SNS COLLEGE OF ENGINEERING

Kurumbapalayam(Po), Coimbatore – 641 107

An Autonomous Institution

Accredited by NAAC-UGC with 'A' Grade

Approved by AICTE, Recognized by UGC & Affiliated to Anna University, Chennai

DEPARTMENT OF ARTIFICIAL INTELLIGENCE & DATA SCIENCE

Course Code and Name : 23ITB203 / Principles of Operating Systems

II YEAR / IV SEMESTER

Unit 2 : Scheduling Criteria & Algorithms

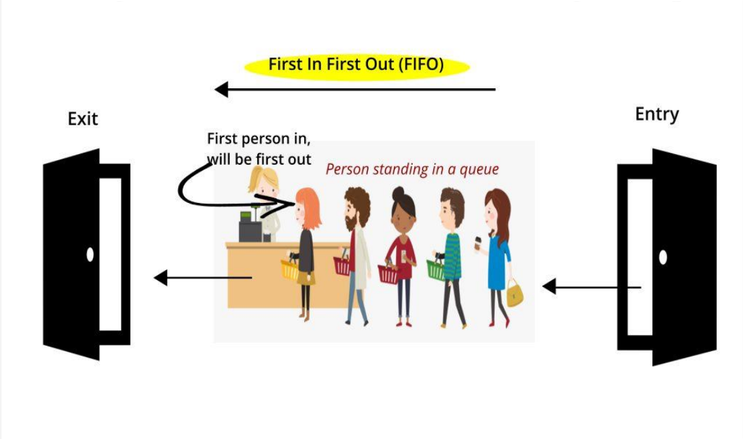
Scheduling Criteria

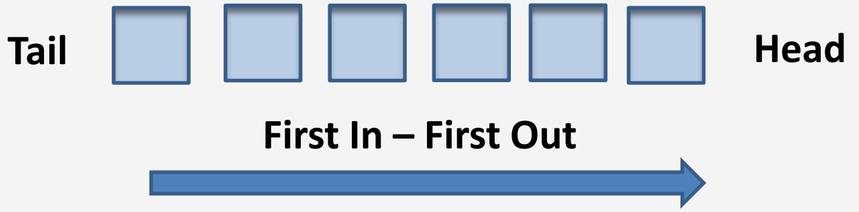
- CPU utilization
- Throughput
- Turnaround Time
- Waiting Time
- Response Time

Scheduling Algorithms

(First-Come, First-Served Scheduling)

- By far the simplest scheduling algorithm
- The process that requests the CPU first is allocated the CPU first.
- The implementation of FCFS policy is easily managed with a FIFO queue.





- When a process enters the ready queue, its PCB is linked onto the tail of the queue.
- When the CPU is free, it is allocated to the process at the head of the queue.
- The running process is then removed from the queue.

The average waiting time under the FCFS policy, is often quite long.
Consider the following set of processes that arrive at time 0.

Process	Burst Time (ms)
P1	24
P2	3
P3	3

If the process arrives in the order P1, P2, P3, and are served in FCFS order, we get the result shown in the following Gantt chart.

Waiting time for P1 = 0 ms
Waiting time for P2 = 24 ms
Waiting time for P3 = 27 ms

Average waiting time =(0+24+27)/3=17ms

If the process come in the order P2, P3, P1, however the result will be shown in the following Gantt chart:

Waiting time for P1 = 6 ms
Waiting time for P2 = 0 ms
Waiting time for P3 = 3 ms

Average waiting time = (6+0+3)/3 = 3 ms

This reduction is substantial. Thus, the average waiting time under an FCFS policy is generally not minimal and may vary substantially if the process's CPU burst times vary greatly.

The FCFS scheduling algorithm is nonpreemptive

- Once the CPU has been allocated to a process, that process keeps the CPU until it releases the CPU, either by terminating or requesting I/O.
- The FCFS algorithm is troublesome for time-sharing systems, where it is important that each user get a share of the CPU at regular intervals.

