22/8/17

CPU scheduling

It arises from the multiprogramming concept. → multiple prog.s run in the system simultaneously.

(Every prog. consists of some I/O & some executable instructions.)

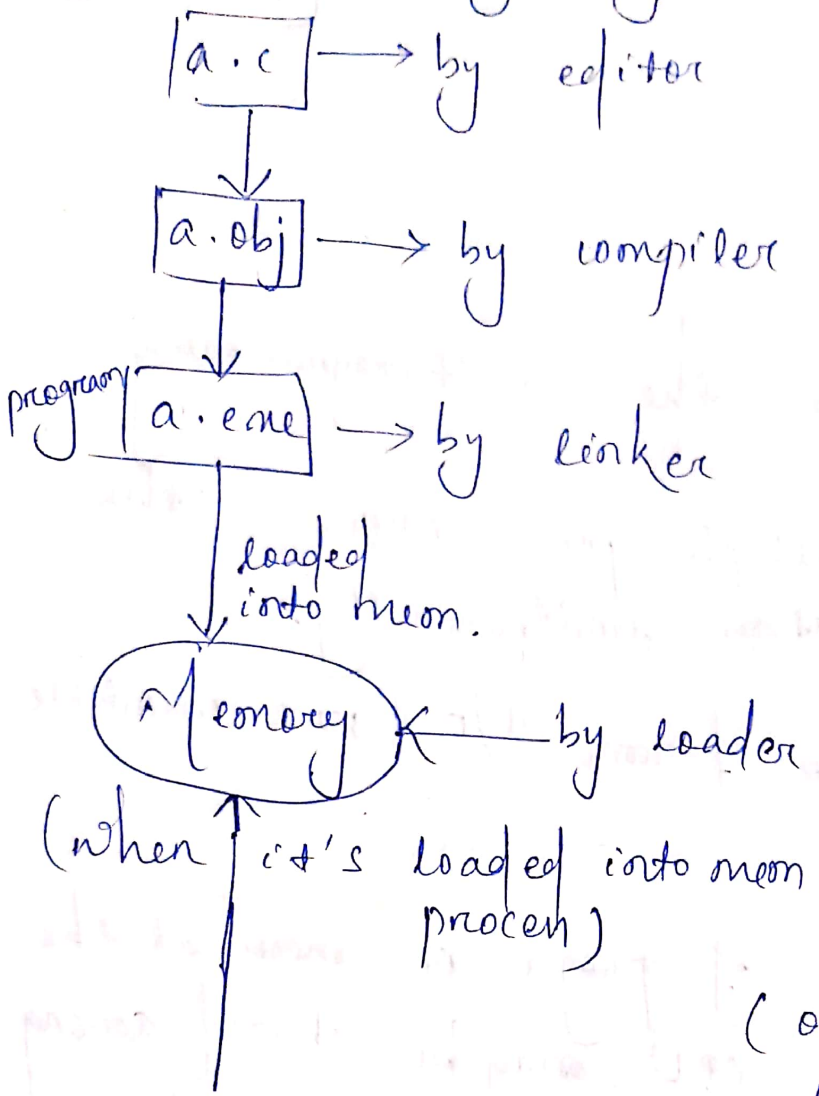
∴ By having no. of prog.s in mem. at the same time, CPU may be shared among them. This improves the overall efficiency of the system by getting more works done in less time.

- The benefits of this scheme are:
  - Increased CPU utilization
  - " Job throughput
    → amount of jobs that can be completed within a time interval.

Procen - Prog. under execution.

Prog - Set of self contained instructions. There

must be a beginning & an end.



(when it's loaded into mem. it's called procen)

executed by OS

(one prog. can lead to many procen)

Program	Procen
- static in nature	- dynamic
- Takes space in the disk	- It has a life. It takes birth & dies.
- Permanent	- Temporary

(If a prog. is run 5 times then 5 processes are created, each are identical. B'cz each process is a unique PID (Process Identification) number.)

Process → It's a prog. in execution.

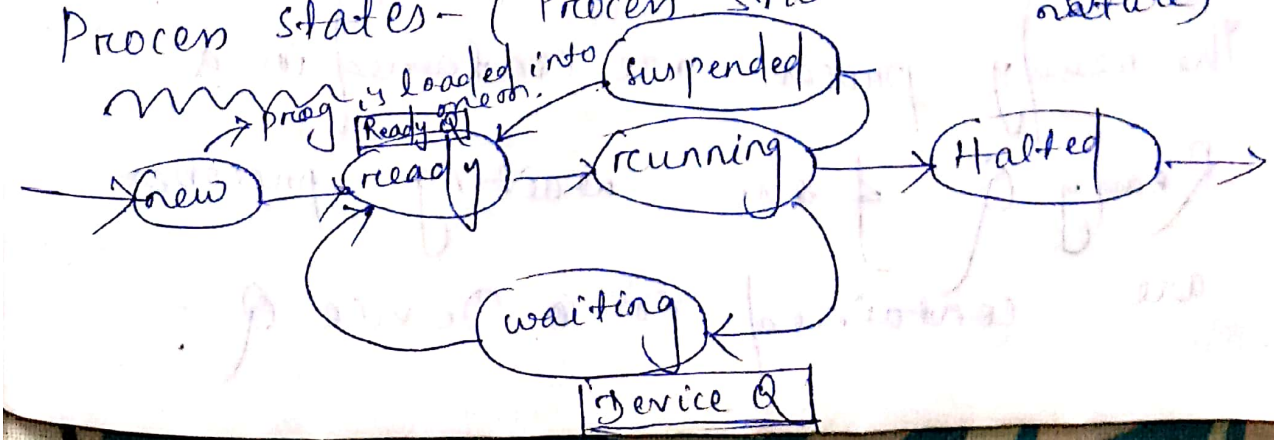
Process execution | CPU | I/O | CPU | I/O | CPU |

Process execution - It's a cycle of CPU execution & I/O wait, beginning & ending with a CPU burst.

