CSE 323/L-24/17.05.2024/

✳ Priority Scheduling

⇒ A priority number is associated with each process.

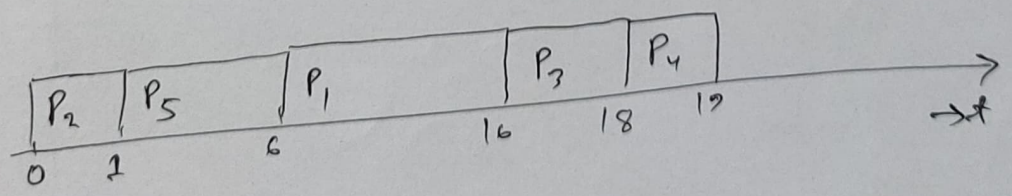
Lower the number ⇒ Higher the priority

✳ SJF is a priority scheduling, where priority is the inverse of predicted next CPU burst time.

Examples: Non-Preemptive

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2
	19	

Gantt Chart:



Waiting time:

$P_1 = 6$ $P_4 = 18$

$P_2 = 0$ $P_5 = 1$

$P_3 = 16$

Average $= \frac{6+0+16+18+1}{5} = 8.2$ ms

* if two ~~pai~~ process has the same priority then arrival time ~~with~~ or table sequence will consider.

* Problem:

  - Starvation
    - low priority processes may never execute.

      Because of newly added process have higher priority

* Solution:

  - Aging
  - priority will increase over time. (Certain amount of time)

* Round Robin (RR)

- Process, which arrive first will execute first.
- But there is a fixed amount of time for process to run at once. Time Quantum q.
- if a process run time exceeded the defined time quantum, then the process will be preemted and back to end of the ready queue.
- So, Round Robin works for the preemtive process.

* q must be large with respect to context switch, otherwise, overhead is too high.

- higher average turnaround time than SJF, but better response.

⇒ context switch time < 10 μsec

⇒ q usually 10ms to 100 ms

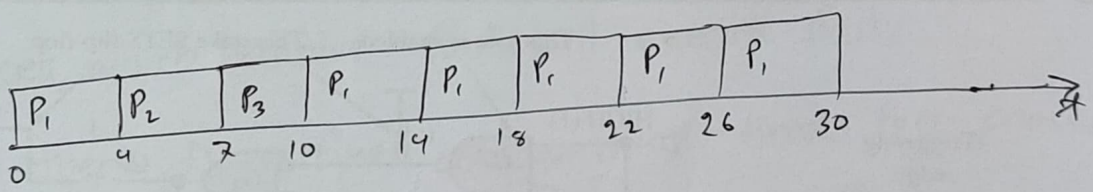
Example:

Process	Burst Time
P_1	24 - 20 - 16 - 12 - 8 - 4 - 0
P_2	3 - 0
P_3	3 - 0

Time Quantum = 4

Remaining

Gantt chart:



Waiting time:

$P_1 = 10 - 4 = 6$

$P_2 = 4$

$P_3 = 7$

Average = $\frac{6+4+7}{3} = 5.66$ ms

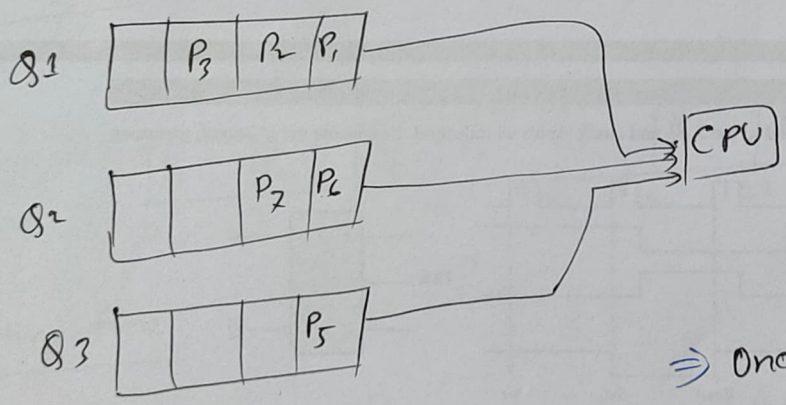
Time Quantum ↑ ↓ Content switch
↓ ↑

* if we want more multi-programing, then we need to take small time quantum. Not less than content switch time.

