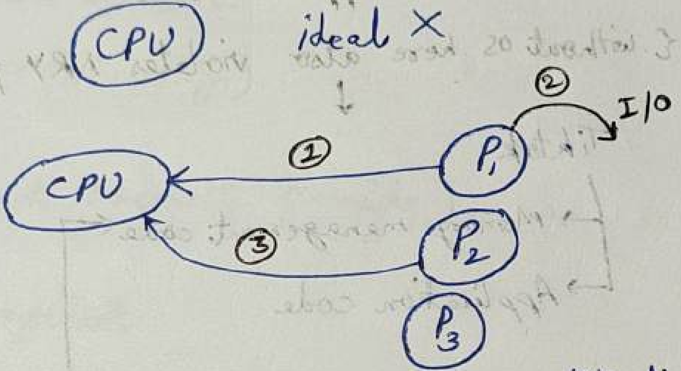
Types of OS

- Goals

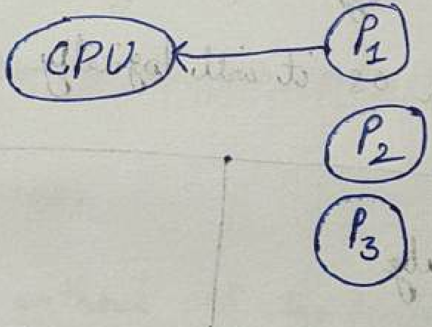
① Maximum CPU utilization



→ In this we aim for maximum utilization of CPU so it isn't ideal.

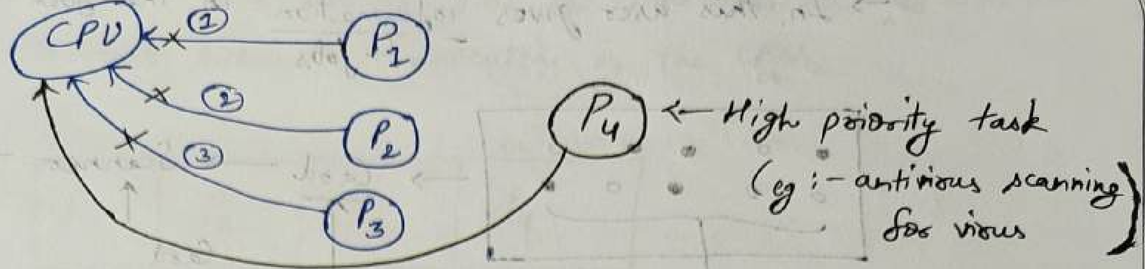
⇒ Like in above example the process P_1 is executed and then gone for I/O operations then CPU is ideal so we want CPU to not be ideal so P_2 is executed.

② Process shouldn't starve (Process Starvation ×)



→ In this the process P_1 is executed but it is very big so P_2 & P_3 will starve as they won't get CPU.

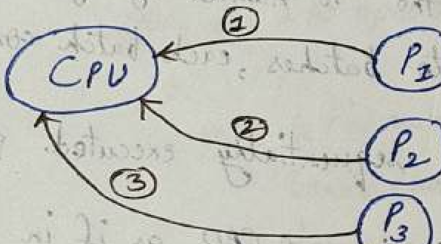
③ High Priority job execution



→ So OS should support execution of high priority tasks first.

Types Of OS

① Single Process OS



→ At a time only one process will be executed (maximum utilization of CPU x)

