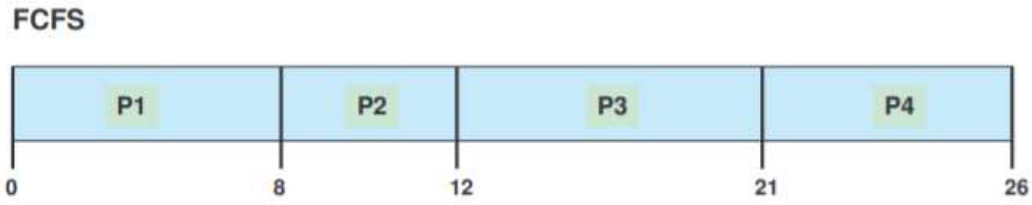
Unit-II
Lecture: 10
(Scheduling)
(Part-II)

Gantt chart:

A Gantt chart is a horizontal bar chart used to represent operating systems CPU scheduling in graphical view that help to plan, coordinate and track specific CPU utilization factor like throughput, waiting time, turnaround time etc.

In general, a Gantt chart illustrates a project schedule. This chart lists the tasks to be performed on the vertical axis and time intervals on the horizontal axis. The width of the horizontal bars in the graph shows the duration of each activity.

e.g. Gantt Chart for 4 Processes P1,P2,P3,P4 having CPU time of 8,4,9 and 5 respectively.



First Come First Serve (FCFS) Scheduling Algorithm:

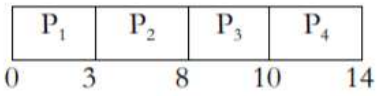
- In this, the process **which requests the CPU first is allocated to the CPU first.**
- The implementation of FCFS algorithm is easily managed with a FIFO queue.
- The average waiting time under FCFS policy is quiet long.
- The FCFS algorithm is non preemptive means once the CPU has been allocated to a process then the process keeps the CPU until the release of the CPU either by terminating or requesting I/O.

Consider the following example:

Process	CPU time
P_1	3
P_2	5
P_3	2
P_4	4

Using FCFS algorithm find the average waiting time and average turnaround time if the order is P_1, P_2, P_3, P_4.

Solution: If the process arrived in the order P_1, P_2, P_3, P_4 then according to the FCFS the Gantt chart will be:



The waiting time for process $P_1 = 0$, $P_2 = 3$, $P_3 = 8$, $P_4 = 10$ then the turnaround time for process $P_1 = 0 + 3 = 3$, $P_2 = 3 + 5 = 8$, $P_3 = 8 + 2 = 10$, $P_4 = 10 + 4 = 14$.

Then average waiting time = $(0 + 3 + 8 + 10)/4 = 21/4 = 5.25$

Average turnaround time = $(3 + 8 + 10 + 14)/4 = 35/4 = 8.75$

FCFS can also block the system in a busy dynamic system in another way, known as the convoy effect.

Convoy Effect: ***Convoy effect is phenomenon associated with the First Come First Serve (FCFS) algorithm, in which the whole Operating System slows down due to few slow processes.***

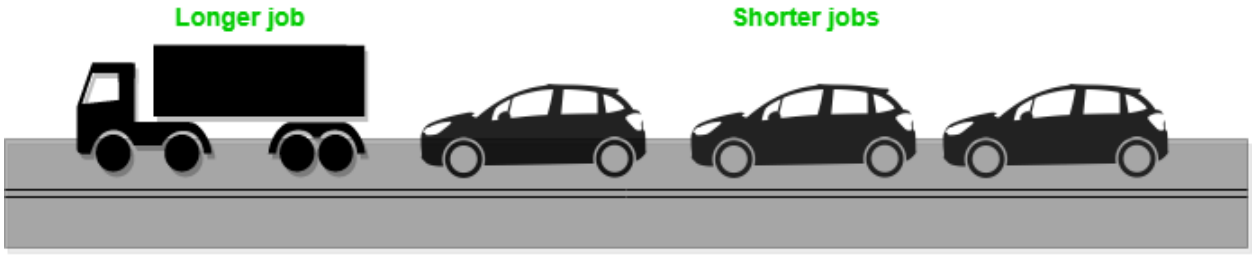


Figure - The Convey Effect, Visualized

FCFS algorithm is non-preemptive in nature, that is, once CPU time has been allocated to a process, other processes can get CPU time only after the current process has finished. This property of FCFS scheduling leads to the situation called Convoy Effect.

