L3

Process → Program Under execution

Program + runtime = Process.
(code) activity
↓
Instruction ↳ operands and other info?

- An instance of a program
- Sheduable/Dispatchable unit (CPU)
- Unit of execution (CPU)
- Locus of Control (OS).

Process as a Data structure

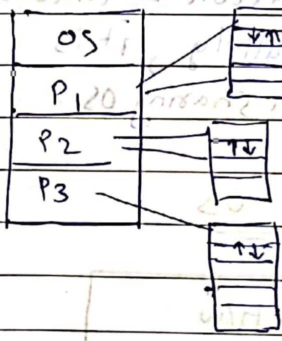
i) Defination → code or only prog.
ii) Representation/Implementation
→ how process stored in memory
iii) Operation
iv) Attributes

Representation of a Process

Stack ↓	
↑	
Heap	Dynamic memory allocation
Data Section	→ Public or Static Variable.
Code or text section	→ Defination of ~~code~~ process, Instruction

Heap → Dynamic memory alloctn.
Stack → Local Variable, function parameters, return addresses (activation record).

- Data section, code section are ~~Same si~~ limited Size.
- while Stack, heap are not fixed ~~memory~~ Size, and space between them is for if necessary we can increase the size of Heap or Stack.



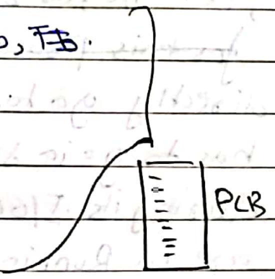
each process has its own Stack, heap, its own Data Section, its own code section

Operation on Process

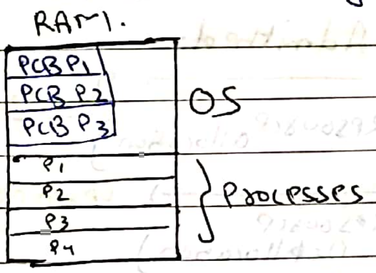
1) Create (Resource allocation)
2) Shedule, Run
3) wait/Block
4) Suspend, Resume
5) Terminate (Resource Deallocation)

• Attributes of a process

1) PID -> Process ID
It is used to uniqly identify each and every process
2) List of Device
3) PC, GPR
3) Type -> Background, Foreground
4) Size -> in Mb, ~~FB~~.
5) Memory limits.
6) Priority.
7) State
8) List of files

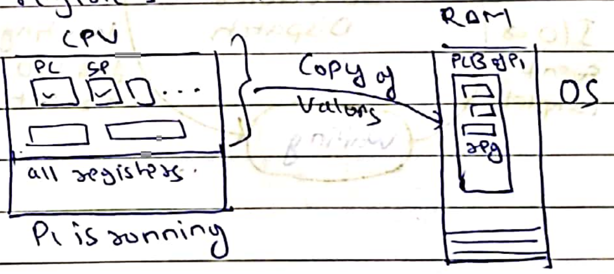


PCB -> Process Control Block.
-> It is also known as process descriptor.
-> all the attributes of a process above are in the PCB.
-> PCB is in control of OS.



Inside RAM where Programs runs or Execute (process) [many process are present inside RAM with OS some part of it] every process has its own ~~Represent~~ Representation (OS, CS, stack, heap). and all the attributes of process present inside PCB, which is in control of OS.

When ~~Proc~~ Program Execute. it need registers, Program Counter, several GPRs while it is Executing, for some reason if it has to leave the CPU, So, to Save all these GPR, PC values, OS take a copy of each and every values of GPR, PC and save it into PCB of that process, So, that when it resumes, it can start from where this process left.
In PCB, also some GPRs and PC registers are there.



The content of PCB of a process are collectively known as '<u>Context</u>' of that process.
if.
~~When any~~ other process want to ~~run~~ execute, then, from PCB of that process, OS send its current values of GPR, PC etc to CPU, So, that it resumes.

The whole process of copying of context of each process to PCB to context load of another process from PCB to CPU is called <u>Context Switching</u>

Context Switching -> Context Save + context load.

