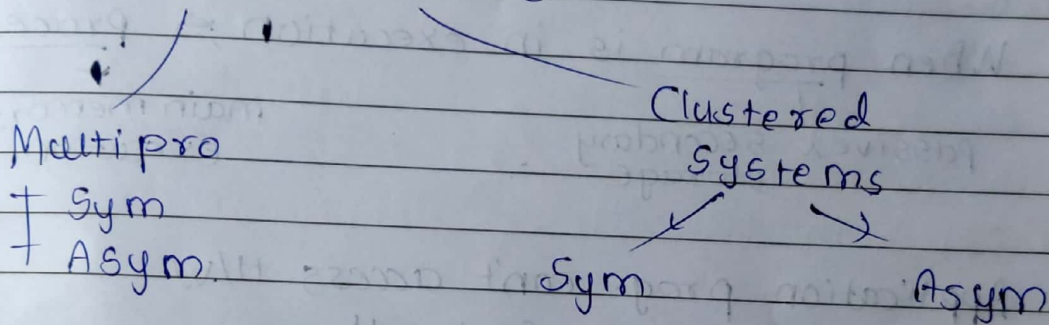
* Unit 1 :

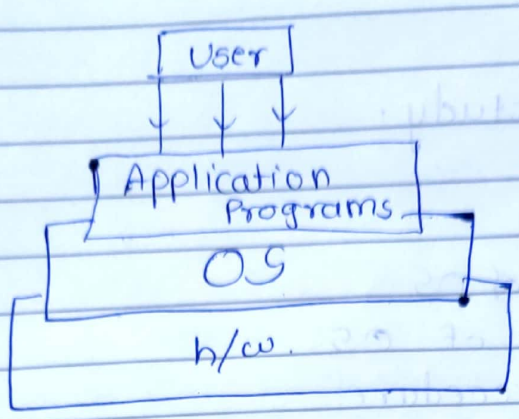
Subtopics to study:

1) OS
2) Components of OS
3) Major goals of OS
4) Booting procedure
5) Understand OS role
6) General Structure



7) Evolution of OS
   - Serial
   - Batch
   - Buff & Spooling
   - Multiprogram
   - Multiprocessing.
   - Time Sharing
   - Parallel systems
   - Distributed & Real time.
8) System Calls
9) System Program
10) OS structure
    (simple, layered and Kernel approach)
11) Virtual M/Cs.

* OS definition



OS ⇒ to access h/w, interface.

When program is in execution ⇒ process
↓ passive ↓ secondary storage
↓ main memory execution.

Application prog. can't access H/W alone or else confusion !!
Only OS can access it.

* Boot procedure
Power on self Test (POST)
(to check if H/W works or not)
↓
In ROM,
Firmware ⇒ activate software.
↓
HDD activated
↓
Bootstrap will load OS in MM.
↓
Once OS in MM ⇒ GUI appears.

* Functionality of OS
(services provided!)
i). Process management.
ii). Memory management.
iii). Device management.
iv). File Management.
v). Security
vi). Secondary storage management.

Basic goals of OS

Primary → User friendly / convin.
Secondary → Efficient.

" Those who are habituated with something, will not change it "

1. Process management:
deletion / creation of process.
scheduling, resuming, suspending and terminating staff.
Process Control Block → PCB

2. Memory management:
Partitions; types, sizes, free for allocation, deallocation.

3. Device management:
I/O device management.
(resources).

9/1

* Evolution of OS:

* Earlier systems
* Serial processing:
  Operator executes the program, then operator executes the program one by one in a serial manner

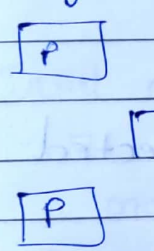
* Batch processing:
  Job with similar needs are grouped in a batch, that batch is executed together. Mantain same environment.
  Not serial, advantage: a little faster
  disadvantage: If speed mismatch, CPU will be idle for longer time ☹.

* Buffering & Spooling:
* i/p, stored in buffer, so next i/p must be ready by the time CPU processes first input, hence a bit of performance ↑.
  Spooling: Simultaneous Peripheral Operation Online. It will overlap i/p and o/p. Card reader: prog. stored in disk is read and stored directly and sent to processor.

