Operating System

Unit – 2
(Part-C)
Process

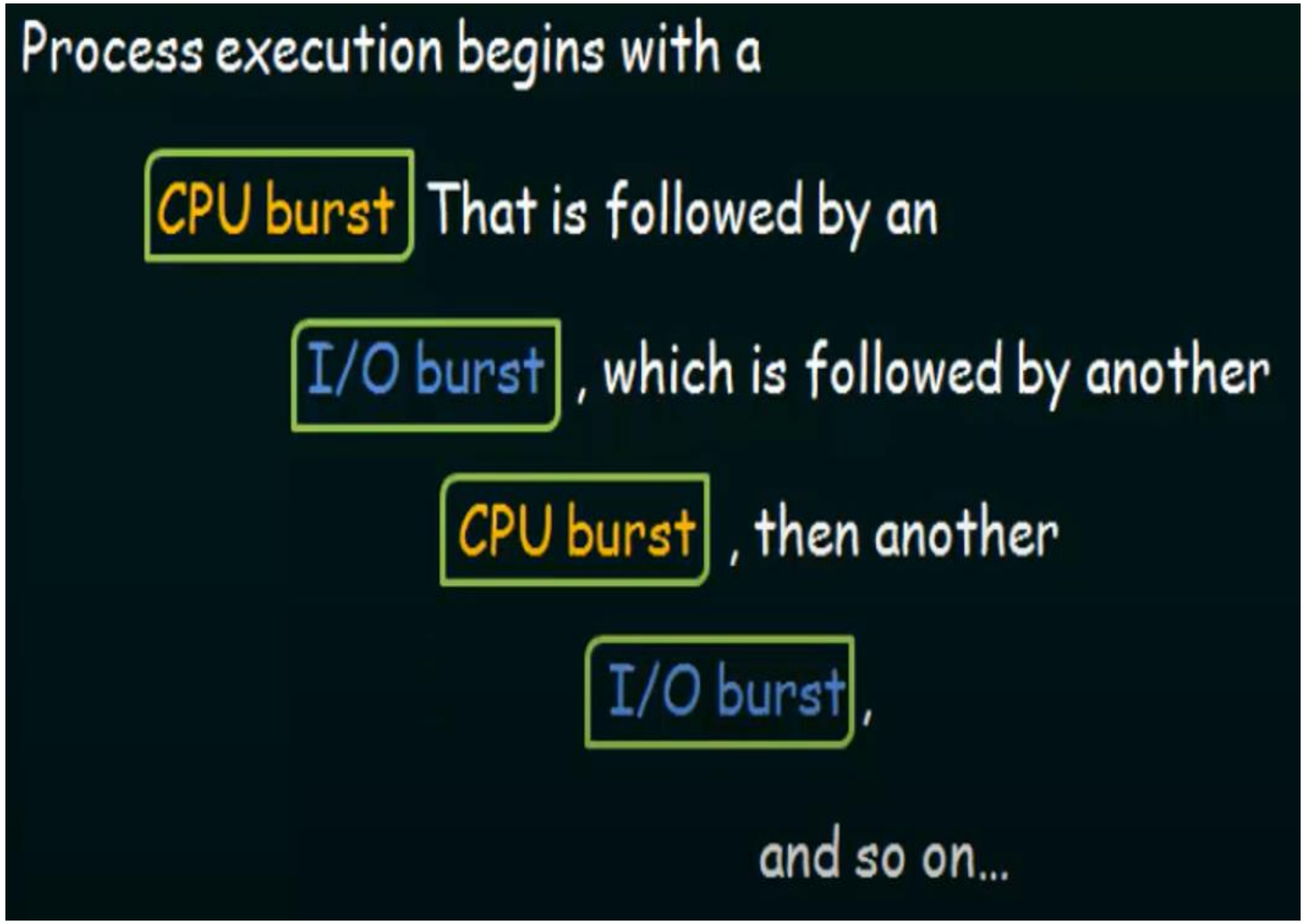
Mayank Mishra
School of Electronics Engineering
KIIT-Deemed to be University

CPU and I/O Burst Cycles

- Process execution consists of a cycle of **CPU execution** and **I/O wait**. Processes alternate between these two states.

- **CPU Burst** is when the process is being executed in the CPU.

- **I/O Burst** is when the CPU is waiting for I/O for further execution.


Process execution begins with a

CPU burst That is followed by an

I/O burst, which is followed by another

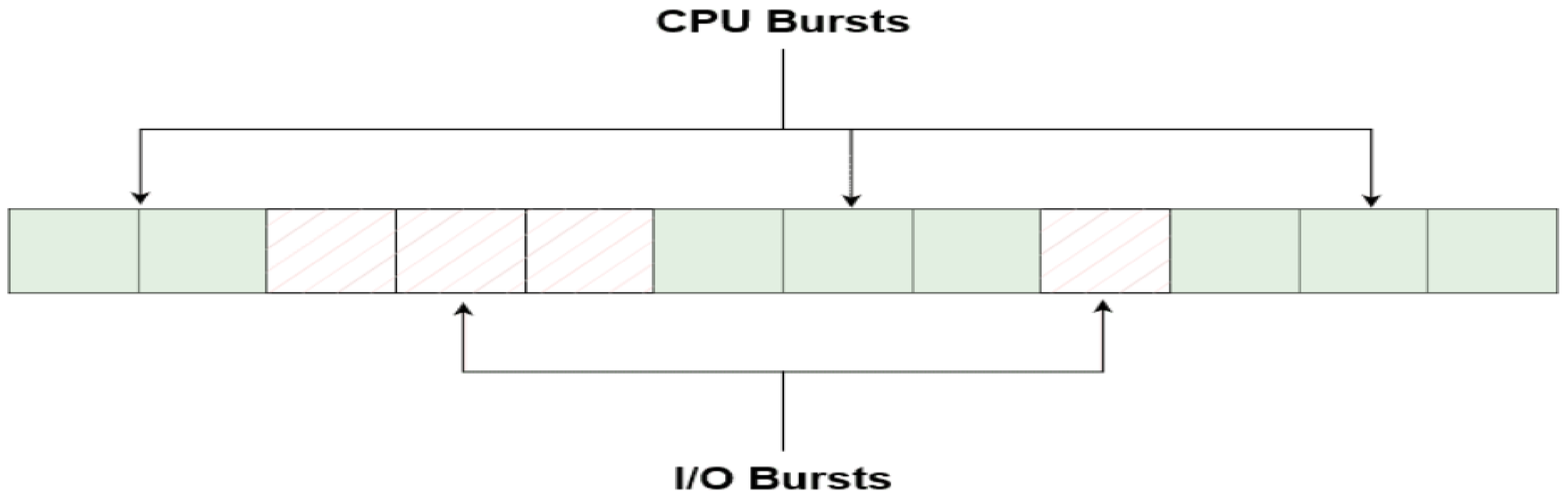
CPU burst, then another

I/O burst,

and so on...

CPU and I/O Burst Cycles (Contd.)

- In many real-world scenarios, CPU and I/O bursts are interconnected. For example, a web server may experience CPU bursts while processing requests and I/O bursts when reading or writing data to storage. Balancing the utilization of CPU and I/O resources is essential for optimizing system performance.



In the figure above, we can see how one process or program on a system changes between CPU and I/O utilization while running.

