Sumedh

- ## CPU Scheduling

The CPU scheduler selects from among the processes in memory that are ready to execute and allocates CPU to one of them

CPU scheduling takes place when
a process switches from running to wait / steady / terminated
or wait to ready

- ## Scheduling criteria

i) CPU utilization - Keep CPU as busy as possible
Heavy load - 90%
light - 40%

ii) Throughput - no of processes completed / unit time

iii) Response time - time take when a req submits till response comes

iv) waiting time - amt of time it waits in ready queue

v) Turnaround time - time to complete a press from submission to complition

- ## FCFS Scheduling (First come, first serve) [Non - pre-emptive]

The first job entered is first one to be serviced

eg process arrives in order P_1, P_2, P_3

	BT	CT	TAT	WT / RT
P_1	24	24	24	0
P_2	3	27	27	24
P_3	3	30	30	27
			Avg = 27	avg 17

P_1	P_2	P_3

0 24 27 30

CT − AT

TAT − BT

if Order P_2, P_3, P_1

P_2	P_3	P_1

0 3 6 30

	BT	CT	TAT	WT
P_1	6	30	30	6
P_2	0	3	3	0
P_3	3	6	0	3

- disadv

i) Short jobs stuck b/w big ones
ii) Non-pre-emptive, so if process executes for long time, baaki processes wait

- **SJF Scheduling (Shortest-job) [Non-pre-emptive]**

Here, all jobs arrive at same time, priority to one with less Burst time

eg P_1 : BT = 24
P_2 : " 3
P_3 " 3

P_2	P_3	P_1

0 3 6 30

	BT	CT	TAT	WT
P_1	24	30	30	6
P_2	3	3	3	0
P_3	3	6	6	3

eg	Process	AT	BT	CT	TAT	WT
		given in Q			CT - AT	TAT - BT
	P1	0	7	7	7	0
	P2	2	4	12	10	6
	P3	4	1	8	4	3
	P4	5	4	16	11	7

Gantt chart

P1	P3	P2	P4

0 7 8 12 16

- **Adv** : gives optimal WT
- **disadv** : long jobs wait for short ones

- **Priority Scheduling** [Pre-emptive / Non - Pre-emp]

CPU alloc to High priority process

eg (NP)

Process	BT	P	WT
A	8	2	0
B	1	1	1
C	1	3	9

B	A	C

0 1 9 10

eg (P)

PID	Priority	Arrival Time	Burst Time	Completion Time(CT)	Turn Around Time(TAT)	Waiting Time (WT)
P1	2(low)	0	~~4~~ 3	25	25	21
P2	4	1	~~2~~ 1	22	21	19
P3	6	2	~~3~~ 2	21	19	16
P4	10	3	~~5~~ ~~4~~ ~~3~~	12	9	4
P5	8	4	1	19	15	14
P6	12(high)	5	~~4~~ 0	9	4	0
P7	9	6	~~6~~	18	12	6

P1	P2	P3	P4	P4	P6	P4	P7	P8	P3	P2	P1

0 1 2 3 4 5 9 12 18 19 21 22 25

└ first check AT , then check priority and just do 1 unit of work until High prior process arrives then just complete rest process from High to low prior

- **Adv** : High prior jobs first
- **Disadv** : low prior jobs never execute completely till the ene

• RR scheduling (Round Robin)

Each process executes for only particular time unit given by time quantum so the process added again in ready queue

eg

Process	BT
P_1	~~53~~ 33
P_2	17
P_3	68
P_4	24

Time Qua = 20

Ready queue : P_1 P_2 P_3 P_4

Gantt chart

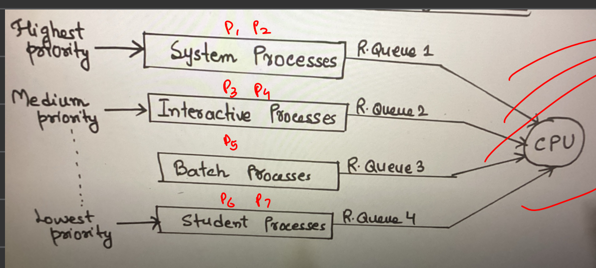
P_1	P_2	P_3	P_4	P_1	P_3	P_4	P_1	P_3	P_4

0 20 37 57 77 97 . . .

