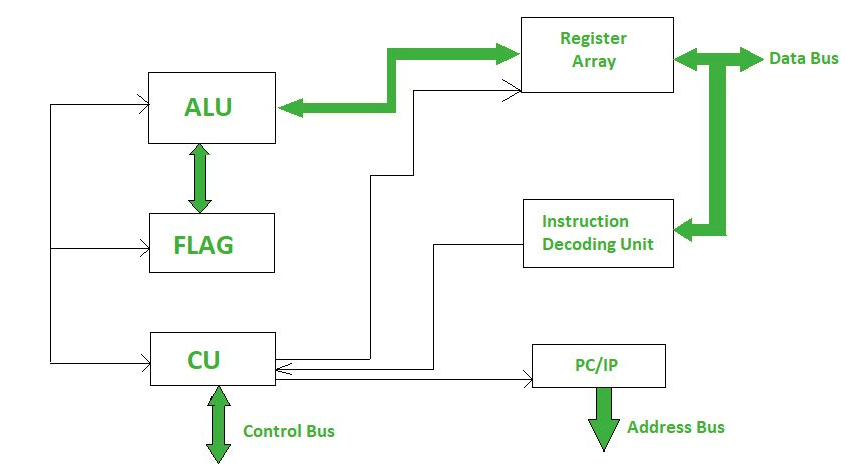
What is a microprocessor?

A Microprocessor is a programmable device that takes in input perform some arithmetic and logical operations over it and produce the desired output. In simple words, a Microprocessor is a digital device on a chip which can fetch instruction from memory, decode and execute them and give results.  
  
**Basics of Microprocessor:**  
A Microprocessor takes a bunch of instructions in machine language and executes them, telling the processor what it has to do. Microprocessor performs three basic things while executing the instruction:

1. It performs some basic operations like addition, subtraction, multiplication, division and some logical operations using its Arithmetic and Logical Unit (ALU). New Microprocessors also perform operations on floating-point numbers also.
2. Data in the Microprocessor can move from one location to another.
3. It has a Program Counter (PC) register that stores the address of next instruction based on the value of PC, Microprocessor jumps from one location to another and takes the decision.

A typical Microprocessor structure looks like this.  
  
[](https://www.geeksforgeeks.org/introduction-of-microprocessor/)

How does compiler compiles the interlinked libraries?

When a [C program is compiled](https://www.geeksforgeeks.org/compiling-a-c-program-behind-the-scenes/), the compiler generates object code. After generating the object code, the compiler also invokes the linker. One of the main tasks for linker is to make code of library functions (eg printf(), scanf(), sqrt(), ..etc) available to your program. A linker can accomplish this task in two ways, by copying the code of library function to your object code, known as **Static Linking**. Or by making some arrangements so that the complete code of library functions is not copied, but made available at run-time, known as **Dynamic Linking**.

* **Static Linking and Static Libraries** is the result of the linker making copy of all used library functions to the executable file. Static Linking creates larger binary files, and need more space on disk and main memory. Examples of static libraries (libraries which are statically linked) are, ***.a*** files in Linux and ***.lib***files in Windows.
* **Dynamic linking and Dynamic Libraries** Dynamic Linking doesn't require the code to be copied, it is done by just placing the name of the library in the binary file. The actual linking happens when the program is run when both the binary file and the library are in memory. Examples of Dynamic libraries (libraries which are linked at run-time) are, ***.so*** in Linux and ***.dll*** in Windows.

Explain the implementation of virtual methods?

A [virtual function](https://www.geeksforgeeks.org/virtual-function-cpp/) a member function which is declared within a base class and is re-defined(Overriden) by a derived class. When you refer to a derived class object using a pointer or a reference to the base class, you can call a virtual function for that object and execute the derived class's version of the function.

* Virtual functions ensure that the correct function is called for an object, regardless of the type of reference (or pointer) used for function call.
* They are mainly used to achieve[Runtime polymorphism](https://www.cdn.geeksforgeeks.org/polymorphism-in-c/)
* Functions are declared with a **virtual**keyword in base class.
* The resolving of function call is done at Run-time.