Question 1:

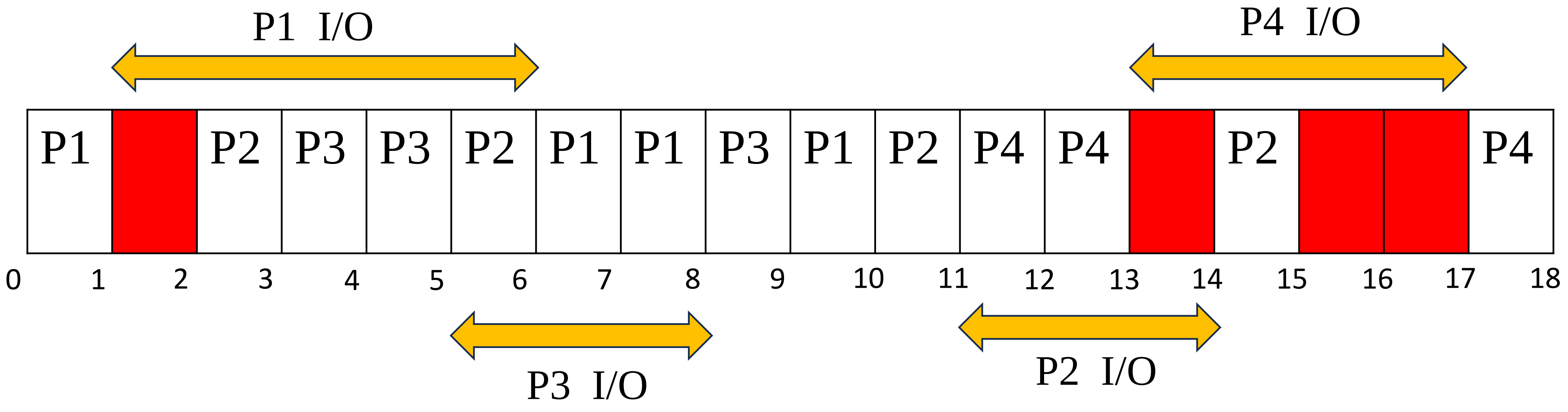
An operating system uses Priority scheduling algorithm for pre-emptive scheduling of processes. Considering the set of 4 processes whose arrival time, priority (0 is the highest priority), and CPU burst and I/O burst time are given below:

Process No.	Arrival Time	Priority	CPU	I/O	CPU
P1	0	2	1	5	3
P2	2	3	3	3	1
P3	3	1	2	3	1
P4	3	4	2	4	1

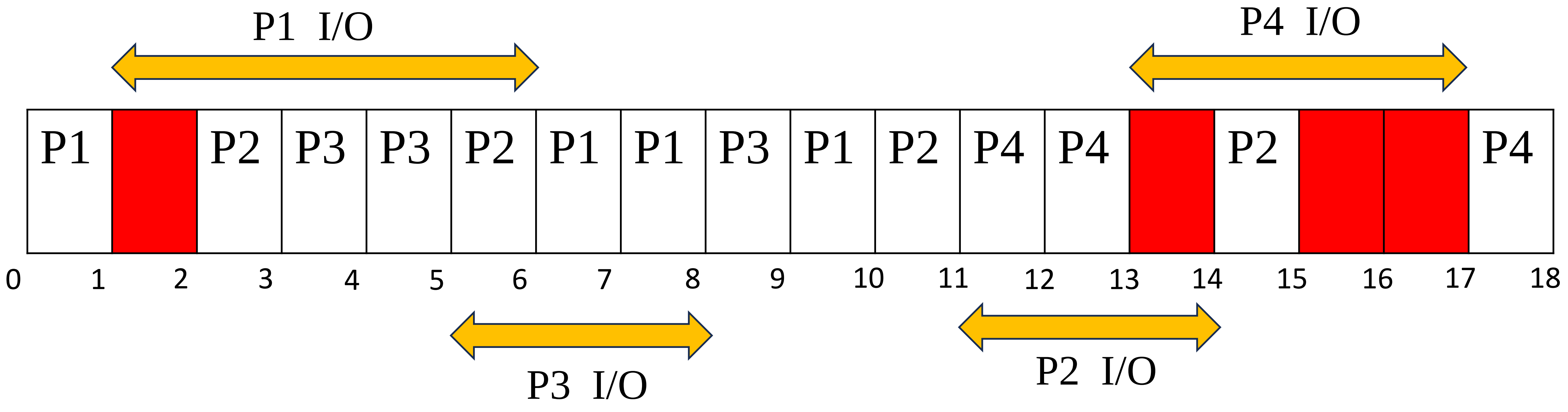
Find the completion time of P1, P2, P3, P4.

Process No.	Arrival Time	Priority	CPU	I/O	CPU
P1	0	2	1	5	3
P2	2	3	3	3	1
P3	3	1	2	3	1
P4	3	4	2	4	1

Gantt Chart



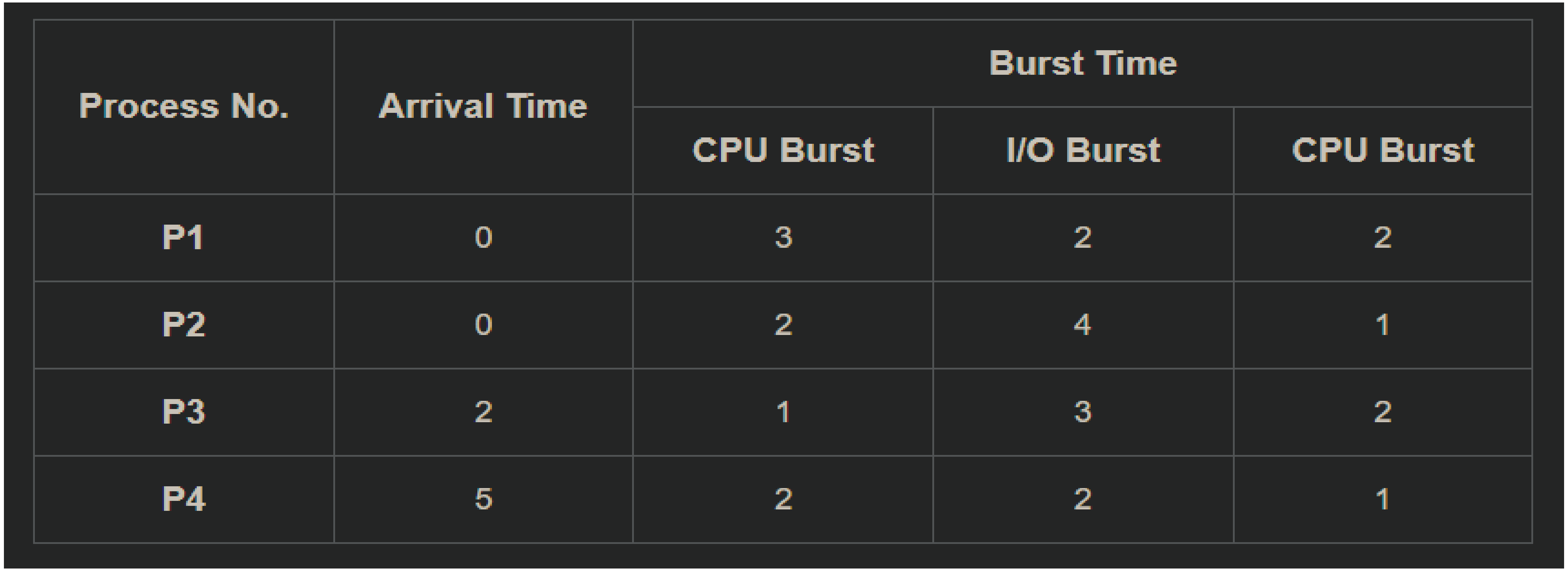
Gantt Chart



Process No.	Arrival Time	Priority	CPU	I/O	CPU	CT
P1	0	2	1	5	3	10
P2	2	3	3	3	1	15
P3	3	1	2	3	1	9
P4	3	4	2	4	1	18

