**CHAPTER-2**

**PROCESSES, THREAD & PROCESS SCHEDULING:** Processes: Definition, Process Relationship, Different states of a Process, Process State transitions, Process Control Block (PCB), Context switching.

**Thread:** Definition, Various states, Benefits of threads, Types of threads, Concept of multithreads.

**Process Scheduling:** Foundation and Scheduling objectives, Types of Schedulers, Scheduling criteria: CPU utilization,Throughput, Turnaround Time, Waiting Time, Response Time; Scheduling algorithms: Pre-emptive and Non pre- emptive, FCFS, SJF, RR.

**Thread:** Definition, Various states, Benefits of threads, Types of threads, Concept of multithreads.

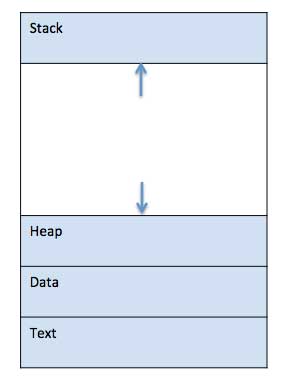
## Process

A process is basically a program in execution. The execution of a process must progress in a sequential fashion.

*A process is defined as an entity which represents the basic unit of work to be implemented in the system.*

To put it in simple terms, we write our computer programs in a text file and when we execute this program, it becomes a process which performs all the tasks mentioned in the program.

When a program is loaded into the memory and it becomes a process, it can be divided into four sections ─ stack, heap, text and data. The following image shows a simplified layout of a process inside main memory −



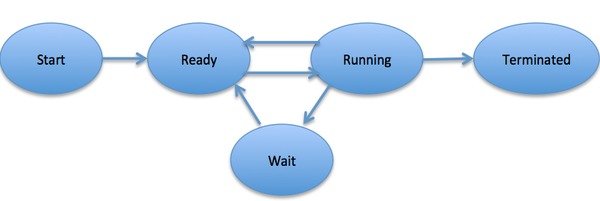
|  |  |
| --- | --- |
| **S.N.** | **Component & Description** |
| 1 | **Stack**  The process Stack contains the temporary data such as method/function parameters, return address and local variables. |
| 2 | **Heap**  This is dynamically allocated memory to a process during its run time. |
| 3 | **Text**  This includes the current activity represented by the value of Program Counter and the contents of the processor's registers. |
| 4 | **Data**  This section contains the global and static variables. |

## Process Life Cycle

When a process executes, it passes through different states. These stages may differ in different operating systems, and the names of these states are also not standardized.

In general, a process can have one of the following five states at a time.

|  |  |
| --- | --- |
| **S.N.** | **State & Description** |
| 1 | **Start**  This is the initial state when a process is first started/created. |
| 2 | **Ready**  The process is waiting to be assigned to a processor. Ready processes are waiting to have the processor allocated to them by the operating system so that they can run. Process may come into this state after **Start** state or while running it by but interrupted by the scheduler to assign CPU to some other process. |
| 3 | **Running**  Once the process has been assigned to a processor by the OS scheduler, the process state is set to running and the processor executes its instructions. |
| 4 | **Waiting**  Process moves into the waiting state if it needs to wait for a resource, such as waiting for user input, or waiting for a file to become available. |
| 5 | **Terminated or Exit**  Once the process finishes its execution, or it is terminated by the operating system, it is moved to the terminated state where it waits to be removed from main memory. |

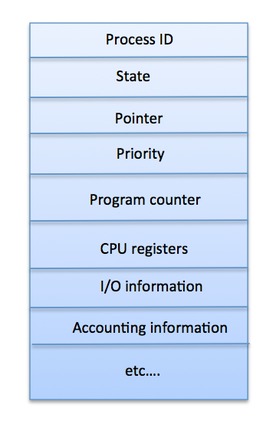


## Process Control Block (PCB)

A Process Control Block is a data structure maintained by the Operating System for every process. The PCB is identified by an integer process ID (PID). A PCB keeps all the information needed to keep track of a process as listed below in the table −

|  |  |
| --- | --- |
| **S.N.** | **Information & Description** |
| 1 | **Process State**  The current state of the process i.e., whether it is ready, running, waiting, or whatever. |
| 2 | **Process privileges**  This is required to allow/disallow access to system resources. |
| 3 | **Process ID**  Unique identification for each of the process in the operating system. |
| 4 | **Pointer**  A pointer to parent process. |
| 5 | **Program Counter**  Program Counter is a pointer to the address of the next instruction to be executed for this process. |
| 6 | **CPU registers**  Various CPU registers where process need to be stored for execution for running state. |
| 7 | **CPU Scheduling Information**  Process priority and other scheduling information which is required to schedule the process. |
| 8 | **Memory management information**  This includes the information of page table, memory limits, Segment table depending on memory used by the operating system. |
| 9 | **Accounting information**  This includes the amount of CPU used for process execution, time limits, execution ID etc. |
| 10 | **IO status information**  This includes a list of I/O devices allocated to the process. |

The architecture of a PCB is completely dependent on Operating System and may contain different information in different operating systems. Here is a simplified diagram of a PCB −



The PCB is maintained for a process throughout its lifetime, and is deleted once the process terminates.