**Example:**

CPP

*// CPP program to illustrate*

*// concept of Virtual Functions*

#include *<iostream>*

**using** **namespace** **std**;

**class** **base** {

**public**:

**virtual** void print()

{

cout << "print base class" << endl;

}

void show()

{

cout << "show base class" << endl;

}

};

**class** **derived** : **public** base {

**public**:

void print()

{

cout << "print derived class" << endl;

}

void show()

{

cout << "show derived class" << endl;

}

};

int main()

{

base\* bptr;

derived d;

bptr = &d;

*// virtual function, binded at runtime*

bptr->print();

*// Non-virtual function, binded at compile time*

bptr->show();

}

**Output:**

print derived class

show base class

Explain the implementation of Dynamic Binding?

In **Dynamic binding** compiler doesn't decide the method to be called. Overriding is a perfect example of dynamic binding. In overriding both parent and child classes have the same method.  
  
**Let's see an example:**

Java

**public** **class** **NewClass**

{

**public** **static** **class** **superclass**

{

void print()

{

System.out.println("print in superclass.");

}

}

**public** **static** **class** **subclass** **extends** superclass

{

@Override

void print()

{

System.out.println("print in subclass.");

}

}

**public** **static** void main(String[] args)

{

superclass A = **new** superclass();

superclass B = **new** subclass();

A.print();

B.print();

}

}

**Output:**

print in superclass.

print in subclass.

Let's break down the code and understand it thoroughly.

* Methods are not static in this code.
* During compilation, the compiler has no idea as to which print has to be called since compiler goes only by referencing variable not by the type of object and therefore the binding would be delayed to runtime and therefore the corresponding version of the print will be called based on type on object.

Explain Virtual table or vtable?

The idea is that virtual functions are called according to the type of the object instance pointed to or referenced, not according to the type of the pointer or reference.  
  
In other words, virtual functions are resolved late, at runtime. So the compiler maintains two things to perform runtime resolution:

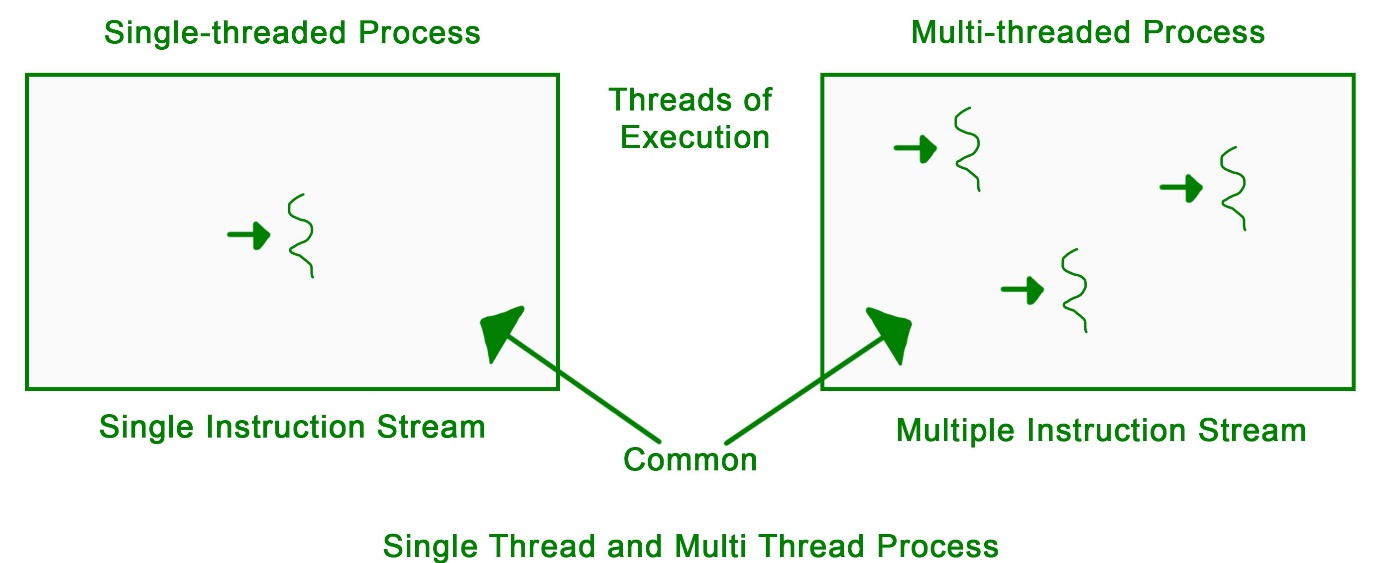
1. **vtable:** vtable is short form for Virtual Function Table. The virtual table is a lookup table of functions used to resolve function calls in a dynamic/late binding manner. The compiler creates a static table per class and the data consists of pointers to the virtual function definitions. They are automatically initialized by the compiler's constructor code. Since virtual function pointers are stored in each instance, the compiler is enabled to call the correct virtual function at runtime. First, every class that uses virtual functions (or is derived from a class that uses virtual functions) is given its own virtual table. Second, the compiler also adds a hidden pointer to the base class, which we will call \*\_\_vptr. \*\_\_vptr is set (automatically) when a class instance is created so that it points to the virtual table for that class.
2. **vptr:** A pointer to vtable, maintained per object instance (see [this](http://geeksquiz.com/c-virtual-functions-question-12/) for an example).