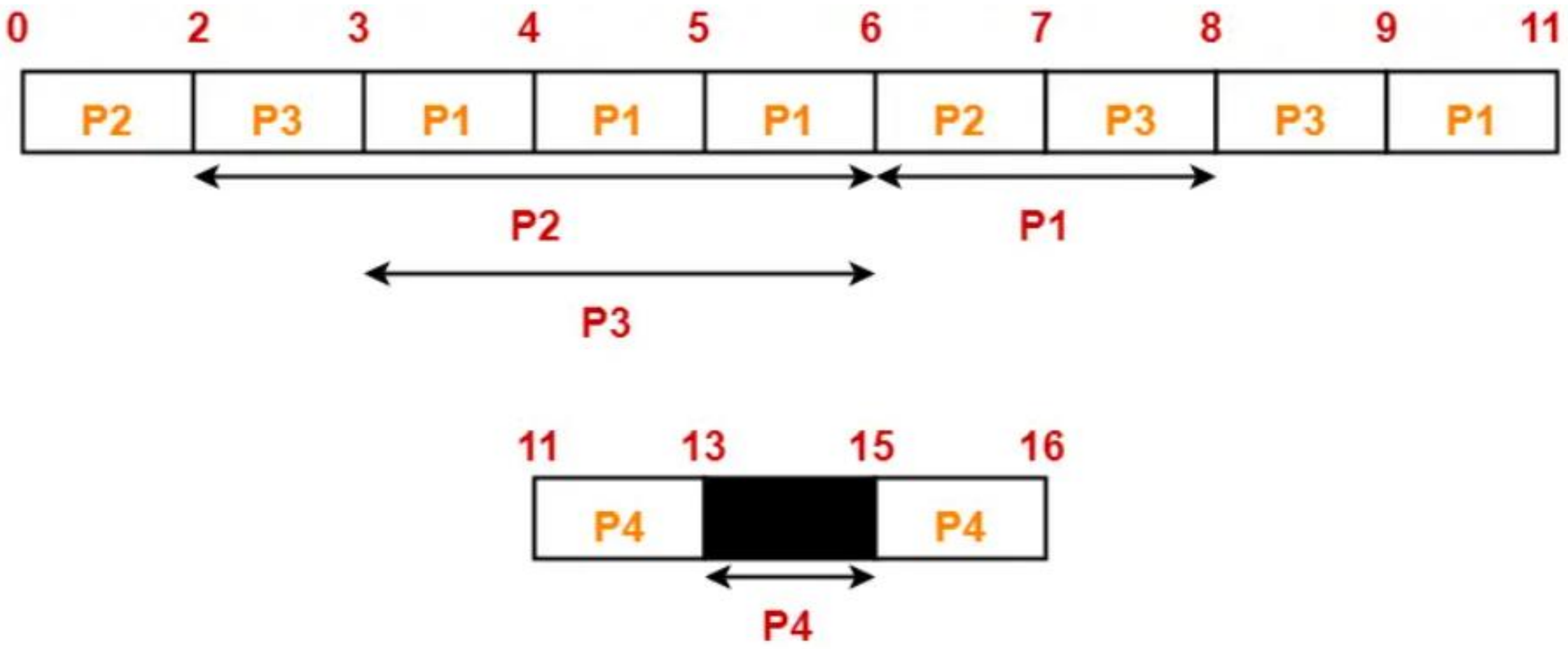
Question 2:

Consider the set of 4 processes whose arrival time and burst time are given below-:

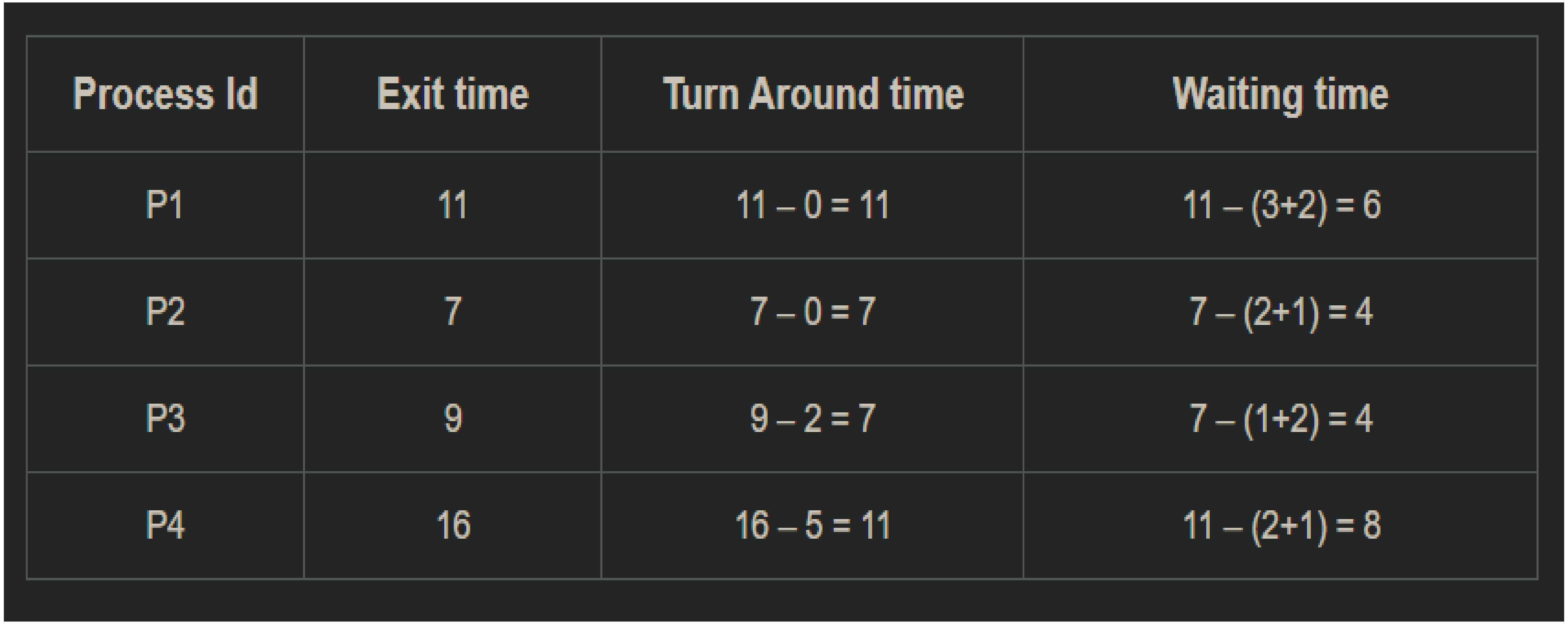


Process No.	Arrival Time	Burst Time		
		CPU Burst	I/O Burst	CPU Burst
P1	0	3	2	2
P2	0	2	4	1
P3	2	1	3	2
P4	5	2	2	1