(CPU burst → Requirement of CPU time.)

- During this the process goes through diff^n process states.

Process states - (Process states are cyclic in nature)



ready → process requires some resources, resources are allocated to it, but the CPU is not free.

running → CPU time is assigned

waiting → waiting for I/O opn

Halted → all the allocated resources are released from the process.

suspended → sometimes some of the processes may be suspended. lower priority processes are taken out of the mem. temporarily. (their state is saved)

Applⁿ
- CPU bound → I/O is less, CPU time requirement is more → Ex → Scientific applⁿs
- I/O bound → I/O is more, CPU time requirement is less → Ex → commercial applⁿ

Many processes are in ready state but only 1 " is " running ".

The ready process are contained in a Ready Q of the waiting processes are contained in a Device Q.

1 CPU → 1 ready Q

each device has its own device Q.

Process Control Block (PCB)

(when a process created, its birth record, i.e. PCB, is created)

- Each process is represented by a PCB, which is a data block or record containing the following info.
  - pid-no. (process identification no.)
    - ↳ assigned by OS.
  - Process state (current state of process)
  - Program Counter (PC register is one value of it holds the value in the currently running prog.)
  - CPU registers

(When CPU switches from 1 process to another, the CPU register values of old process are loaded to the corresponding PCB, the register values of the new process are loaded from PCB into the CPU registers. This is called context switching & the time required for this switching is called context switching time.)

- CPU scheduling info. (it requires priority)
  ↳ (what's the priority of the current procen)
- Mem. management info.
  (LBR (Lower Bound Register) & UBR (Upper Bound Register))
  ↳ the procen is in which part of mem.
- I/O status info.
  ↳ what are the files opened by your procen & a particular file is read upto which record, i.e. a file pointer.
- Accounting info.

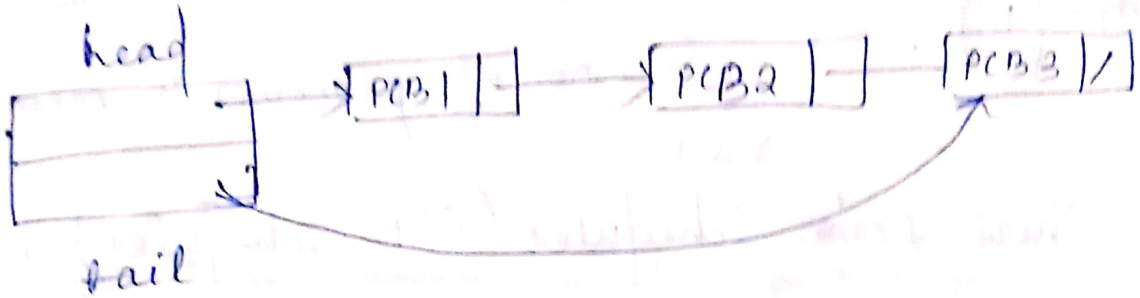
10 procen in mem ⇒ 10 PCB in mem.
3 " are suspended ⇒ 7 procenes in mem.
⇒ 10 PCB in mem.

A procen is suspended ⇒ OS's PCB is not suspended from mem.
from mem.
A procen is terminated ⇒ PCB is destroyed from mem.

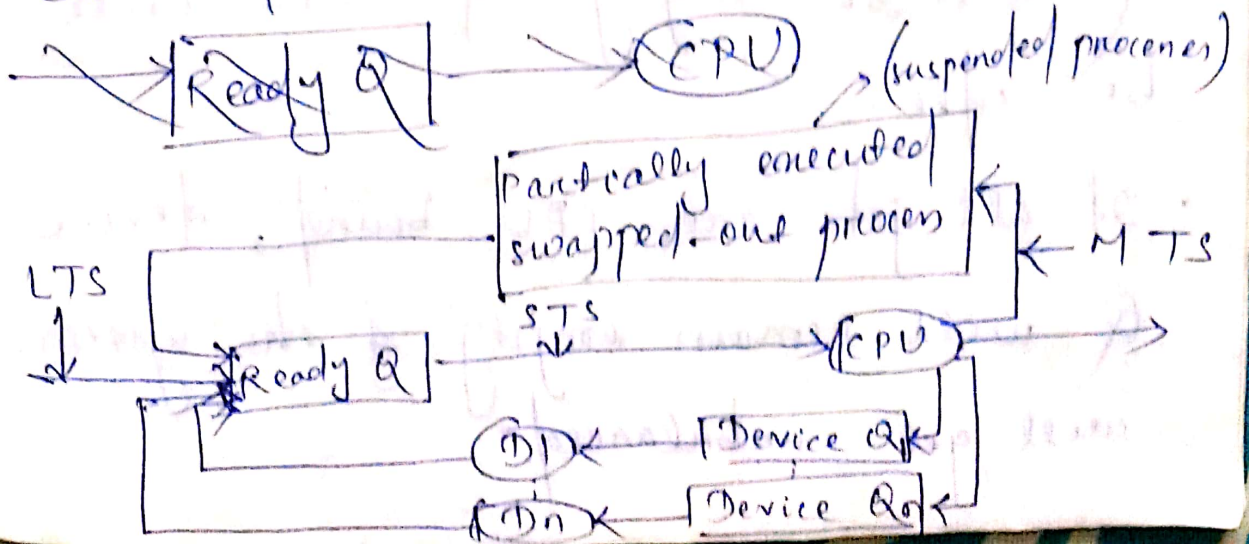
PCB must be stored in main. (Therefore it's called data structure)

