UNIT 2 - CPU Scheduling

1. Explain the FCFS, preemptive and non preemptive versions of Shortest-Job-First and Round Robin (time slice = 2) scheduling algorithms with Gantt Charts for the four processes given. Compare their average turn around and waiting time. **(12)** (May 2015)

Process	Arrival time	Burst time
P1	0	10
P2	1	6
P3	2	12
P4	3	15

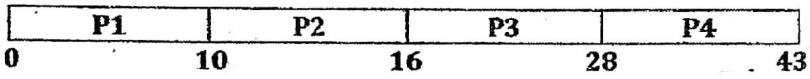
****

1) Gantt Chart

A rectangle marked off horizontally in time units, marked off at end of each job or job segment. It shows the distribution of time-bursts in time. It is used to determine total and average statistics on processes, by formulating various scheduling algorithms on it.

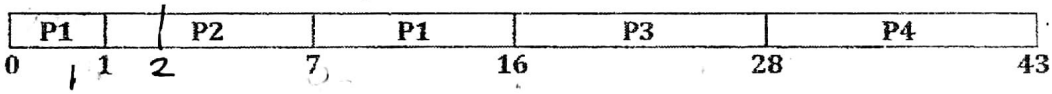
a) First Come First Serve (FCFS)

- The process that requests the CPU first is allocated the CPU first.



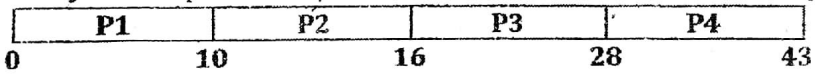
b) Preemptive Shortest Job First (or) Shortest Remaining Time First(SRTF)

- Preemptive SJF algorithm will preempt the currently running process if the burst time of newly arrived process(**NAP**) is smaller than the remaining burst time of currently running process(**CRP**).
- **Stop CRP if BT of NAP < Remaining BT of CRP**



c) Non Preemptive Shortest Job First

- Non Preemptive SJF algorithm will allow the currently running process even though the burst time of the newly arrived process is smaller than the burst time of the currently running process.



d) Round Robin(time slice = 2 ms)

- It is similar to FCFS scheduling, but preemption is added to switch between processes.
- A small unit of time called a time slice or time quantum is defined.

P1(10)	2	2	2	2	2			
P2(6)	2	2	2					
P3(12)	2	2	2	2	2	2		
P4(15)	2	2	2	2	2	2	2	1

P1	P2	P3	P4	P1	P2	P3	P4	P1	P2	P3	P4	P1	P3	P4	P1	P3	P4	P3	P4	P4	P4
0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40	42	43

2) Turnaround time (TAT)

Turnaround time = **Completed time – Arrival time.**

Process	FCFS	Preemptive SJF	Non Preemptive SJF	Round Robin
P1	10 – 0 = 10	16 – 0 = 16	10 – 0 = 10	32 – 0 = 32
P2	16 – 1 = 15	7 – 1 = 6	16 – 1 = 15	20 – 1 = 19
P3	28 – 2 = 26	28 – 2 = 26	28 – 2 = 26	38 – 2 = 36
P4	43 – 3 = 40	43 – 3 = 40	43 – 3 = 40	43 – 3 = 40
Total TAT	91 ms	88 ms	91 ms	127 ms
Average TAT	91/4 = 22.75 ms	88/4 = 22 ms	91/4 = 22.75 ms	127/4 = 31.75 ms

3) **Waiting time (WT)**

Waiting time = **Turnaround time – Burst time**

Process	FCFS	Preemptive SJF	Non Preemptive SJF	Round Robin
P1	10 – 10 = 0	16 – 10 = 6	10 – 10 = 0	32 – 10 = 22
P2	15 – 6 = 9	6 – 6 = 0	15 – 6 = 9	19 – 6 = 13
P3	26 – 12 = 14	26 – 12 = 14	26 – 12 = 14	36 – 12 = 24
P4	40 – 15 = 25	40 – 15 = 25	40 – 15 = 25	40 – 15 = 25
Total WT	**48 ms**	**45 ms**	**48 ms**	**84 ms**
Average WT	**48/4 = 12 ms**	**45/4 = 11.25 ms**	**48/4 = 12 ms**	**84/4 = 21 ms**

4) **Result**

Algorithm	Average Turnaround Time(ms)	Average Waiting Time(ms)
FCFS	**22.75**	**12**
Preemptive SJF	**22**	**11.25**
Non Preemptive SJF	**22.75**	**12**
Round Robin	**31.75**	**21**

It can be noted that **Preemptive Shortest Job First** gives minimum average waiting time(**11.25** ms) and is optimal

****

2. Consider the following set of processes, with the length of the CPU burst time given in milliseconds

Process	Burst Time	Priority
P1	10	3
P2	1	1
P3	2	3
P4	1	4
P5	5	2

The processes are arrived in the order P1, P2, P3, P4, P5 all at time 0. **(May 2012, Nov 2015)**

a) What is a Gantt chart? Explain how it is used. **(4)**

b) Draw four Gantt charts illustrating the execution of these processes using FCFS, SJF, Non preemptive priority(a smaller priority number implies a higher priority), and RR(quantum = 1) scheduling. **(3)**

c) What is the turnaround time of each process for each of the scheduling algorithms in part b ? **(3)**

d) What is the waiting time of each process for each of the scheduling algorithms in part b ? **(3)**

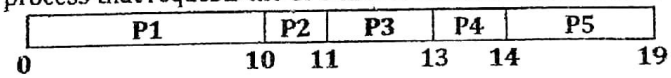
e) Which of the schedules in part b results in the minimal average waiting time(over all processes) ? **(3)**

1) **Gantt Chart**

A rectangle marked off horizontally in time units, marked off at end of each job or job segment. It shows the distribution of time-bursts in time. It is used to determine total and average statistics on processes, by formulating various scheduling algorithms on it.

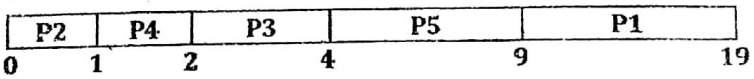
a) **First Come First Serve (FCFS)**

- The process that requests the CPU first is allocated the CPU first.



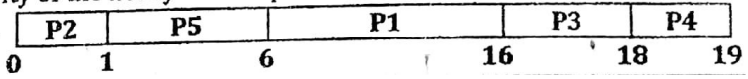
b) **Shortest Job First**

- Allocates the CPU to the process that has the smallest burst time.
- If two processes have the same next CPU burst, then FCFS algorithm is used.



c) **Non Preemptive Priority**

- A priority is associated with each process and the CPU is allocated to process with highest priority.
- Non Preemptive priority will allow the currently running process to finish its work even though the priority of the newly arrived process is higher than the priority of the currently running process



d) Round Robin(time quantum = 1 ms)

- It is similar to FCFS scheduling, but preemption is added to switch between processes.
- A small unit of time called a time slice or time quantum is defined.

