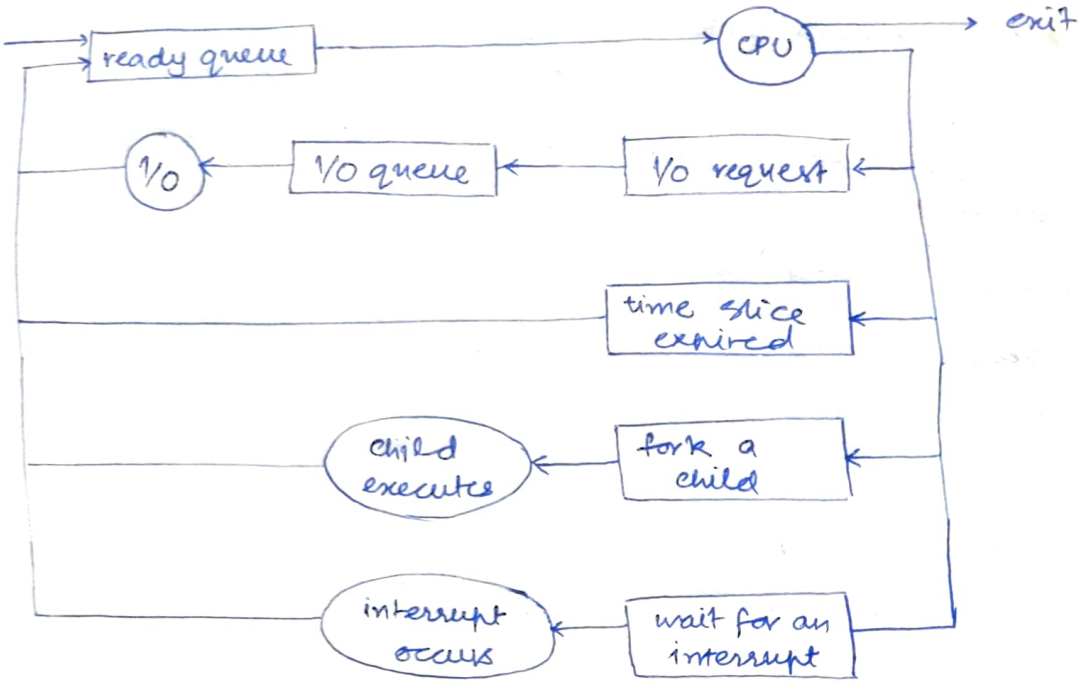




⇒ There is no need for running queue becz only 1 process runs at a time.

Queue-diagram representation of process scheduling

Each rectangle box represents a queue :-



Exp:

1) Initially a process enters ready queue
2) Then it may be allotted CPU according.
3) Then it can either
   - 3.1) EXIT : After execution
   - 3.2) ENTER OTHER QUEUES :
     - 3.2.1) Request for an I/O and enter the queue of that I/O device.
     - 3.2.2) Round Robin, time slice expired or time quantum exhausted.
     - 3.2.3) Thread?
     - 3.2.4) Preemption!
4) Then again enter the ready queue

- Two types of queues are present :
  1) The ready queue
  2) A set of device queues.
- The circles represent the resources that serve the queue.
- ~~The circles represent~~ A new process is initially put in the ready queue. It waits until it selected for execution or dispatched.

2.6 Schedulers

A process migrates among the various scheduling queue throughout its lifetime. The operating system must select (for scheduling purposes) processes from these queue in some fashion. The selection process is carried out by the appropriate scheduler.

Types of Schedulers

1. Long Term Schedulers (LTS) / Spooler: In multiprogramming OS, more processes are submitted than can be executed immediately. These processes are spooled to a mass-storage device (typically a disk), where they are kept for later execution. The long-term scheduler, or job scheduler, selects processes from this pool & loads them into memory for execution.
   - low frequency (of work)
   - Decides the no. of processes in MM at runtime.
   - touches process only once
2. Short Term Scheduler (STS): The short-term scheduler, or CPU scheduler, selects from among the processes that are ready to execute & allocates CPU to one of them.
   - High frequency: more frequently has to select processes for the CPU.

a) कोई Process secondary memory से main memory mein kab jayega से LTS / Spooler decide karta hai!

- maintains 'the degree of multiprogramming' : थोड़ी-थोड़ी देर में कुछ specific process memory में load करेगा: usi rate se process generate kare jis rate se bahar jaa rahe hain thruput
- कोई Processes (LTS में से सिर्फ एक बार ही गुजरता है) (Ya max swap in swap out का time) क्योंकि हर Process सिर्फ एक ही बार LTS से CPU mein jata hai!

c) STS decides ki fhaal konsa process ready queue me se CPU mein jayega.

And a processes can go thru STS multiple times as it might not complete execution at once (time quatum expired, I/O request, thread, os interrupt etc)

3. Middle/Medium term scheduler

Maintains/MANAGES the degree of multiprogramming a bit bit it.