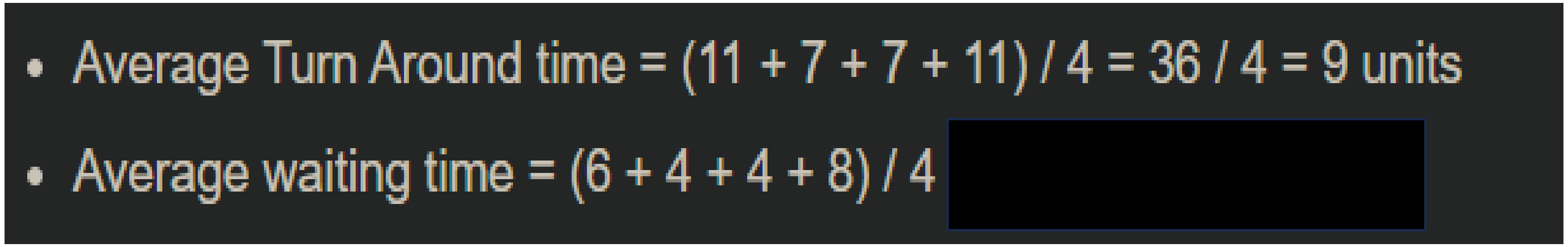
If the CPU scheduling policy is Shortest Remaining Time First, calculate the average waiting time and average turn around time.



Gantt Chart



Process Id	Exit time	Turn Around time	Waiting time
P1	11	11 – 0 = 11	11 – (3+2) = 6
P2	7	7 – 0 = 7	7 – (2+1) = 4
P3	9	9 – 2 = 7	7 – (1+2) = 4
P4	16	16 – 5 = 11	11 – (2+1) = 8



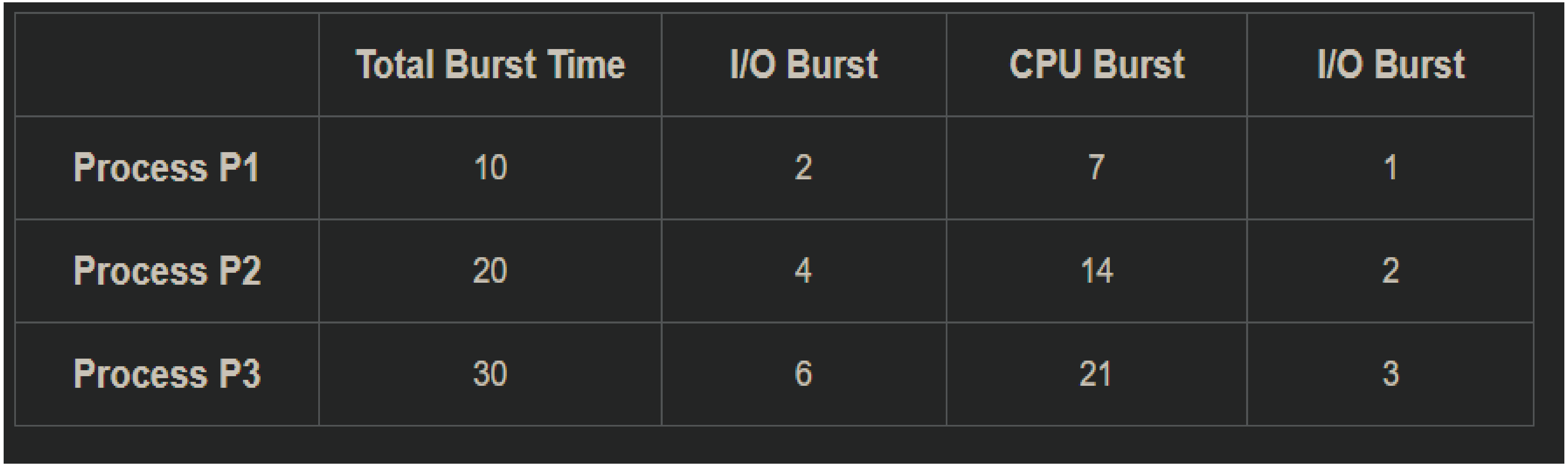
- Average Turn Around time = (11 + 7 + 7 + 11) / 4 = 36 / 4 = 9 units
- Average waiting time = (6 + 4 + 4 + 8) / 4 [redacted]

Question 3:

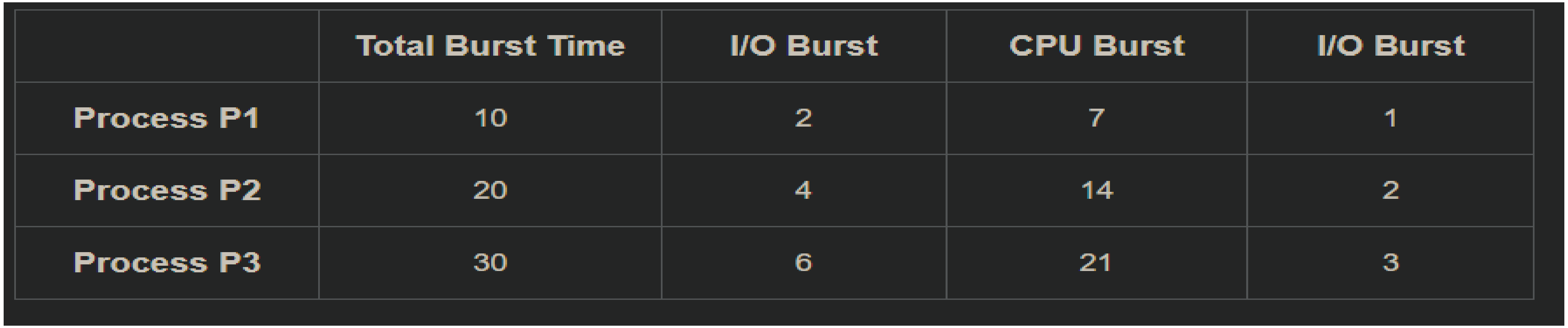
Consider three process, all arriving at time zero, with total execution time of 10, 20 and 30 units respectively. Each process spends the first 20% of execution time doing I/O, the next 70% of time doing computation, and the last 10% of time doing I/O again. The operating system uses a shortest remaining compute time first scheduling algorithm and schedules a new process either when the running process gets blocked on I/O or when the running process finishes its compute burst. Assume that all I/O operations can be overlapped as much as possible. For what percentage of does the CPU remain idle?

Consider three process, all arriving at time zero, with total execution time of 10, 20 and 30 units respectively. Each process spends the first 20% of execution time doing I/O, the next 70% of time doing computation, and the last 10% of time doing I/O again. The operating system uses a shortest remaining compute time first scheduling algorithm and schedules a new process either when the running process gets blocked on I/O or when the running process finishes its compute burst. Assume that all I/O operations can be overlapped as much as possible. For what percentage of does the CPU remain idle?