P1(10)	1	1	1	1	1	1	1	1	1	1
P2(1)	1									
P3(2)	1	1								
P4(1)	1									
P5(5)	1	1	1	1	1					

P1	P2	P3	P4	P5	P1	P3	P5	P1	P5	P1	P5	P1	P5	P1	P1	P1	P1	P1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

2) Turnaround time (TAT)

Turnaround time = **Completed time – Arrival time**

Process	FCFS	SJF	Non Preemptive Priority	Round Robin
P1	10 – 0 = 10	19 – 0 = 19	16 – 0 = 16	19 – 0 = 19
P2	11 – 0 = 11	1 – 0 = 1	1 – 0 = 1	2 – 0 = 2
P3	13 – 0 = 13	4 – 0 = 4	18 – 0 = 18	7 – 0 = 7
P4	14 – 0 = 14	2 – 0 = 2	19 – 0 = 19	4 – 0 = 4
P5	19 – 0 = 19	9 – 0 = 9	6 – 0 = 6	14 – 0 = 14
Total TAT	**67 ms**	**35 ms**	**60 ms**	**46 ms**
Average TAT	67/5 = **13.4 ms**	35/5 = **7 ms**	60/5 = **12 ms**	46/5 = **9.2 ms**

3) Waiting time (WT)

Waiting time = **Turnaround time – Burst time**

Process	FCFS	SJF	Non Preemptive Priority	Round Robin
P1	10 – 10 = 0	19 – 10 = 9	16 – 10 = 6	19 – 10 = 9
P2	11 – 1 = 10	1 – 1 = 0	1 – 1 = 0	2 – 1 = 1
P3	13 – 2 = 11	4 – 2 = 2	18 – 2 = 16	7 – 2 = 5
P4	14 – 1 = 13	2 – 1 = 1	19 – 1 = 18	4 – 1 = 3
P5	19 – 5 = 14	9 – 5 = 4	6 – 5 = 1	14 – 5 = 9
Total WT	**48 ms**	**16 ms**	**41 ms**	**27 ms**
Average WT	48/5 = **9.6 ms**	16/5 = **3.2 ms**	41/5 = **8.2**	27/5 = **5.4 ms**

4) Result

Algorithm	Average Turnaround Time(ms)	Average Waiting Time(ms)
FCFS	**13.4**	**9.6**
SJF	**7**	**3.2**
Non Preemptive Priority	**12**	**8.2**
Round Robin	**9.2**	**5.4**

It can be noted that **Shortest Job First(SJF)** gives the minimum average waiting time(**3.2** ms) and is optimal.

3. Explain the FCFS, preemptive and non preemptive versions of Shortest-Job-First and Round Robin(time slice = 2) scheduling algorithms with Gantt Charts for the four processes given. Compare their average turn around and waiting time. **(12)** (Nov 2012)

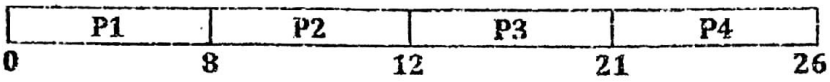
Process	Arrival time	Burst time
P1	0	8
P2	1	4
P3	2	9
P4	3	5

1) Gantt Chart

A rectangle marked off horizontally in time units, marked off at end of each job or job segment. It shows the distribution of time-bursts in time. It is used to determine total and average statistics on processes, by formulating various scheduling algorithms on it.

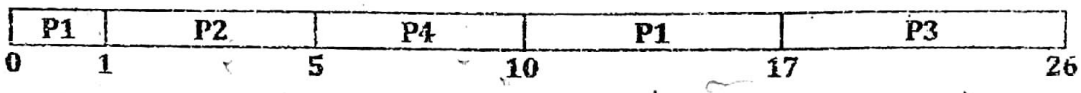
a) First Come First Serve (FCFS)

- The process that requests the CPU first is allocated the CPU first.



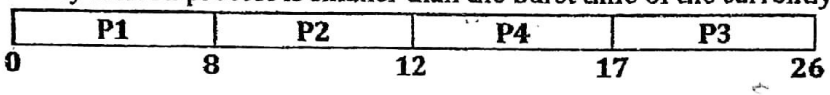
b) Preemptive Shortest Job First (or) Shortest Remaining Time First(SRTF)

- Preemptive SJF algorithm will preempt the currently running process if the burst time of newly arrived process(**NAP**) is smaller than the remaining burst time of currently running process(**CRP**).
- **Stop CRP if BT of NAP < Remaining BT of CRP**



c) Non Preemptive Shortest Job First

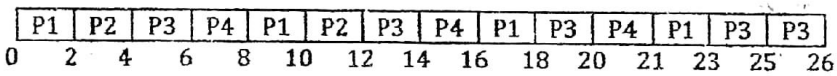
- Non Preemptive SJF algorithm will allow the currently running process even though the burst time of the newly arrived process is smaller than the burst time of the currently running process.



d) Round Robin(time slice = 2 ms)

- It is similar to FCFS scheduling, but preemption is added to switch between processes.
- A small unit of time called a time slice or time quantum is defined.

P1(8)	2	2	2	2	
P2(4)	2	2			
P3(9)	2	2	2	2	1
P4(5)	2	2	1		



2) Turnaround time (TAT)

Turnaround time = **Completed time – Arrival time**

Process	FCFS	Preemptive SJF	Non Preemptive SJF	Round Robin
P1	8 – 0 = 8	17 – 0 = 17	8 – 0 = 8	23 – 0 = 23
P2	12 – 1 = 11	5 – 1 = 4	12 – 1 = 11	12 – 1 = 11
P3	21 – 2 = 19	26 – 2 = 24	26 – 2 = 24	26 – 2 = 24
P4	26 – 3 = 23	10 – 3 = 7	17 – 3 = 14	21 – 3 = 18
Total TAT	**61ms**	**52 ms**	**57 ms**	**76 ms**
Average TAT	61/4 = **15.25 ms**	52/4 = **13 ms**	57/4 = **14.25 ms**	76/4 = 19 ms

3) Waiting time (WT)

Waiting time = **Turnaround time – Burst time**

Process	FCFS	Preemptive SJF	Non Preemptive SJF	Round Robin
P1	8 – 8 = 0	17 – 8 = 9	8 – 8 = 0	23 – 8 = 15
P2	11 – 4 = 7	4 – 4 = 0	11 – 4 = 7	11 – 4 = 7
P3	19 – 9 = 10	24 – 9 = 15	24 – 9 =15	24 – 9 = 15
P4	23 – 5 = 18	7 – 5 = 2	14 – 5 = 9	18 – 5 = 13
Total WT	**35 ms**	**26 ms**	**31 ms**	**50 ms**
Average WT	35/4 = **8.75 ms**	26/4 = **6.5 ms**	31/4 = **7.75 ms**	50/4 = **12.5 ms**

4) **Result**

Algorithm	Average Turnaround Time(ms)	Average Waiting Time(ms)
FCFS	15.25	8.75
Preemptive SJF	13	6.5
Non Preemptive SJF	14.25	7.75
Round Robin	19	12.5

It can be noted that **Preemptive Shortest Job First** gives minimum average waiting time(6.5 ms) and is optimal

****

4. Consider the following five processes, with the length of the CPU burst time given in milliseconds.

Process	Burst time
P1	10
P2	29
P3	3
P4	7
P5	12

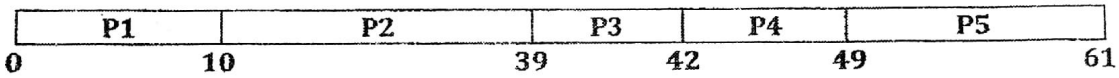
Consider the First Come First Serve(FCFS), Non Preemptive Shortest Job First(SJF), and Round Robin(RR) (quantum = 10 milliseconds) scheduling algorithms. Illustrate the scheduling using Gantt chart. Which algorithm will give the minimum average waiting time? Discuss. **(16)** **(Nov 2007, May 2006)**

****

1) **Gantt Chart**

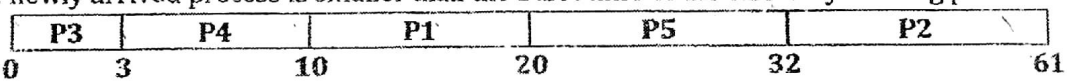
a) **First Come First Serve (FCFS)**

- The process that requests the CPU first is allocated the CPU first.



b) **Non Preemptive Shortest Job First(SJF)**

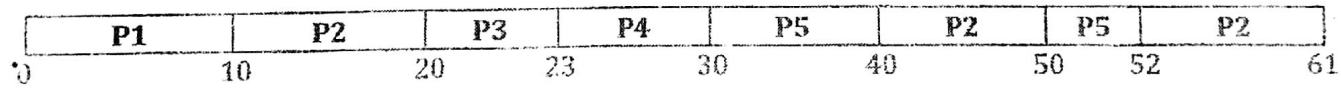
- Non Preemptive SJF algorithm will allow the currently running process even though the burst time of the newly arrived process is smaller than the burst time of the currently running process.



c) **Round Robin(quantum = 10 ms)**

- It is similar to FCFS scheduling, but preemption is added to switch between processes.
- A small unit of time called a time slice or time quantum is defined.