# Operating System - Process Scheduling

## Definition

The process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy.

Process scheduling is an essential part of a Multiprogramming operating systems. Such operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.

Ezoic

## Categories of Scheduling

There are two categories of scheduling:

1. **Non-preemptive:** Here the resource can’t be taken from a process until the process completes execution. The switching of resources occurs when the running process terminates and moves to a waiting state.
2. **Preemptive:** Here the OS allocates the resources to a process for a fixed amount of time. During resource allocation, the process switches from running state to ready state or from waiting state to ready state. This switching occurs as the CPU may give priority to other processes and replace the process with higher priority with the running process.

[Explore our](https://www.tutorialspoint.com/latest/courses?utm_source=tutorialspoint&utm_medium=tutorials_3p&utm_campaign=internal" \t "_blank)**[latest online courses](https://www.tutorialspoint.com/latest/courses?utm_source=tutorialspoint&utm_medium=tutorials_3p&utm_campaign=internal" \t "_blank)**[and learn new skills at your own pace. Enroll and become a certified expert to boost your career.](https://www.tutorialspoint.com/latest/courses?utm_source=tutorialspoint&utm_medium=tutorials_3p&utm_campaign=internal" \t "_blank)

## Process Scheduling Queues

The OS maintains all Process Control Blocks (PCBs) in Process Scheduling Queues. The OS maintains a separate queue for each of the process states and PCBs of all processes in the same execution state are placed in the same queue. When the state of a process is changed, its PCB is unlinked from its current queue and moved to its new state queue.

The Operating System maintains the following important process scheduling queues −

* **Job queue** − This queue keeps all the processes in the system.
* **Ready queue** − This queue keeps a set of all processes residing in main memory, ready and waiting to execute. A new process is always put in this queue.
* **Device queues** − The processes which are blocked due to unavailability of an I/O device constitute this queue.



The OS can use different policies to manage each queue (FIFO, Round Robin, Priority, etc.). The OS scheduler determines how to move processes between the ready and run queues which can only have one entry per processor core on the system; in the above diagram, it has been merged with the CPU.

Two-State Process Model

Two-state process model refers to running and non-running states which are described below −

|  |  |
| --- | --- |
| **S.N.** | **State & Description** |
| 1 | **Running**  When a new process is created, it enters into the system as in the running state. |
| 2 | **Not Running**  Processes that are not running are kept in queue, waiting for their turn to execute. Each entry in the queue is a pointer to a particular process. Queue is implemented by using linked list. Use of dispatcher is as follows. When a process is interrupted, that process is transferred in the waiting queue. If the process has completed or aborted, the process is discarded. In either case, the dispatcher then selects a process from the queue to execute. |

Schedulers

Schedulers are special system software which handle process scheduling in various ways. Their main task is to select the jobs to be submitted into the system and to decide which process to run. Schedulers are of three types −

* Long-Term Scheduler
* Short-Term Scheduler
* Medium-Term Scheduler

## Long Term Scheduler

It is also called a **job scheduler**. A long-term scheduler determines which programs are admitted to the system for processing. It selects processes from the queue and loads them into memory for execution. Process loads into the memory for CPU scheduling.

The primary objective of the job scheduler is to provide a balanced mix of jobs, such as I/O bound and processor bound. It also controls the degree of multiprogramming. If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system.

On some systems, the long-term scheduler may not be available or minimal. Time-sharing operating systems have no long term scheduler. When a process changes the state from new to ready, then there is use of long-term scheduler.

Short Term Scheduler

It is also called as **CPU scheduler**. Its main objective is to increase system performance in accordance with the chosen set of criteria. It is the change of ready state to running state of the process. CPU scheduler selects a process among the processes that are ready to execute and allocates CPU to one of them.

Short-term schedulers, also known as dispatchers, make the decision of which process to execute next. Short-term schedulers are faster than long-term schedulers.

Medium Term Scheduler

Medium-term scheduling is a part of **swapping**. It removes the processes from the memory. It reduces the degree of multiprogramming. The medium-term scheduler is in-charge of handling the swapped out-processes.

A running process may become suspended if it makes an I/O request. A suspended processes cannot make any progress towards completion. In this condition, to remove the process from memory and make space for other processes, the suspended process is moved to the secondary storage. This process is called **swapping**, and the process is said to be swapped out or rolled out. Swapping may be necessary to improve the process mix.