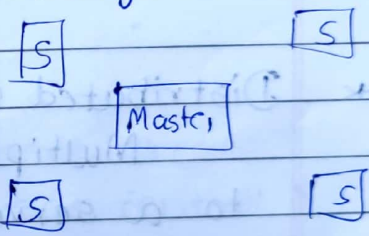
* Throughput:
  No. of tasks performed per unit time. Speed up time.

* Multiple Programming:
Main memory stores multiple ready to run programs. Simultaneous operation.
Single processor executes multiple programs.
Here CPU won't be idle for long.
Memory utilization is efficient as progs stored.
Throughput is high.
Hence better than serial, batch, buffer & spool.
But more scheduling is required.
Good CPU scheduling is needed.

* Multiprocessing:
Multiple processors are involved. Can be
Symmetric OR Assymetric

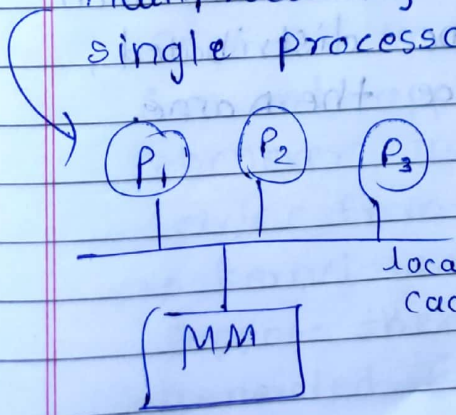


All are peers (P)
Independent control.
System can't collapse
More scheduling

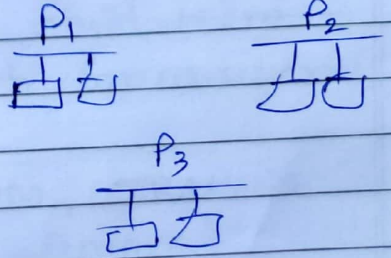


If Master fails
pcch x
Less scheduling

Multiprocessing is better than multiple single processor because cost & reliability

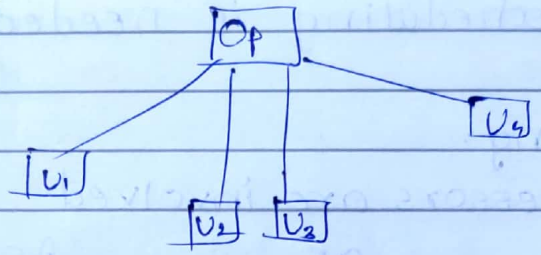


Multiple single proc



* Timesharing / Multitasking:

Giving turns to all users, all users are handled in round about fashion. Every user will have a time slot. Similar to concept of lazy loading. (loading slowly ... because everyone is taken care of.. but switching is so fast that users can feel as if they are using alone).

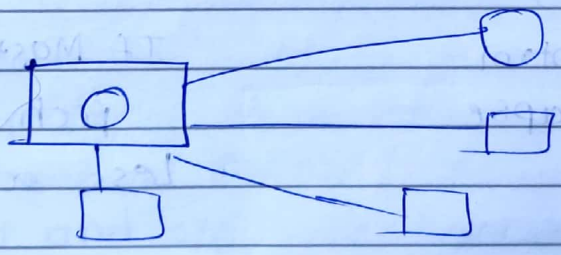


U_i = users

Take turns and simultaneously time sharing

* Distributed Systems

14/1. Multiple computers are connected to a server kind of thing.



like home branch and local branch of a bank. Data can be distributed, globally accessed, hence the name.

* Real time systems:
Like air traffic control, medical imagery, industrial controlling systems. Depends on deadline, real time / to produce output.
eg: take off, landing timing.
eg: heart rate during operations.
Types:
Soft: some delay is okk.
Hard: Deadline, timely manner, etc. strict.

* Dual Boot operation:
Operating system operation.
(sorry wrong title)

* Dual mode operation:
It is an OS operation. There are 2 modes (depends on who has control).

User Mode
User has control.
(1)

Kernel mode
(System / Supervised / Privileged mode).
(0).

Single bit is attached to comp. hardware
Once OS is loaded, → Kernel mode
GUI and appl. prog. → User mode
whenever user wants to ~~shift from~~ service from OS, switch from user to kernel mode.
Suppose there is a problem, TRAP is generated (related to software),

then user → kernel, message displayed on monitor, OS does this.
This is a system call.