P1(10)	10		
P2(29)	10	10	9
P3(3)	3		
P4(7)	7		
P5(12)	10	2	



2) **Turnaround time (TAT)**

Turnaround time = **Completed time – Arrival time**

Process	FCFS	Non Preemptive SJF	Round Robin
P1	10 – 0 = 10	20 – 0= 20	10 – 0 = 10
P2	39 – 0 = 39	61 – 0= 61	61 – 0 = 61
P3	42 – 0 = 42	3 – 0 = 3	23 – 0 = 23
P4	49 – 0 = 49	10 – 0 = 10	30 – 0 = 30
P5	61 – 0 = 61	32 – 0 = 32	52 – 0 = 52
Total TAT	201 ms	126 ms	176 ms
Average TAT	201/5 = 40.2 ms	126/5 = 25.2 ms	176/5 = 35.2 ms

3) **Waiting time (WT)**

Waiting time = **Turnaround time – Burst time**

Process	FCFS	Non Preemptive SJF	Round Robin
P1	10 – 10 = 0	20 – 10 = 10	10 – 10 = 0
P2	39 – 29 = 10	61 – 29 = 32	61 – 29 = 32
P3	42 – 3 = 39	3 – 3 =0	23 – 3 = 20
P4	49 – 7 = 42	10 – 7 = 3	30 – 7 = 23
P5	61 – 12 = 49	32 – 12 =20	52 – 12 = 40
Total WT	**140 ms**	**65 ms**	**115 ms**
Average WT	140/5 =**28 ms**	65/5 = **13 ms**	115/5 = **23 ms**

4) **Result**

Algorithm	Average Turnaround Time(ms)	Average Waiting Time(ms)
FCFS	40.2	28
Non Preemptive SJF	25.2	13
Round Robin	35.2	23

It can be noted that Non Preemptive Shortest Job First gives minimum average waiting time(13 ms) and is optimal

****

5. Assume the following workload in a system. All jobs arrive at time 0 in the order given.

Job	Burst time(ms)	Priority
A	8	2
B	4	1
C	5	4
D	2	2
E	1	3

i) Draw a Gantt chart illustrating the execution of these jobs using FCFS, RR (quantum = 4) and non preemptive priority(a smaller priority number implies a higher priority) and SJF CPU scheduling **(8)**

ii) Calculate the average waiting time and average turnaround time for each of the above scheduling algorithm. **(8)** **(Nov 2006)**

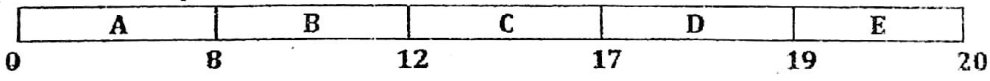
****

1) **Gantt Chart**

A rectangle marked off horizontally in time units, marked off at end of each job or job segment. It shows the distribution of time-bursts in time. It is used to determine total and average statistics on processes, by formulating various scheduling algorithms on it.

a) **First Come First Serve (FCFS)**

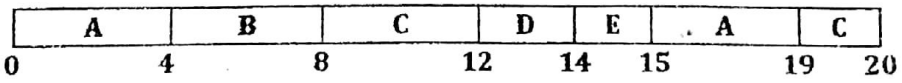
- The process that requests the CPU first is allocated the CPU first.



b) **Round Robin(quantum = 4 ms)**

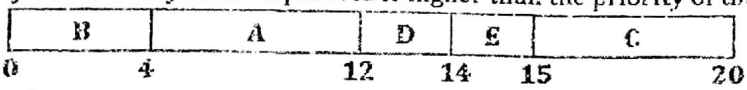
- It is similar to FCFS scheduling, but preemption is added to switch between processes.
- A small unit of time called a time slice or time quantum is defined.

A(8)	4	4
B(4)	4	
C(5)	4	1
D(2)	2	
E(1)	1	



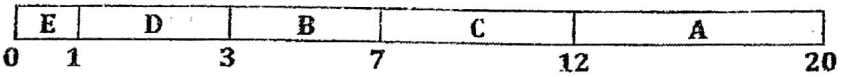
c) **Non Preemptive Priority**

- A priority is associated with each process and the CPU is allocated to process with highest priority.
- Non Preemptive priority will allow the currently running process to finish its work even though the priority of the newly arrived process is higher than the priority of the currently running process



d) **Shortest Job First**

- Allocates the CPU to the process that has the smallest burst time.
- If two processes have the same next CPU burst, then FCFS algorithm is used.



2) **Turnaround time (TAT)**

Turnaround time = **Completed time – Arrival time**

Process	FCFS	Round Robin	Non Preemptive Priority	SJF
A	8 – 0 = 8	19 – 0 = 19	12 – 0 = 12	20 – 0 = 20
B	12 – 0 = 12	8 – 0 = 8	4 – 0 = 4	7 – 0 = 7
C	17 – 0 = 17	20 – 0 = 20	20 – 0 = 20	12 – 0 = 12
D	19 – 0 = 19	14 – 0 = 14	14 – 0 = 14	3 – 0 = 3
E	20 – 0 = 20	15 – 0 = 15	15 – 0 = 15	1 – 0 = 1
Total TAT	**76 ms**	**76 ms**	**65 ms**	**53 ms**
Average TAT	76/5 = **15.2 ms**	76/5 = **15.2 ms**	65/5 = **13 ms**	53/5 = **10.6 ms**

3) **Waiting time (WT)**

Waiting time = **Turnaround time – Burst time**

Process	FCFS	Round Robin	Non Preemptive Priority	SJF
P1	8 – 8 = 0	19 – 8 = 11	12 – 8 = 4	20 – 8 = 12
P2	12 – 4 = 8	8 – 4 = 4	4 – 4 = 0	7 – 4 = 3
P3	17 – 5 = 12	20 – 5 = 15	20 – 5 = 15	12 – 5 = 7
P4	19 – 2 = 17	14 – 2 = 12	14 – 2 = 12	3 – 2 = 1
P5	20 – 1 = 19	15 – 1 = 14	15 – 1 = 14	1 – 1 = 0
Total WT	**56 ms**	**56 ms**	**45 ms**	**23 ms**
Average WT	56/5 = **11.2 ms**	56/5 = **11.2 ms**	45/5 = **9** ms	23/5 = **4.6 ms**

4) **Result**

Algorithm	Average Turnaround Time(ms)	Average Waiting Time(ms)
FCFS	**15.2**	**11.2**
Round Robin	**15.2**	**11.2**
Non Preemptive Priority	**13**	**9**
SJF	**10.6**	**4.6**

It can be noted that **Shortest Job First(SJF)** gives the minimum average waiting time(**4.6** ms) and is optimal.

****

6. Assume the following processes arrive for execution at the time indicated and also mention with the length of the CPU burst time given in milliseconds.

Job	Burst time(ms)	Priority	Arrival time(ms)
P1	6	2	0
P2	2	2	1
P3	3	4	1
P4	1	1	2
P5	2	3	2

i) Give a Gantt chart illustrating the execution of these processes using FCFS , Round Robin(quantum = 1) and Priority(Preemptive and Non Preemptive). **(4)**

ii) Calculate the average waiting time and average turnaround time for each of the above scheduling algorithm. **(12).**

(Nov 2008)

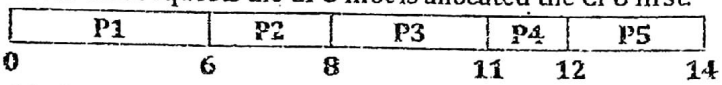
****

1) Gantt Chart

A rectangle marked off horizontally in time units, marked off at end of each job or job segment. It shows the distribution of time-bursts in time. It is used to determine total and average statistics on processes, by formulating various scheduling algorithms on it.