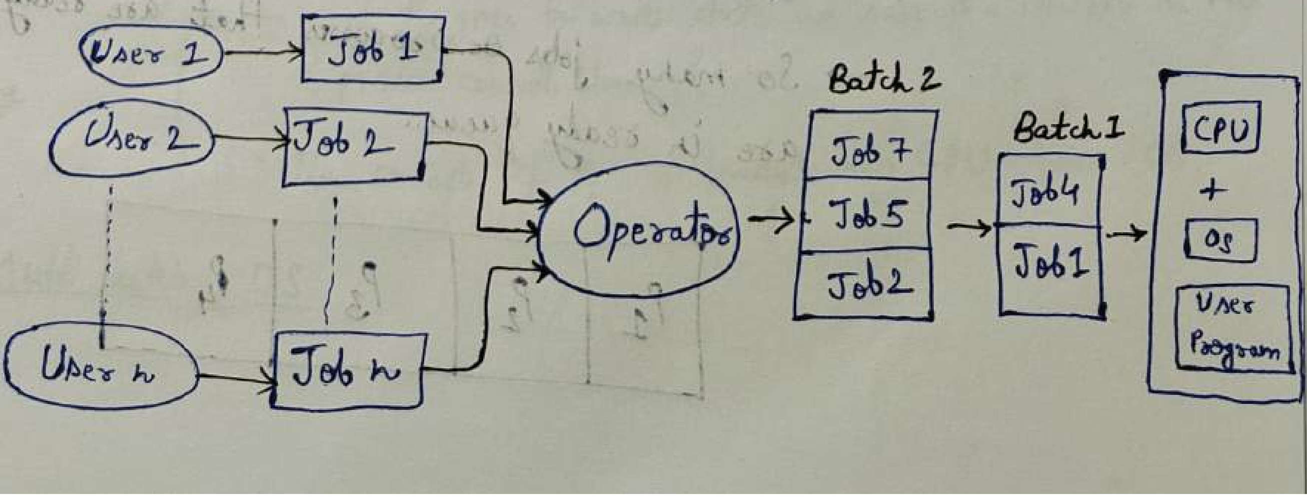
→ If P_1 goes for I/O operation then CPU is sitting ideal only

→ P_2 & P_3 will also starve if P_1 is a very large task.

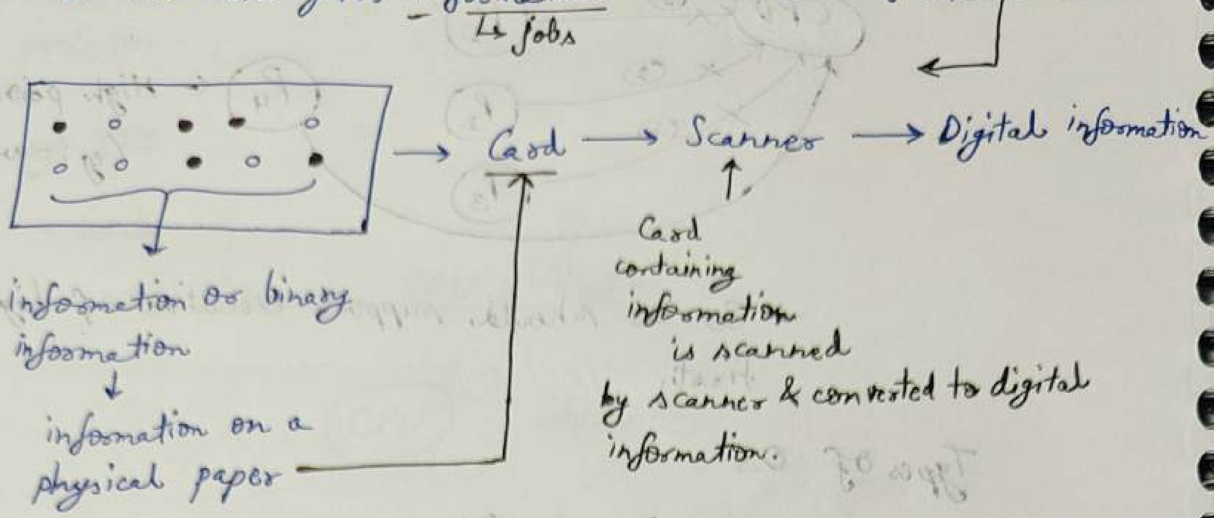
→ High priority jobs won't be executed

eg :- MS DOS {first OS}
↳ Disk Operating System

② Batch Processing OS



→ In this user gives information (↳ jobs) in the form of punch card.


Card
Scanner
Digital information
information or binary information
information on a physical paper
Card containing information is scanned by scanner & converted to digital information.

→ Operator sorts the n number of jobs given by n users and divides it into batches, each batch consists of similar jobs.

→ Each batch will be sequentially executed {OS is still Single Process}

→ No maximum utilization of CPU as if in Batch 1, Job 4 is executed then it goes for I/O operations then CPU will sit ideal for that time.