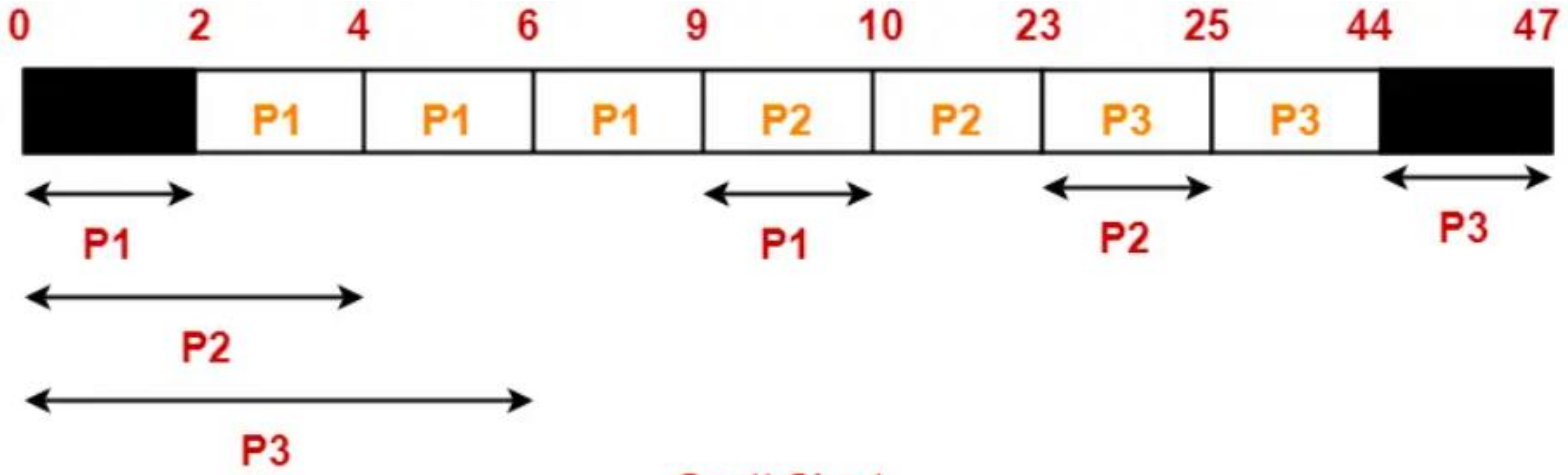
Solution:



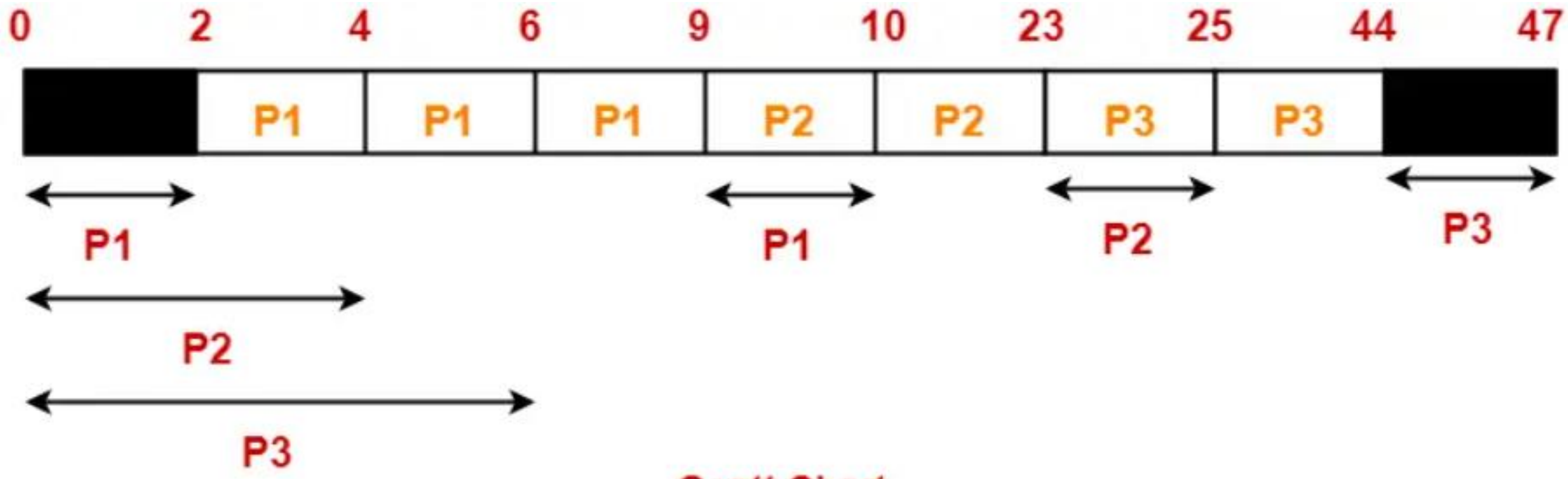
	Total Burst Time	I/O Burst	CPU Burst	I/O Burst
Process P1	10	2	7	1
Process P2	20	4	14	2
Process P3	30	6	21	3



	Total Burst Time	I/O Burst	CPU Burst	I/O Burst
Process P1	10	2	7	1
Process P2	20	4	14	2
Process P3	30	6	21	3