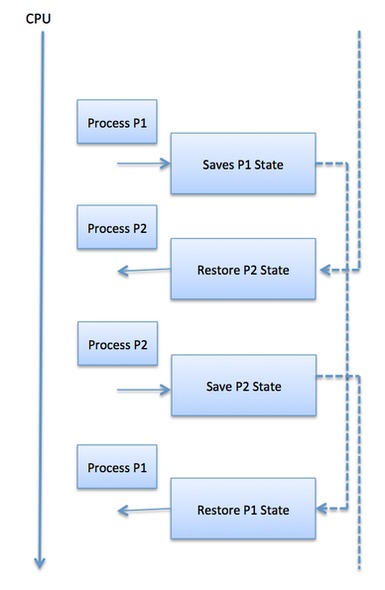
Comparison among Scheduler

|  |  |  |  |
| --- | --- | --- | --- |
| **S.N.** | **Long-Term Scheduler** | **Short-Term Scheduler** | **Medium-Term Scheduler** |
| 1 | It is a job scheduler | It is a CPU scheduler | It is a process swapping scheduler. |
| 2 | Speed is lesser than short term scheduler | Speed is fastest among other two | Speed is in between both short and long term scheduler. |
| 3 | It controls the degree of multiprogramming | It provides lesser control over degree of multiprogramming | It reduces the degree of multiprogramming. |
| 4 | It is almost absent or minimal in time sharing system | It is also minimal in time sharing system | It is a part of Time sharing systems. |
| 5 | It selects processes from pool and loads them into memory for execution | It selects those processes which are ready to execute | It can re-introduce the process into memory and execution can be continued. |

Context Switching

A context switching is the mechanism to store and restore the state or context of a CPU in Process Control block so that a process execution can be resumed from the same point at a later time. Using this technique, a context switcher enables multiple processes to share a single CPU. Context switching is an essential part of a multitasking operating system features.

When the scheduler switches the CPU from executing one process to execute another, the state from the current running process is stored into the process control block. After this, the state for the process to run next is loaded from its own PCB and used to set the PC, registers, etc. At that point, the second process can start executing.

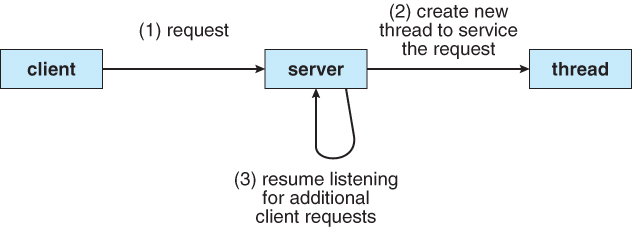


Context switches are computationally intensive since register and memory state must be saved and restored. To avoid the amount of context switching time, some hardware systems employ two or more sets of processor registers. When the process is switched, the following information is stored for later use.

* Program Counter
* Scheduling information
* Base and limit register value
* Currently used register
* Changed State
* I/O State information
* Accounting information

***Motivation***

* Threads are very useful in modern programming whenever a process has multiple tasks to perform independently of the others.
* This is particularly true when one of the tasks may block, and it is desired to allow the other tasks to proceed without blocking.
* For example in a word processor, a background thread may check spelling and grammar while a foreground thread processes user input ( keystrokes ), while a third thread loads images from the hard drive, and a fourth does periodic automatic backups of the file being edited.
* Another example is a web server - Multiple threads allow for multiple requests to be satisfied simultaneously, without having to service requests sequentially or to fork off separate processes for every incoming request. ( The latter is how this sort of thing was done before the concept of threads was developed. A daemon would listen at a port, fork off a child for every incoming request to be processed, and then go back to listening to the port. )

  
**Figure - Multithreaded server architecture**

#### **Benefits**

There are four major categories of benefits to multi-threading:

* **Responsiveness -** One thread may provide rapid response while other threads are blocked or slowed down doing intensive calculations.
* **Resource sharing -** By default threads share common code, data, and other resources, which allows multiple tasks to be performed simultaneously in a single address space.
* **Economy -** Creating and managing threads ( and context switches between them ) is much faster than performing the same tasks for processes.
* **Scalability**, i.e. Utilization of multiprocessor architectures - A single threaded process can only run on one CPU, no matter how many may be available, whereas the execution of a multi-threaded application may be split amongst available processors. ( Note that single threaded processes can still benefit from multi-processor architectures when there are multiple processes contending for the CPU, i.e. when the load average is above some certain threshold. )

**Thread in Operating System**

A thread is a single sequence stream within a process. Threads are also called lightweight processes as they possess some of the properties of processes. Each thread belongs to exactly one process. In an operating system that supports multithreading, the process can consist of many threads. But threads can be effective only if the CPU is more than 1 otherwise two threads have to context switch for that single CPU.

**What is Thread in Operating Systems?**

In a process, a thread refers to a single sequential activity being executed. these activities are also known as thread of execution or thread control. Now, any operating system process can execute a thread. we can say, that a process can have multiple threads.