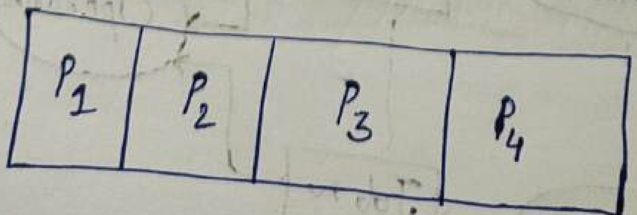
→ Similarly process will also starve. In this there will be batch starvation also. Like if batch 1 takes too much time for execution then batch 2 will starve.

→ Similarly high priority task will also not be executed. eg:- ATLAS

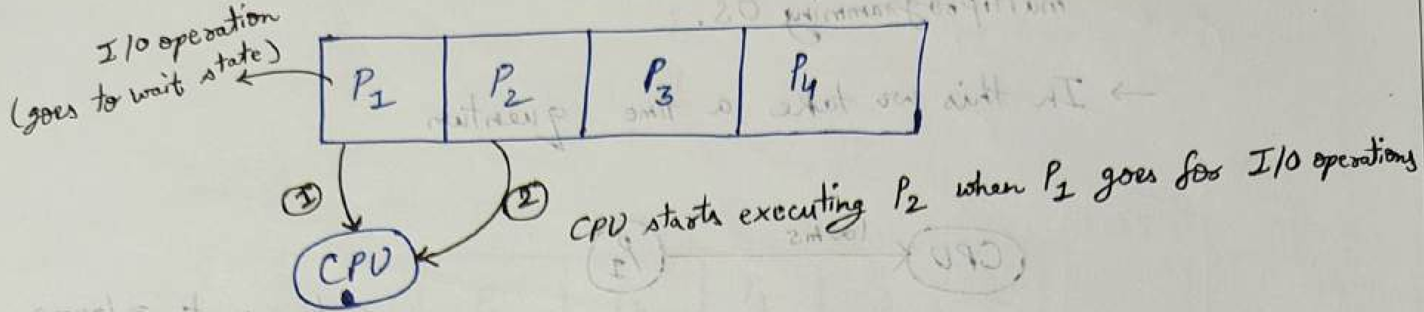
③ Multiprogramming OS

→ CPU is still single CPU

→ So many jobs or processes that are ready to be executed are in ready queue.



→ So in this if any process goes for I/O operation or wait state then the next process is executed by the CPU.


I/O operation (goes to wait state)
P1
P2
P3
P4
1
2
CPU
CPU starts executing P2 when P1 goes for I/O operations

Context switching ★★

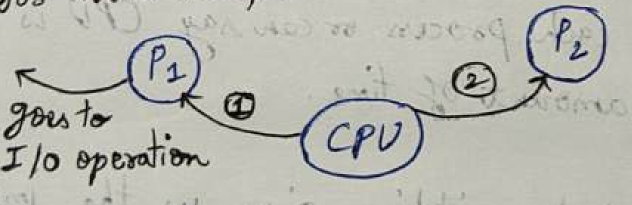
↳ This concept is used to tell CPU to execute P_2 when P_1 goes for I/O operations.

eg :- Let's assume we are studying physics for ~~of~~ our board exam but from 11:00 am we will study chemistry.

So we will close and bookmark till where we have studied physics and will open chemistry book from where we left before.

→ So it's like we store the information of till where we read the physics book and load the information of chemistry book as to from where we must continue to read. eg :- THE

→ So for above example



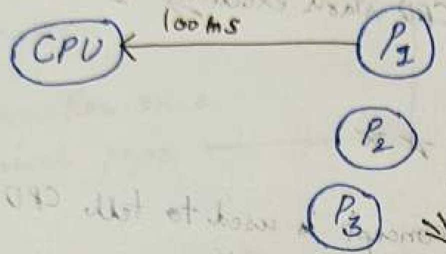
Context Switching ← { • When P_1 goes to wait state we save it's context in PCB (process control block).
• Then context of P_2 is loaded into CPU from PCB. }