Multilevel Queue

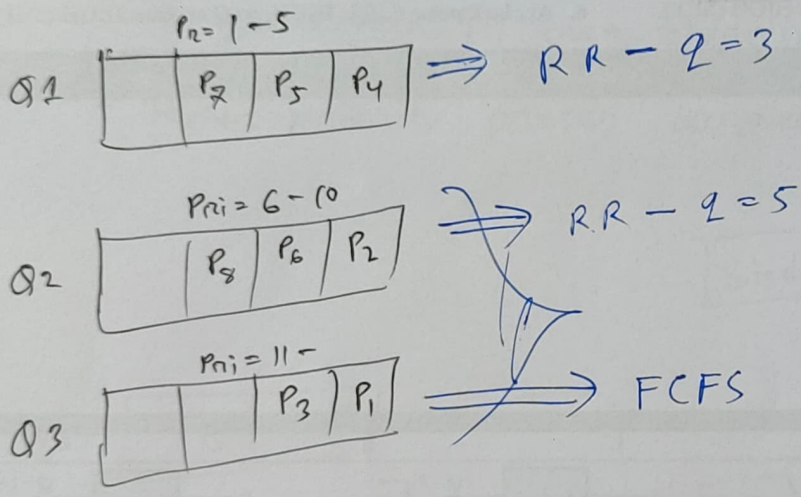
⇒ There are multiple Ready Queue.
- Its queue has its own scheduling algorithm.
- Higher priority process assigned in upper queue.

Slide - 6.17

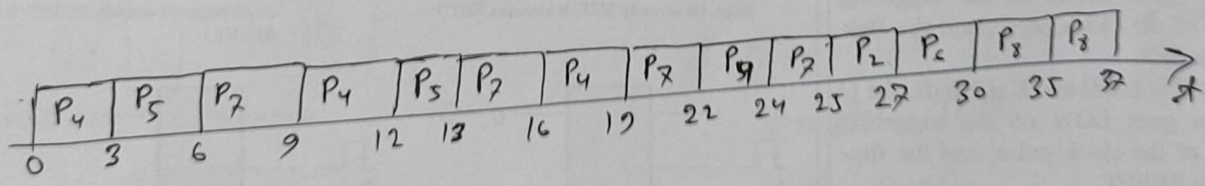


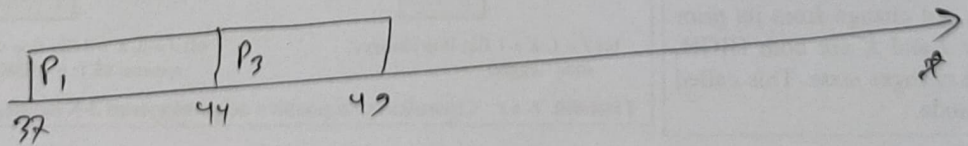
⇒ Once first queue became empty, then second queue will run.

Process	Burst Time	Priority
P_1	7	12
P_2	2	9
P_3	5	15
P_4	11 8 5 2	4
P_5	4 1	1
P_6	3	7
P_7	10 7 4 1	2
P_8	7 2	6
	49	



Gantt chart:





Waiting time:

$P_1 = 37$

$P_2 = 25$

$P_3 = 44$

$P_4 = 22 - 3 - 3 - 3 = 13$

$P_5 = 12 - 3 = 9$

$P_6 = 27$

$P_7 = 24 - 3 - 3 - 3 = 15$

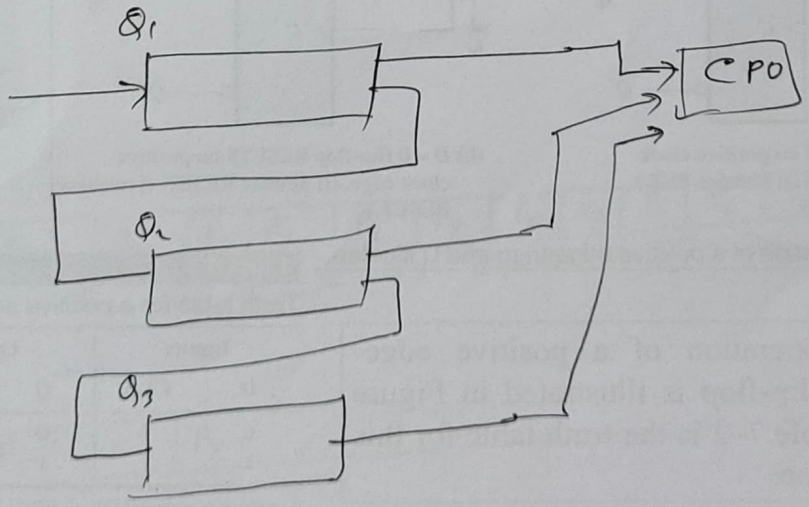
$P_8 = 30$

Average: $\frac{37 + 25 + 44 + 13 + 9 + 27 + 15 + 30}{8}$

$= \frac{200}{8} = 25$

Multilevel Feedback Queue

⇒ Almost same as multilevel queue, but here, process ~~flow~~ flow like a pipeline architecture.



Slide-6.19

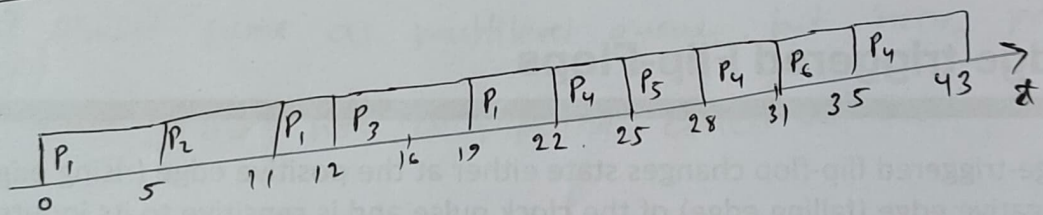
⇒ Process first enter in the first queue. if process can not run completely due to Time Quantum, then it will move to 2nd Queue. And the last Queue is FCFS so the process must be end there.

Complex Example, important for exam.

Process	Priority	Arrival time	Burst Time
P_1	8	0	9
P_2	2	5	6
P_3	5	12	7
P_4	15	16	14
P_5	6	25	3
P_6	10	31	4
			43

Priority scheduling

Gantt Chart:



Waiting time:

$P_1 = 19 - 6 - 0 = 13$

$P_2 = 5 - 5 = 0$

$P_3 = 12 - 12 = 0$

$P_4 = 35 - 3 - 3 = 29 - 16 = 13$

$P_5 = 25 - 25 = 0$

$P_6 = 31 - 31 = 0$

Average

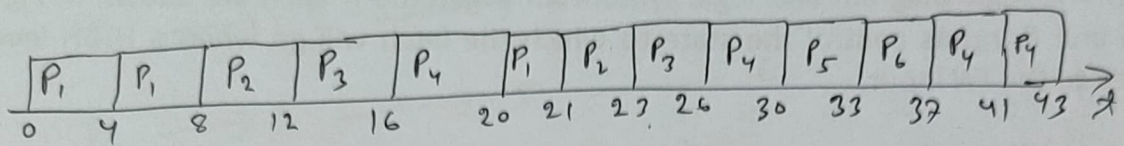
$$= \frac{13 + 5 + 12 + 29 + 25 + 31}{6}$$

$$= \frac{13 + 0 + 0 + 13 + 0 + 0}{6}$$

$$= \frac{26}{6} = 4.3$$

RR : Q = 4

Gantt chart:



Waiting time:

$P_1 = 20 - 8 - 0 = 12$

$P_2 = 21 - 4 - 5 = 12$

$P_3 = 23 - 4 - 12 = 7$

$P_4 = 37 - 8 - 16 = 13$

$P_5 = 30 - 25 = 5$

$P_6 = 33 - 31 = 2$

$$\text{Average} = \frac{12+12+7+13+5+2}{6}$$

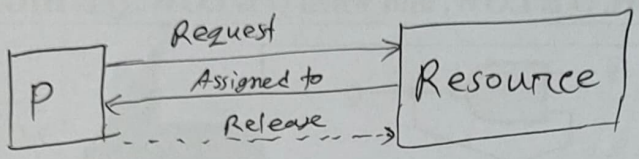
$$= 8.5$$

Quiz-2
Upto this
25.05.2024

L-25/18.05.2024/

Chapter-7

Deadlocks



✳ Resource utilization by a process in three steps:

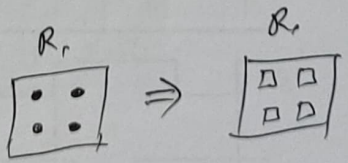
- request
- use
- release

✳ Resource-Allocation Graph

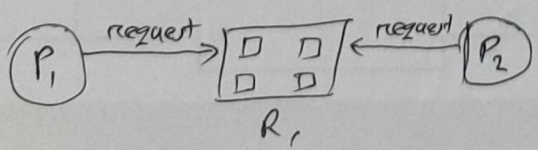
For process ⇒ use circle ⇒ (P) ⇒ (P1) (P2) (P3)

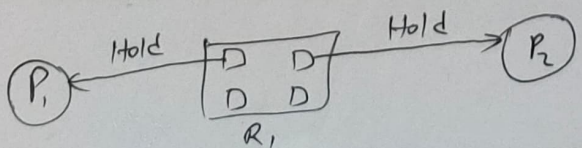
For Resource ⇒ use square ⇒ [R] ⇒ [R1] [R2]

⇒ Resource may have multiple instances:



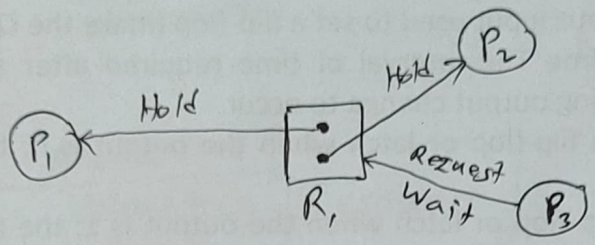
⇒ that means, group of resources of same type.



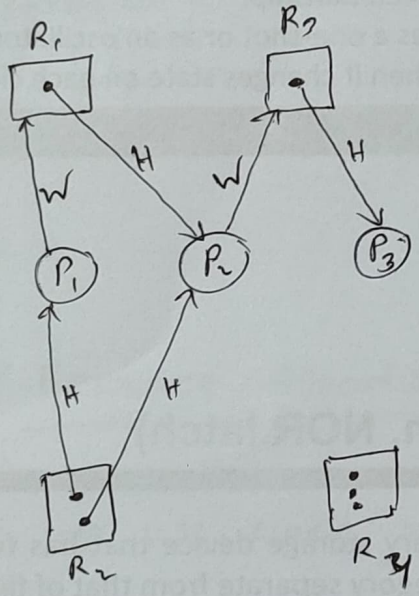


* Request edge ⇒ directed as $P_i \rightarrow R_j$

* Assignment / hold edge ⇒ directed as $R_j \rightarrow P_i$



Example:



* if process can't run for resource allocation, then it will be deadlock.

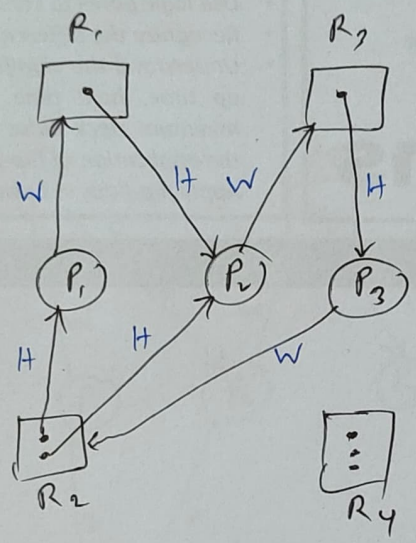
* For avoiding deadlock, we need to make sure that each process can get all resource as it need.

* In the given example, if we run these process in this sequence $P_3 \Rightarrow P_2 \Rightarrow P_1$ then, there will be no deadlock.

✱ Question Pattern:

⇒ Given details in text form, draw the resource allocation graph and explain is it in deadlock or not?