Swaps out some processes if the load on main memory is v. high is that the equilibrium is maintained.

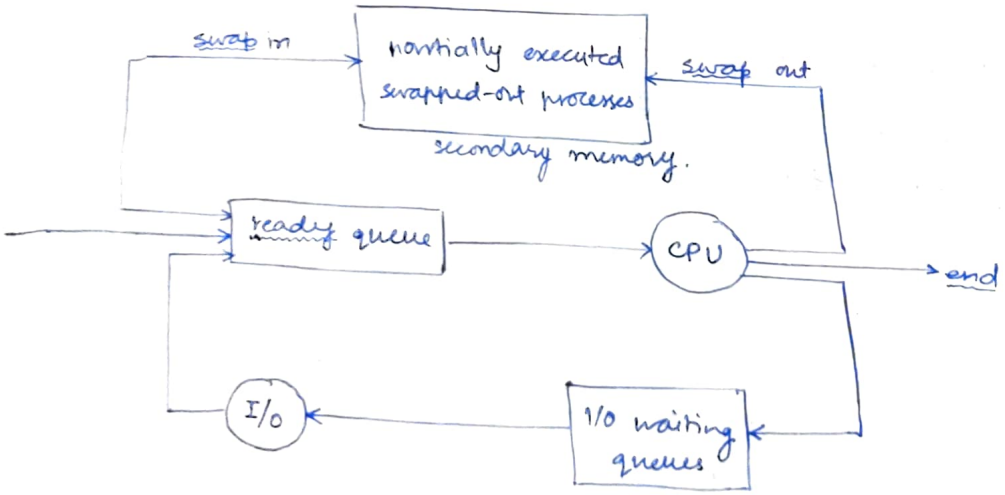
Theory:

The key idea behind a medium-term scheduler is that sometimes it can be advantageous to remove a process from memory (and from active contention for the CPU) and thus reduce the degree of multiprogramming.

Later, the process can be reintroduced into memory, and its execution can be continued where it left off. This scheme is called SWAPPING!

The process is swapping out, and is later swapped in, by the medium-term scheduler. Swapping may be necessary to improve the process mix or because a change in memory requirements has overcommitted available memory, requiring memory to be freed up.

[THRASHING]

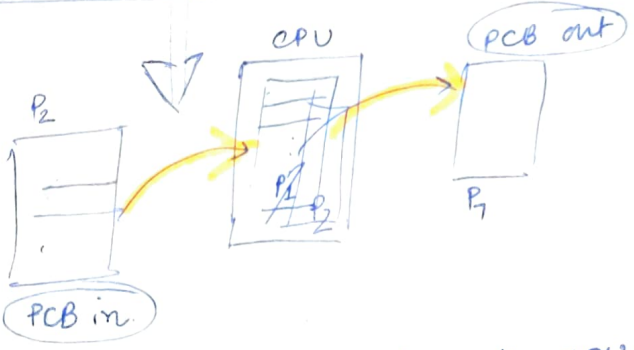


o> The ULTIMATE MASTER OF DEGREE OF MULTIPROGRAMMING IS LTS. Medium-term scheduler can only manage the (DOM)

o> USED IN THRASHING. (Medium-term sch)

4. **Dispatcher**: The Dispatcher is the module that gives control of the CPU to the process selected by the short-term scheduler.

- This function involves the following: switching context, switching to user mode, jumping to the proper location in the user program to restart that program.
- The dispatcher should be as fast as possible since it is invoked during every process switch. The time it takes for the dispatcher to stop one process & start another running is known as the DISPATCH LATENCY.



Taking out P_7 & taking P_2 (This is called Context Switching!)

[STS just decides which process will get the CPU, DISPATCHER dispatches that process to the CPU.]

⑦ Degree of multiprogramming

The number of processes in memory is known as DOM.

- The LTS controls the DOM as it is responsible for bringing in the processes to main memory.
- If the degree of multiprogramming is stable:

The average rate of:

Process creation	=	Departure rate of processes leaving the system

So this means the long-term scheduler may need to be invoked only when a process leaves the system. Bec of the longer interval between executions, the long-term scheduler can afford to take more time to decide which process should be selected for execution.

2.7 CPU Bound and I/O Bound Processes

CPU Bound — generates I/O requests infrequently, using more of its time doing computations

I/O Bound — A process which spends more of its time doing I/O than doing computations.

Resource = CPU + I/O

As a good LTS, you should manage equilibrium b/w CPU-bound & I/O-bound processes. Taki kisi pe bhi line na lage. System ke saare resources properly utilise hon.

- It is important that the LTS selects a **good mix** of I/O-bound & CPU-bound processes.
- If all processes are I/O-bound, the ready queue will almost be empty, and the STS will have a little to do. (The device queues will fill up)
- Similarly, if all processes are CPU bound, the I/O waiting queue will almost be empty, devices will go unused and again the system will be disbalanced.
- So, to have the best system performance, LTS needs to select a good combination of I/O and CPU-bound processes.