④ <u>Multi Tasking OS</u>

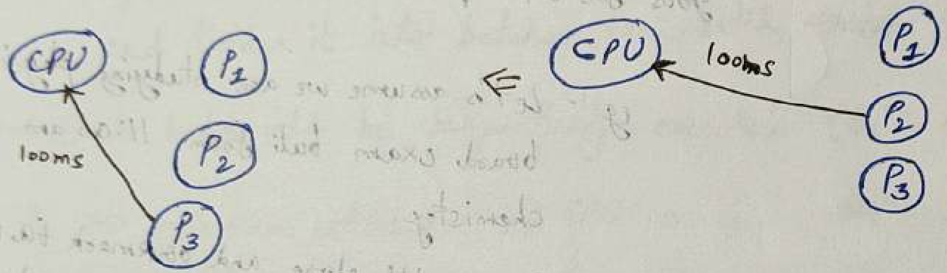
→ Single CPU and context switching is here also.

→ Here time sharing is there which makes it better than multiprogramming OS.

→ In this we take a time quantum



eg :- time quantum = 100ms



→ So in this the responsiveness of applications is increased.

→ In this each process will execute in the give time quantum if it doesn't gets completely executed then it waits for it's next time quantum.

→ So this OS is also called as fair share as equal amount of time is given to each process or can say CPU is allocated to each process for equal amount of time.

→ Therefore this context switching gives us the impression that it is multitasking so it is called as multitasking OS.

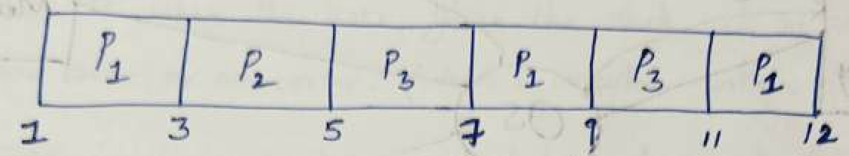
Fair share / Time Sharing / Multitasking OS

eg :- CTSS

eg :- time quantum = 2 time units

Process	CPU Burst time
P_1	5
P_2	2
P_3	4

Gantt chart



→ So in this maximum CPU utilization.

→ No starvation as each process gets CPU allocated to it for a specific time quantum.

→ If high priority task comes then it allocates CPU to high priority task.

⑤ Multi processing OS

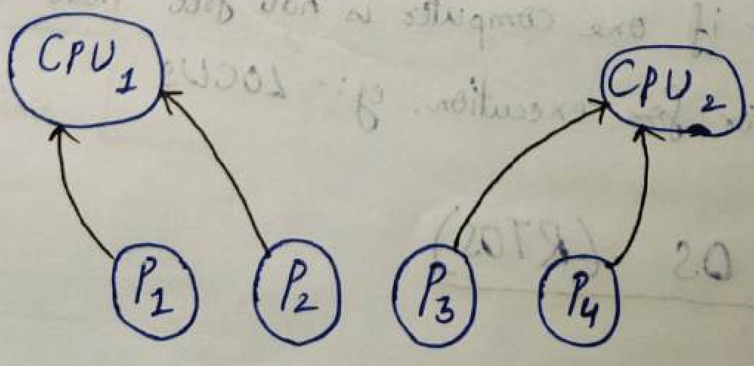
→ In this also context switching, time sharing is there.

→ In this the number of CPU is greater than equal to 1.

$$CPU \geq 1$$

→ More CPU utilization, ~~more~~ less chances of starvation, more apps can run simultaneously.

→ If high priority task comes then if CPU1 is busy then CPU2 will do this task. eg :- Windows