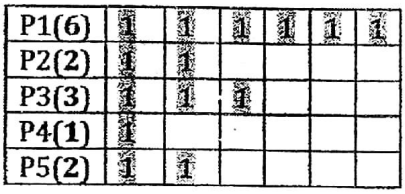
a) First Come First Serve(FCFS)

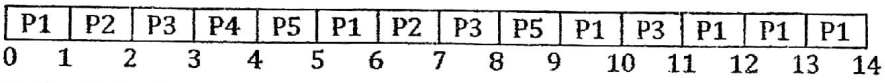
- The process that requests the CPU first is allocated the CPU first.



b) Round Robin(quantum = 1 ms)

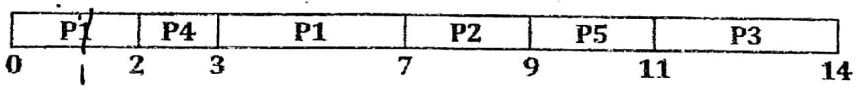
- It is similar to FCFS scheduling, but preemption is added to switch between processes.
- A small unit of time called a time slice or time quantum is defined.





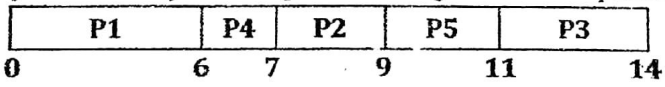
c) Preemptive Priority

- A priority is associated with each process and the CPU is allocated to process with highest priority.
- Preemptive priority will stop the currently running process if the priority of the newly arrived process is higher than the priority of the currently running process
- **Stop CRP** if **priority** of **NAP > priority** of **CRP**



d) Non Preemptive Priority

- A priority is associated with each process and the CPU is allocated to process with highest priority.
- Non Preemptive priority will allow the currently running process to finish its work even though the priority of the newly arrived process is higher than the priority of the currently running process



2) Turnaround time (TAT)

Turnaround time = **Completed time – Arrival time**

Process	FCFS	Round Robin	Preemptive Priority	Non Preemptive Priority
A	6 – 0 = 6	14 – 0 = 14	7 – 0 = 7	6 – 0 = 6
B	8 – 1 = 7	7 – 1 = 6	9 – 1 = 8	9 – 1 = 8
C	11 – 1 = 10	11 – 1 = 10	14 – 1 = 13	14 – 1 = 13
D	12 – 2 = 10	4 – 2 = 2	3 – 2 = 1	7 – 2 = 5
E	14 – 2 = 12	9 – 2 = 7	11 – 2 = 9	11 – 2 = 9
Total TAT	**45 ms**	**39 ms**	**38 ms**	**41 ms**
Average TAT	45/5 = **9 ms**	39/5 = **7.8 ms**	38/5 = **7.6 ms**	41/5 = **8.2 ms**

3) **Waiting time (WT)**

Waiting time = **Turnaround time – Burst time**

Process	FCFS	Round Robin	Preemptive Priority	Non Preemptive Priority
P1	6 – 6 = 0	14 – 6 = 8	7 – 6 = 1	6 – 6 = 0
P2	7 – 2 = 5	6 – 2 = 4	8 – 2 = 6	8 – 2 = 6
P3	10 – 3 = 7	10 – 3 = 7	13 – 3 = 10	13 – 3= 10
P4	10 – 1 = 9	2 – 1 = 1	1 – 1 = 0	5 – 1 = 4
P5	12 – 2 = 10	7 – 2 = 5	9 – 2 = 7	9 – 2 = 7
Total WT	**31 ms**	**25 ms**	**24 ms**	**27 ms**
Average WT	**31/5 = 6.2 ms**	**25/5 = 5 ms**	**24/5 = 4.8 ms**	**27/5 = 5.4 ms**

4) **Result**

Algorithm	Average Turnaround Time(ms)	Average Waiting Time(ms)
FCFS	**9**	**6.2**
Round Robin	**7.8**	**5**
Preemptive Priority	**7.6**	**4.8**
Non Preemptive Priority	**8.2**	**5.4**

It can be noted that **Preemptive Priority** gives the minimum average waiting time(**4.8** ms) and is optimal.

****

7. Suppose that the following processes arrive for execution at the times indicated. Each process will run the listed amount of time.

Process	Arrival time	Burst time
P1	0.0	8
P2	0.4	4
P3	1.0	1

a) What is the average turnaround time for these processes using FCFS and SJF scheduling?

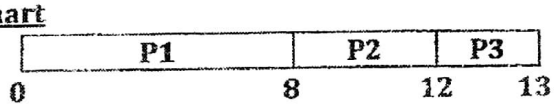
b) Compute what the average turnaround time will be if the CPU is left idle for the first 1 unit and then SJF scheduling is used.

****

1) **First Come First Serve(FCFS)**

The process that requests the CPU first is allocated the CPU first.

a) **Gantt chart**



b) **Turnaround time(TAT)**

Process	Completed - Arrival	TAT
P1	8 - 0	8
P2	12 - 0.4	11.6
P3	13 - 1	12
		TAT = 31.6

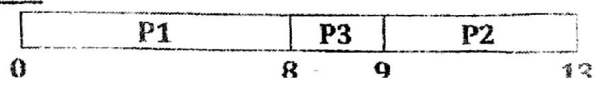
Turn Around Time = 31.6 ms

Average Turn Around Time = 31.6/3 = 10.53 ms

2) **Shortest Job First (SJF) (assuming time from 0)**

Allocates the CPU to the process that has the smallest burst time.

a) **Gantt chart**



b) Turnaround time(TAT)

Process	Completed - Arrival	TAT
P1	8 - 0	8
P2	13 - 0.4	12.6
P3	9 - 1	8
		TAT = 28.6

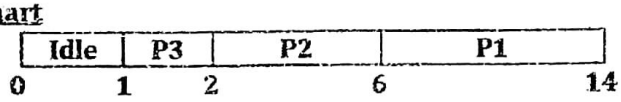
Turn Around Time = 28.6 ms

Average Turn Around Time = 28.6/3 = 9.53 ms

3) Shortest Job First (SJF) (assuming CPU idle during first 1 ms)

Allocates the CPU to the process that has the smallest burst time.

a) Gantt chart



b) Turnaround time(TAT)

Process	Completed - Arrival	TAT
P1	14 - 0	14
P2	6 - 0.4	5.2
P3	2 - 1	1
		TAT = 20.6

Turn Around Time = 20.6 ms

Average Turn Around Time = 20.6/3 = 6.86 ms

****

8. Consider the following set of processes, with the length of the CPU burst time given in milliseconds

Process	Burst Time	Priority
P1	8	3
P2	3	1
P3	4	4
P4	2	2
P5	6	5

The processes are assumed to have arrived in the order P1, P2, P3, P4 and P5 all at time 0.

a) Draw four Gantt charts illustrating the execution of these processes using FCFS, SJF, Non preemptive priority(a smaller priority number implies higher priority), and RR(Quantum = 2) scheduling.

b) What is the waiting time and turnaround time for each of the scheduling algorithms?

c) Which of the schedules in part (a) results in the minimal average waiting time (over all processes)?

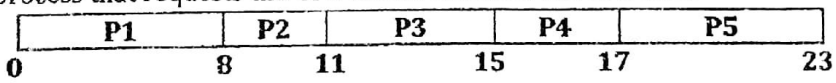
****

1) Gantt Chart

A rectangle marked off horizontally in time units, marked off at end of each job or job segment. It shows the distribution of time-bursts in time. It is used to determine total and average statistics on processes, by formulating various scheduling algorithms on it.

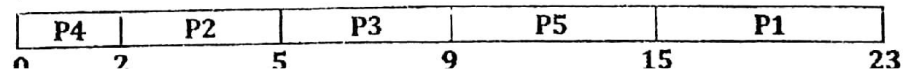
a) First Come First Serve (FCFS)

- The process that requests the CPU first is allocated the CPU first.