- **Adv** : good for small task
- **disadv** : More context switching

• Multi-lvl queue Scheduling

Here processes are classified into groups, basically seperate ready queues based on property and in that put diff processes

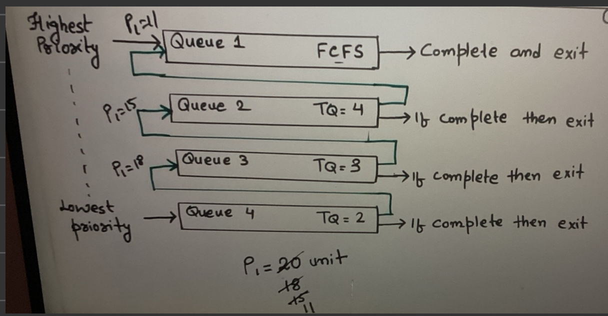


Can use diff scheduling algos for diff queues i.e FCFS, SJF, RR etc

- **Disadv** : Starvation

- **Multilevel Feedback Scheduling**

Here, only diff from above is that processes can move b/w diff ready queues



- **Adv** : No starvation

Considering the following example of a system, check whether the system is safe or not using Banker's algorithm. Determine the sequence if it is safe.

Process	Allocation	Max.	Available (Work)	Need (max-allocation)
	A B C	A B C	A B C	A B C
P_0	0 1 0	7 5 3	~~3 3 2~~	7 4 3
P_1	2 0 0	3 2 2	~~5 3 2~~	1 2 2
P_2	3 0 2	9 0 2	~~7 4 3~~	6 0 0
P_3	2 1 1	2 2 2	~~7 4 5~~	0 1 1
P_4	0 0 2	4 3 3	~~7 5 5~~ 10 5 7	4 3 1

$\langle P_1, P_3, P_4, P_0, P_2 \rangle$

Banker's algorithm:

$Need_i \le work \Rightarrow work = work + allocation$

P_0 $743 \le 332$ ✗

P_1 $122 \le 332$ ✓ $w = w + alloc = 332 + 200 = 532$

P_2 $600 \le 532$ ✗

P_3 $011 \le 532$ ✓ $w = w + alloc = 532 + 211 = 743$

P_4 $431 \le 532$ ✓ $w = w + alloc = 743 + 002 = 745$

P_0 $743 \le 745$ ✓ $w = w + alloc = 745 + 010 = 755$

P_2 $600 \le 755$ ✓ $w = w + alloc = 755 + 302 = 10\ 5\ 7$

Resource request algorithm.

1. $Request_i \leq need_i$ goto (2)
2. $Request_i \leq available$ goto (3)
3. $available = available - request_i$

   $allocation = allocation + request_i$

   $need_i = need_i - request_i$
4. Check if new state is safe or not (using Banker's algorithm)

	Allocation	Max	(Work) Available	Need
P_0	010	753	332	743
P_1	200	322		020
P_2	302	902		600
P_3	211	222		011
P_4	002	433		431

If P_1 requests (1,0,2), determine if it can be granted immediately.

$$P_1 \rightarrow R(1\,0\,2)$$

$$\begin{aligned} Need(P_1) &= Max - Alloc. \\ &= 322 - 200 \\ &= 122 \end{aligned}$$

$$\begin{aligned} Need(P_1) &= Max - Alloc. \\ &= 322 - 200 \\ &= 122 \end{aligned}$$

1. Request ≤ need

   $102 \leq 122$ ✓

2. Request ≤ avail

   $102 \leq 332$ ✓

3. avail = avail − request

   $= 332 - 102 = 230$

   alloc = alloc + request

   $= 200 + 102 = 302$

   need = need − request

   $= 122 - 102 = 020.$

(New table)

	Allocation	Max	Available	Need
P_0	010	753	230	743
P_1	302	322		020
P_2	302	902		600
P_3	211	222		011
P_4	002	433		431

Need ≤ Available