Whenever services from OS are needed, system calls are involved.

* OS Services:

(Friendly)
User handling
- User Interface
- I/O operations
- Prog. execution
- Error detection
- Communicator

Efficient
- Resource sharing
- Accounting
- Protection & Security

- Command line
- Batch interface
- GUI

* Types of prog:

i). System prog: Compiler, provides environment for user, directly access H/W with
ii). Application prog: Specific tasks, user oriented
eg: VLC, spreadsheet, etc

System prog. with the help of OS, can directly interact with H/W.

Assignment: "Hello" program system call.

16/1

* Operating System:

→ Traditional Systems

- MS DOS
- UNIX
- MS DOS – not layered.
  - no well defined structure
  - functionalities not segregated
  - (ideally users should not have direct access to hardware)
- UNIX – not layered either (slightly more than MS DOS)
  - limited functionality case

MS-DOS

Application prog.
I/O system
Resident system
MS DOS device driver
ROM BIOS device driver

UNIX

Application Prog.
File CPU, mem, sys-sched, management
terminal device mem.

Kernel →

H/W →

Components in kernel call in one layer.

→ Layered Structure:

- not efficient.
- time consuming. (need to access all layers to reach h/w)
- chain heirarchy.

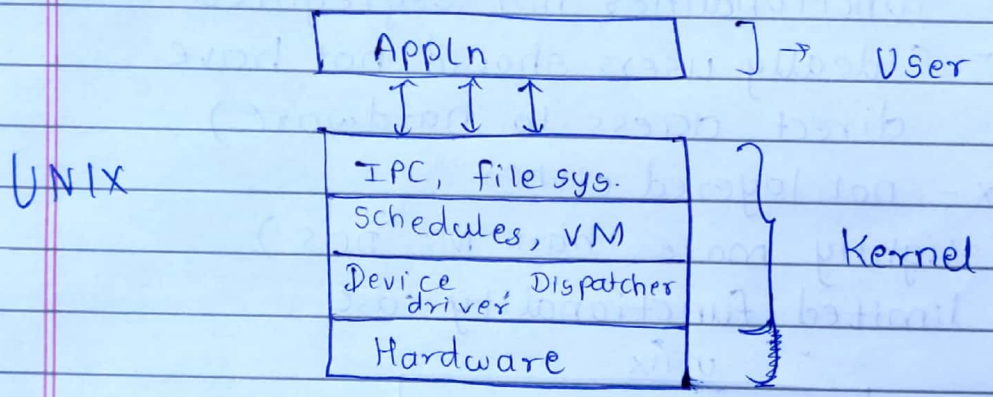
layer 5	User prog.
	I/O devices
	device driver
	memory manag.
	CPU scheduling
layer 0	Hardware

* During booting ⇒ OS loaded in RAM.

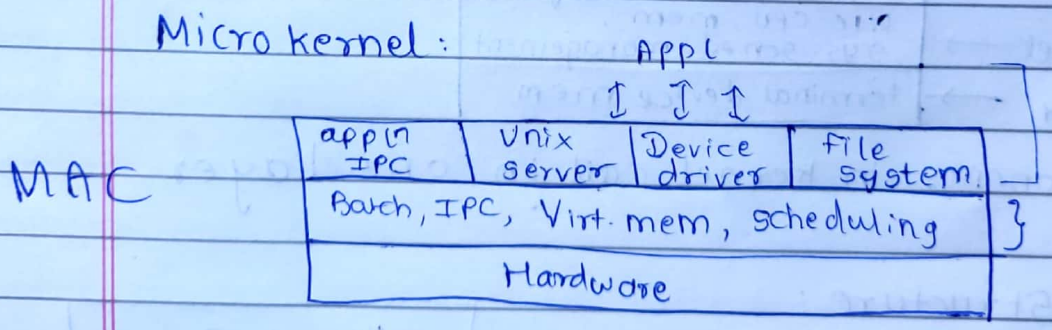
→ Kernel :
(provides basic functionality of OS)
① Monolithic - linux, UNIX
② Microkernel : MAC OSX

monolithic : everything combined together.
microkernel : Only basic func. in kernel.
- kernel compact.
- all other functionalities in user space (via sys. calls).
- can be easily modified unlike monolithic

