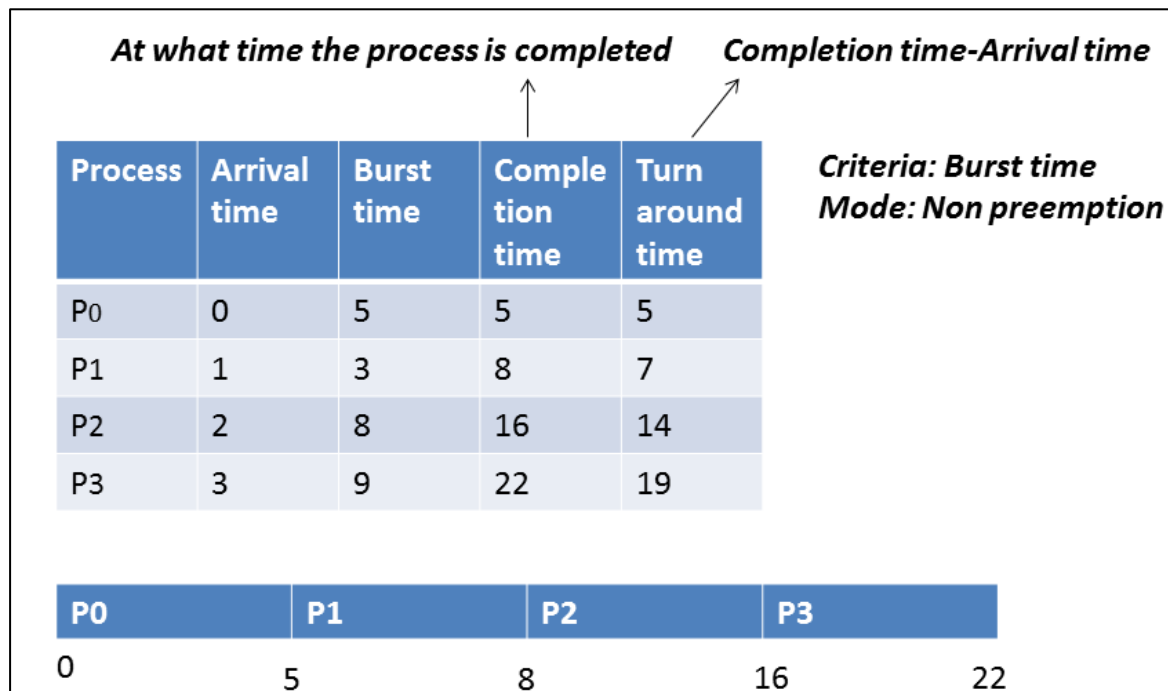
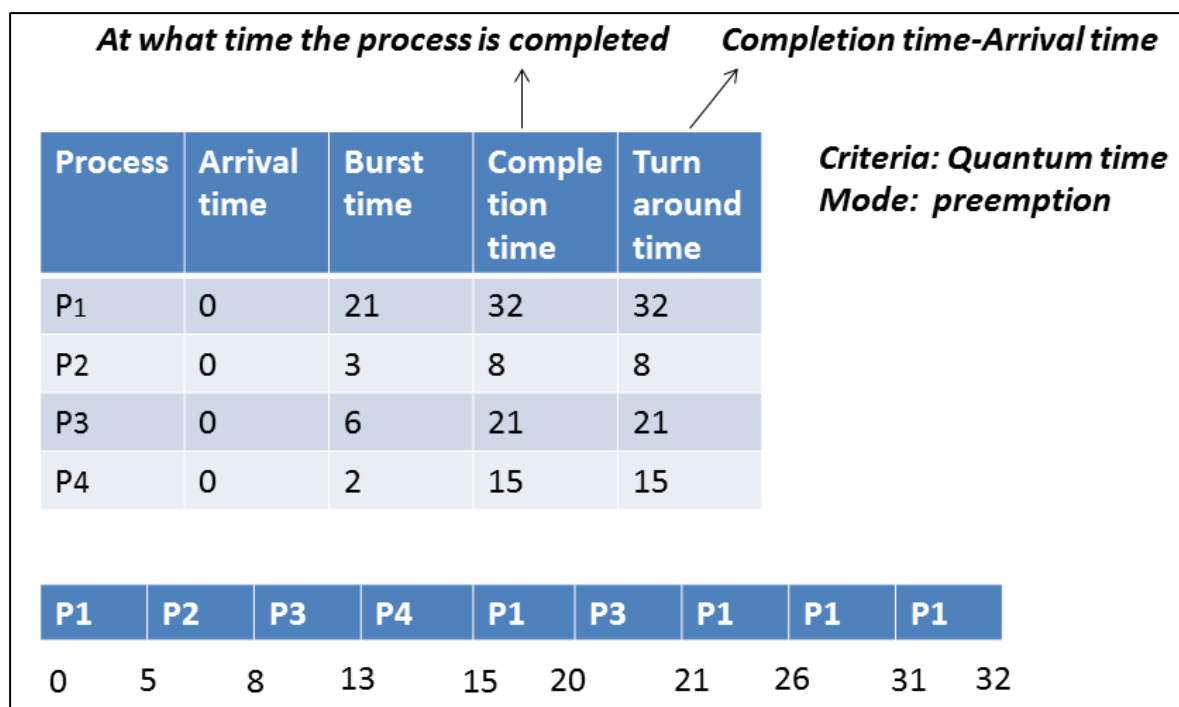