[](https://media.geeksforgeeks.org/wp-content/uploads/VirtualFunction.png)

What is Multithreading?

The concept of **multi-threading** needs proper understanding of these two terms - **a process and a thread**.

* A **process** is a program being executed. A process can be further divided into independent units known as threads.
* A **thread** is like a small light-weight process within a process. Or we can say a collection of threads is what is known as a process.

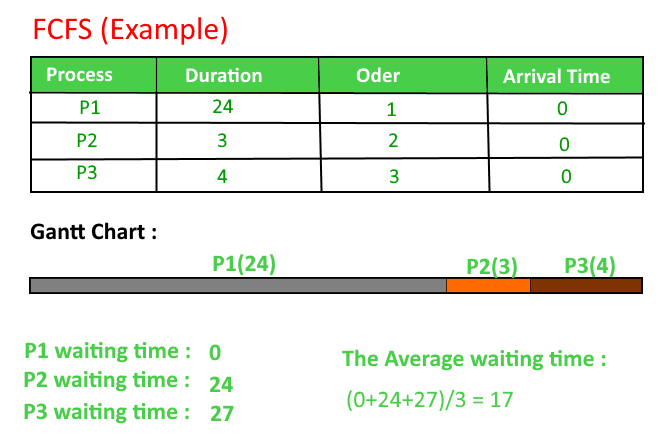
[](https://www.geeksforgeeks.org/multithreading-in-operating-system/)

What is the difference between a thread and a process?

**Difference between Process and Thread:**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **Process** | **Thread** |
| **1.** | **Process means any program is in execution.** | **Thread means segment of a process.** |
| **2.** | **Process takes more time to terminate.** | **Thread takes less time to terminate.** |
| **3.** | **It takes more time for creation.** | **It takes less time for creation.** |
| **4.** | **It also takes more time for context switching.** | **It takes less time for context switching.** |
| **5.** | **Process is less efficient in term of communication.** | **Thread is more efficient in term of communication.** |
| **6.** | **Process consume more resources.** | **Thread consume less resources.** |
| **7.** | **Process is isolated.** | **Threads share memory.** |

Explain FCFS Scheduling?

First in, first out (FIFO), also known as first come, first served (FCFS), is the simplest scheduling algorithm. FIFO simply queues processes in the order that they arrive in the ready queue.  
In this, the process that comes first will be executed first and the next process starts only after the previous gets fully executed.  
Here we are considering that arrival time for all processes is 0.  
  
  
  
  
  
**Implementation:**

1- Input the processes along with their burst time (bt).

2- Find waiting time (wt) for all processes.