Gantt Chart



Gantt Chart

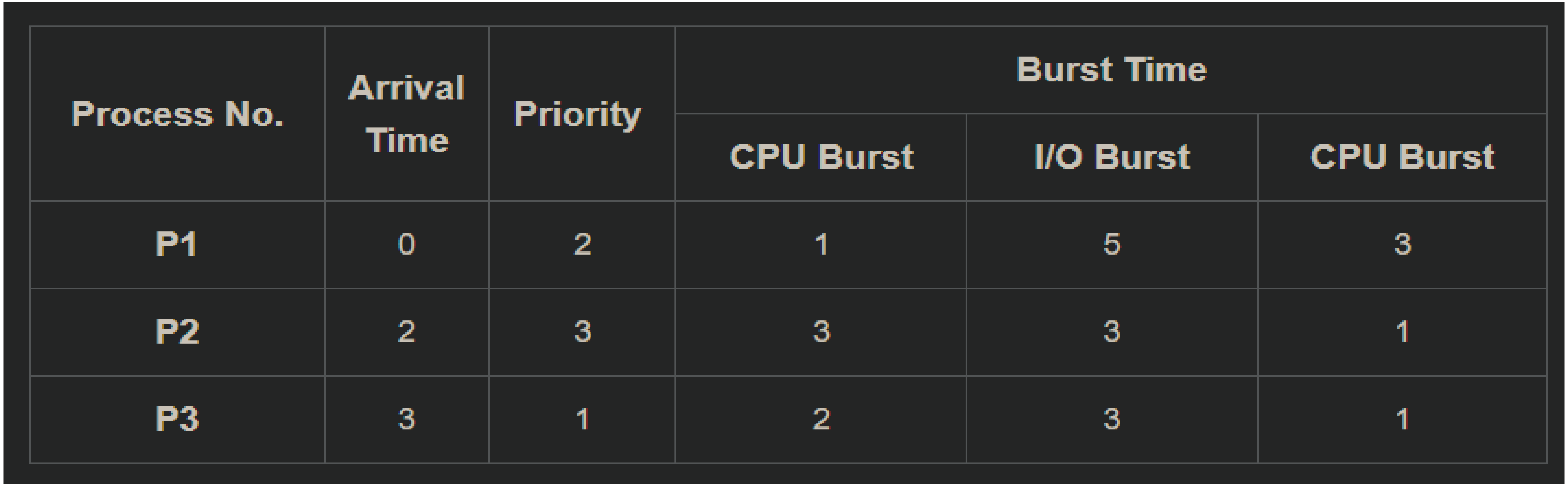
Percentage of time CPU remains idle

= (5 / 47) x 100

= 10.638%

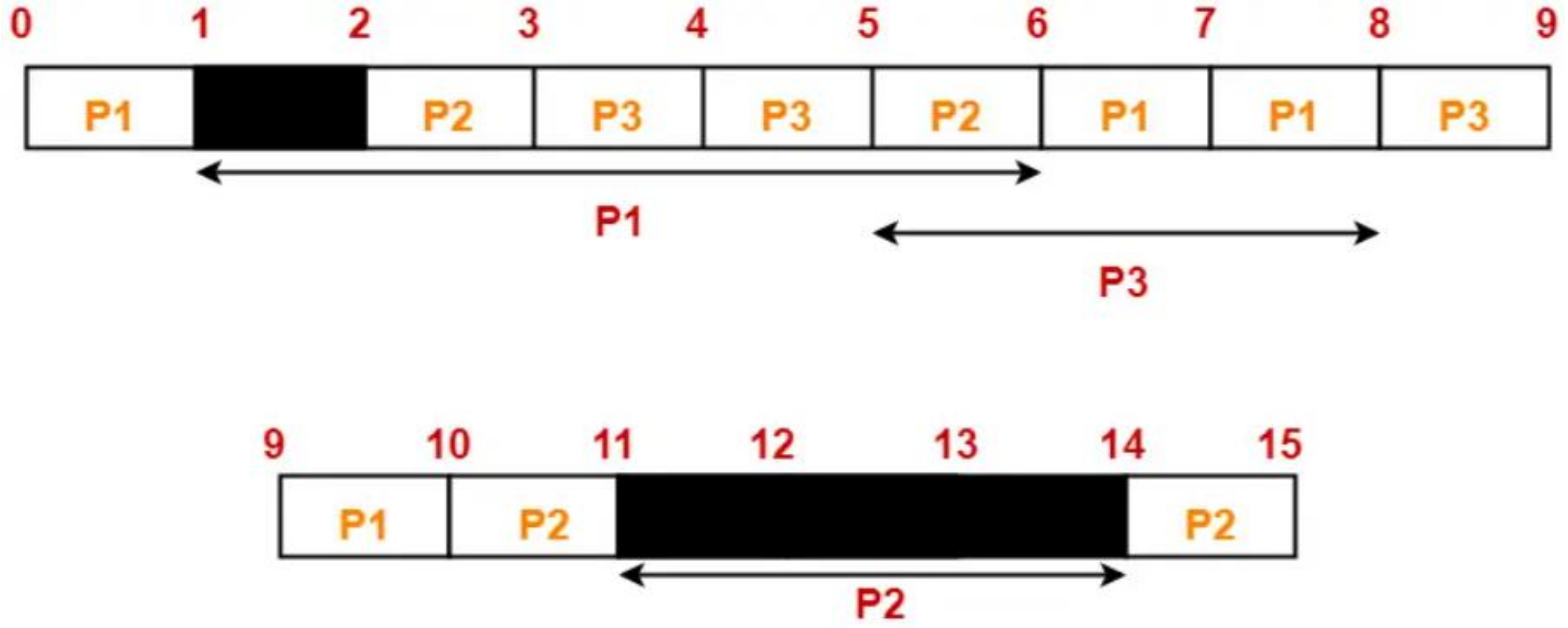
Question 4:

Consider the set of 4 processes whose arrival time and burst time are given below-

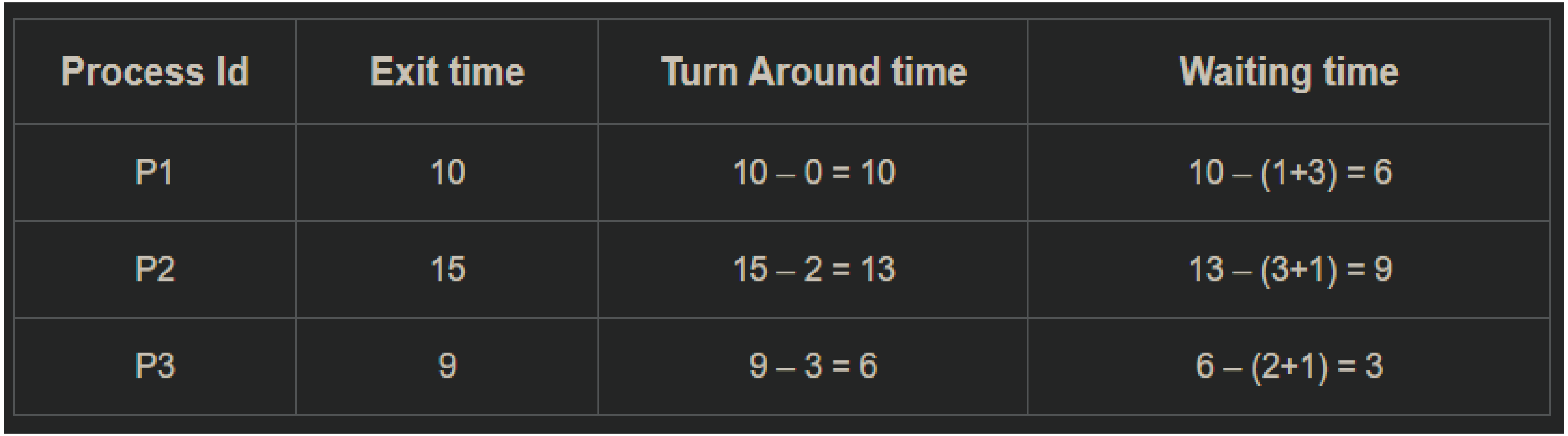


Process No.	Arrival Time	Priority	Burst Time		
			CPU Burst	I/O Burst	CPU Burst
P1	0	2	1	5	3
P2	2	3	3	3	1
P3	3	1	2	3	1

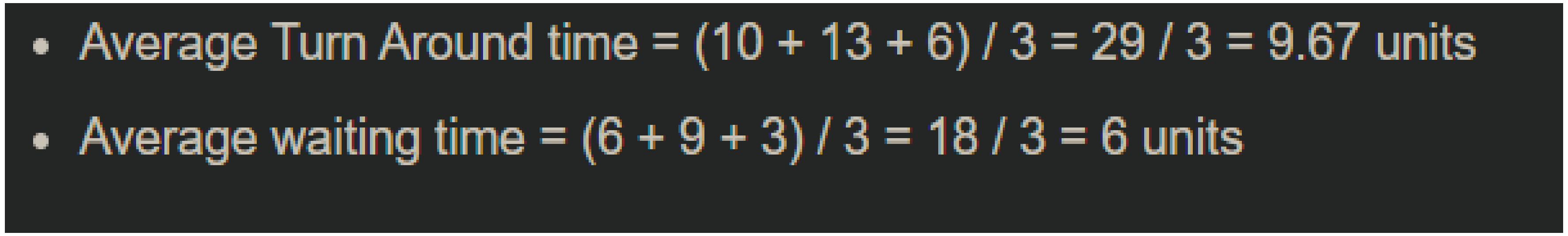
If the CPU scheduling policy is Priority Scheduling, calculate the average waiting time and average turn around time. (Lower number means higher priority)


0
1
2
3
4
5
6
7
8
9
P1
P2
P3
P3
P2
P1
P1
P3
P1
P3
9
10
11
12
13
14
15
P1
P2
P2
P2

Gantt Chart



Process Id	Exit time	Turn Around time	Waiting time
P1	10	10 – 0 = 10	10 – (1+3) = 6
P2	15	15 – 2 = 13	13 – (3+1) = 9
P3	9	9 – 3 = 6	6 – (2+1) = 3



- Average Turn Around time = (10 + 13 + 6) / 3 = 29 / 3 = 9.67 units
- Average waiting time = (6 + 9 + 3) / 3 = 18 / 3 = 6 units

Multilevel Queue Scheduling

- All the scheduling algorithms we saw until now, we just take one ready queue there. And, in ready queue, out of all the processes, some processes based on scheduling algorithms are picked and executed on CPU. In real time, process can be of different type.