Example:



✱ If all process associated with ~~he~~ Hold and wait at the same time, then it is in deadlock.

✱ Deadlock Characterization:

If the given 4 conditions holds simultaneously, then it is in deadlock:

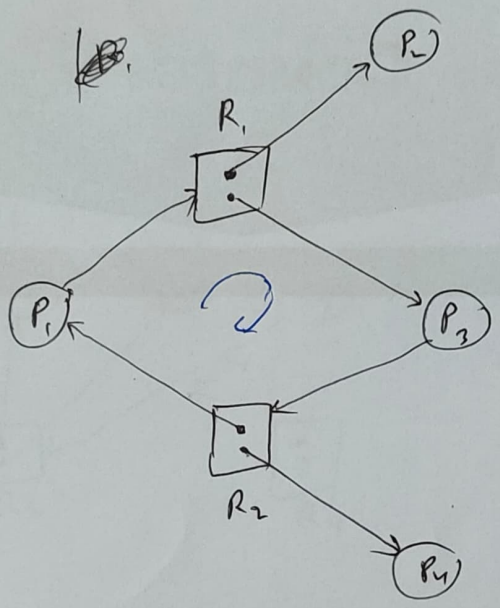
i. Mutual Exclusion ⇒ one resource instance = one proces

ii. Hold & Wait ⇒ each resource associated with hold and wait at the same time.

iii. No Preemption ⇒ before completing the task, process can't be terminated.

iv. Circular wait ⇒

✱ Example:



circular wait but not a deadlock. So, 4 conditions need to be hold at the same time.

✱ Facts of Deadlock

⇒ if graph contains no cycles ⇒ no deadlock

⇒ if graph contains cycles

- if only one instance in every resource
  ⇒ then deadlock

- if ~~not~~ more than one instance
  ⇒ possibility of deadlock

Deadlock Prevention:

⇒ We can make sure that at least any one condition is false.

i. Mutual Exclusion:

- not required for shareable resource
  - read only file
- must hold for non-shareable resource.

ii. Hold & Wait:

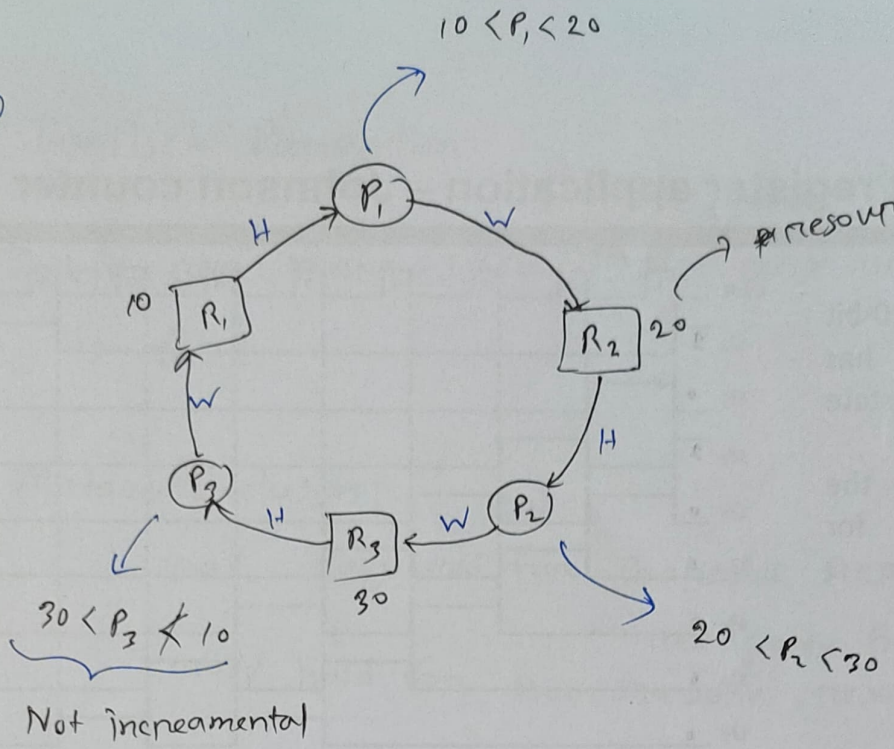
- when a process request for a resource, we need to make sure that, it doesn't hold any other resources.

iii. No preemption:

- if process request new resources while holding others, then the process need to ~~terminate or~~ be interrupted and release other resources.
- will restart when all the required resources are free.

iv. Circular Wait:

- Increamental request.
- each process request resources in an increasing order of enumeration.