scheduling Qs → Ready Q
→ Device Q

Ready Q / Device Q



- Processes which are ready / waiting to execute are kept in the ready Q.
- The list of processes waiting for a particular I/O device are put into a device Q.
- Each device has its own device Q.



Different types of ~~process~~ Schedulers

(1) Long term Scheduler (Job scheduler) → (LTS)

- It determines which jobs are admitted to the system for processing.
- It controls the degree of multiprogramming (DOM) (P). → no. of process in mem.

(2) short term Scheduler (CPU scheduler) → STS

- It selects from among the jobs which are ready to execute & allocates CPU to one of them.

(3) Medium term Scheduler (Swapper) → MTS

- If all jobs are I/O bound, ready Q will be empty & STS will be idle.
- If all jobs are CPU bound, device Q will remain empty & the system will get unbalanced.

- Therefore, a job mix up both the categories are required.
- If required the MTS removes active process from mem. & reduces the DMP.
- At some later time that process can be brought into mem. to resume its op^n/~~exe~~ further execution.
- This scheme is otherwise called swapping.

25/8/15

CPU scheduling Algorithm

- It decides which of the process in RQ (Ready Q) is allocated the CPU next.

Performance Criteria of the algorithms

(a) Maximise:
   (i) CPU utilisation
   (ii) Throughput

(b) Minimise :

(i) Turnaround time

(Time gap b/w job submission & job completion)

↳ during this time the process is sometimes in the RQ, sometimes in running state & sometimes in waiting state

(ii) Waiting time

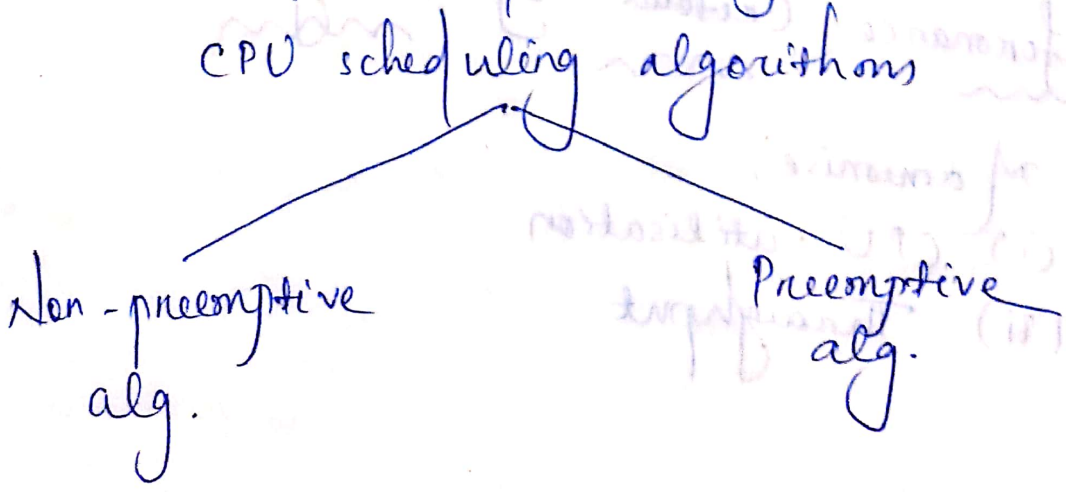
(Turn around time - Running time)

(iii) Response time

(Time gap b/w job submission & getting the 1st response from the CPU.)

(Generally turnaround time & waiting time are taken into consideration to compare diff^rt algorithms.)

CPU scheduling algorithms



(preempt → to take away forcefully)

Non-preemptive alg.

1. FCFS (First-Come-First-Served)
2. SJF (Shortest Job First)
3. Priority

Preemptive alg.

1. RR (Round-Robin) (equivalent to FCFS)
2. SRTF (Shortest remaining time first) (equivalent to SJF)
3. Priority (equivalent to priority without preemption)

(Unix uses RR alg. It's the truely timesharing alg.)

(i) First-come-first-served (FCFS) alg.

- Using this alg. the process which requests the CPU first is allocated the CPU first.
- when a process enters the ready Q, it's linked to the tail of the Q.

when the CPU becomes free, it's allocated to the process at the head of the ready Q.

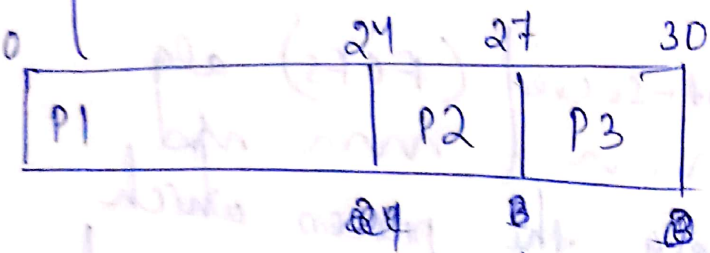
Process	CPU Burst Time (BT)	TAT	WT
P1	24	24	0
P2	3	27	24
P3	3	30	27
		81	51

TAT → Turn Around Time = CPU BT + WT

WT → Waiting Time = TAT − CPU BT

Process execution ~~CPU utilization~~ is depicted through a chat, called Gantt chart.

