**CPU Scheduling**

• CPU scheduling is the basis of multi-programmed operating systems. By switching the CPU among processes, the operating system can make the computer more productive.

• **Basic Concepts**

The idea of multiprogramming is relatively simple. A process is executed until it must wait, typically for the completion of some I/O request. In a simple computer system, the CPU would then just sit idle. Scheduling is a fundamental operating-system function. Almost all computer resources are scheduled before use.

• CPU - I/O Burst Cycle

The success of CPU scheduling depends on the following observed property of processes: Process execution consists of a cycle of CPU execution and I/O wait. Processes alternate back and forth between these two states.

• Context Switch

To give each process on a multi-programmed machine a fair share of the CPU, a hardware clock generates interrupts periodically. This allows the operating system to schedule all processes in main memory (using scheduling algorithm) to run on the CPU at equal intervals. Each switch of the CPU from one process to another is called a context switch.

• Preemptive Scheduling

**CPU scheduling decisions may take place under the following four circumstances:**

1. When a process switches from the running state to the waiting state (for. example, I/O request, or invocation of wait for the termination of one of the child processes).

2. When a process switches from the running state to the ready state (for example, when an interrupt occurs).

3. When a process switches from the waiting state to the ready state (for example, completion of I/O).

4. When a process terminates.

Different CPU scheduling algorithms have different properties and may favor one class of processes over another. In choosing which algorithm to use in a particular situation, we must consider the properties of the various algorithms. Many criteria have been suggested for comparing CPU scheduling algorithms. Criteria that are used include the following:

• CPU utilization.

• Throughput.

• Turnaround time.

• Waiting time.

• Response time.

### Goals of Scheduling (objectives)

In this section we try to answer following question: What the scheduler try to achieve?

Many objectives must be considered in the design of a scheduling discipline. In particular, a scheduler should consider fairness, efficiency, response time, turnaround time, throughput, etc., Some of these goals depends on the system one is using for example batch system, interactive system or real-time system, etc. but there are also some goals that are desirable in all systems.

### General Goals

**Fairness**      
        Fairness is important under all circumstances. A scheduler makes sure that each process gets its fair share of the CPU and no process can suffer indefinite postponement. Note that giving equivalent or equal time is not fair. Think of *safety control* and *payroll* at a nuclear plant.

**Policy Enforcement**

        The scheduler has to make sure that system's policy is enforced. For example, if the local policy is safety then the *safety control processes* must be able to run whenever they want to, even if it means delay in *payroll processes*.

**Efficiency**      
        Scheduler should keep the system (or in particular CPU) busy cent percent of the time when possible. If the CPU and all the Input/Output devices can be kept running all the time, more work gets done per second than if some components are idle.

**Response Time**

        A scheduler should minimize the response time for interactive user.

**Turnaround**  
        A scheduler should minimize the time batch users must wait for an output.

**Throughput**  
        A scheduler should maximize the number of jobs processed per unit time.

A little thought will show that some of these goals are contradictory. It can be shown  that any scheduling algorithm that favors some class of jobs hurts another class of jobs. The amount of CPU time available is finite, after all.

### Preemptive Vs Non-preemptive Scheduling

The Scheduling algorithms can be divided into two categories with respect to how they deal with clock interrupts.

**Nonpreemptive Scheduling**

A scheduling discipline is nonpreemptive if, once a process has been given the CPU, the CPU cannot be taken away from that process.

Following are some characteristics of nonpreemptive scheduling

1. In nonpreemptive system, short jobs are made to wait by longer jobs but the overall treatment of all processes is fair.
2. In nonpreemptive system, response times are more predictable because incoming high priority jobs can not displace waiting jobs.
3. In nonpreemptive scheduling, a schedular executes jobs in the following two situations.
   1. When a process switches from running state to the waiting state.
   2. When a process terminates.

**Preemptive Scheduling**

A scheduling discipline is preemptive if, once a process has been given the CPU can taken away.

The strategy of allowing processes that are logically runable to be temporarily suspended is called Preemptive Scheduling and it is contrast to the "run to completion" method.

**CPU Scheduling Algorithms**

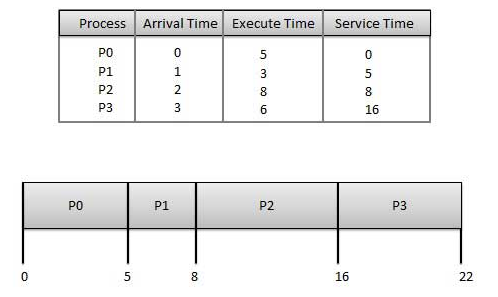
A Process Scheduler schedules different processes to be assigned to the CPU based on particular scheduling algorithms. There are six popular process scheduling algorithms which we are going to discuss in this chapter −

* First-Come, First-Served (FCFS) Scheduling
* Shortest-Job-Next (SJN) Scheduling
* Priority Scheduling
* Shortest Remaining Time
* Round Robin(RR) Scheduling
* Multiple-Level Queues Scheduling

These algorithms are either non-preemptive or preemptive. Non-preemptive algorithms are designed so that once a process enters the running state, it cannot be preempted until it completes its allotted time, whereas the preemptive scheduling is based on priority where a scheduler may preempt a low priority running process anytime when a high priority process enters into a ready state.

## First Come First Serve (FCFS)

* Jobs are executed on first come, first serve basis.
* It is a non-preemptive, pre-emptive scheduling algorithm.
* Easy to understand and implement.
* Its implementation is based on FIFO queue.
* Poor in performance as average wait time is high.