L-26 / 20.05.2024

Deadlock Avoidance

- Predict the deadlock possibility in near future and take action before that.
  - need some priority information available.
    - how much resource and which type it may need.
  - if there is no possibility of deadlock,
    ⇒ safe state (#required resource < # available resource)
  - if there is a possibility of deadlock.
    ⇒ unsafe state

- Therefore, when a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state.

* Safe State ⇒ No deadlocks

unsafe ~~Set~~ State ⇒ Possibility of deadlocks Slide-7.16

⇒ we need to make sure that, system never enter an unsafe state.

* Example:

resource = 12
A

Process	Max Needs (at a time)	Current Needs (Hold)
P_0	10	5
P_1	4	2
P_2	9	2

* if all resource can run in worst case, then it is safe state. Otherwise unsafe state.

⇒ Total Resource = 12

Currently Using = 9

Available = $3 - 2(P_1) = 1 + 4(P_1 \text{ end}) = 5 - 5(P_0) = 0 + 10(P_0 \text{ end}) = 10 - 7(P_2) = 3$
$+ 9(P_2 \text{ end}) = 12$

* in current situation, we can't run P_0 and P_2 in worst case. Because available is only 3. But we can run P_1 in worst case. Then P_0 will run and end the process to release all other resources.

$P_1 \Rightarrow P_0 \Rightarrow P_2$ ⤵ Safe State

Process	Max need	currently need
P_0	10	5
P_1	4	2+2 ⇒ 0
P_2	9	2+1=3

→ one extra request. if it is allowed system will enter in unsafe state.

Total Resource = 12

Currently Using = 5+2+2+1 = 10

Available = 12−10 = 2 − 2 (P_1) = 0 + 4 (P_1 end) = 4

we can run P_1 at first.

P_1 ⇒ others process can't run in worst case

⇒ unsafe state (Possibility of deadlock)

Chapter-8
Main Memory

- Memory Unit only sees a stream of:
  - address + read request
  - address + data + write request
- Register access time 1 CPU Clock
  - Main memory access time = many cycles ⇒ stall
  - Cache sits between main memory and CPU registers

* two register hold the address of a process.

base & limit

process address must be ~~base ≤ process address < base + limit~~

base ≤ process address < base + limit

Slide - 8.4

* Address Binding:

⇒ Convert the logical address to physical address.

⇒ array[] = int[10]

array[3] = 10

↳ logical address used in program language.

Physical address generated by OS.

* Address binding happen at three different stages

⇒ Compile Time:

- logical address converted into physical address during the compilation time.
- For any reason, if we need to change the logical address, then we need to compile the program again.

⇒ Load Time:

- if process reside location in memory is unknown then, compiler need to generate relocatable code. Final address generated during load time.
- if we need to change the address, then we need to reload the program.

⇒ Execution Time:

- if process can move one segment to another during execution, then final address need to generate during execution time.
  - Need hardware support
    - base + limit registers access
  - most popular

(*) Logical Address / Virtual Address ⇒ generated by CPU
Physical Address ⇒ seen by the memory unit.

(*) During
- Compile time
- Load time

} Logical Address = Physical Address

(*) Logical Address space ⇒ set of all logical addresses generated by a program

* Physical Address Space ⇒ set of all physical address generated by a program. (mmu)

* MCQ:

- Base Register ⇒ relocation register
- MS-DOS on Intell 80x86 used 4 relocation register

Slide-8.10 | 8.11

* Swapping:

⇒ When main memory gets full, we need to move old process to run new process. For that OS use the secendary memory as a backing store. Its moved for a temporary time.

- its increase the multi-programing.

Slide-8.12

Final Syllabus End 05.06.2024

* For swap in and swap out, it takes the time twiteh.

⇒ Therefore, context switch in such a swapping system is fairly high.