**Why Do We Need Thread?**

* Threads run in parallel improving the application performance. Each such thread has its own CPU state and stack, but they share the address space of the process and the environment.
* Threads can share common data so they do not need to use [inter-process communication](https://www.geeksforgeeks.org/inter-process-communication-ipc). Like the processes, threads also have states like ready, executing, blocked, etc.
* Priority can be assigned to the threads just like the process, and the highest priority thread is scheduled first.
* Each thread has its own [Thread Control Block (TCB)](https://www.geeksforgeeks.org/thread-control-block-in-operating-system). Like the process, a context switch occurs for the thread, and register contents are saved in (TCB). As threads share the same address space and resources, synchronization is also required for the various activities of the thread.

**Similarity Between Threads and Process**

* Only one thread or process is active at a time in an operating system.
* Within the process, both execute in a sequential manner.
* Both can create children.
* Both can be scheduled by the operating system: Both threads and processes can be scheduled by the operating system to execute on the [CPU](https://www.geeksforgeeks.org/difference-between-cpu-and-gpu/) . The operating system is responsible for assigning CPU time to the threads and processes based on various [scheduling algorithms](https://www.geeksforgeeks.org/cpu-scheduling-in-operating-systems/) .
* Both have their own execution context: Each thread and process has its own execution context, which includes its own register set, program counter, and stack. This allows each thread or process to execute independently and make progress without interfering with other threads or processes.
* Both can communicate with each other: Threads and processes can communicate with each other using various [inter-process communication](https://www.geeksforgeeks.org/inter-process-communication-ipc/) (IPC) mechanisms such as shared memory, message queues, and pipes. This allows threads and processes to share data and coordinate their activities.
* Both can be preempted: Threads and processes can be preempted by the operating system, which means that their execution can be interrupted at any time. This allows the operating system to switch to another thread or process that needs to execute.
* Both can be terminated: Threads and processes can be terminated by the operating system or by other threads or processes. When a thread or process is terminated, all of its resources, including its execution context, are freed up and made available to other threads or processes.

**Differences Between Threads and Process**

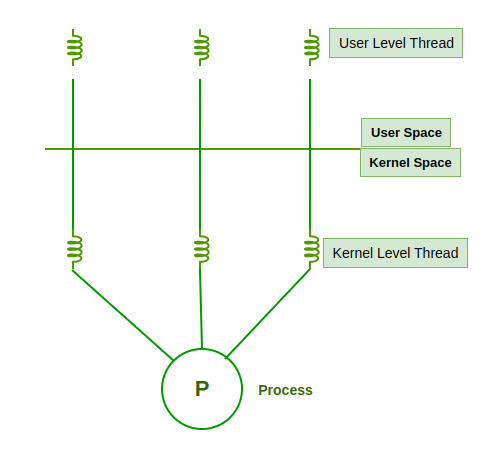
* Resources: Processes have their own address space and resources, such as memory and file handles, whereas threads share memory and resources with the program that created them.
* Scheduling: Processes are scheduled to use the processor by the operating system, whereas threads are scheduled to use the processor by the operating system or the program itself.
* Creation: The operating system creates and manages processes, whereas the program or the operating system creates and manages threads.
* Communication: Because processes are isolated from one another and must rely on inter-process communication mechanisms, they generally have more difficulty communicating with one another than threads do. Threads, on the other hand, can interact with other threads within the same Programme directly.

**Components of Threads**

* These are the basic components of the Operating System.
* Stack Space
* Register Set
* [Program Counter](https://www.geeksforgeeks.org/what-is-program-counter/)

**Types of Thread in Operating System**

* Threads are of two types. These are described below.
* User Level Thread
* Kernel Level Thread



*Threads*

**1. User Level Threads**

User Level Thread is a type of thread that is not created using system calls. The kernel has no work in the management of user-level threads. User-level threads can be easily implemented by the user. In case when user-level threads are single-handed processes, kernel-level thread manages them. Let’s look at the advantages and disadvantages of User-Level Thread.

**Advantages of User-Level Threads**

* Implementation of the User-Level Thread is easier than Kernel Level Thread.
* [Context Switch](https://www.geeksforgeeks.org/context-switch-in-operating-system) Time is less in User Level Thread.
* User-Level Thread is more efficient than Kernel-Level Thread.
* Because of the presence of only Program Counter, Register Set, and Stack Space, it has a simple representation.

**Disadvantages of User-Level Threads**

* There is a lack of coordination between Thread and Kernel.
* In case of a page fault, the whole process can be blocked.

**2. Kernel Level Threads**

A [kernel Level Thread](https://www.geeksforgeeks.org/kernel-level-threads-in-operating-system) is a type of thread that can recognize the Operating system easily. Kernel Level Threads has its own thread table where it keeps track of the system. The operating System Kernel helps in managing threads. Kernel Threads have somehow longer context switching time. Kernel helps in the management of threads.