1. First Come, First Serve (FCFS)

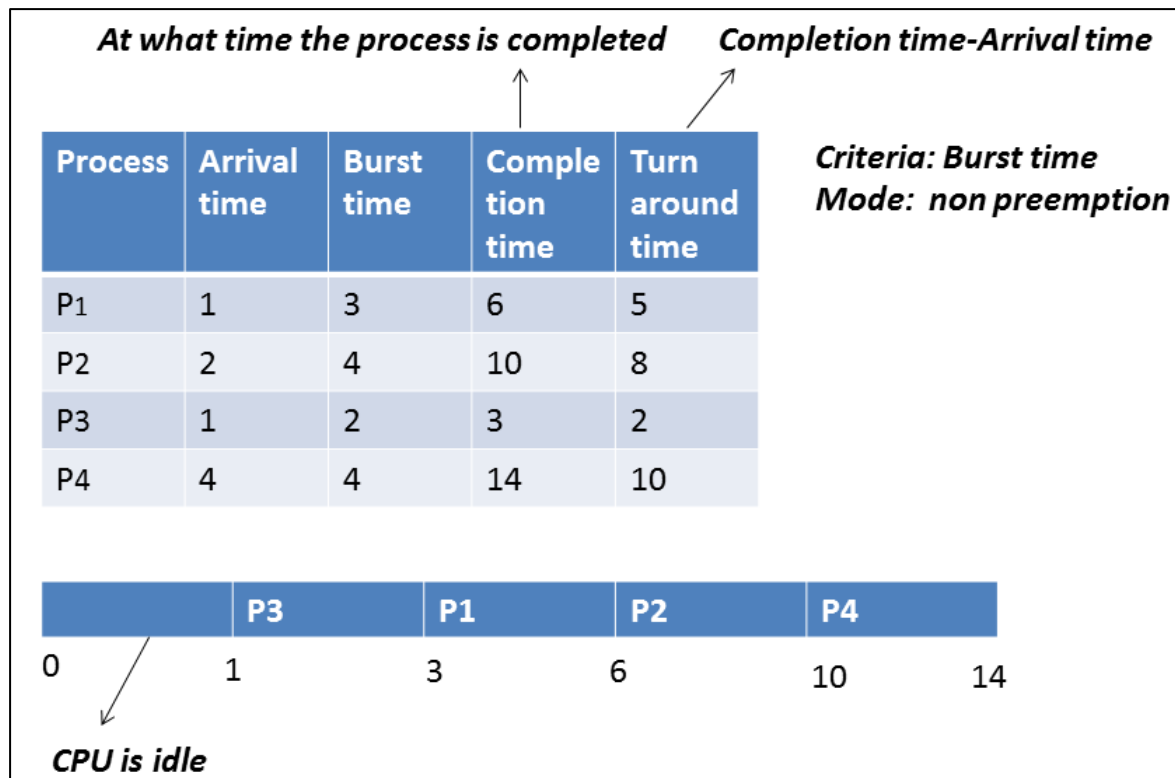


2. Round Robin

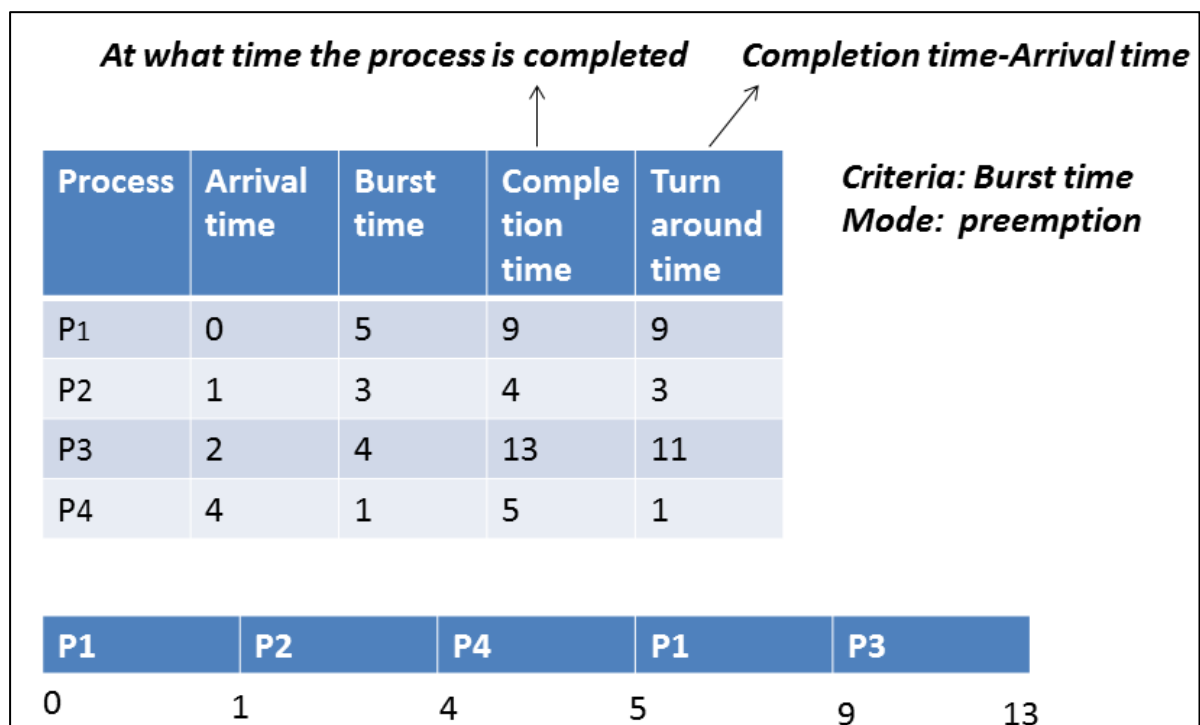
Quantum Time = 5



3. Shortest Time Job First (SJF) – Non Preemptive



4. Shortest Time Job First (SJF) - Preemptive



5. Priority Scheduling Algorithm

At what time the process is completed *Completion time-Arrival time*

Process	Arrival time	Burst time	Priority	Completion time	Turn around time
P1	0	4	4	4	4-0=4
P2	1	3	3	15	15-1=14
P3	2	1	2	12	12-2=10
P4	3	5	1	8	9-3=6
P5	4	2	1	11	11-4=7

Criteria: priority
Mode: Non preemption

P1	P4	P5	P3	P2	
0	4	9	11	12	15

Safety Algorithm

It is a safety algorithm used to check whether or not a system is in a safe state or follows the safe sequence in a banker's algorithm:

1. There are two vectors **Wok** and **Finish** of length m and n in a safety algorithm.

Initialize: Work = Available

Finish[i] = false; for i = 0, 1, 2, 3, 4... n - 1.

2. Check the availability status for each type of resources [i], such as:

Need[i] <= Work

Finish[i] == false if the i does not exist, go to step 4.

3. **Work = Work + Allocation(i)**

Finish[i] = true

Go to step 2 to check the status of resource availability for the next process.

4. If Finish[i] == true; it means that the system is safe for all processes.

Resource Request Algorithm

A resource request algorithm checks how a system will behave when a process makes each type of resource request in a system as a request matrix.

Let create a resource request array $R[i]$ for each process $P[i]$. If the Resource Request_i [j] equal to 'K', which means the process $P[i]$ requires 'k' instances of Resources type $R[j]$ in the system.