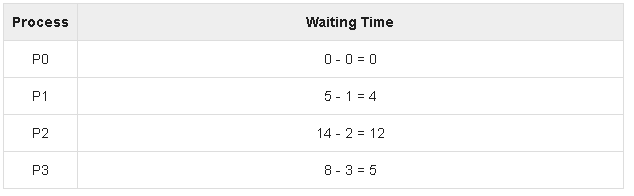


## Shortest Job Next (SJN)

* This is also known as **shortest job first**, or SJF
* This is a non-preemptive, pre-emptive scheduling algorithm.
* Best approach to minimize waiting time.
* Easy to implement in Batch systems where required CPU time is known in advance.
* Impossible to implement in interactive systems where required CPU time is not known.
* The processer should know in advance how much time process will take.

Given: Table of processes, and their Arrival time, Execution time

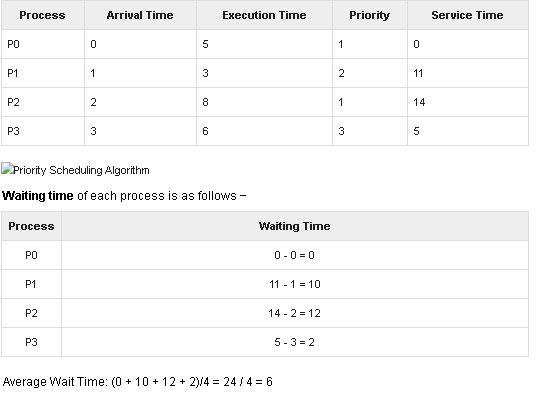




## Priority Based Scheduling

* Priority scheduling is a non-preemptive algorithm and one of the most common scheduling algorithms in batch systems.
* Each process is assigned a priority. Process with highest priority is to be executed first and so on.
* Processes with same priority are executed on first come first served basis.
* Priority can be decided based on memory requirements, time requirements or any other resource requirement.

Given: Table of processes, and their Arrival time, Execution time, and priority. Here we are considering 1 is the lowest priority.

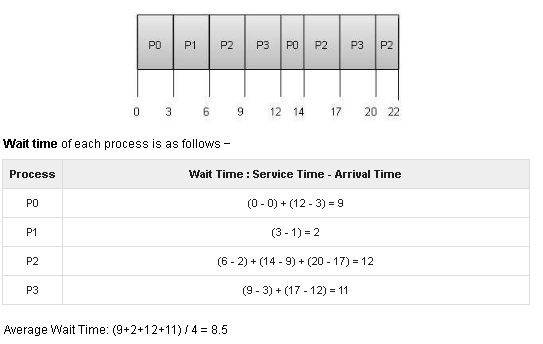


## Shortest Remaining Time

* Shortest remaining time (SRT) is the preemptive version of the SJN algorithm.
* The processor is allocated to the job closest to completion but it can be preempted by a newer ready job with shorter time to completion.
* Impossible to implement in interactive systems where required CPU time is not known.
* It is often used in batch environments where short jobs need to give preference.

## Round Robin Scheduling

* Round Robin is the preemptive process scheduling algorithm.
* Each process is provided a fix time to execute, it is called a **quantum**.
* Once a process is executed for a given time period, it is preempted and other process executes for a given time period.
* Context switching is used to save states of preempted processes.



**Scheduling Criteria**

There are several different criteria to consider when trying to select the "best" scheduling algorithm for a particular situation and environment, including:

* **CPU utilization** - Ideally the CPU would be busy 100% of the time, so as to waste 0 CPU cycles. On a real system CPU usage should range from 40% ( lightly loaded ) to 90% ( heavily loaded. )
* **Throughput**- Number of processes completed per unit time. May range from 10 / second to 1 / hour depending on the specific processes.
* **Turnaround time** - Time required for a particular process to complete, from submission time to completion. ( Wall clock time. )
* **Waiting time** - How much time processes spend in the ready queue waiting their turn to get on the CPU.
* ( **Load average** - The average number of processes sitting in the ready queue waiting their turn to get into the CPU. Reported in 1-minute, 5-minute, and 15-minute averages by "uptime" and "who". )

Video Links: <https://www.youtube.com/watch?v=2h3eWaPx8SA&list=PLBlnK6fEyqRiVhbXDGLXDk_OQAeuVcp2O&index=19>

<https://www.youtube.com/watch?v=4DhFmL-6SDA&list=PLBlnK6fEyqRiVhbXDGLXDk_OQAeuVcp2O&index=38>

<https://www.youtube.com/watch?v=7DoP1L9nAAs&list=PLBlnK6fEyqRiVhbXDGLXDk_OQAeuVcp2O&index=40>

<https://www.youtube.com/watch?v=t0g9b3SJECg&list=PLBlnK6fEyqRiVhbXDGLXDk_OQAeuVcp2O&index=43>

REFERENCES:

1. Galvin, Peter B., Silberchatz, A., “*Operating System Concepts*”, Addison Wesley, 8th Edition.
2. Flynn, *“Operating Systems”*, Cengage Learning.
3. Dhamdhere, D.M., "*Operating System:A Concept Based Approach*",  
   Tata Mc-Graw- Hill.
4. https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-828-operating-system-engineering-fall-2012/lecture-notes-and-readings/
5. **https://www.youtube.com/watch?v=4hCih9eLc7M&list=PL3-wYxbt4yCjpcfUDz-TgD\_ainZ2K3MUZ&index=19**
6. <https://computing.llnl.gov/tutorials/>
7. https://nptel.ac.in/courses/106/105/106105214
8. <https://www.guru99.com/operating-system-tutorial.html>
9. <https://www.geeksforgeeks.org/operating-systems/>