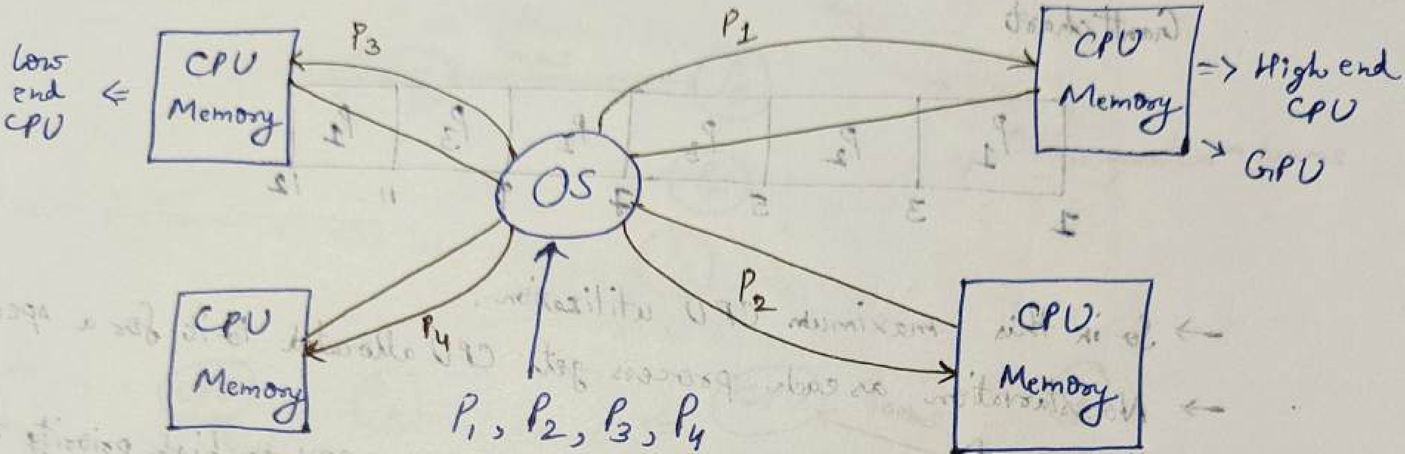


time quantum = t time units

→ If any one CPU fails then other CPU will do the work
$\Downarrow$
increased reliability

⑥ Distributed OS

↳ loosely coupled



→ In this we can keep CPU of different configurations also

→ In this there are loosely coupled autonomous interconnected computers

→ Multiple users can be there.

eg:- In leetcode if we submit our code it goes to a LINUX machine to check our code.

→ So in this there are many computers which are connected over the internet so to handle this type of system we use distributive OS.

→ In this also if one computer is not free then the task is given to other computer for execution. eg:- LOCUS

⑦ Real Time OS (RTOS)

→ Used where there is no chance of error, execution should be fast.

eg :- ATCS (Air Traffic Control system)
Industrial applications

→ Used where the response should be fast.

→ In this OS when the user gives the task it responses back very fast with low error. or no error. (task is executed within a deadline)

→ An RTOS is designed to process data and execute tasks within a strict time constraint, providing predictable and deterministic responses to real-time events.

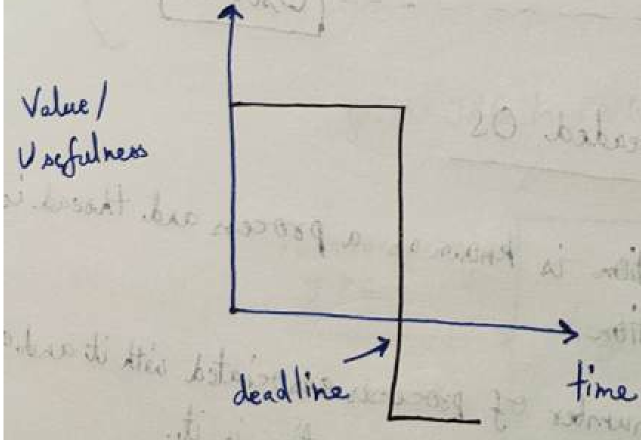
→ It is essential for applications requiring precise timing and reliability.

$$\begin{tikzcd}
 & \text{Real-time OS} \arrow[dl] \arrow[dr] & \\
\text{Hard Real-time OS} & & \text{Soft Real-time OS}
\end{tikzcd}$$

Hard Real-time OS

→ Ensures tasks are completed within strict deadlines.

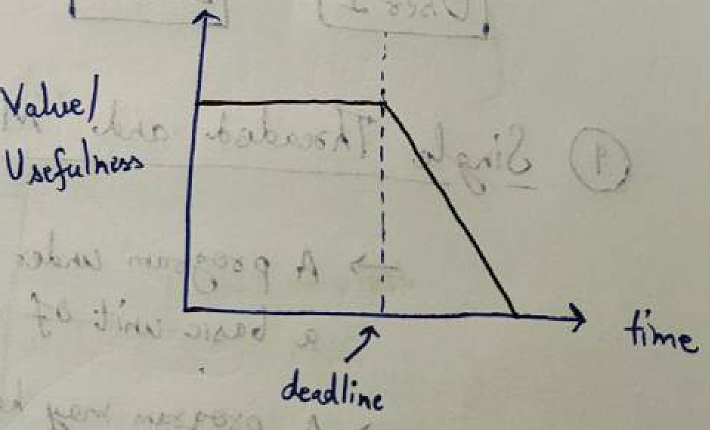
→ Delays are not acceptable.