1. When the number of **requested resources** of each type is less than the **Need** resources, go to step 2 and if the condition fails, which means that the process $P[i]$ exceeds its maximum claim for the resource. As the expression suggests:

If **Request(i) <= Need**

Go to step 2;

2. And when the number of requested resources of each type is less than the available resource for each process, go to step (3). As the expression suggests:

If **Request(i) <= Available**

Else Process $P[i]$ must wait for the resource since it is not available for use.

3. When the requested resource is allocated to the process by changing state:

Available = Available - Request

Allocation(i) = Allocation(i) + Request (i)

Need_i = Need_i - Request_i

When the resource allocation state is safe, its resources are allocated to the process $P(i)$. And if the new state is unsafe, the Process $P(i)$ has to wait for each type of Request $R(i)$ and restore the old resource-allocation state.

Example: Consider a system that contains five processes P1, P2, P3, P4, P5 and the three resource types A, B and C. Following are the resources types: A has 10, B has 5 and the resource type C has 7 instances.

Process	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P1	0	1	0	7	5	3	3	3	2
P2	2	0	0	3	2	2			
P3	3	0	2	9	0	2			
P4	2	1	1	2	2	2			
P5	0	0	2	4	3	3			

Answer the following questions using the banker's algorithm:

1. What is the reference of the need matrix?
2. Determine if the system is safe or not.
3. What will happen if the resource request (1, 0, 0) for process P1 can the system accept this request immediately?