**Advantages of Kernel-Level Threads**

* It has up-to-date information on all threads.
* Applications that block frequency are to be handled by the Kernel-Level Threads.
* Whenever any process requires more time to process, Kernel-Level Thread provides more time to it.

**Disadvantages of Kernel-Level threads**

* Kernel-Level Thread is slower than User-Level Thread.
* Implementation of this type of thread is a little more complex than a user-level thread.

## Difference Between User-Level Thread and Kernel-Level Thread

| **Parameters** | **User Level Thread** | **Kernel Level Thread** |
| --- | --- | --- |
| **Implemented by** | User threads are implemented by user-level libraries. | Kernel threads are implemented by Operating System (OS). |
| **Recognize** | The operating System doesn’t recognize user-level threads directly. | Kernel threads are recognized by [Operating System](https://www.geeksforgeeks.org/what-is-an-operating-system/). |
| **Implementation** | Implementation of User threads is easy. | Implementation of Kernel-Level thread is complicated. |
| **Context switch time** | Context switch time is less. | Context switch time is more. |
| **Hardware support** | No hardware support is required for context switching. | Hardware support is needed. |
| **Blocking operation** | If one user-level thread performs a blocking operation then the entire process will be blocked. | If one kernel thread performs a blocking operation then another thread can continue execution. |
| **Multithreading** | Multithreaded applications cannot take full advantage of multiprocessing. | Kernels can be multithreaded. |
| **Creation and Management** | User-level threads can be created and managed more quickly. | Kernel-level level threads take more time to create and manage. |
| **Operating System** | Any operating system can support user-level threads. | Kernel-level threads are operating system-specific. |
| **Thread Management** | Managed by a thread library at the user level. | The application code doesn’t contain thread management code; it’s an [API](https://www.geeksforgeeks.org/what-is-an-api/) to the kernel mode. |
| **Example** | POSIX threads, Mach C-Threads. | Java threads, [POSIX threads](https://www.geeksforgeeks.org/posix-threads-in-os/) on Linux. |
| **Advantages** | Simple and quick to create, more portable, does not require kernel mode privileges for context switching. | Allows for true parallelism, multithreading in kernel routines, and can continue execution if one thread is blocked. |
| **Disadvantages** | Cannot fully utilize multiprocessing, entire process blocked if one thread blocks. | Requires more time to create/manage, involves mode switching to kernel mode. |
| **Memory management** | In user-level threads, each thread has its own stack, but they share the same address space. | In kernel-level threads have their own stacks and their own separate address spaces, so they are better isolated from each other. |
| **Fault tolerance** | User-level threads are less fault-tolerant than kernel-level threads. If a user-level thread crashes, it can bring down the entire process. | Kernel-level threads can be managed independently, so if one thread crashes, it doesn’t necessarily affect the others. |
| **Resource utilization** | Limited access to system resources, cannot directly perform I/O operations. | It can access to the system-level features like I/O operations. |
| **Portability** | User-level threads are  more portable than kernel-level threads. | Less portable due to dependence on OS-specific kernel implementations. |

**Multithreading Models**

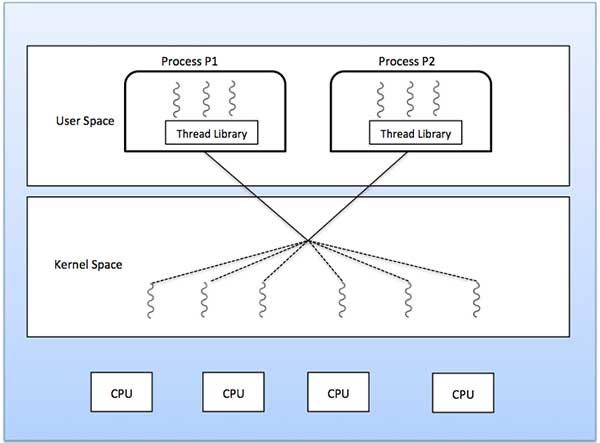
Some operating system provide a combined user level thread and Kernel level thread facility. Solaris is a good example of this combined approach. In a combined system, multiple threads within the same application can run in parallel on multiple processors and a blocking system call need not block the entire process. Multithreading models are three types

* Many to many relationship.
* Many to one relationship.
* One to one relationship.

**Many to Many Model**

The many-to-many model multiplexes any number of user threads onto an equal or smaller number of kernel threads.