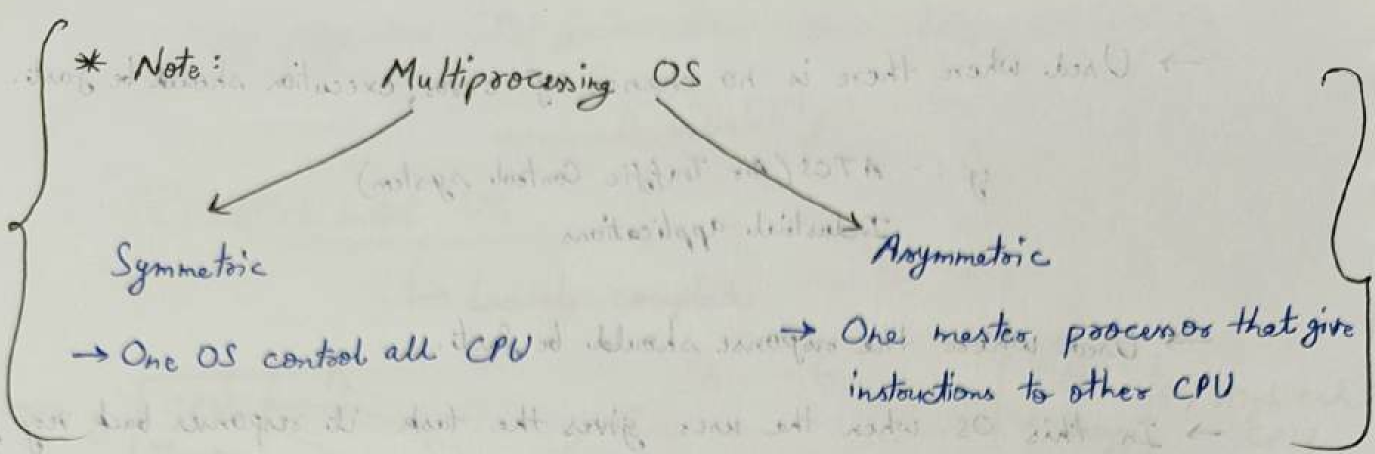
Soft Real-time OS

→ Allows for some flexibility in deadlines

→ Delays are acceptable but kept to a minimum.



* Note:



→ One OS control all CPU

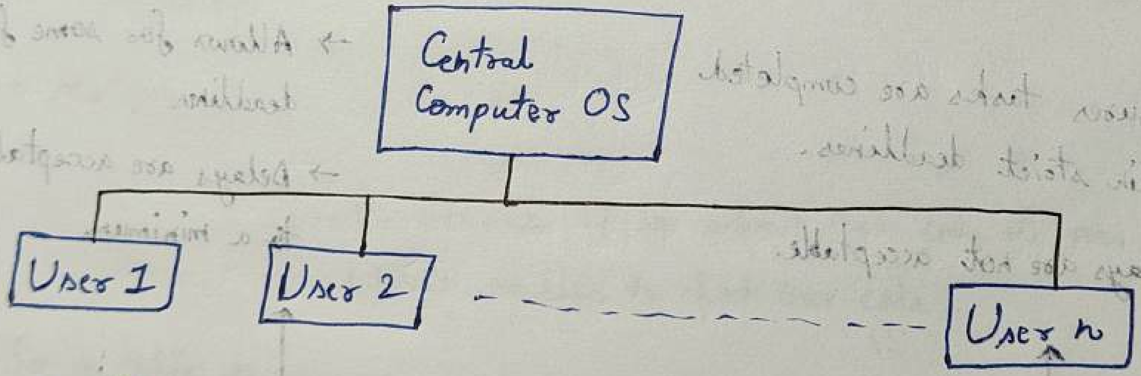
→ One master processor that give instructions to other CPU

⑧ Multiuser OS

→ Multiuser OS are designed to allow multiple users to access and interact with a computer system simultaneously.