Gantt chart



Avg. TAT
$= \frac{81}{3} = 27$

Avg. WT
$= \frac{51}{3} = 17$

(assume that all processes arrive at the same time.)

(For non ~~of~~ preemptive, order of arrival is imp., arrival time is not imp. In preemptive, arrival time is imp)

Adv. of this alg.

- It's simple to understand & implement.

Disadv.

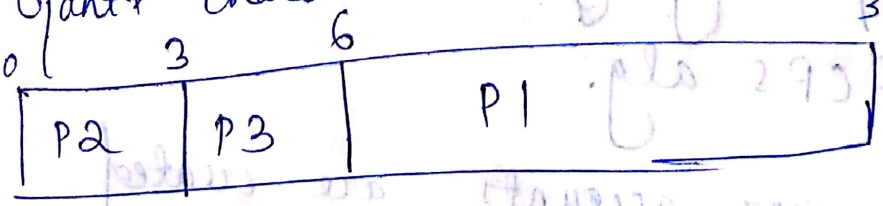
- The avg. TAT & WT becomes large.

(ii) Shortest Job First (SJF) alg.

- This alg. anociates with each job the length of its next CPU burst. (predicted value)

when the CPU is available, it's assigned to the job with the smallest next CPU burst. If there are more than one jobs with the same next CPU burst then FCFS is used among them.

Gantt chart



TAT	WT
30	6
3	0
6	3
39	9

Avg. TAT = $\frac{39}{3} = 13$

Avg. WT = $\frac{9}{3} = 3$

Adv.

- SJF is optimal because it gives minimum avg. WT & TAT for a given set of jobs.
- SJF also decreases the WT for shorter jobs.

Disadv.

- SJF is based on a predicted value which may or may not be true.

(iii) Priority alg.

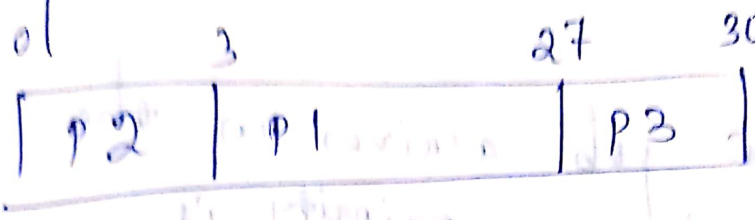
- Here a priority is associated with each job & CPU is allocated to the job with the highest priority.
- Equal priority jobs are scheduled using FCFS alg.

(When user accounts are created by the super user, every account has a priority, when you run a prog. it runs according to the priority of your account.)

Process	Priority	CPU BT	TAT	WT
P_1	1	24	27	3
P_2	0	3	3	0
P_3	2	3	30	27
			60	30

Avg. $= \frac{60}{3} = 20$ Avg $= \frac{30}{3} = 10$

Gantt chart



(Real life appln require priority)

Adv.

- It gives priority to time sensitive appln.

Disadv.

- It can sometimes lead to the prob. of Starvation (indefinite blocking).

(We should use the priority with a soln for starvation.

- Solution to the prob. of Starvation is Aging.

→ if a process waits for a longer period of time then its priority increases to the next level of priority

- Aging is a technique of gradually increasing the priority of jobs that wait in the system for a long time.

1/9/15

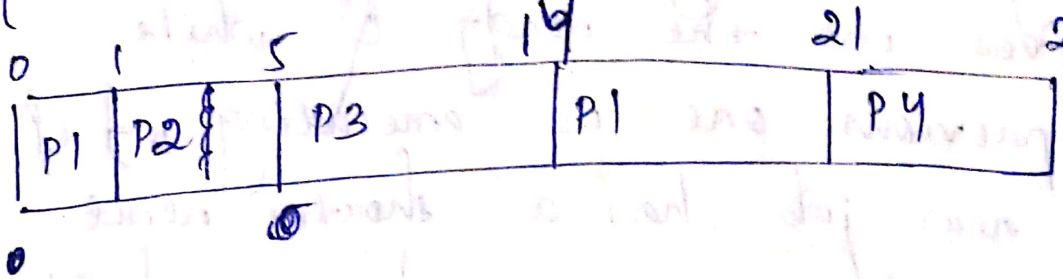
Preemptive alg.

(i) Priority

- When a new job arrives in the ready queue, its priority is compared with the priority of currently running process.
- If the priority of the newly arrived job is higher as compared to currently running process, then CPU is preempted from the currently running process & is allocated to newly added job.