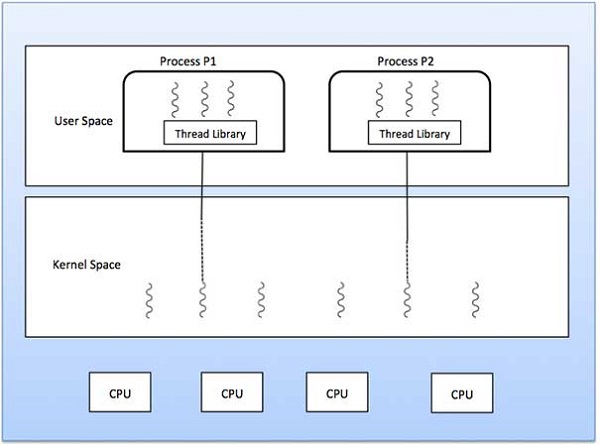
The following diagram shows the many-to-many threading model where 6 user level threads are multiplexing with 6 kernel level threads. In this model, developers can create as many user threads as necessary and the corresponding Kernel threads can run in parallel on a multiprocessor machine. This model provides the best accuracy on concurrency and when a thread performs a blocking system call, the kernel can schedule another thread for execution.



Many to One Model

Many-to-one model maps many user level threads to one Kernel-level thread. Thread management is done in user space by the thread library. When thread makes a blocking system call, the entire process will be blocked. Only one thread can access the Kernel at a time, so multiple threads are unable to run in parallel on multiprocessors.

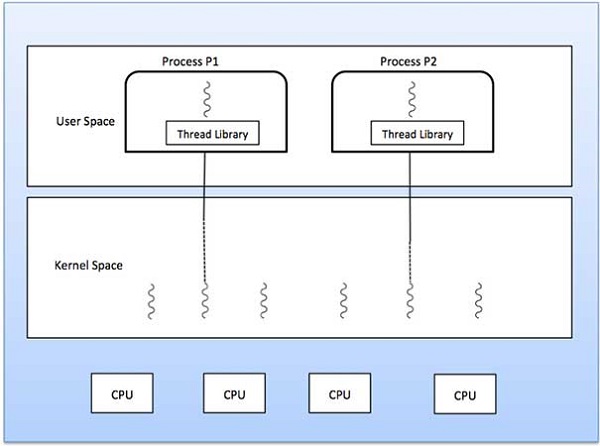
If the user-level thread libraries are implemented in the operating system in such a way that the system does not support them, then the Kernel threads use the many-to-one relationship modes.



One to One Model

There is one-to-one relationship of user-level thread to the kernel-level thread. This model provides more concurrency than the many-to-one model. It also allows another thread to run when a thread makes a blocking system call. It supports multiple threads to execute in parallel on microprocessors.

Disadvantage of this model is that creating user thread requires the corresponding Kernel thread. OS/2, windows NT and windows 2000 use one to one relationship model.



**Difference Between Process and Thread**

The primary difference is that threads within the same process run in a shared memory space, while processes run in separate memory spaces. Threads are not independent of one another like processes are, and as a result, threads share with other threads their **code section, data section, and OS resources** (like open files and signals). But, like a process, a thread has its own [program counter (PC)](https://www.geeksforgeeks.org/what-is-program-counter), register set, and stack space.

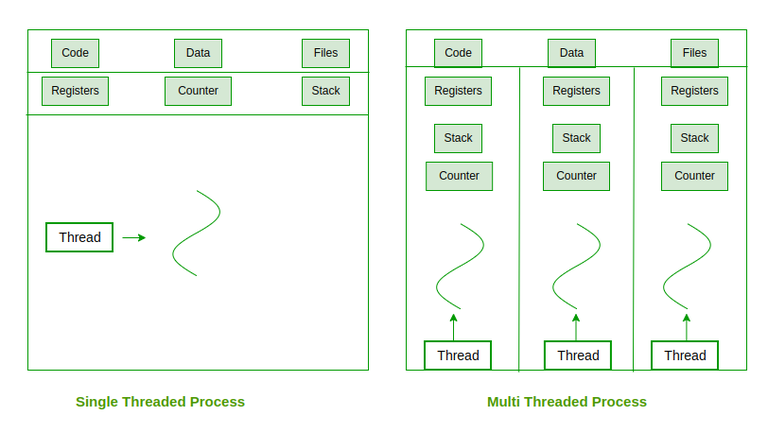
The table below represents the difference between process and thread.

| **Process** | **Thread** |
| --- | --- |
| Process means any program is in execution. | Thread means a segment of a process. |
| The process takes more time to terminate. | The thread takes less time to terminate. |
| It takes more time for creation. | It takes less time for creation. |
| It also takes more time for context switching. | It takes less time for context switching. |
| The process is less efficient in terms of communication. | Thread is more efficient in terms of communication. |
| Multiprogramming holds the concepts of multi-process. | We don’t need multi programs in action for multiple threads because a single process consists of multiple threads. |
| The process is isolated. | Threads share memory. |
| The process is called the heavyweight process. | A Thread is lightweight as each thread in a process shares code, data, and resources. |
| Process switching uses an interface in an operating system. | Thread switching does not require calling an operating system and causes an interrupt to the kernel. |
| If one process is blocked, then it will not affect the execution of other processes. | If a user-level thread is blocked, then all other user-level threads are blocked. |
| The process has its own Process Control Block, Stack, and Address Space. | Thread has Parents’ PCB, its own Thread Control Block, and Stack and common Address space. |
| Changes to the parent process do not affect child processes. | Since all threads of the same process share address space and other resources so any changes to the main thread may affect the behavior of the other threads of the process. |
| A system call is involved in it. | No system call is involved, it is created using APIs. |
| The process does not share data with each other. | Threads share data with each other. |