3- As first process that comes need not to wait so

waiting time for process 1 will be 0 i.e. wt[0] = 0.

4- Find **waiting time** for all other processes i.e. for

process i ->

wt[i] = bt[i-1] + wt[i-1] .

5- Find **turnaround time** = waiting\_time + burst\_time

for all processes.

6- Find **average waiting time** =

total\_waiting\_time / no\_of\_processes.

7- Similarly, find **average turnaround time** =

total\_turn\_around\_time / no\_of\_processes.

Explain Shortest Job First (or SJF) Scheduling?

**Shortest job first (SJF)** or **shortest job next**, is a scheduling policy that selects the waiting process with the smallest execution time to execute next. SJN is a non-preemptive algorithm.

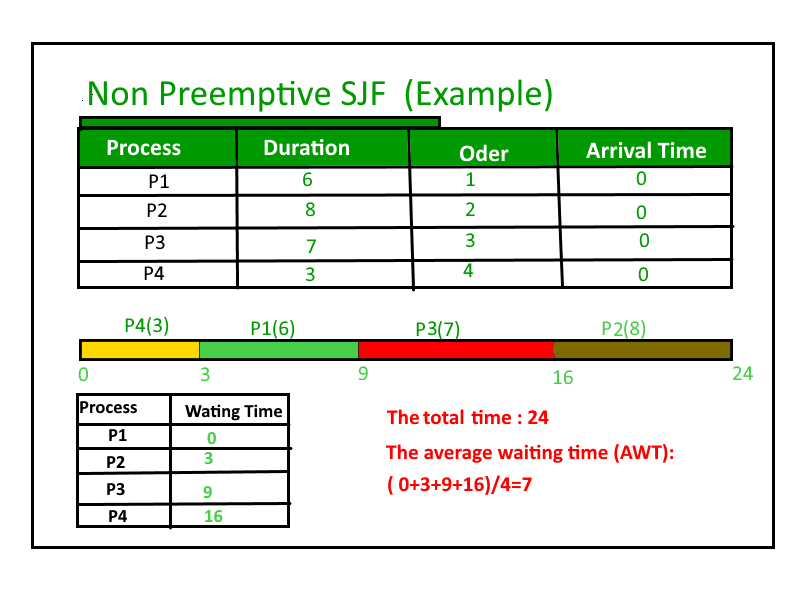
* Shortest Job first has the advantage of having a minimum average waiting time among all scheduling algorithms.
* It is a Greedy Algorithm.
* It may cause starvation if shorter processes keep coming. This problem can be solved using the concept of ageing.
* It is practically infeasible as Operating System may not know burst time and therefore may not sort them. While it is not possible to predict execution time, several methods can be used to estimate the execution time for a job, such as a weighted average of previous execution times. SJF can be used in specialized environments where accurate estimates of running time are available.

**How to compute below times in SJF using a program?**

1. Completion Time: Time at which process completes its execution.
2. Turn Around Time: Time Difference between completion time and arrival time. Turn Around Time = Completion Time - Arrival Time
3. Waiting Time(W.T): Time Difference between turn around time and burst time. Waiting Time = Turn Around Time - Burst Time

**Algorithm:**

1. Sort all the process according to the arrival time.
2. Then select that process which has minimum arrival time and minimum Burst time.
3. After completion of process make a pool of process which after till the completion of previous process and select that process among the pool which is having minimum Burst time.



Explain Shortest Remaining Time First (SRTF) Scheduling?

In the **Shortest Remaining Time First (SRTF**) scheduling algorithm, the process with the smallest amount of time remaining until completion is selected to execute. Since the currently executing process is the one with the shortest amount of time remaining by definition, and since that time should only reduce as execution progresses, processes will always run until they complete or a new process is added that requires a smaller amount of time.