- With both priority and round-robin scheduling, all processes may be placed in a single queue, and the scheduler then selects the process with the highest priority to run.

- In practice, it is often easier to have separate queues for each distinct priority, and priority scheduling simply schedules the process in the highest-priority queue. This approach—known as **multilevel queue.**

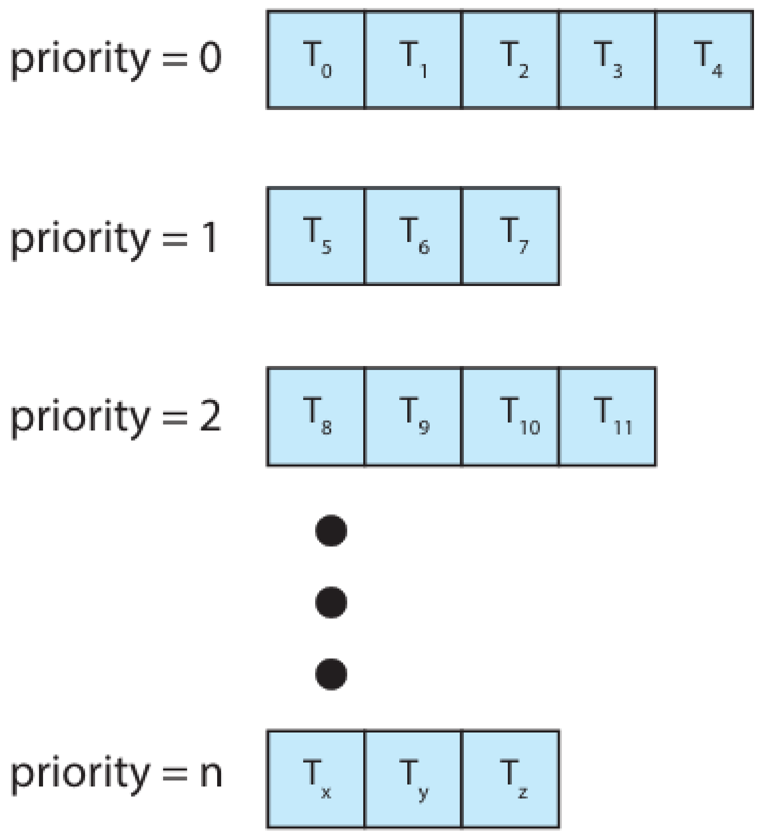


Fig. Separate queues for each priority

- **Multilevel Queue Scheduling is a CPU scheduling mechanism where the process is divided into several hierarchy queues and each queue possesses a different priority, and process type.**

- A multilevel queue scheduling algorithm can also be used to partition processes into several separate queues based on the process type (as shown in the Fig.). For example, a common division is made between **foreground** (interactive) processes and **background** (batch) processes.

- These two types of processes have different response-time requirements and so may have different scheduling needs. In addition, foreground processes may have priority (externally defined) over background processes.

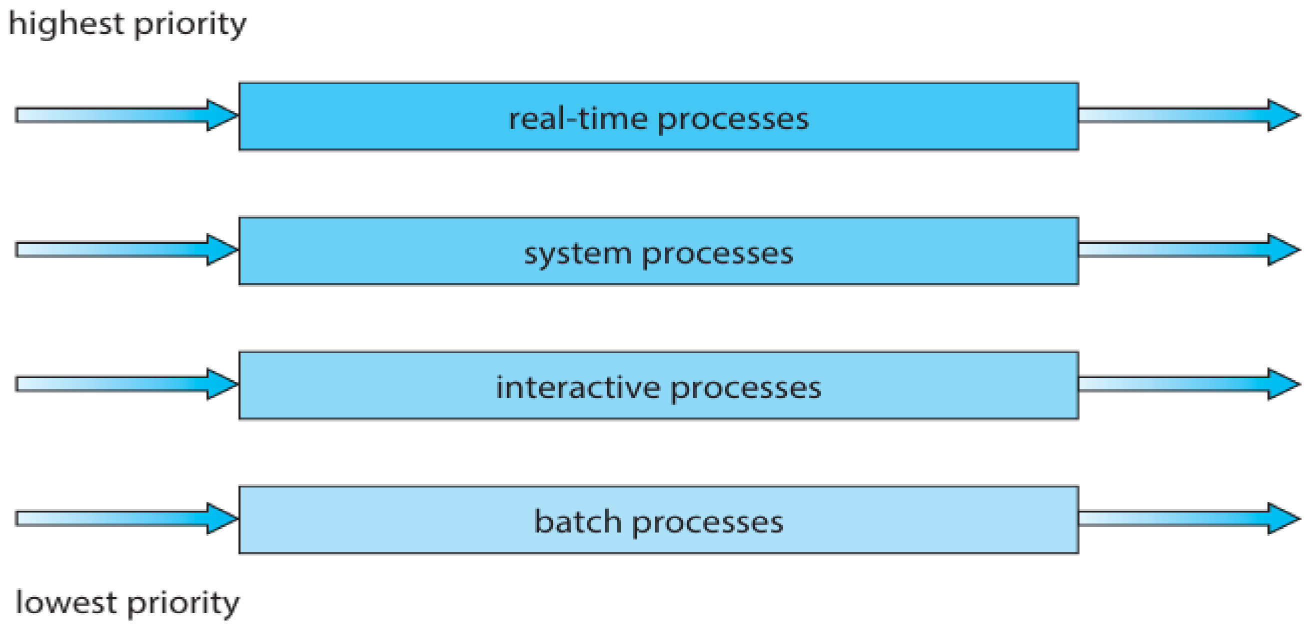


Fig. Multilevel queue scheduling

- Separate queues might be used for foreground and background processes, and each queue might have its own scheduling algorithm. The foreground queue might be scheduled by an RR algorithm, for example, while the background queue is scheduled by an FCFS algorithm.

- In addition, there must be scheduling among the queues, which is commonly implemented as fixed-priority preemptive scheduling.

- A multilevel queue scheduling algorithm with four queues, listed below in order of priority:

  1. **Real-time processes**
  2. **System processes**
  3. **Interactive processes**
  4. **Batch processes**

- **Each queue has absolute priority over lower-priority queues.** No process in the batch queue, for example, could run unless the queues for real-time processes, system processes, and interactive processes were all empty. If an interactive process entered the ready queue while a batch process was running, the batch process would be preempted.

Multilevel Queue Scheduling (Contd.)

Advantage:

- You can use multilevel queue scheduling to apply different scheduling methods to distinct processes.
- It will have low overhead in terms of scheduling.

Disadvantage:

- There is a risk of starvation for lower priority processes.
- It is rigid in nature.