**What is Multi-Threading?**

A thread is also known as a lightweight process. The idea is to achieve parallelism by dividing a process into multiple threads. For example, in a [browser](https://www.geeksforgeeks.org/web-browser), multiple tabs can be different threads. MS Word uses multiple threads: one thread to format the text, another thread to process inputs, etc. More advantages of multithreading are discussed below.

Multithreading is a technique used in operating systems to improve the performance and responsiveness of computer systems. Multithreading allows multiple threads (i.e., lightweight processes) to share the same resources of a single process, such as the CPU, [memory](https://www.geeksforgeeks.org/memory-management-in-operating-system), and[I/O devices](https://www.geeksforgeeks.org/input-and-output-devices).



*Single Threaded vs Multi-threaded Process*

**Benefits of Thread in Operating System**

* **Responsiveness*:***If the process is divided into multiple threads, if one thread completes its execution, then its output can be immediately returned.
* **Faster context switch*:***Context switch time between threads is lower compared to the process context switch. Process context switching requires more overhead from the CPU.
* **Effective utilization of multiprocessor system*:***If we have multiple threads in a single process, then we can schedule multiple threads on multiple processors. This will make process execution faster.
* **Resource sharing*:***Resources like code, data, and files can be shared among all threads within a process. Note: Stacks and registers can’t be shared among the threads. Each thread has its own stack and registers.
* **Communication*:***Communication between multiple threads is easier, as the threads share a common address space. while in the process we have to follow some specific communication techniques for communication between the two processes.
* **Enhanced throughput of the system*:***If a process is divided into multiple threads, and each thread function is considered as one job, then the number of jobs completed per unit of time is increased, thus increasing the throughput of the system.

**Process Scheduling:** Foundation and Scheduling objectives, Types of Schedulers, Scheduling criteria: CPU utilization,Throughput, Turnaround Time, Waiting Time, Response Time; Scheduling algorithms: Pre-emptive and Non pre- emptive, FCFS, SJF, RR.

# Lightbox

Categories of Scheduling

Scheduling falls into one of two categories:

* Non-Preemptive: In this case, a process’s resource cannot be taken before the process has finished running. When a running process finishes and transitions to a waiting state, resources are switched.
* Preemptive: In this case, the OS assigns resources to a process for a predetermined period. The process switches from running state to ready state or from waiting state to ready state during resource allocation. This switching happens because the CPU may give other processes priority and substitute the currently active process for the higher priority process.

# Process Scheduling in OS (Operating System)

Operating system uses various schedulers for the process scheduling described below.

### 1. Long term scheduler

Long term scheduler is also known as job scheduler. It chooses the processes from the pool (secondary memory) and keeps them in the ready queue maintained in the primary memory.

Long Term scheduler mainly controls the degree of Multiprogramming. The purpose of long term scheduler is to choose a perfect mix of IO bound and CPU bound processes among the jobs present in the pool.

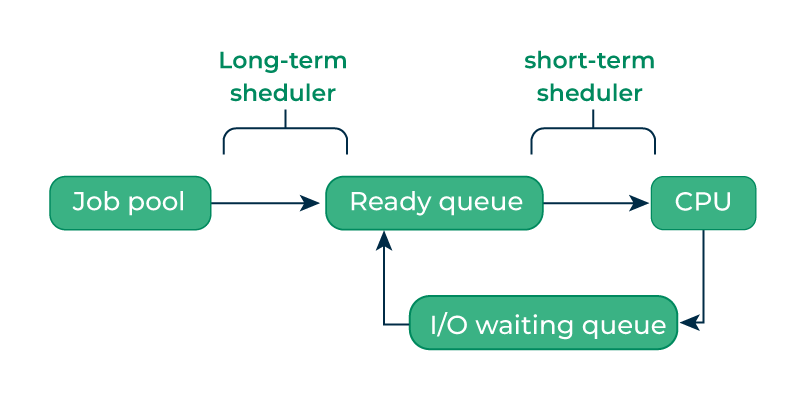
If the job scheduler chooses more IO bound processes then all of the jobs may reside in the blocked state all the time and the CPU will remain idle most of the time. This will reduce the degree of Multiprogramming. Therefore, the Job of long term scheduler is very critical and may affect the system for a very long time.

### 2. Short term scheduler

Short term scheduler is also known as CPU scheduler. It selects one of the Jobs from the ready queue and dispatch to the CPU for the execution.