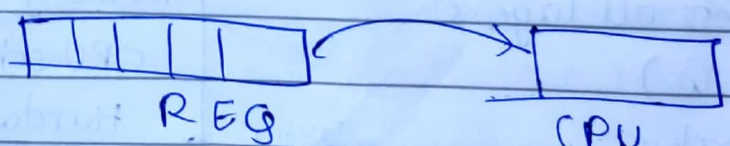
Monolithic



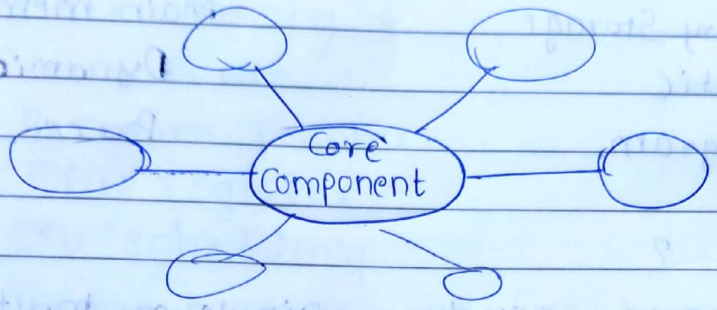
Micro kernel :



Dispatcher : which process from ready queue (scheduled) to be dispatched into CPU.



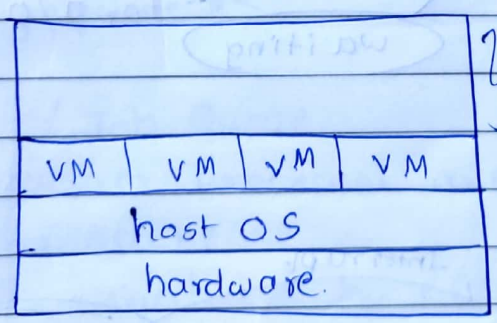
* Module based structure:
  - Related to object oriented programming
    eg: Solaris OS.
  - Core component in kernel - all of them being dynamically loadable modules.
  - can be loaded as and when needed / called.



dynamically loaded / linked.

∴ After traditional system, all others are layered.

* Virtual Machine:
  - VMware virtual box.
  - can create multiple Vmachines on top of host OS - called guest operating systems.



} separate processors & memory for the VM.

  - can switch to any guest OS (virtual time multiplexing).
  - diff. from dual boot - can run only one OS at a time.

29/1.

* Unit 2 : Process & Threads *

Process:

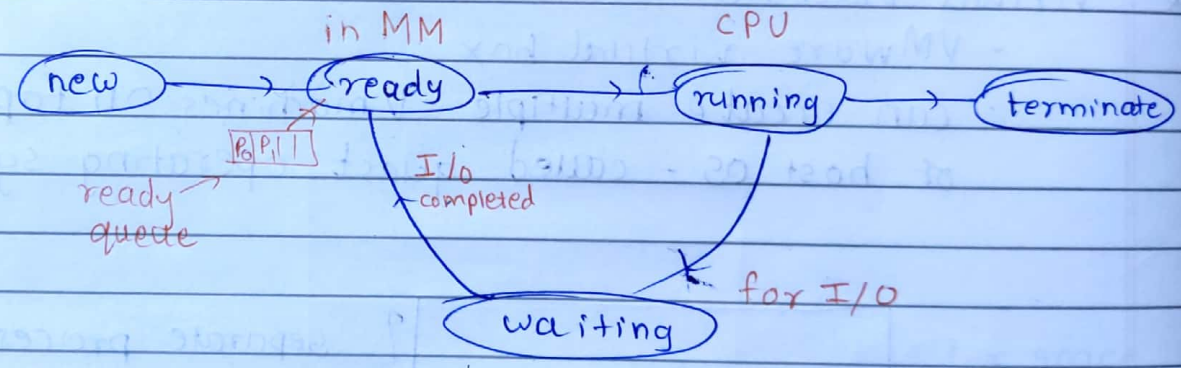
Program. currently executing.
Dynamic active entity