**Shortest Remaining Time First (Preemptive SJF): Example**

**P1 waiting time: 4-2 = 2  
P2 waiting time: 0  
The average waiting time(AWT): (0 + 2) / 2 = 1**

**Implementation:**

1- Traverse until all process gets completely

executed.

a) Find process with minimum remaining time at

every single time lap.

b) Reduce its time by 1.

c) Check if its remaining time becomes 0

d) Increment the counter of process completion.

e) Completion time of current process =

current\_time +1;

e) Calculate waiting time for each completed

process.

wt[i]= Completion time - arrival\_time-burst\_time

f)Increment time lap by one.

2- Find turnaround time (waiting\_time+burst\_time).

Explain Longest Remaining Time First (LRTF) CPU Scheduling?

This is a pre-emptive version of **Longest Job First (LJF) scheduling** algorithm. In this scheduling algorithm, we find the process with the maximum remaining time and then process it. We check for the maximum remaining time after some interval of time(say 1 unit each) to check if another process having more Burst Time arrived up to that time.  
  
**Procedure:**

* **Step-1:** First, sort the processes in increasing order of their Arrival Time.
* **Step-2:** Choose the process having least arrival time but with most Burst Time. Then process it for 1 unit. Check if any other process arrives upto that time of execution or not.
* **Step-3:** Repeat the above both steps until execute all the processes.

**Example-1:** Consider the following table of arrival time and burst time for four processes P1, P2, P3 and P4.

Process Arrival time Burst Time

P1 1 ms 2 ms

P2 2 ms 4 ms

P3 3 ms 6 ms

P4 4 ms 8 ms

**Working: (for input 1):**

1. At t = 1, Available Process : P1. So, select P1 and execute 1 ms.
2. At t = 2, Available Process : P1, P2. So, select P2 and execute 1 ms (since BT(P1)=1 which is less than BT(P2) = 4)
3. At t = 3, Available Process : P1, P2, P3. So, select P3 and execute 1 ms (since, BT(P1) = 1 , BT(P2) = 3 , BT(P3) = 6).
4. Repeat the above steps until the execution of all processes.

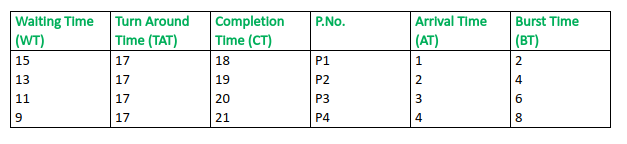
**Note** that CPU will be idle for 0 to 1 unit time since there is no process available in the given interval.  
  
Gantt chart will be as following below,  
  
  
  
Since, complietion time (CT) can be directly determined by Gantt chart, and

Turn Around Time (TAT)

= (Complition Time) - (Arival Time)

Also, Waiting Time (WT)

= (Turn Around Time) - (Burst Time)

Therefore, final table look like,  
  
  
  
**Output:**

Total Turn Around Time = 68 ms

So, Average Turn Around Time = 68/4 = 17.00 ms

And, Total Waiting Time = 48 ms

So Average Waiting Time = 48/4 = 12.00 ms

Explain Priority CPU Scheduling?

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

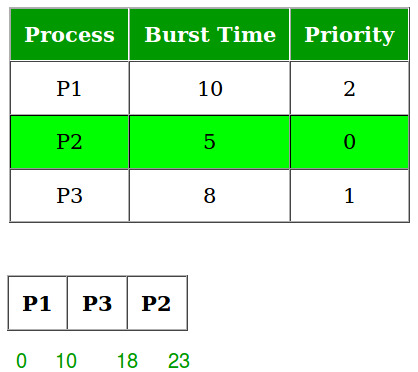
1- First input the processes with their burst time

and priority.

2- Sort the processes, burst time and priority

according to the priority.

3- Now simply apply [FCFS](https://www.geeksforgeeks.org/program-fcfs-scheduling-set-1/) algorithm.

[](https://media.geeksforgeeks.org/wp-content/uploads/PRIORITYsCHEDULING.jpg)  
  
**Note:** A major problem with priority scheduling is indefinite blocking or starvation. A solution to the problem of indefinite blockage of the low-priority process is ageing. Ageing is a technique of gradually increasing the priority of processes that wait in the system for a long period of time.