Suppose there is one CPU intensive (large burst time) process in the ready queue, and several other processes with relatively less burst times but are Input/Output (I/O) bound (Need I/O operations frequently).

Steps are as following below:

- The I/O bound processes are first allocated CPU time. As they are less CPU intensive, they quickly get executed and go to I/O queues.
- Now, the CPU intensive process is allocated CPU time. As its burst time is high, it takes time to complete.
- While the CPU intensive process is being executed, the I/O bound processes complete their I/O operations and are moved back to ready queue.
- However, the I/O bound processes are made to wait as the CPU intensive process still hasn't finished. **This leads to I/O devices being idle.**
- When the CPU intensive process gets over, it is sent to the I/O queue so that it can access an I/O device.
- Meanwhile, the I/O bound processes get their required CPU time and move back to I/O queue.
- However, they are made to wait because the CPU intensive process is still accessing an I/O device. As a result, **the CPU is sitting idle now**.

Hence in Convoy Effect, one slow process slows down the performance of the entire set of processes, and leads to wastage of CPU time and other devices.

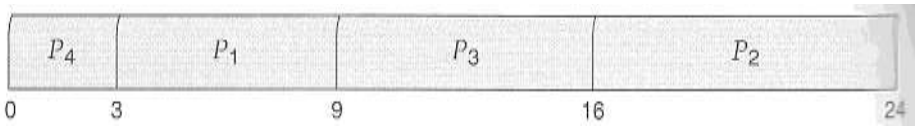
To avoid Convoy Effect, preemptive scheduling algorithms like Round Robin Scheduling can be used – as the smaller processes don't have to wait much for CPU time – making their execution faster and leading to less resources sitting idle.

Shortest Job First (SJF) Scheduling:

- Process which have the shortest burst time are scheduled first. If two processes have the same burst time then FCFS is used to break the tie.
- The idea behind the SJF algorithm is to pick the quickest fastest little job that needs to be done, get it out of the way first and then pick the next smallest fastest job to do next.
    - Technically, this algorithm picks a process based on the next shortest CPU burst, not the overall process time.

e.g. the Gantt chart below is based upon the following CPU burst times (assuming all jobs arrive at the same time):

Process	Burst Time
P1	6
P2	8
P3	7
P4	3



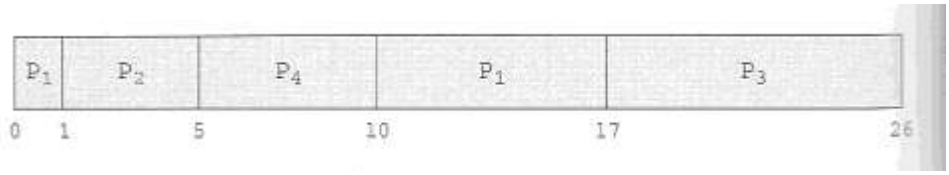
Average wait time is: (0+3+9+16)/4 = 7.0 ms

- **SJF can be either preemptive or non-preemptive.**
    - **Preemptive SJF is referred to as shortest remaining time first scheduling.**
    - Preemption occurs when a new process arrives in the ready queue that has a predicted burst time shorter than the time remaining in the process whose burst is currently on the CPU.

e.g. Consider the following data:

Process	Arrival Time	Burst Time
P1	0	8
P2	1	4
P3	2	9
p4	3	5

The Gantt chart is as below:



The average waiting time is: ((10-1)+(17-2)+(5-3))/4 = 26/4 = 6.5 ms

SJF is optimal as it gives minimum average waiting time for a given set of processes.

But it is not practical as its difficult to predict burst time i.e. How do you know how long the next CPU burst is going to be? i.e. learning to predict the future.