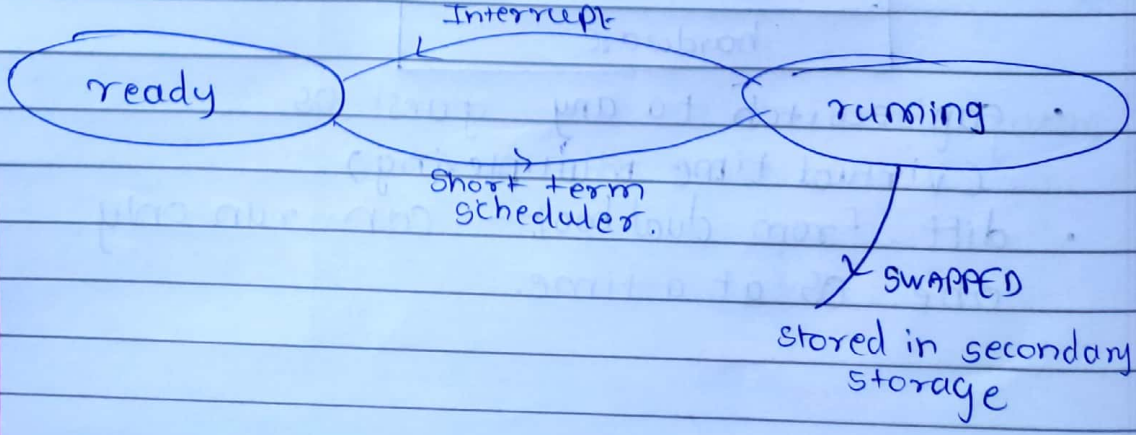
Secondary Storage	Main memory
Static	Dynamic
Program.	Process.

Threads: ?

A process can have single or multiple threads. (consisting of).

States of process.

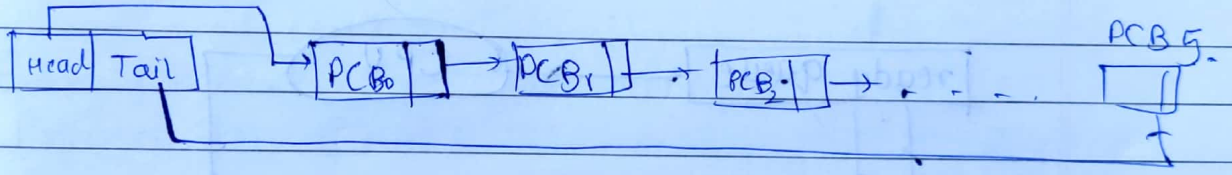




Whenever, process is executed, every ~~state~~ process will have Process Control Block (PCB), when process is over, PCB is released.

Process Control Block: (Task Ctrl Block)
Tasks are storing of:
i). Process State
ii). Program Counter.
iii). CPU register.
iv). CPU scheduling
v). Memory management (paging, seg.)
vi). Accounting info: (ID, process duration, etc.)

Storage of PCBs in ready queue



Scheduling queue:

i). I/p / Job Queue.
whatever processes are enqueued.
ii). Ready queue:
As scheduled by CPU, ready to execute.
iv). Device Queue:
Waiting list for I/O operations.