Process	AT (Arrival Time)	Priority	CPU BT	TAT (Finished time − AT)	WT
P1	0	1	8-17	21	13
P2	1	0	4	4	0
P3	2	0	9	12	3
P4	3	2	5	23	18

Gantt chart



Avg. TAT = $\frac{6+20+15}{4} = 15$

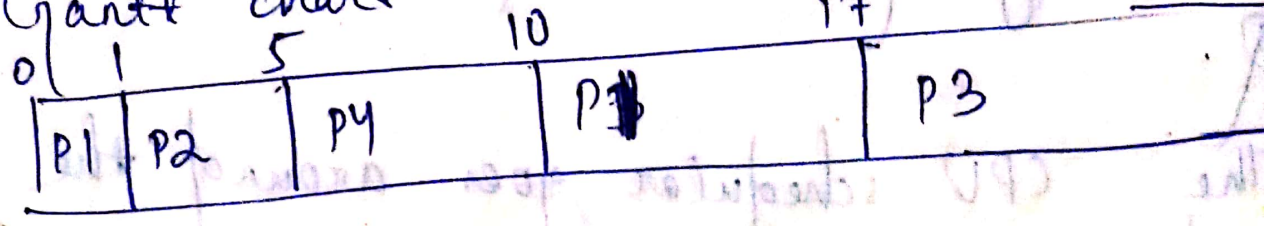
Avg. WT = $\frac{34}{4} = 8.5$

(Adv. & disadv. are same with the prev. priority alg.)

(ii) Shortest Remaining Time First (SRTF)

Process	AT	CPU BT	TAT	WT
P1	0	8-1=7	17	9
P2	1	4	4	0
P3	2	9	24	15
P4	3	5	7	2

Gantt chart



Avg. TAT = $\frac{52}{4} = 13$

Avg. WT = $\frac{26}{4} = 6.5$

- Using this alg. when a new jobs arrives in the ready Q while a previous one is executing, if the new job has a shorter next CPU burst as compared to what's ~~left of~~ the left over time of currently executing job, then CPU will be preempted from the currently running job & be allocated to the newly added job.

(Adv. & disadv. are same as SJF)
(Starvation may happen here)

(iii) Round Robin (RR) Alg.

- Using this alg. a time quantum (TQ) is defined.
- The ready Q is treated as a circular Q.
- The CPU scheduler goes around the ready Q, allocating the CPU to each process for a time interval of 1 time quantum.

- The scheduler picks up the 1st job from the ready Q, sets a timer to interrupt after 1 TQ & then switches to the next job from the ready Q. This switching is known as Context Switching.

- This requires the CPU to save the registers for the old process & load the registers for the new process.

- If the CPU burst is less than the TQ, then the process will release the CPU voluntarily for the next process.

- On the other hand, if the CPU burst is greater than the time quantum then the timer will go off & will cause an interrupt so that CPU can proceed with the next job.

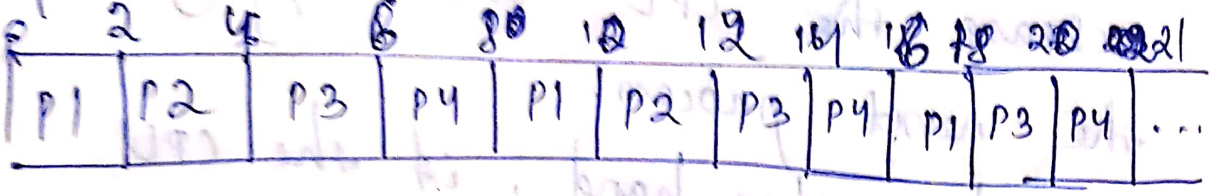
- The TQ value should be choosen very carefully. Because if it becomes

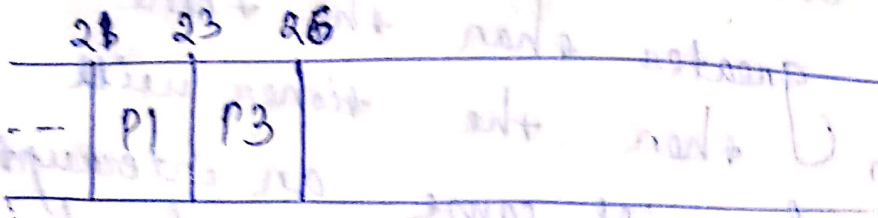
too large, if become equivalent to FCFS if it is too low, it leads to very frequent context switching which is not desirable.

Process	AT	CPU BT	TAT	WT
P1	0	8-2=6-2=4 =2-2=0	23	15
P2	1	4-2=2-2=0	11	7
P3	2	9-2=7-2 =5-2=3-2 =1	24	15
P4	3	5-2=3-2=1-1 =0	18	13

TQ = 2

Gantt chart





Avg. TAT = $\frac{76}{4} = 19$

Avg. WT = $\frac{50}{4} = 12.5$

Adv.
- There will be no starvation.
- In the true sense it's a time sharing system, which is used by the popular alg.. Unix.

(Unix uses RR along with priority. Generally there's a ready Q per CPU. But Unix uses more than one ready Q for one CPU.)