Scheduling Algorithms

(Shortest-Job-First Scheduling)

- This algorithm associates with each process the length of the process's next CPU burst.
- When the CPU is available, it is assigned to the process that has the smallest next CPU burst.
- If the next CPU bursts of two processes are the same, FCFS scheduling is used to break the tie.

The SJF algorithm can be either **preemptive** or **nonpreemptive**

A more appropriate term for this scheduling method would be the

Shortest-Next-CPU-Burst Algorithm

because scheduling depends on the length of the next CPU burst of a process, rather than its total length.

Example of SJF Scheduling (Non-Preemptive)

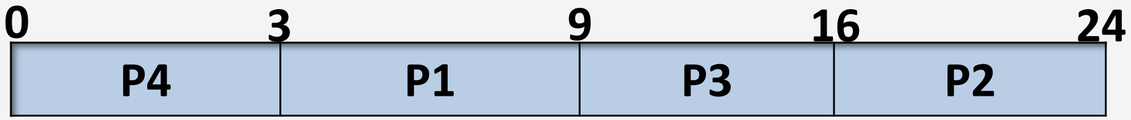
Consider the following set of processes, with the length of the CPU burst given in milliseconds:

Process ID	Burst Time (ms)
P1	6
P2	8
P3	7
P4	3

Waiting time for **P1** = 3 ms
Waiting time for **P2** = 16 ms
Waiting time for **P3** = 9 ms
Waiting time for **P4** = 0 ms

Average waiting Time
= (3+16+9+0)/4 = **7 ms**

Gantt Chart:



By comparison, if we were using the FCFS scheduling scheme, the average waiting time would be 10.25 milliseconds

Example of SJF Scheduling (Preemptive)

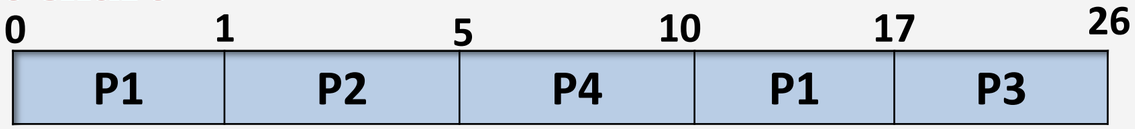
Consider the following four processes, with the length of the CPU burst given in milliseconds and the processes arrive at the ready queue at the times shown:

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

Waiting time for **P1** = (10-1-0) = 9 ms
Waiting time for **P2** = (1-0-1) = 0 ms
Waiting time for **P3** = (17-0-2) = 15 ms
Waiting time for **P4** = (5-0-3) = 2 ms

Average waiting Time
= (9+0+15+2)/4 = **6.5 ms**

Gantt chart



Waiting time = **Total waiting time** – **No. of milliseconds process executed** – **Arrival Time**
Preemptive SJF scheduling is sometimes called **Shortest Remaining Time First Scheduling**

Problems with SJF Scheduling

The real difficulty with the SJF algorithm is knowing the length of the next CPU request.
Although the SJF algorithm is optimal, it cannot be implemented at the level of short-term CPU
There is no way to know the length of the next CPU burst.

One approach is

- To try to approximate SJF Scheduling
- We may not know the length of the next CPU burst, but we may be able to predict its value.
- We expect that the next CPU burst will be similar in length to the previous ones.
- Thus, by computing an approximation of the length of the next CPU burst, we can pick the process with the shortest predicted CPU burst.

Scheduling Algorithms

(Priority Scheduling)

- A priority is associated with each process, and the CPU is allocated to the process with the highest priority.
- Equal priority processes are scheduled in FCFS order.
- An SJF algorithm is simply a priority algorithm where priority is the inverse of the (predicted) next CPU burst. The larger the CPU burst, the lower the priority and vice-versa.