1) Secondary I/P Queue → MM ready queue

Long term scheduler

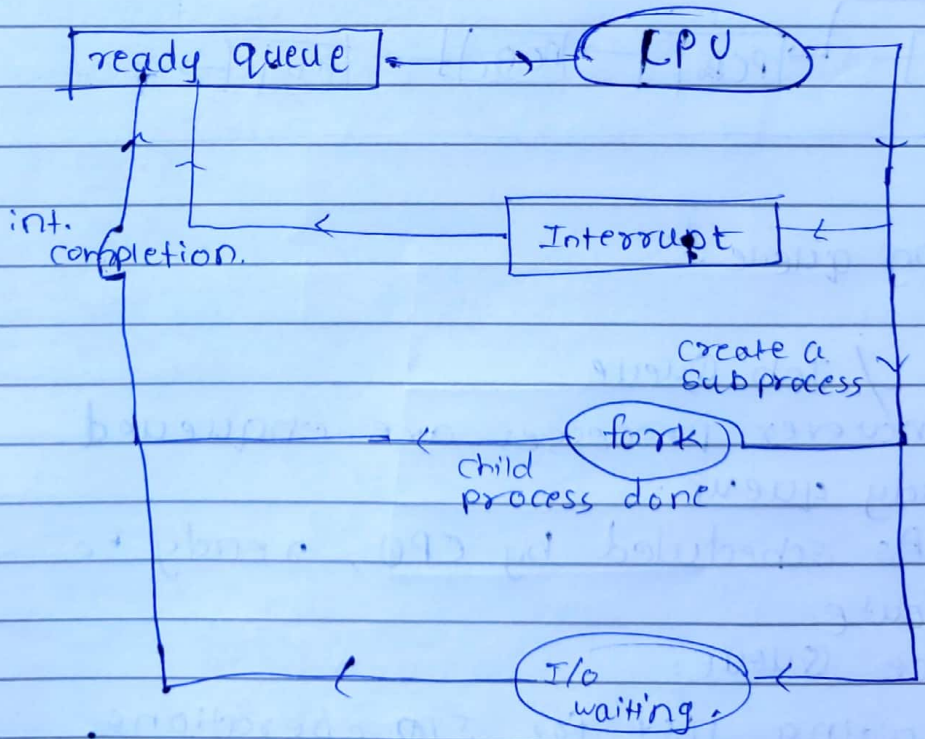
2) MM ready → CPU

Short term scheduler.

3) CPU MM → secondary (Swapping)

Medium term scheduler.

* Process in Execution:



* Context switch:
(Dispatch latency).

Process to process switching.
Old PCB stored,
New PCB created.

Till tomorrow, whatever will be taught ⟹ T-1 portion.

30/1.

Process operation:
- Creation
- Termination.

CPU burst - I/O wait cycle.
Scheduling criteria.

Every process, has id, unique Process ID (pid).
Parent process can create child process.
Sometimes,
i) Parent has to wait for child.
ii). Parent & child execute together.

Assignment 3

#include <sys/type.h>
<stdio.h>
<unistd.h>
} header files required to deal with systemCalls
stdin, stdout, stderr.

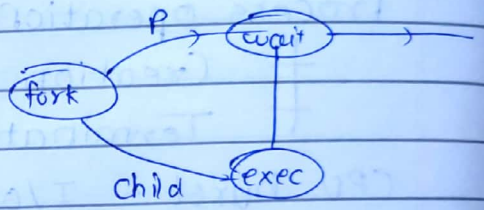
```
int main ( ) {

    pid_t  pid;
    pid = fork( )
```

→ in order to create a new process.

Generally parent & child proc. should have pid > 0

```
if (pid < 0) {

fprintf ( stderr,
" fork failed");

    return 1

}

else if ( pid == 0 ) {

    execlp ("/bin/ls", "ls", NULL);
        exit

}
else
    wait (NULL)
    printf (" child completed ");

    return

}
```

Parent is exec., creat new child, wait for child and then resum



Generally, parent terminates after
i) child is done
ii) child timer expires.

* CPU burst:

| CPU Burst
| I/O
| CPU Burst
| I/O
:

Process whenever is executing it will alternate / go through these states.

and so on..

CPU → process terminated.

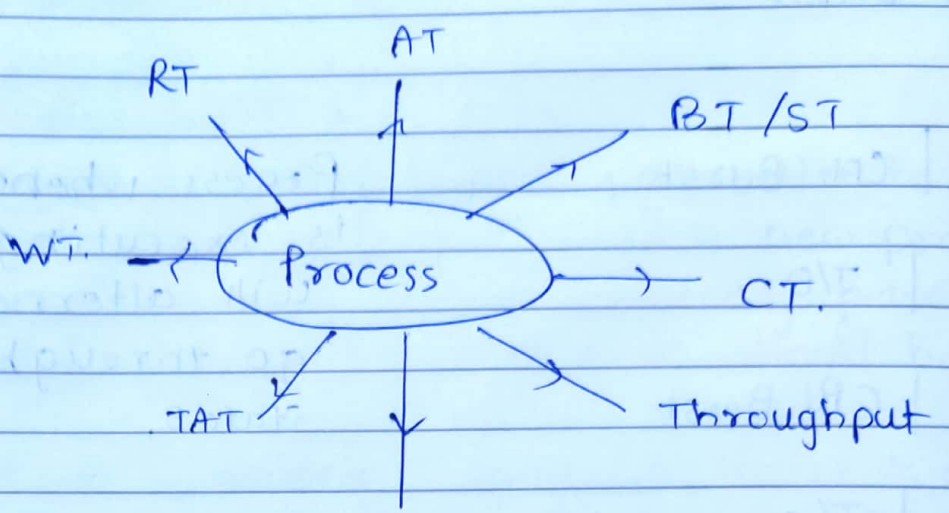
* Scheduling:

1) running → waiting (for I/O)
2) running → ready (interrupt)
3) waiting → ready (I/O done)
4) Process termination

these four types require scheduling.