Job — foreground job
Job — background job
↳ these jobs have lower priority

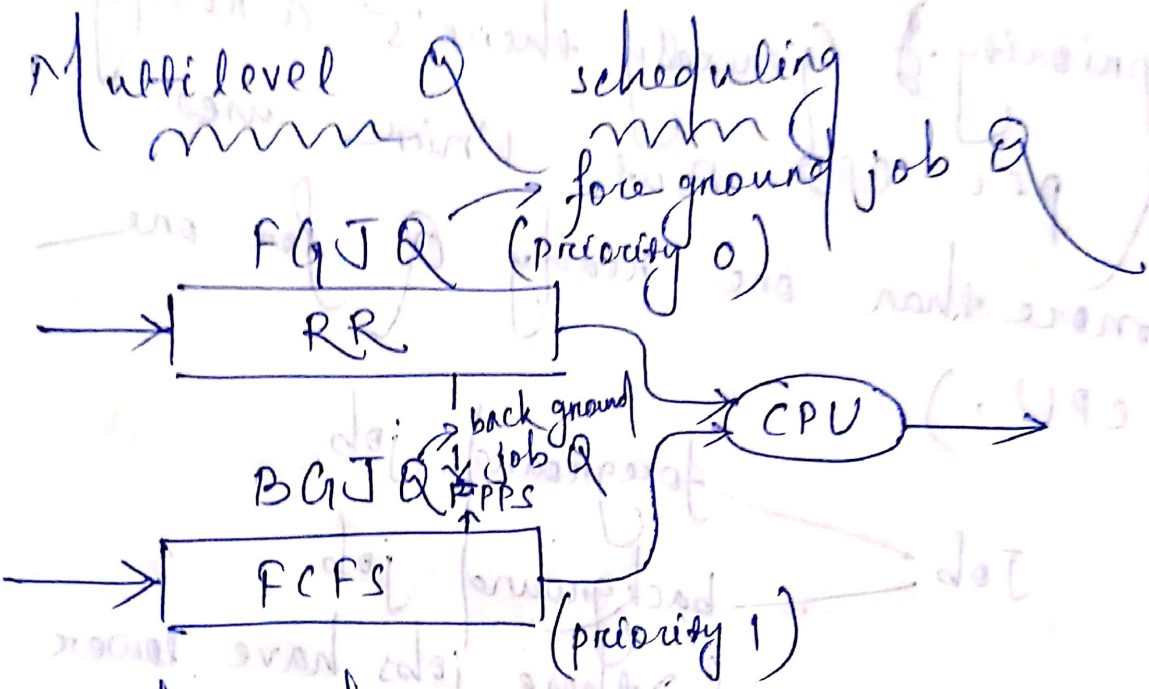
In Unix if we give the command

$ prog1 &

then prog1 will run in the background & we can give another command immediately.
But if the command written without '&', then job will run in foreground

mode. In this case we can give another command only after the prog. is executed. Unix maintains 2 ready Q for one CPU: one for background jobs & another for foreground jobs.

Multilevel Q scheduling



Whether the job of 1st Q or 2nd Q will be executed depends upon their priority. BGJQ has lower priority than FGJQ. This is called fixed priority scheduling. Foreground jobs used round robin alg. & background jobs use FCFS alg.

✓ Jobs are classified into diff categories like foreground jobs & background jobs.

- The ready Q is partitioned into separate Q's.
- The jobs are permanently assigned to one of the Q's depending upon the job type.
- First there will be a scheduling b/w the Q's which's fixed priority preemptive schedule.
- Each Q will have its own scheduling alg., which is round robin for foreground job Q & FCFS for background job Q.
- This can lead to starvation.

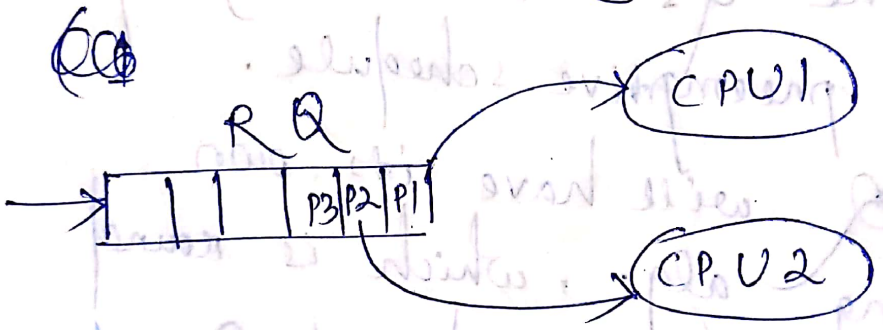
Multilevel feedback Q scheduling

- It allows a job to move b/w the Q's.
- If a job uses too much of

CPU time it will be moved to a lower priority. Similarly if a job waits too long in a lower priority Q, it can be moved to a higher priority Q.

Multiple Processor Scheduling

(1) Homogenous (All the processors are of the same type)
→ it uses load sharing

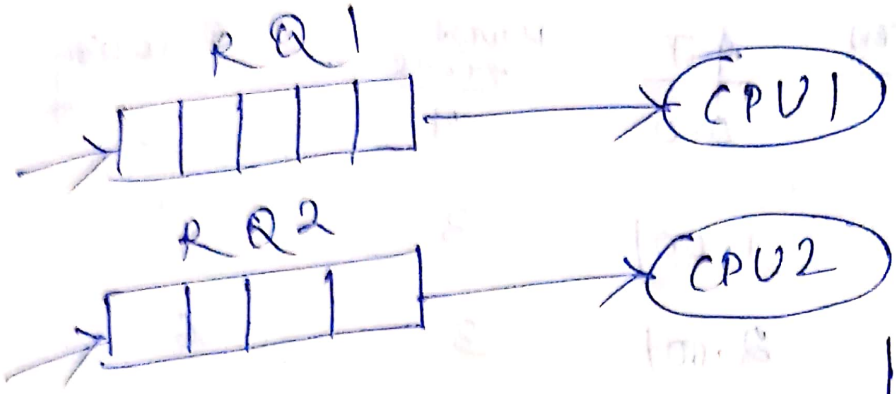


(1 CPU, 1 RQ → Time sharing
Multiple CPU, 1 RQ → Load ")

- If there are several homogenous processors, then load sharing is used & a common ready Q is used where all the jobs are put & are scheduled to

any available processor.