Multilevel Feedback Queue Scheduling

- Normally, when the multilevel queue scheduling algorithm is used, processes are permanently assigned to a queue when they enter the system. If there are separate queues for foreground and background processes, for example, processes do not move from one queue to the other, since processes do not change their foreground or background nature. This setup has the advantage of low scheduling overhead, **but it is inflexible**.

- The **multilevel feedback queue scheduling** algorithm, in contrast, **allows a process to move between queues.**

- The idea is to separate processes according to the characteristics of their CPU bursts. If a process uses too much CPU time, it will be moved to a lower-priority queue.

- This scheme leaves I/O-bound and interactive processes—which are typically characterized by short CPU bursts —in the higher-priority queues.

- In addition, a process that waits too long in a lower-priority queue may be moved to a higher-priority queue. **This form of aging prevents starvation**.

Multilevel Feedback Queue Scheduling (Contd.)

- For example, consider a multilevel feedback queue scheduler with three queues, numbered from 0 to 2 (as shown in Fig.)

- The scheduler first executes all processes in queue 0. Only when queue 0 is empty will it execute processes in queue 1. Similarly, processes in queue 2 will be executed only if queues 0 and 1 are empty.

- A process that arrives for queue 1 will preempt a process in queue 2. A process in queue 1 will in turn be preempted by a process arriving for queue 0.

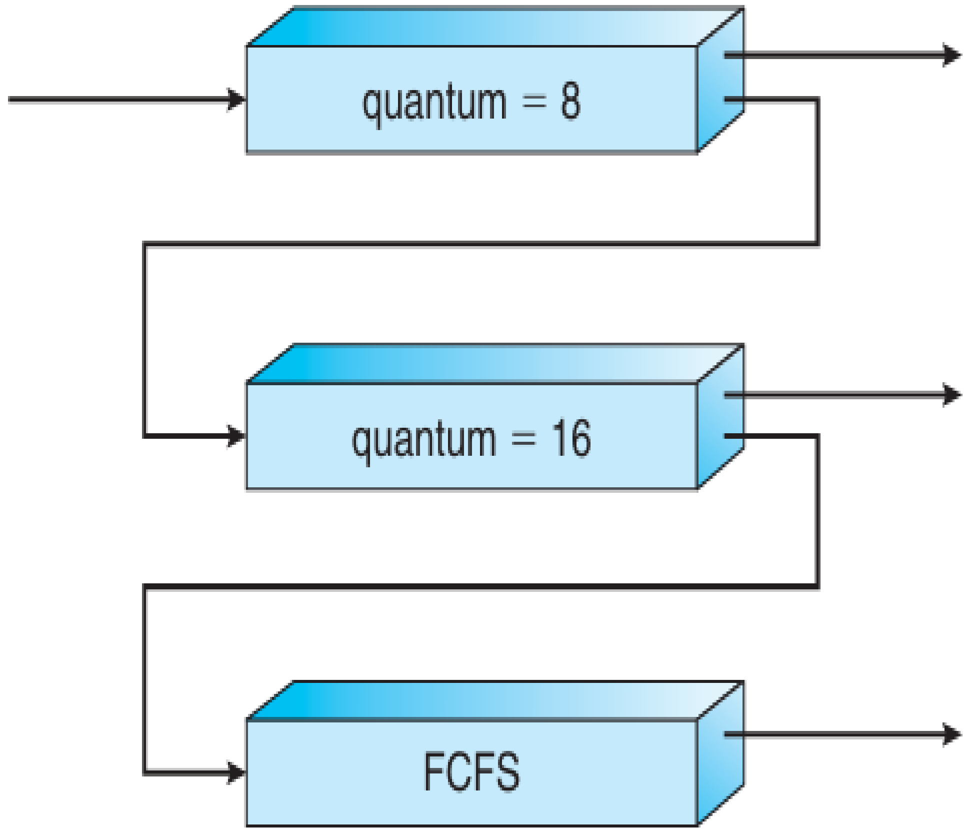


Fig. Multilevel feedback queues

Multilevel Feedback Queue Scheduling (Contd.)

- A process in queue 0 is given a time quantum of 8 milliseconds. If it does not finish within this time, it is moved to the tail of queue 1.
- If queue 0 is empty, the process at the head of queue 1 is given a quantum of 16 milliseconds. If it does not complete, it is preempted and is put into queue 2.
- A process that arrives for queue 1 will preempt a process in queue 2. A process in queue 1 will in turn be preempted by a process arriving for queue 0.
- Processes in queue 2 are run on an FCFS basis but are run only when queues 0 and 1 are empty.
- To prevent starvation, a process that waits too long in a lower-priority queue may gradually be moved to a higher-priority queue.

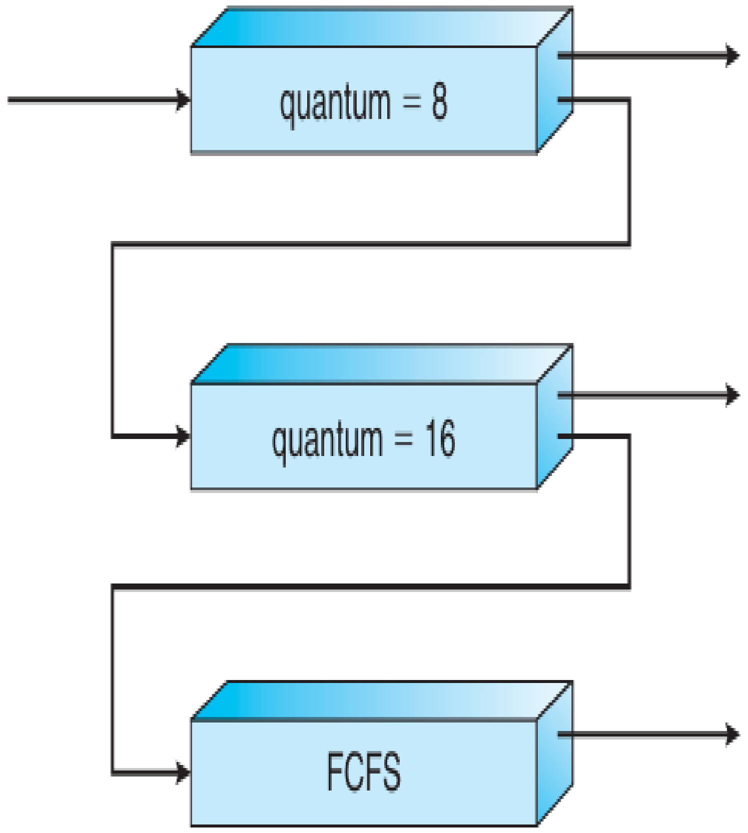


Fig. Multilevel feedback queues

Multilevel Feedback Queue Scheduling (Contd.)

In general, a multilevel feedback queue scheduler is defined by the following parameters:

- The number of queues
- The scheduling algorithm for each queue
- The method used to determine when to upgrade a process to a higher priority queue
- The method used to determine when to demote a process to a lower priority queue
- The method used to determine which queue a process will enter when that process needs service

References

1. Abraham Silberschatz, Peter Baer Galvin, Greg Gagne, “Operating System Concepts,” Eleventh Edition (Willey).

2. Andrew S. Tanenbaum, “Modern Operating Systems”, Fourth Edition (Pearson Publications), 2014.

3. https://www.geeksforgeeks.org/

4. https://www.javatpoint.com/

5. https://www.tutorialspoint.com/

6. https://www.nesoacademy.org/

7. https://www.baeldung.com/