For conditions (1) and (4), processes are non preemptive / cooperative. (ziddi processes)

For conditions (2) and (3), it can wait, it gives chance to higher priority process / interrupt



WT : Waiting time, how much time it will wait in the ready queue.

TAT : Turn Around Time

RT : Response time . After submission, how long does it take to respond.

BT/ST : Burst time / Service Time.

CT : Completion Time

AT : Arrival Time.

Considered while scheduling.

* Scheduling Algo :

i) FCFS : First come, First service.
AT, non preemptive, convoy effect.

ii) SJF : Shortest Job First.
Least BT, will be fed to processor.
AT + BT, non preemptive.
same BT, then use FCFS.

iii) SRTF : Shortest remaining Time first.

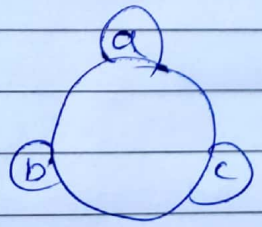
P1	P2

A.T: 0.01ms 0.9ms
B.T: more less

P1 is 'understanding'
=> At 0.9, P1 removes, goes to queue, P2 served first.

iv) Priority :
Based on priority.

v) Round Robin



Arrival of multiple ✓
Each one get a chance in order (like timesharing).

SJF

* Few terms:

i) Schedule length = $\max(C_i) - \min(A_i)$

ii) Waited TAT = $\frac{C_i - A_i}{X_i}$

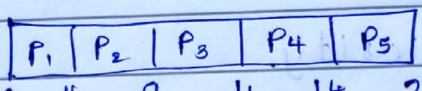
iii).

* Solve:

1.

Pno.	AT	BT	CT	TAT	WT.
1	0	4	0 4	4	0
2	1	5	4 9	8	4
3	2	2	9 11	9	9
4	3	3	11 14	11	11
5	4	6	20	16	14

FCFS:



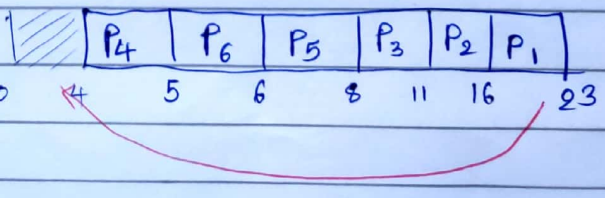
Gantt chart.

(will be CT)

2.

Pno.	AT	BT.
1	0	7
2	1	5
3	2	3
4	3	1
5	4	2
6	5	1

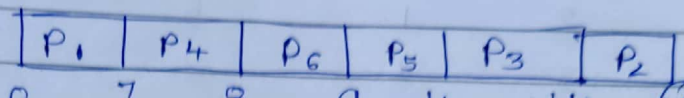
SJF



~~SRTF:~~ Make sure, don't keep idle.

Corrected:

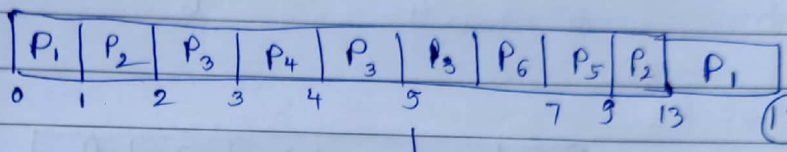
SJF



Same

SRTF:

(Preem



Don't do more context switch if it is equal / already there.

T1

4/2

* Priority Scheduling:

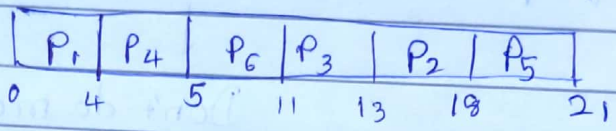
Preemptive (AT + priority.)

Non preemptive.