② Heterogenous processors



- If the processors are heterogenous, then each processor will have its own Q & its own scheduling alg.

8/9/15

TAT → Ending time − AT
or WT + CPU burst time
WT → Starting time − AT
or TAT − AT

Process	A.T	Processing time
A	0.0	3
B	1.001	6
C	4.001	4
D	6.001	2

Draw the Gantt chart & find out the TAT & WT using

(i) FCFS
(ii) SJF
(iii) SRTF
(iv) RR (TQ = 2)
(v) RR (TQ = 1)

Q. ②

Procen	AT	Burst time	Priority
A	0.0	4	3
B	1.00	3	4
C	2.00	3	6
D	3.00	5	5

Draw the Gantt chart, find out TAT & WT using
(i) Preemptive priority
(ii) Non preemptive "

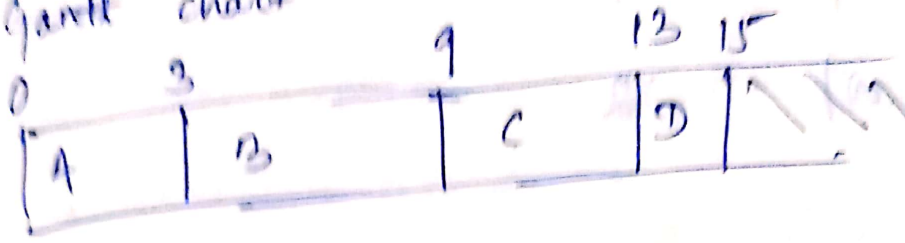
Ans

① FCFS

Procen	AT	Procesing time
A	0.0	3
B	1.00	6
C	4.00	4
D	6.00	2

TAT	WT
3	0
7.999	1.999
8.999	4.999
8.999	6.999

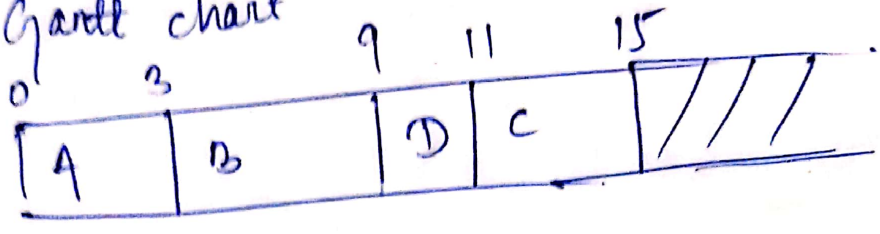
Gantt chart



SJF

Procen	AT	Processing time	TAT	WT
A	0.0	3		
B	1.00	6		
C	4.00	4		
D	6.00	2		

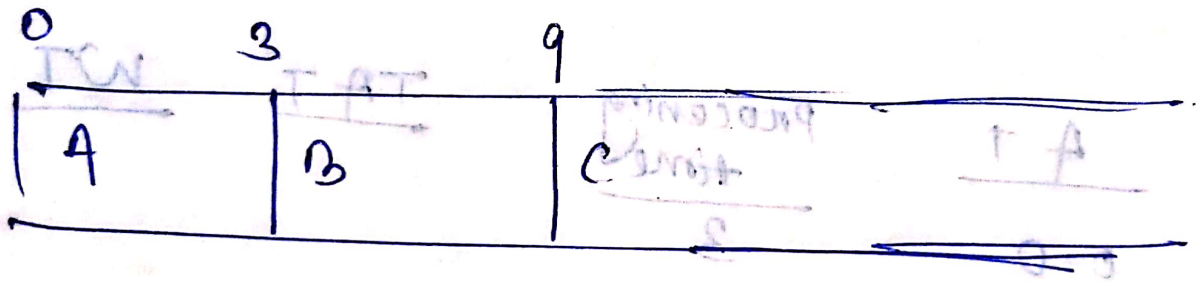
Gantt chart



SRTF

Procen	AT	Procening time	TAT	WT
A	0.0	3		
B	1.00	6		
C	4.00	4		
D	6.00	2		

Gantt chart

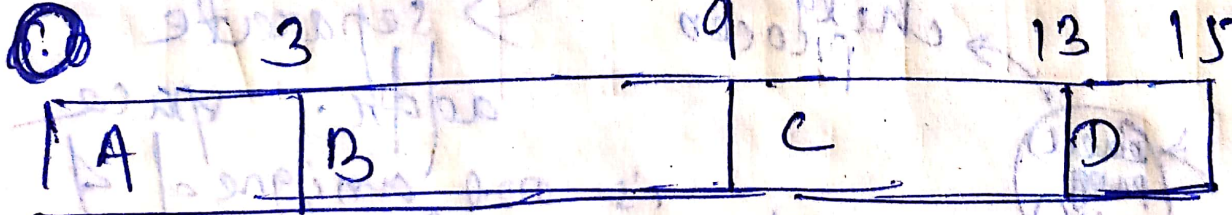


Ans of 8/9/15 H.W

(1)(i) FCFS

Process	AT	Processing time	TAT	WT
A	0.0	3	3	0
B	1.001	6	7.999	1.999
C	4.001	4	8.999	4.999
D	6.001	2	8.999	6.999
			28.997	13.997