→ These operating systems provide mechanisms for managing user accounts, enforcing access control policies and facilitating concurrent access to system resources.

eg:- Unix, Linux



⑨ Single Threaded and Multithreaded OS

→ A program under execution is known as a process and thread is a basic unit of execution.

→ A program may have a number of processes associated with it and each process can have a number of threads executing in it.

→ Thread is kind of light weight process which execute independently.

• Single threaded OS

→ In a single threaded OS each process can execute only one task or instruction at a time.

→ The OS allocates CPU time to processes sequentially, allowing each process to run to completion before switching to next process.

→ This type of OS may suffer from poor performance and inefficiency

eg :- DOS

• Multiple threaded OS

→ In a multithreaded OS each process can be divided into multiple threads, each of which can execute independently.

→ Threads within the same process share the same memory space and resources allowing them to communicate and coordinate with each other more efficiently than separate processes.

→ Multithreaded OS can exploit parallelism and concurrency, improving system responsiveness, throughput and resource utilization.

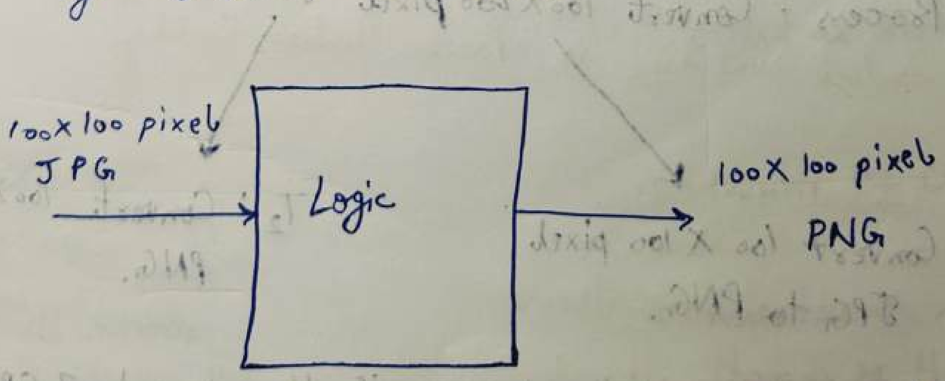
In Single

$$\begin{tikzcd}\text{Process} \arrow[r, squiggly] & \text{thread}\end{tikzcd}$$

In multi

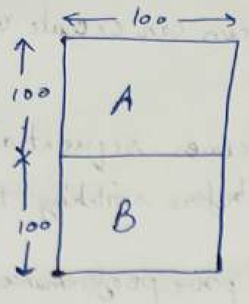
$$\begin{tikzcd} & \text{thread 1} \\ \text{Process} \arrow[ur, squiggly] \arrow[r, squiggly] \arrow[dr, squiggly] & \text{thread 2} \\ & \text{thread 3} \\ & \vdots \end{tikzcd}$$

eg :- JPG to PNG converter

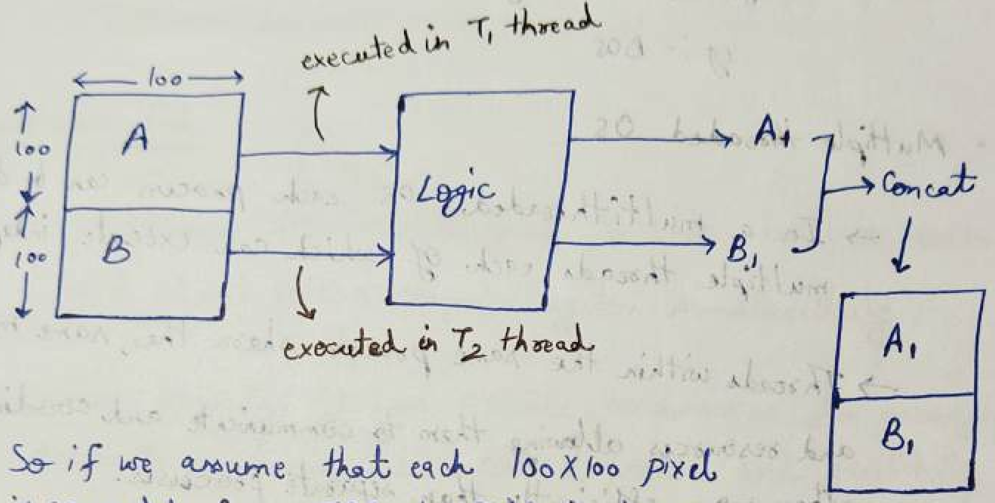


→ So let's assume our converter can only convert 100X100 pixel JPG to PNG

So if we take image of 100 X 200 pixels then



So divided in 2 parts as can only convert 100 x 100 pixel images.
Since both of these are not independent of each other so both can be simultaneously executed in thread in multithreaded OS.