• Process Cannot access its own PCB

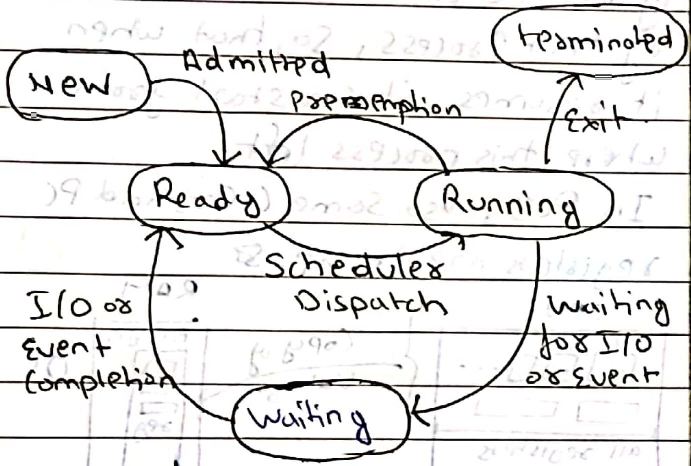
Stop a running process and run another process is called Context Switching.

• PCB stored in OS protected area

• Context Switch is done by dispatcher.

L-4

Process States



1) New state → All installed application process not running. means, only stored in MM, and not running ~~means~~ in new state, All installed process are known to be in new state.

Before Running the processes

2) Ready state → The state in which the processes are ready to Run, waiting for CPU to run process, All process which are waiting to run on CPU are known to be in Ready state

3) Running state → process Executing in CPU, running state.

4) terminating → If process Executed and complete its operation then it terminates.

If process want to use I/O devices during ~~Exect~~ Execution (running state), it has to leave the CPU and wait for I/O event, it means process is in Waiting state, ~~it~~ if I/O, or event is completed ~~for this process~~ then it cannot directly go to running state, it has to go in ready state, because during its I/O event, other process are in Running state.

• If the process is in running state and forcefully take out from running state ~~is called~~ it means this process get preempted and after preemption it goes to ready state.

Process $\text{New} \xrightarrow[\text{(Resource allocation)}]{\text{Admitted}} \text{Ready}$.

$\text{Running} \xrightarrow[\text{(Resource Deallocation)}]{} \text{terminated}$.

NOTE:
Process Can go from Running state to terminated state or Running to waiting state by its own wish. and other transitions are decided by OS.

New to Ready : when Process admitted by OS
Ready to Running : " " dispatched to
Running to termin : " " completed
Running to Blocked : " " go for I/O
Running to Ready : " " pre-empted
Blocked to Ready : " " completes I/O

Q -> if there are n processes are admitted, we have m no. of CPUs, but m < n.

States	(Process) min	(Process) max.
Running	0	m
Ready	0	n
Blocked	0	n.

TWO TYPES OF PROCESS

1) CPU Bound -> ~~must~~ if the process is intensive in terms of CPU operation.

2) I/O Bound -> If the process is intensive in terms of I/O operations.

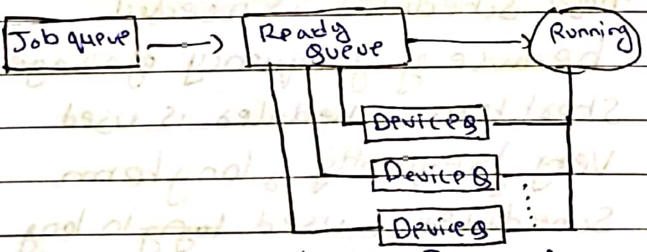
• A process which has just terminated but has to relinquish its resources is called zombie process

L-5

Process Scheduling

-> Scheduling is needed for better resource utilization.

• Job queue -> all the processes which are in New state ~~job queue~~ are kept in a queue (Job queue).
• Ready queue -> all the processes which are in ready state are kept in a queue called Ready queue.
• Device queue -> all those processes which waiting for a specific device are kept in a queue (Device queue).