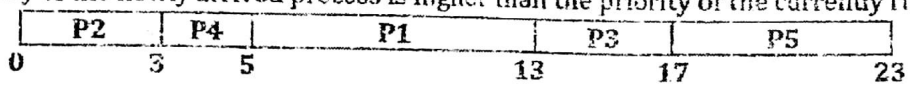
b) Shortest Job First(SJF)

- Allocates the CPU to the process that has the smallest burst time.
- If two processes have the same next CPU burst, then FCFS algorithm is used.



d) **Non Preemptive Priority**

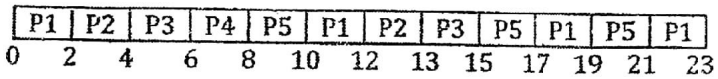
- A priority is associated with each process and the CPU is allocated to process with highest priority.
- Non Preemptive priority will allow the currently running process to finish its work even though the priority of the newly arrived process is higher than the priority of the currently running process



e) **Round Robin(quantum = 2 ms)**

- It is similar to FCFS scheduling, but preemption is added to switch between processes.
- A small unit of time called a time slice or time quantum is defined.

P1(8)	2	2	2	2
P2(3)	2	1		
P3(4)	2	2		
P4(2)	2			
P5(6)	2	2	2	



2) **Turnaround time (TAT)**

Turnaround time = **Completed time – Arrival time**

Process	FCFS	SJF	Non Preemptive Priority	Round Robin
P1	8 – 0 = 8	23 – 0 = 23	13 – 0 = 13	23 – 0 = 23
P2	11 – 0 = 11	5 – 0 = 5	3 – 0 = 3	13 – 0 = 13
P3	15 – 0 = 15	9 – 0 = 9	17 – 0 = 17	15 – 0 = 15
P4	17 – 0 = 17	2 – 0 = 2	5 – 0 = 5	8 – 0 = 8
P5	23 – 0 = 23	15 – 0 = 15	23 – 0 = 23	21 – 0 = 21
Total TAT	**74 ms**	**54 ms**	**61 ms**	**80 ms**
Average TAT	74/5 = **14.8 ms**	54/5 = **10.8** ms	61/5 = **12.2** ms	80/5 = **16 ms**

3) **Waiting time (WT)**

Waiting time = **Turnaround time – Burst time**

Process	FCFS	SJF	Non Preemptive Priority	Round Robin
P1	8 – 8 = 0	23 – 8 = 15	13 – 8 = 5	23 – 8 = 15
P2	11 – 3 = 8	5 – 3 = 2	3 – 3 = 0	13 – 3 = 10
P3	15 – 4 = 11	9 – 4 = 5	17 – 4 = 13	15 – 4 = 11
P4	17 – 2 = 15	2 – 2 = 0	5 – 2 = 3	8 – 2 = 6
P5	23 – 6 = 17	15 – 6 = 9	23 – 6 = 17	21 – 6 = 15
Total WT	**51 ms**	**31 ms**	**38 ms**	**57 ms**
Average WT	51/5 = **10.2 ms**	31/5 = **6.2** ms	38/5 = **7.6** ms	57/5 = **11.4** ms

4) **Result**

Algorithm	Average Turnaround Time(ms)	Average Waiting Time(ms)
FCFS	**14.8**	**10.2**
SJF	**10.8**	**6.2**
Non Preemptive Priority	**12.2**	**7.6**
Round Robin	**16**	**11.4**

It can be noted that **Shortest Job First(SJF)** gives the minimum average waiting time(6.2 ms) and is optimal.

****

UNIT 2 – Bankers Algorithm

1) An operating system contains 3 resources namely R1, R2 and R3. The number of instances of each resource type are **7, 7,10**. The current resource allocation state is shown below,

Process	Current Allocation			Maximum Need		
	R1	R2	R3	R1	R2	R3
P1	2	2	3	3	6	8
P2	2	0	3	4	3	3
P3	1	2	4	3	4	4

a) Is the current allocation in a safe state? **(4)**

b) Can the request made by process **P1**(1, 1, 0) be granted. **(Nov 2015, May 2008)**

****

Solution

a) i) **Available**

	Current Allocation		
	R1	R2	R3
	2	2	3
	2	0	3
	1	2	4
Sum =	**5**	**4**	**10**

Available = Number of instances - Sum of Allocation
= (7,7,10) - (5, 4, 10) = (2, 3, 0)

ii) The contents of **need matrix** are

Process	Need = Max - Allocation		
	R1	R2	R3
P1	1	4	5
P2	2	3	0
P3	2	2	0

iii) **Safe Sequence**

1) If Need ≤ Available, then allocate the resources to that process.
   New available = Prev Available + Allocation
2) If Need > Available, move to next process.

Process	Need (N)	Available(A)	Condition N ≤ A	New Available
P1	(1, 4, 5)	(2, 3, 0)	False	
P2	(2, 3, 0)	(2, 3, 0)	True	**(4, 3, 3)**
P3	(2, 2, 0)	(4, 3, 3)	True	**(5, 5, 7)**
P1	(1, 4, 5)	(5, 5, 7)	True	**(7, 7, 10)**

Safe sequence is **<P2, P3, P1>**. So the system is in a **safe state** if it can allocate resources in the sequence **<P2, P3, P1>**.

b) **Process P1 New request (1, 1, 0)**

Since P1 request (1, 1, 0) ≤ P1 Need (1, 4, 5) and P1 request (1, 1, 0) ≤ Available (2, 3, 0), pretend that this request can be granted immediately.

P1 Need = Previous P1 Need – P1 New Request
P1 Allocation = P1 Allocation + P1 New Request
Available = Available – P1 New Request

Process	Need			Allocation			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	0	3	5	3	3	3	1	2	0
P2	2	3	0	2	0	3			
P3	2	2	0	1	2	4			

Safe Sequence

1) If Need ≤ Available, then allocate the resources to that process.
   New available = Prev Available + Allocation
2) If Need > Available, move to next process.

Process	Need(N)	Available(A)	Condition N ≤ A	New Available
P0	(0, 3, 5)	(1, 2, 0)	False	
P1	(2, 3, 0)	(1, 2, 0)	False	
P2	(2, 2, 0)	(1 , 2, 0)	False	

Since the request from process P1(1, 1, 0) results in an unsafe state, it cannot be granted immediately.
i.e) System is in an **unsafe state** and process P1 must wait for request (1, 1, 0).

****

2) Consider the following snapshot of a system:

Process	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	0	0	1	2	0	0	1	2	1	5	2	0
P1	1	0	0	0	1	7	5	0				
P2	1	3	5	4	2	3	5	6				
P3	0	6	3	2	0	6	5	2				
P4	0	0	1	4	0	6	5	6				

Answer the following questions based on the banker's algorithm **(May 2014)**

a) What is the content of Need matrix? **(6)**
b) Is the system in a safe state? **(2)**
c) Which processes may cause deadlock if the system is not safe. **(3)**
d) If a request from process P1 arrives for (0, 4, 3, 1) can the request be granted immediately? Justify. **(5)**

****

a) Need matrix

The contents of need matrix are

Process	Need = Max – Allocation			
	A	B	C	D
P0	0	0	0	0
P1	0	7	5	0
P2	1	0	0	2
P3	0	0	2	0
P4	0	6	4	2

b) Safe sequence

1) If Need ≤ Available, then allocate the resources to that process.
   New available = Prev Available + Allocation
2) If Need > Available, move to next process.