So if we assume that each 100 X 100 pixel is converted from JPG to PNG in 10 sec then in this it takes 10 sec to process both.

Total Time = 10 sec.

Whereas if we do it sequentially then it will take

Total Time = $\underbrace{10}_{\text{Time to convert A from JPG to PNG}} + \underbrace{10}_{\text{Time to convert B from JPG to PNG}}$ = 20 sec

Process : Convert 100 X 200 pixel JPG to PNG

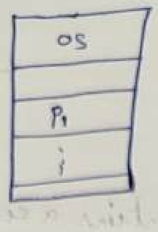
$$\begin{tikzcd} & \text{Process: Convert 100 X 200 pixel JPG to PNG} \arrow[dl] \arrow[dr] & \\ T_1 : \text{Convert 100 X 100 pixel JPG to PNG.} & & T_2 : \text{Convert 100 X 100 pixel JPG to PNG.} \end{tikzcd}$$

T_1 : Convert 100 X 100 pixel JPG to PNG.

T_2 : Convert 100 X 100 pixel JPG to PNG.

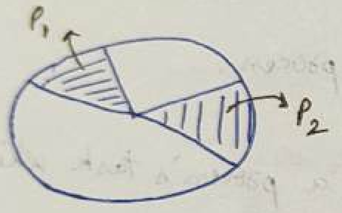
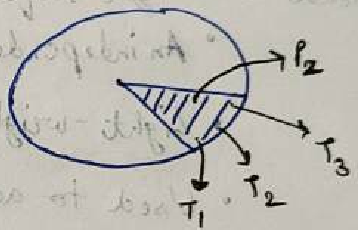
* Note: No use in multi threaded OS if there is only 1 CPU as with 1 CPU work will be done sequentially only so above example the total time will be 20 sec only. So multiple CPU is better.

* Note :- let's assume process P_1 is alloted a space in memory



So all the threads will also share same space as thread are part of the same process so there is no isolation between threads also as part of same process.

Multitasking vs Multithreading

Multitasking	Multithreading
→ More than 1 processes concept.	→ More than 1 threads.
→ Isolation & memory protection (as between processes)	→ No Isolation & memory protection. (as between threads) ↳ part of process
	
→ Process scheduled	→ Threads scheduled
→ The execution of more than one task simultaneously is called as multitasking.	→ A process is divided into several different sub-tasks called as threads which has its own path of execution. This concept is called as multithreading.
→ Number of CPU 1.	→ Number of CPU ≥ 1.
→ OS must allocate separate resources to each program that CPU is executing.	→ Resources are shared among threads of that process

* Note: For best practice
Number of CPU = Number of thread
Defeats purpose if Number of thread > Number of CPU

Definitions

Program: A program is an executable file which contains a certain set of instructions written to complete the specific job or operation on your computer.
→ It's a compiled code. Ready to be executed.
→ Stored in disk

Process: Program under execution. Resides in Computer's primary memory (RAM).

Thread:
- Single sequence stream with a process.
- An independent path of execution in a process.
- Light-weight process
- Used to achieve parallelism by dividing a process's task which are independent path of execution.

Thread Scheduling: Threads are scheduled for execution based on their priority. Even though threads are executing within the runtime, all threads are assigned processor time slices by the OS.

Thread Context Switching vs Process Context Switching

Thread Context Switching	Process Context Switching
→ OS saves current state of thread & switches to another thread of same process.	→ OS saves current state of process & switches to another process by restoring its state.

→ Doesn't includes switching of memory address space.	→ Includes switching of memory address space.
→ Fast switching	→ Slow switching
→ CPU's cache state is preserved. (as other thread might use)	→ CPU's cache state is flushed