Priority scheduling can be either **preemptive** or **nonpreemptive**

A preemptive priority scheduling algorithm will preempt the CPU if the priority of the newly arrived process is higher than the priority of the current running process.

A nonpreemptive priority scheduling algorithm will simply put the new process at the head of the ready queue.

Consider the following four processes, assumed to have arrived at time 0, in the order P1, P2, P3, P4, P5 with the length of the CPU burst given in milliseconds:

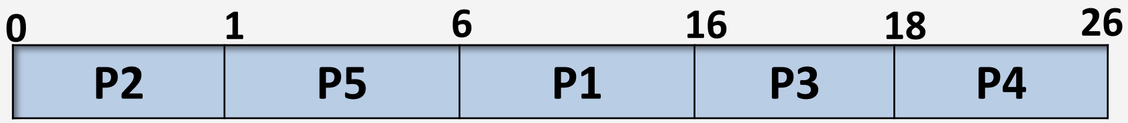
Process ID	Burst Time	Priority
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2

Waiting time for **P1** = 6 ms
Waiting time for **P2** = 0 ms
Waiting time for **P3** = 16 ms
Waiting time for **P4** = 18 ms
Waiting time for **P5** = 1 ms

Average waiting Time

Using priority scheduling, we would schedule these Processes according to the following Gantt chart:

=(6+0+16+18+1)/5
=41/5 = 8.2 ms



Problem with Priority Scheduling

- A major problem with priority scheduling algorithms is **indefinite blocking**, or **starvation**
- A process that is ready to run but waiting for the CPU can be considered blocked.
- A priority scheduling algorithm can leave some low priority processes waiting indefinitely
- In a heavily loaded computer system, a steady stream of higher-priority processes can prevent a low-priority process from ever getting the CPU.

Solution to the Problem

- A solution to the problem of indefinite blockage of low-priority processes is **aging**.
- Aging is a technique of gradually increasing the priority of processes that wait in the system for a long time.
  - For example,
  - If priorities range from 127 (low) to 0 (high), we could increase the priority

Of a waiting process by 1 every 15 minutes.

Eventually, even a process with an initial priority of 127 would have the highest priority in the system and would be executed.

Problem with Priority Scheduling

- A major problem with priority scheduling algorithms is **indefinite blocking**, or **starvation**
- A process that is ready to run but waiting for the CPU can be considered blocked.
- A priority scheduling algorithm can leave some low priority processes waiting indefinitely
- In a heavily loaded computer system, a steady stream of higher-priority processes can prevent a low-priority process from ever getting the CPU.

Solution to the Problem

- A solution to the problem of indefinite blockage of low-priority processes is **aging**.
- Aging is a technique of gradually increasing the priority of processes that wait in the system for a long time.
  - For example,
  - If priorities range from 127 (low) to 0 (high), we could increase the priority

Of a waiting process by 1 every 15 minutes.

Eventually, even a process with an initial priority of 127 would have the highest priority in the system and would be executed.

Problem with Priority Scheduling

- A major problem with priority scheduling algorithms is **indefinite blocking**, or **starvation**
- A process that is ready to run but waiting for the CPU can be considered blocked.
- A priority scheduling algorithm can leave some low priority processes waiting indefinitely
- In a heavily loaded computer system, a steady stream of higher-priority processes can prevent a low-priority process from ever getting the CPU.

Solution to the Problem

- A solution to the problem of indefinite blockage of low-priority processes is **aging**.
- Aging is a technique of gradually increasing the priority of processes that wait in the system for a long time.
  - For example,
  - If priorities range from 127 (low) to 0 (high), we could increase the priority