Process	Need (N)	Available(A)	Condition N ≤ A	New Available
P0	(0, 0, 0, 0)	(1, 5, 2, 0)	True	(1, 5, 3, 2)
P1	(0, 7, 5, 0)	(1, 5, 3, 2)	False	
P2	(1, 0, 0, 2)	(1, 5, 3, 2)	True	(2, 8, 8, 6)
P3	(0, 0, 2, 0)	(2, 8, 8, 6)	True	(2, 14, 11, 8)
P4	(0, 6, 4, 2)	(2, 14, 11, 8)	True	(2, 14, 12, 12)
P1	(0, 7, 5, 0)	(2, 14, 12, 12)	True	(3, 14, 12, 12)

Safe sequence is <**P0, P2, P3, P4, P1**>. Yes the system is in a safe state if it can allocate resources in the sequence <**P0, P2, P3, P4, P1**>

c) The system is in a safe state if it can allocate resources in the sequence <**P0, P2, P3, P4, P1**>. If no suc sequence exists, then the system state is said to be **unsafe**. A safe state is not a deadlock state. Process P may cause deadlock if the system is not **safe.**

Safety algorithm

- The algorithm for finding out whether or not a system is in a safe state

1. Let Work and Finish be vectors of length m and n, respectively. Initialize:
   Work = Available
   Finish[i] = false for i=1,2,...,n.
2. Find an i such that both:
   (a) Finish[i] = false
   (b) **$Need_i \leq Work$**
   If no such i exists, go to step 4.
3. **$Work = Work + Allocation_i$**
   Finish[i] = true Go to step 2.
4. If Finish[i] == true for all i, then the system is in a safe state.

d) P1 New request (0, 4, 3, 1)

P1 request (0, 4, 3, 1) > P1 Need (0, 7, 5, 0), raise an error condition, since the process P1 has exceeded its maximum claim. So a request (0, 4, 3, 1) from process P1 cannot be granted immediately.

Resource-Request Algorithm

1. If $Request_i \leq Need_i$ go to step 2. Otherwise, raise error, since process has exceeded its maximum claim.
2. If $Request_i \leq$ Available, go to step 3. Otherwise P_i must wait, since the resources are not available.
3. The system pretend to have allocated the requested resources to process P_i by modifying
   $Available = Available - Request_i$
   $Allocation_i = Allocation_i + Request_i$
   $Need_i = Need_i - Request_i$

- If the resulting resource-allocation state is safe, the resources are allocated to process P_i.
- If the new state is unsafe then P_i must wait, and the old resource-allocation state is restored.

****

3) Consider the following snapshot of a system: **(May 2013)**

Process	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3	3	3	2
P1	2	0	0	3	2	2			
P2	3	0	2	9	0	2			
P3	2	1	1	2	2	2			
P4	0	0	2	4	3	3			

Answer the following questions based on banker's algorithm

a) What is the content of NEED matrix? **(2)**

b) Is the system in a safe state? **(3)**

c) If a request from process P0 arrives for (0, 2, 0) can the request be granted immediately? **(3)**

****

a) Need matrix

The contents of need matrix are

Process	Need = Max – Allocation		
	A	B	C
P0	7	4	3
P1	1	2	2
P2	6	0	0
P3	0	1	1
P4	4	3	1

b) Safe sequence

1) If Need ≤ Available, then allocate the resources to that process.
   New available = Prev Available + Allocation
2) If Need > Available, move to next process.

Process	Need (N)	Available(A)	Condition N ≤ A	New Available
P0	**(7, 4, 3)**	**(3, 3, 2)**	**False**	
P1	(1, 2, 2)	(3, 3, 2)	True	(5, 3, 2)
P2	**(6, 0, 0)**	**(5, 3, 2)**	**False**	
P3	(0, 1, 1)	(5, 3, 2)	True	(7, 4, 3)
P4	(4, 3, 1)	(7, 4, 3)	True	(7, 4, 5)
P0	(7, 4, 3)	(7, 4, 5)	True	(7, 5, 5)
P2	(6, 0, 0)	(7, 5, 5)	True	(10, 5, 7)

Safe sequence is **<P1, P3, P4, P0, P2>**. Yes the system is in a safe state if it can allocate resources in the sequence **<P1, P3, P4, P0, P2>**

c) P0 New request (0, 2, 0)

Since P0 request (0, 2, 0) ≤ P0 Need (7, 4, 3) and P0 request (0, 2, 0) ≤ Available (3, 3, 2), pretend that this request can be granted immediately.

P0 Need = Previous P0 Need – P0 New Request
P0 Allocation = P0 Allocation + P0 New Request
Available = Available – P0 New Request

Process	Need			Allocation			Available		
	A	B	C	A	B	C	A	B	C
P0	**7**	**2**	**3**	**0**	**3**	**0**	**3**	**1**	**2**
P1	1	2	2	2	0	0			
P2	6	0	0	3	0	2			
P3	0	1	1	2	1	1			
P4	4	3	1	0	0	2			

Safe Sequence

Process	Need(N)	Available(A)	Condition N ≤ A	New Available
P0	**(7, 2, 3)**	**(3, 1, 2)**	**False**	
P1	**(1, 2, 2)**	**(3, 1, 2)**	**False**	
P2	**(6, 0, 0)**	**(3, 1, 2)**	**False**	
P3	(0, 1, 1)	(3, 1, 2)	True	(5, 2, 3)
P4	**(4, 3, 1)**	**(5, 2, 3)**	**False**	
P0	**(7, 2, 3)**	**(5, 2, 3)**	**False**	
P1	(1, 2, 2)	(5, 2, 3)	True	(7, 2, 3)
P2	(6, 0, 0)	(7, 2, 3)	True	(10, 2, 5)
P4	**(4, 3, 1)**	**(10, 2, 5)**	**False**	

Since the request from process P0(0, 2, 0) results in an unsafe state, it cannot be granted immediately.
i.e) System is in an **unsafe state** and process P0 must wait for request (0, 2, 0).

****

4) Consider the following snapshot of a system: **(May 2012, Nov 2010)**

Process	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	0	0	1	2	0	0	1	2	1	5	2	0
P1	1	0	0	0	1	7	5	0				
P2	1	3	5	4	2	3	5	6				
P3	0	6	3	2	0	6	5	2				
P4	0	0	1	4	0	6	5	6				

Answer the following questions based on the banker's algorithm

a) Define safety algorithm. **(4)**
b) What is the content of the matrix Need? **(2)**
c) Is the system in a safe state? **(5)**
d) If a request from process P1 arrives for (0, 4, 2, 0) can the request be granted immediately? (5)

a) **Safety algorithm**

- A state is **safe** if the system can allocate resources to each process in some order and still avoid a deadlock. More formally, a system is in a **safe state** only if there exists a **safe sequence**. If **no** such **sequence** exists, then the system state is said to be **unsafe.**
- A safe state is not a deadlock state. Conversely, a deadlock state is an unsafe state.
- Not all **unsafe states** are deadlocks. An unsafe state ***may*** lead to a deadlock
- The algorithm for finding out whether or not a system is in a safe state

1. Let Work and Finish be vectors of length m and n, respectively. Initialize:
   Work = Available
   Finish[i] = false for i=1,2,...,n.
2. Find an i such that both:
   (a) Finish[i] = false
   (b) $\mathbf{Need_i \leq Work}$
   If no such i exists, go to step 4.
3. $\mathbf{Work = Work + Allocation_i}$
   Finish[i] = true Go to step 2.
4. If Finish[i] == true for all i, then the system is in a safe state.

b) **Need matrix**

The contents of need matrix are

Process	Need = Max – Allocation			
	A	B	C	D
P0	0	0	0	0
P1	0	7	5	0
P2	1	0	0	2
P3	0	0	2	0
P4	0	6	4	2

c) **Safe sequence**

1) If Need ≤ Available, then allocate the resources to that process.
   New available = Prev Available + Allocation
2) If Need > Available, move to next process.

Process	Need (N)	Available(A)	Condition N ≤ A	New Available
P0	(0, 0, 0, 0)	(1, 5, 2, 0)	True	(1, 5, 3, 2)
P1	**(0, 7, 5, 0)**	**(1, 5, 3, 2)**	**False**	
P2	(1, 0, 0, 2)	(1, 5, 3, 2)	True	(2, 8, 8, 6)
P3	(0, 0, 2, 0)	(2, 8, 8, 6)	True	(2, 14, 11, 8)
P4	(0, 6, 4, 2)	(2, 14, 11, 8)	True	(2, 14, 12, 12)
P1	(0, 7, 5, 0)	(2, 14, 12, 12)	True	(3, 14, 12, 12)

Safe sequence is **<P0, P2, P3, P4, P1>**. Yes the system is in a safe state if it can allocate resources in the sequence **<P0, P2, P3, P4, P1>**

d) **P1 New request (0, 4, 2, 0)**

Since P1 request (0, 4, 2, 0) ≤ P1 Need (0, 7, 5, 0) and P1 request (0, 4, 2, 0) ≤ Available (1, 5, 2, 0), pretend that this request can be granted immediately.