1. Context of the need matrix is as follows:

$\text{Need}[i] = \text{Max}[i] - \text{Allocation}[i]$

Need for P1: $(7, 5, 3) - (0, 1, 0) = 7, 4, 3$

Need for P2: $(3, 2, 2) - (2, 0, 0) = 1, 2, 2$

Need for P3: $(9, 0, 2) - (3, 0, 2) = 6, 0, 0$

Need for P4: $(2, 2, 2) - (2, 1, 1) = 0, 1, 1$

Need for P5: $(4, 3, 3) - (0, 0, 2) = 4, 3, 1$

Process	Need		
	A	B	C
P1	7	4	3
P2	1	2	2
P3	6	0	0
P4	0	1	1
P5	4	3	1

Hence, we created the context of need matrix.

2. Apply the Banker's Algorithm:

Available Resources of A, B and C are 3, 3, and 2.

Now we check if each type of resource request is available for each process.

Step 1: For Process P1:

Need \leq Available

7, 4, 3 \leq 3, 3, 2 condition is **false**.

So, we examine another process, P2.

Step 2: For Process P2:

Need \leq Available

1, 2, 2 \leq 3, 3, 2 condition **true**

New available = available + Allocation

(3, 3, 2) + (2, 0, 0) \Rightarrow 5, 3, 2

Similarly, we examine another process P3.

Step 3: For Process P3:

P3 Need \leq Available

6, 0, 0 \leq 5, 3, 2 condition is **false**.

Similarly, we examine another process, P4.

Step 4: For Process P4:

P4 Need \leq Available

0, 1, 1 \leq 5, 3, 2 condition is **true**

New Available resource = Available + Allocation

5, 3, 2 + 2, 1, 1 \Rightarrow 7, 4, 3

Similarly, we examine another process P5.

Step 5: For Process P5:

P5 Need \leq Available

4, 3, 1 \leq 7, 4, 3 condition is **true**

New available resource = Available + Allocation

7, 4, 3 + 0, 0, 2 \Rightarrow 7, 4, 5

Now, we again examine each type of resource request for processes P1 and P3.

Step 6: For Process P1:

P1 Need \leq Available

7, 4, 3 \leq 7, 4, 5 condition is **true**

New Available Resource = Available + Allocation

7, 4, 5 + 0, 1, 0 \Rightarrow 7, 5, 5

So, we examine another process P2.

Step 7: For Process P3:

P3 Need \leq Available

6, 0, 0 \leq 7, 5, 5 condition is true

New Available Resource = Available + Allocation

7, 5, 5 + 3, 0, 2 \Rightarrow 10, 5, 7

Hence, we execute the banker's algorithm to find the safe state and the safe sequence like P2, P4, P5, P1 and P3.

3. For granting the Request (1, 0, 2), first we have to check that **Request \leq Available**, that is (1, 0, 2) \leq (3, 3, 2), since the condition is true. So the process P1 gets the request immediately.

1. First In First Out (FIFO)

Page reference: 1, 3, 0, 3, 5, 6, 3

1	3	0	3	5	6	3
		0	0	0	0	3
	3	3	3	3	6	6
1	1	1	1	5	5	5
Miss	Miss	Miss	Hit	Miss	Miss	Miss

Total Page Fault = 6

2. Least Recently Used (LRU)

Page reference: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 3

No. of Page frame - 4

7	0	1	2	0	3	0	4	2	3	0	3	2	3
			2	2	2	2	2	2	2	2	2	2	2
		1	1	1	1	1	4	4	4	4	4	4	4
	0	0	0	0	0	0	0	0	0	0	0	0	0
7	7	7	7	7	3	3	3	3	3	3	3	3	3
Miss	Miss	Miss	Miss	Hit	Miss	Hit	Miss	Hit	Hit	Hit	Hit	Hit	Hit

Total Page Fault = 6

Here LRU has same number of page fault as optimal but it may differ according to question.

3. Optimal Page Replacement

Page reference 7,0,1,2,0,3,0,4,2,3,0,3,2,3 **No. of Page frame - 4**

7	0	1	2	0	3	0	4	2	3	0	3	2	3
			2	2	2	2	2	2	2	2	2	2	2
		1	1	1	1	1	4	4	4	4	4	4	4
	0	0	0	0	0	0	0	0	0	0	0	0	0
7	7	7	7	7	3	3	3	3	3	3	3	3	3
Miss	Miss	Miss	Miss	Hit	Miss	Hit	Miss	Hit	Hit	Hit	Hit	Hit	Hit

Total Page Fault = 6