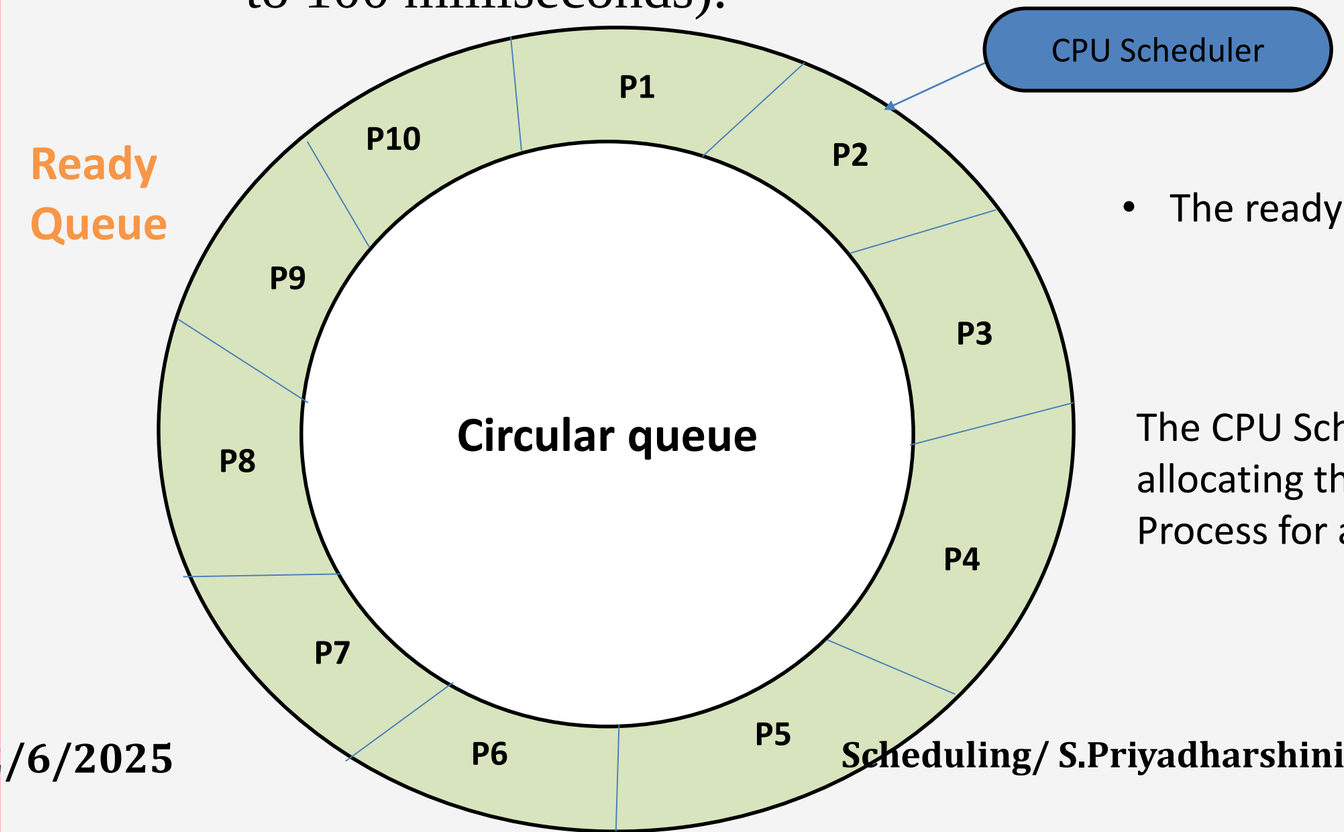
Of a waiting process by 1 every 15 minutes.

Eventually, even a process with an initial priority of 127 would have the highest priority in the system and would be executed.

Scheduling Algorithms

(Round-Robin Scheduling)

- The round-robin (RR) scheduling algorithm is designed especially for time sharing systems.
- It is similar to FCFS scheduling, but preemption is added to switch between processes.
- A small unit of time, called a time quantum or time slice, is defined (generally from 10 to 100 milliseconds).



- The ready queue is treated as Circular queue.

The CPU Scheduler goes around the ready queue, allocating the CPU to each
Process for a time interval of upto 1 time quantum