P1 Need = Previous P1 Need – P1 New Request
P1 Allocation = P1 Allocation + P1 New Request
Available = Available – P1 New Request

Process	Need				Allocation				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	0	0	0	0	0	0	1	2	**1**	**1**	**0**	**0**
P1	**0**	**3**	**3**	**0**	**1**	**4**	**2**	**0**				
P2	1	0	0	2	1	3	5	4				
P3	0	0	2	0	0	6	3	2				
P4	**0**	**6**	**4**	**2**	0	0	1	4				

Safe sequence

Process	Need (N)	Available (A)	Condition N ≤A	New Available
P0	(0, 0, 0, 0)	(1, 1, 0, 0)	True	(1, 1, 1, 2)
P1	**(0, 3, 3, 0)**	**(1, 1, 1, 2)**	**False**	
P2	(1, 0, 0, 2)	(1, 1, 1, 2)	True	(2, 4, 6, 6)
P3	(0, 0, 2, 0)	(2, 4, 6, 6)	True	(2, 10, 9, 8)
P4	(0, 6, 4, 2)	(2, 10, 9, 8)	True	(2, 10, 10, 12)
P1	(0, 3, 3, 0)	(2, 10, 10, 12)	True	(3, 14, 12, 12)

Safe sequence is **<P0, P2, P3, P4, P1>**. Yes the system is in a safe state if it can allocate resources in the sequence **<P0, P2, P3, P4, P1>**

****

5) Consider the following snapshot of a system: **(May 2010)**

Process	Allocation				Max. Demand				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	0	0	1	2	0	0	1	2	2	1	0	0
P1	2	0	0	0	2	7	5	0				
P2	0	0	3	4	6	6	5	6				
P3	2	3	5	4	4	3	5	6				
P4	0	3	3	2	0	6	5	2				

Write the banker's algorithm and find out the following with the help of banker's algorithm.

a) How many resources the system still needs?

b) Is the system currently safe? If it is in a safe state, write the safe sequence.

c) If a request from process P2 arrives for (0, 1, 0, 0) can it be granted immediately? **(16)**

****

Safety algorithm

- A state is **safe** if the system can allocate resources to each process in some order and still avoid a deadlock. More formally, a system is in a **safe state** only if there exists a **safe sequence**. If no such **sequence** exists, then the system state is said to be **unsafe**.
- A safe state is not a deadlock state. Conversely, a deadlock state is an unsafe state.
- Not all **unsafe states** are deadlocks. An unsafe state ***may*** lead to a deadlock
- The algorithm for finding out whether or not a system is in a safe state

1. Let Work and Finish be vectors of length m and n, respectively. Initialize:
   Work = Available
   Finish[i] = false for i=1,2,...,n.
2. Find an i such that both:
   (a) Finish[i] = false
   (b) **$Need_i \leq Work$**
   If no such i exists, go to step 4.
3. **$Work = Work + Allocation_i$**
   Finish[i] = true Go to step 2.
4. If Finish[i] == true for all i, then the system is in a safe state.

a) **Need matrix**

The contents of need matrix are

Process	Need = Max – Allocation			
	A	B	C	D
P0	0	0	0	0
P1	0	7	5	0
P2	6	6	2	2
P3	2	0	0	2
P4	0	3	2	0

b) Safe sequence

1) If Need ≤ Available, then allocate the resources to that process.
   New available = Prev Available + Allocation
2) If Need > Available, move to next process.

Process	Need (N)	Available(A)	Condition N ≤ A	New Available
P0	(0, 0, 0, 0)	(2, 1, 0, 0)	True	(2, 1, 1, 2)
P1	**(0, 7, 5, 0)**	**(2, 1, 1, 2)**	**False**	
P2	**(6, 6, 2, 2)**	**(2, 1, 1, 2)**	**False**	
P3	(2, 0, 0, 2)	(2, 1, 1, 2)	True	(4, 4, 6, 6)
P4	(0, 3, 2, 0)	(4, 4, 6, 6)	True	(4, 7, 9, 8)
P1	(0, 7, 5, 0)	(4, 7, 9, 8)	True	(6, 7, 9, 8)
P2	(6, 6, 2, 2)	(6, 7, 9, 8)	True	(6, 7, 12, 12)

Safe sequence is **<P0, P3, P4, P1, P2>**. Yes the system is in a safe state if it can allocate resources in the sequence **<P0, P3, P4, P1, P2>**

c) P2 New request (0, 1, 0, 0)

Since P2 request (0, 1, 0, 0) ≤ P2 Need (0, 7, 5, 0) and P2 request (0, 1, 0, 0) ≤ Available (2, 1, 0, 0), pretend that this request can be granted immediately.

P2 Need = Previous P2 Need – P2 New Request
P2 Allocation = P2 Allocation + P2 New Request
Available = Available – P2 New Request

Process	Need				Allocation				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	0	0	0	0	0	0	1	2	**2**	**0**	**0**	**0**
P1	0	7	5	0	2	0	0	0				
P2	**6**	**5**	**2**	**2**	**0**	**1**	**3**	**4**				
P3	2	0	0	2	2	3	5	4				
P4	0	3	2	0	0	3	3	2				

Safe sequence

Process	Need (N)	Available (A)	Condition N ≤ A	New Available
P0	(0, 0, 0, 0)	(2, 0, 0, 0)	True	(2, 0, 1, 2)
P1	**(0, 7, 5, 0)**	**(2, 0, 1, 2)**	**False**	
P2	**(6, 5, 2, 2)**	**(2, 0, 1, 2)**	**False**	
P3	(2, 0, 0, 2)	(2, 0, 1, 2)	True	(4, 3, 6, 6)
P4	(0, 3, 2, 0)	(4, 3, 6, 6)	True	(4, 6, 9, 8)
P1	**(0, 7, 5, 0)**	**(4, 6, 9, 8)**	**False**	
P2	**(6, 5, 2, 2)**	**(4, 6, 9, 8)**	**False**	

Since the request from process P2(0, 1, 0, 0) results in an unsafe state, it cannot be granted immediately.
i.e) System is in an **unsafe state** and process P2 must wait for request (0, 1, 0, 0).

****

6) Consider the following snapshot of a system: **(Nov 2006)**

Process	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	2	0	1	2	2	0	1	2	2	4	2	1
P1	1	0	0	0	2	7	5	0				
P2	1	3	5	4	2	3	5	6				
P3	0	6	3	2	0	7	5	2				
P4	0	0	1	4	0	7	5	6				

Execute banker's algorithm to answer the following:

a) Is the system in a safe state? If the system is safe, show how all the process could complete their execution successfully. If the system is unsafe, show how deadlock might occur. Explain. **(6)**

b) If a request from process P1 arrives **(1, 4, 2, 0)** can the request be granted? **(10)**

Safety algorithm

- A state is **safe** if the system can allocate resources to each process in some order and still avoid a deadlock. More formally, a system is in a **safe state** only if there exists a **safe sequence**. If no such **sequence** exists, then the system state is said to be **unsafe.**
- A safe state is not a deadlock state. Conversely, a deadlock state is an unsafe state.
- Not all **unsafe states** are deadlocks. An unsafe state ***may*** lead to a deadlock
- The algorithm for finding out whether or not a system is in a safe state

1. Let Work and Finish be vectors of length m and n, respectively. Initialize:
   Work = Available
   Finish[i] = false for i=1,2,...,n.
2. Find an i such that both:
   (a) Finish[i] = false
   (b) $\mathbf{Need_i \leq Work}$
   If no such i exists, go to step 4.
3. $\mathbf{Work = Work + Allocation_i}$
   Finish[i] = true Go to step 2.
4. If Finish[i] == true for all i, then the system is in a safe state.

a) Need matrix

The contents of need matrix are

Process	Need = Max – Allocation			
	A	B	C	D
P0	0	0	0	0
P1	1	7	5	0
P2	1	0	0	2
P3	0	1	2	0
P4	0	7	4	2

b) Safe sequence

1) If Need ≤ Available, then allocate the resources to that process.
   New available = Prev Available + Allocation
2) If Need > Available, move to next process.