Gantt chart



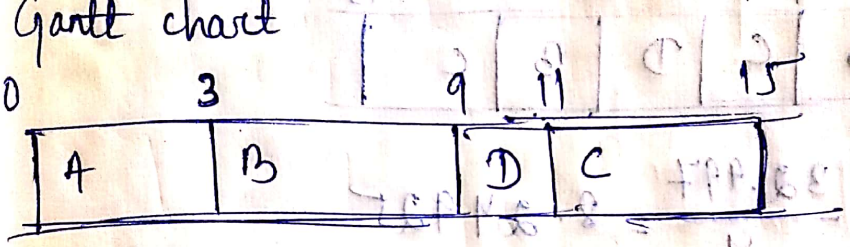
Avg. TAT = $\frac{28.997}{4}$ = 7.24925

Avg. WT = $\frac{13.997}{4}$ = 3.49925

(ii) SJF

Procen	AT	Processing time	TAT	WT
A	0.0	3	3	0
B	1.00	6	7.999	1.999
C	4.00	4	10.999	6.999
D	6.00	2	4.999	2.999
			26.997	11.997

Gantt chart



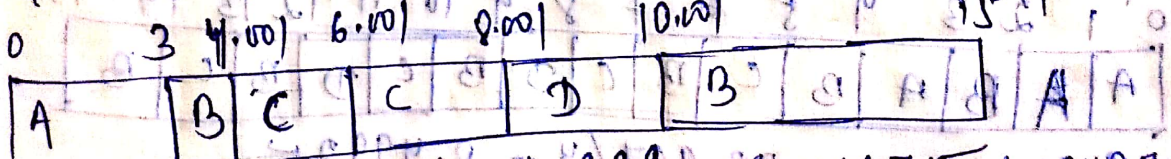
Avg. TAT = $\frac{26.997}{4}$ = 6.74925

Avg. WT = $\frac{11.997}{4}$ = 2.99925

(iii) SRTF

Procen	AT	Processing time	Remaining time	TAT	WT
A	0.0	3	-	3	0
B	1.00	6	4.999	13.999	7.999
C	4.00	4	✗	4	0
D	6.00	2	-	4	2
				24.999	9.999

Gantt chart



Avg. TAT = $\frac{24.999}{4}$ = ~~6.49975~~ 6.24975

Avg. WT = $\frac{8.999}{4}$ = ~~2.99975~~ 2.24975

(iv) RR (TQ = 2)

Procen	AT	Procening time	Remaining time	TAT	WT
A	0.0	3	~~1~~	5	2
B	1.001	6	~~4 2~~	11.999	5.999
C	4.001	4	~~2~~	10.999	6.999
D	6.001	2	-	4.999	2.999
				32.997	~~17~~.997

Gantt chart

0	2	4	5	7	9	11	13	15
A	B	A	B	C	D	B	C	

Avg. TAT = $\frac{32.997}{4} = 8.24925$

Avg. WT = $\frac{~~17~~.997}{4} = ~~1.749 25~~$ 4.99925

(v) RR (TQ = 1)

Procen	AT	Procening time	Remaining time	TAT	WT
A	0.0	3	~~2~~ 1	4	1
B	1.001	6	~~5 4 3 2~~ 1	13.999	~~7~~.999
C	4.001	4	~~3 2~~ 1	9.999	~~5~~.999
D	6.001	2	1	5.999	~~3~~.999
				33.997	18.997

Gantt chart

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	A	B	A	B	C	B	C	D	B	C	D	B	C	B	

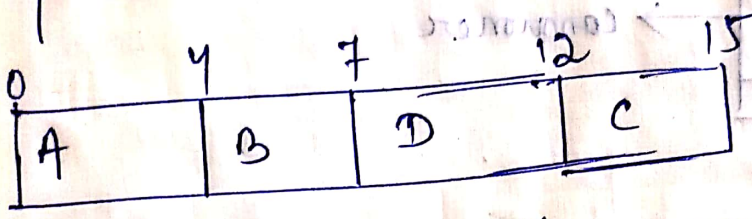
Avg. TAT = 33.997/4 = 8.49925

Avg. WT = 18.997/4 = ~~0.99925~~ 4.74925

② (i) Preemptive priority

Procen	AT	Burst time	Priority	TAT	WT
A	0.0	4	3	4	0
B	1.001	3	4	5.999	2.999
C	2.101	3	6	12.999	9.999
D	3.101	5	5	8.999	3.999
				31.997	16.997

Gantt chart



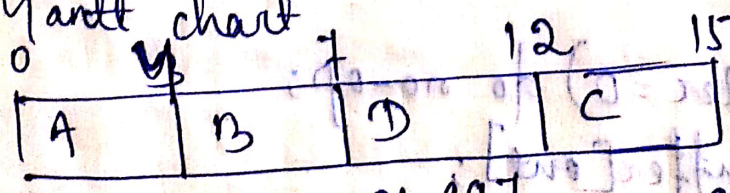
Avg. TAT = $\frac{31.997}{4} = 7.99925$

Avg. WT = $\frac{16.997}{4} = 4.24925$

(ii) Non preemptive priority

Procen	AT	Burst time	Priority	TAT	WT
A	0.0	4	3	4	0
B	1.101	3	4	5.999	2.999
C	2.101	3	6	12.999	9.999
D	3.101	5	5	8.999	3.999
				31.997	16.997

Gantt chart



Avg. TAT = $\frac{31.997}{4} = 7.99925$

Avg. WT = $\frac{16.997}{4} = 4.24925$