• all queues will have PCBs of processes

Types of schedulers

1) Long-term sheduler (Job)
↳ shedule the process from new state to ready state.
two reason for bringing new state to ready state.
① when(if) user runs any program during that time to ~~that~~ execute that Long-term sheduler shedule the process from new to ready.
② for background processes which are not initiated by user.

during newstate to ready State resource allocation takes place.

2) Short-term Scheduler (CPU)

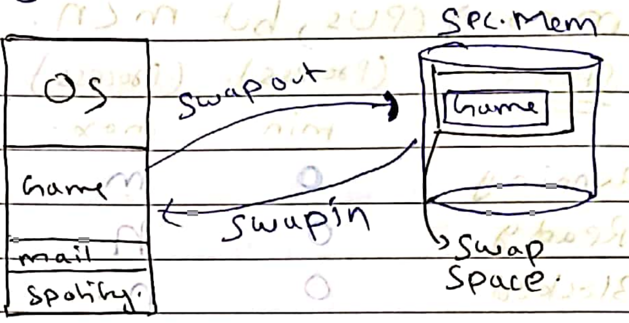
-> There are so many processes in ready state; ~~to run in~~ CPU which process must run in CPU decided by Short term scheduler.
, short term scheduler only selects the process, one of all ready processes to run On CPU

These scheduler is needed because of frequency of usage. Short term scheduler is used very frequently, long term scheduler is used ~~togg~~ in long ~~ter~~ long term.

3) Mid term (Scheduler)

All the processes runs into the RAM, and already some part of RAM is ~~used~~ by OS. occupied by OS and remaining ~~Us~~ is for running the process. let us take an example of running these process :-
1) Suppose you are playing a very big game, it means it occupied some large space of RAM.
During playing, ~~You thought~~ ~~You had an~~ Email ~~which~~

You got a notification of mail and you want to reply that mail, You pause the game and switched to mail, while writing ~~an~~ mail, you are listning to the misc from spotify.
It means, RAM has now these process running -> Spotify, game, mail, and already OS.

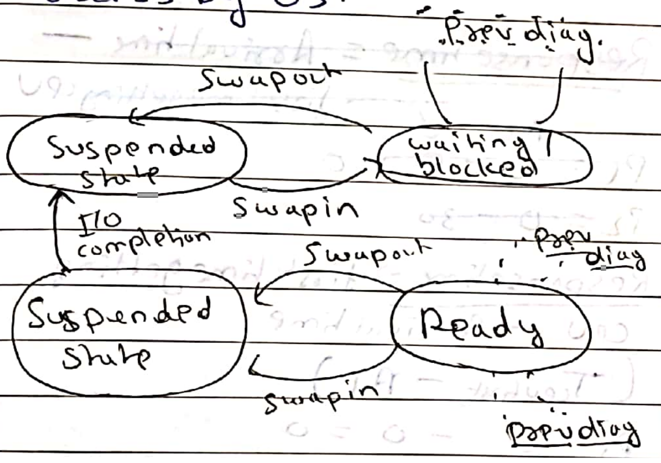


• now, you want to ~~run~~ open whatsapp for some work, But there is no space in RAM
• So, here comes Mid term Scheduler comes into picture it swapout the inactive process (Game) into the Hard disk, so that enough space can be available and can able to use other process (whatsapp)
• after all your work, you go back to game for resuming your enjoyment, game took sometime to load back the saved game from Memory to RAM, and ~~this~~ your progress will not lost.
• This loading from Sec Mem to MM is called Swapin.

The process of Swapin and Swapout is called Swapping Done by Mid term scheduler.

• Swapping is also known as rolling, when Swapping is done based on priority of processes

-> The process which ~~load~~ Swap out from MM to Sec. Mem it kept into Swap Space, only access by OS.


Prev diag.
Swapout
Suspended state
waiting / blocked
Swapin
I/O completion
Swapout
Prev diag
Suspended state
Ready
Swapin
Prev diag