Process	Need (N)	Available(A)	Condition N ≤ A	New Available
P0	(0, 0, 0, 0)	(2, 4, 2, 1)	True	(4, 4, 3, 3)
P1	**(1, 7, 5, 0)**	**(4, 4, 3, 3)**	**False**	
P2	(1, 0, 0, 2)	(4, 4, 3, 3)	True	(5, 7, 8, 7)
P3	(0, 1, 2, 0)	(5, 7, 8, 7)	True	(5, 13, 11, 9)
P4	(0, 7, 4, 2)	(5, 13, 11, 9)	True	(5, 13, 12, 13)
P1	(1, 7, 5, 0)	(5, 13, 12, 13)	True	(6, 13, 12, 13)

Safe sequence is **<P0, P2, P3, P4, P1>**. Yes the system is in a safe state if it can allocate resources in the sequence **<P0, P2, P3, P4, P1>**

c) P1 New request (1, 4, 2, 0)

Since P1 request (1, 4, 2, 0) ≤ P1 Need (1, 7, 5, 0) and P1 request (1, 4, 2, 0) ≤ Available (2, 4, 2, 1), pretend that this request can be granted immediately.

P1 Need = Previous P1 Need – P1 New Request
P1 Allocation = P1 Allocation + P1 New Request
Available = Available – P1 New Request

Process	Need				Allocation				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	0	0	0	0	2	0	1	2	**1**	**0**	**0**	**1**
P1	**0**	**3**	**3**	**0**	**2**	**4**	**2**	**0**				
P2	1	0	0	2	1	3	5	4				
P3	0	1	2	0	0	6	3	2				
P4	0	7	4	2	0	0	1	4				

Safe sequence

Process	Need (N)	Available (A)	Condition N ≤A	New Available
P0	(0, 0, 0, 0)	(1, 0, 0, 1)	True	(3, 0, 1, 3)
P1	(0, 3, 3, 0)	(3, 0, 1, 3)	False	
P2	(1, 0, 0, 2)	(3, 0, 1, 3)	True	(4, 3, 6, 7)
P3	(0, 1, 2, 0)	(4, 3, 6, 7)	True	(4, 9, 9, 9)
P4	(0, 7, 4, 2)	(4, 9, 9, 9)	True	(4, 9, 10, 13)
P1	(0, 3, 3, 0)	(4, 9, 10, 13)	True	(6, 13, 12, 13)

Safe sequence is **<P0, P2, P3, P4, P1>**. Yes the system is in a safe state if it can allocate resources in the sequence **<P0, P2, P3, P4, P1>**

****

7) Consider the following snapshot of a system: **(Nov 2008)**

Process	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	0	0	1	1	0	0	1	1	1	5	2	2
P1	0	1	0	1	1	7	5	1				
P2	1	3	5	1	2	3	5	2				
P3	0	5	3	1	1	6	5	2				
P4	0	0	1	1	5	6	5	1				

Execute banker's algorithm to answer the following:

a) What is the content of a need matrix? **(2)**

b) Is the system in a safe state? If the system is safe, show how all the process could complete their execution successfully. If the system is unsafe, show how deadlock might occur. Explain. **(6)**

c) If a request from process P1 arrives **(0, 3, 2, 0)** can the request be granted? If yes, write the sequence of allocation. **(8)**

****

Safety algorithm

- A state is **safe** if the system can allocate resources to each process in some order and still avoid a deadlock. More formally, a system is in a **safe state** only if there exists a **safe sequence**. If **no** such **sequence** exists, then the system state is said to be **unsafe.**
- A safe state is not a deadlock state. Conversely, a deadlock state is an unsafe state.
- Not all **unsafe states** are deadlocks. An unsafe state ***may*** lead to a deadlock
- The algorithm for finding out whether or not a system is in a safe state

1. Let Work and Finish be vectors of length m and n, respectively. Initialize:
   Work = Available
   Finish[i] =false for i=1,2,...,n.
2. Find an i such that both:
   (a) Finish[i] = false
   (b) **$Need_i \leq Work$**
   If no such i exists, go to step 4.
3. **$Work = Work + Allocation_i$**
   Finish[i] = true Go to step 2.
4. If Finish[i] == true for all i, then the system is in a safe state.

a) **Need matrix**

The contents of need matrix are

Process	Need = Max – Allocation			
	A	B	C	D
P0	0	0	0	0
P1	1	6	5	0
P2	1	0	0	1
P3	1	1	2	1
P4	5	6	4	0

b) **Safe sequence**

1) If Need ≤ Available, then allocate the resources to that process.
   New available = Prev Available + Allocation
2) If Need > Available, move to next process.

Process	Need (N)	Available(A)	Condition N ≤ A	New Available
P0	(0, 0, 0, 0)	(1, 5, 2, 2)	True	(1, 5, 3, 3)
P1	**(1, 6, 5, 0)**	**(1, 5, 3, 3)**	**False**	
P2	(1, 0, 0, 1)	(1, 5, 3, 3)	True	(2, 8, 8, 4)
P3	(1, 1, 2, 1)	(2, 8, 8, 4)	True	(2, 13, 11, 5)
P4	**(5, 6, 4, 0)**	**(2, 13, 11, 5)**	**False**	
P1	(1, 6, 5, 0)	(2, 13, 11, 5)	True	(2, 14, 11, 6)
P4	**(5, 6, 4, 0)**	**(2, 14, 11, 6)**	**False**	

The system is in **unsafe state.** A system is in a safe state only if there exists a **safe sequence.** If **no** such **sequence** exists, then the system state is said to be **unsafe.**

c) **P1 New request (0, 3, 2, 0)**

Since P1 request (0, 3, 2, 0) ≤ P1 Need (1, 6, 5, 0) and P1 request (0, 3, 2, 0) ≤ Available (1, 5, 2, 2), pretend that this request can be granted immediately.

P1 Need = Previous P1 Need – P1 New Request
P1 Allocation = P1 Allocation + P1 New Request
Available = Available – P1 New Request

Process	Need				Allocation				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	0	0	0	0	0	0	1	1	**1**	**2**	**0**	**2**
P1	**1**	**3**	**3**	**0**	**0**	**4**	**2**	**1**				
P2	1	0	0	1	1	3	5	1				
P3	1	1	2	1	0	5	3	1				
P4	5	6	4	0	0	0	1	1				

Safe sequence

Process	Need (N)	Available (A)	Condition N ≤ A	New Available
P0	(0, 0, 0, 0)	(1, 2, 0, 2)	True	(1, 2, 1, 3)
P1	**(1, 3, 3, 0)**	**(1, 2, 1, 3)**	**False**	
P2	(1, 0, 0, 1)	(1, 2, 1, 3)	True	(2, 5, 6, 4)
P3	(1, 1, 2, 1)	(2, 5, 6, 4)	True	(2, 10, 9, 5)
P4	**(5, 6, 4, 0)**	**(2, 10, 9, 5)**	**False**	
P1	(1, 3, 3, 0)	(2, 10, 9, 5)	True	(2, 14, 11, 6)
P4	**(5, 6, 4, 0)**	**(2, 14, 11, 6)**	**False**	

The system is in **unsafe state.** A system is in a safe state only if there exists a **safe sequence.** If **no** such **sequence** exists, then the system state is said to be **unsafe.**