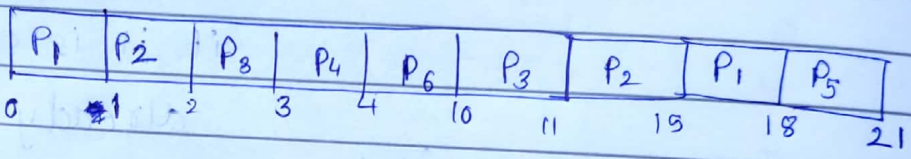
next page

P No.	AT	BT	priority.
1	0	4	4
2	1	5	5
3	2	2	6
4	3	1	8
5	3	3	2
6	4	6	7

Non Preemp.



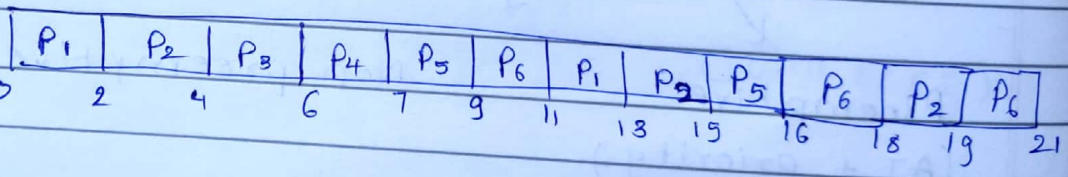
Preemp.



* Round Robin:

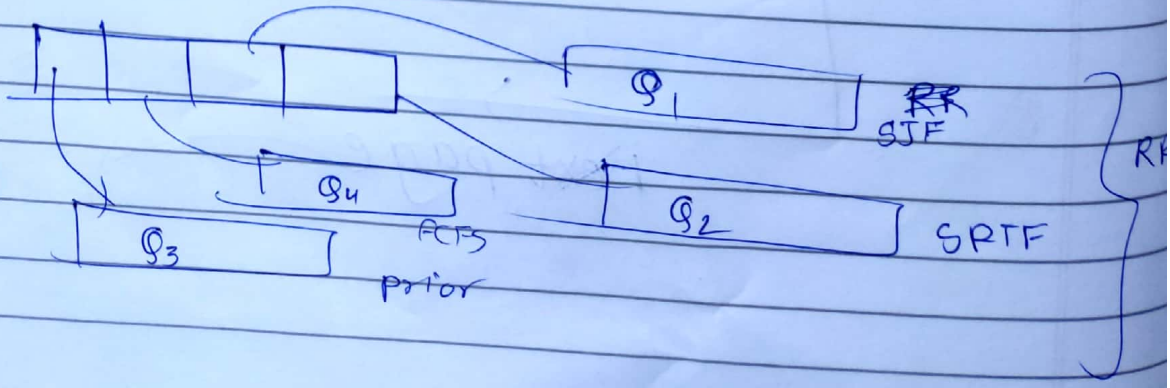
A.T + Time quantum / time slice.

Same as above



* Multilevel queue scheduling:

Ready queue is partitioned into different queues.



Here, every queue is scheduled using different algorithm.

Multilevel feed back queue:
Similar, just that, if there is priority considered Q_1 can take more time of CPU. If at all some process in the queue Q_2 is waiting for long, then it can be scheduled in another queue. Time slice per queue is variable.

Aging: High priority is given to a process of lower priority.

Generally

SJF = optimal algorithm.
Bcoz it calculates ~~min.~~ avg. waiting time.
min. avg waiting

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

T_{n+1} → Predicted CPU burst.
t_n → n^{th} cpu burst.
T_n → Past CPU Burst.

$$0 \leq \alpha \leq 1.$$

If $\alpha = 0$, $\boxed{T_{n+1} = T_n}$

$\alpha = 1$, $\boxed{T_{n+1} = t_n}$.

Consider a system with SJF with exponential avg. technique. what could be the next CPU burst of proc which has earliest completed burst of 5, 8, 3 & 5. Now for an initial value of $T_1 = 10$, $\alpha = 1/2$.

Calculate

$$T_{i+1} = \alpha t_n + (1-\alpha) T_i$$

$$T_{1+1} = 0.5 * 5 + (1-0.5) * 10$$

$$T_2 = 2.5 + 5$$

$$= \underline{7.5}$$

$$T_3 = (0.5 * 8) + (1-0.5) * 7.5$$

$$= 4 + 3.75 = \underline{7.75}.$$

$$T_4 = (0.5 \times 3) + (1-0.5)(7.75)$$

$$= 1.5 + 3.875$$

$$T_5 = \frac{(5 + 5.375)}{2} = \underline{5.1875}.$$