A scheduling algorithm is used to select which job is going to be dispatched for the execution. The Job of the short term scheduler can be very critical in the sense that if it selects job whose CPU burst time is very high then all the jobs after that, will have to wait in the ready queue for a very long time.

This problem is called starvation which may arise if the short term scheduler makes some mistakes while selecting the job.

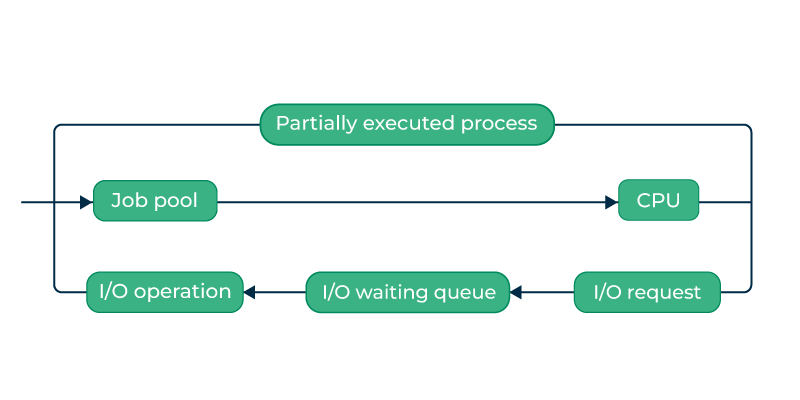


### 3. Medium term scheduler

Medium term scheduler takes care of the swapped out processes.If the running state processes needs some IO time for the completion then there is a need to change its state from running to waiting.

Medium term scheduler is used for this purpose. It removes the process from the running state to make room for the other processes. Such processes are the swapped out processes and this procedure is called swapping. The medium term scheduler is responsible for suspending and resuming the processes.

It reduces the degree of multiprogramming. The swapping is necessary to have a perfect mix of processes in the ready queue.

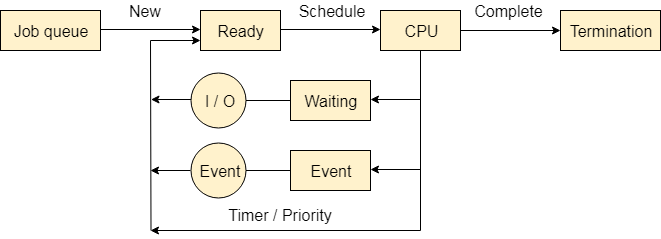


## Comparison Among Scheduler

| **Long Term Scheduler** | **Short Term Schedular** | **Medium Term Scheduler** |
| --- | --- | --- |
| It is a job scheduler | It is a CPU scheduler | It is a process-swapping scheduler. |
| Generally, Speed is lesser than short term scheduler | Speed is the fastest among all of them. | Speed lies in between both short and long-term schedulers. |
| It controls the degree of multiprogramming | It gives less control over how much multiprogramming is done. | It reduces the degree of multiprogramming. |
| It is barely present or nonexistent in the time-sharing system. | It is a minimal time-sharing system. | It is a component of systems for time sharing. |
| It can re-enter the process into memory, allowing for the continuation of execution. | It selects those processes which are ready to execute | It can re-introduce the process into memory and execution can be continued. |

# Process Queues

The Operating system manages various types of queues for each of the process states. The PCB related to the process is also stored in the queue of the same state. If the Process is moved from one state to another state then its PCB is also unlinked from the corresponding queue and added to the other state queue in which the transition is made.



There are the following queues maintained by the Operating system.

### 1. Job Queue

In starting, all the processes get stored in the job queue. It is maintained in the secondary memory. The long term scheduler (Job scheduler) picks some of the jobs and put them in the primary memory.

### 2. Ready Queue

Ready queue is maintained in primary memory. The short term scheduler picks the job from the ready queue and dispatch to the CPU for the execution.

### 3. Waiting Queue

When the process needs some IO operation in order to complete its execution, OS changes the state of the process from running to waiting. The context (PCB) associated with the process gets stored on the waiting queue which will be used by the Processor when the process finishes the IO.

# Various Times related to the Process

### 1. Arrival Time

The time at which the process enters into the ready queue is called the arrival time.

### 2. Burst Time

The total amount of time required by the CPU to execute the whole process is called the Burst Time. This does not include the waiting time. It is confusing to calculate the execution time for a process even before executing it hence the scheduling problems based on the burst time cannot be implemented in reality.

### 3. Completion Time

The Time at which the process enters into the completion state or the time at which the process completes its execution, is called completion time.

### 4. Turnaround time

The total amount of time spent by the process from its arrival to its completion, is called Turnaround time.

### 5. Waiting Time

The Total amount of time for which the process waits for the CPU to be assigned is called waiting time.

### 6. Response Time

The difference between the arrival time and the time at which the process first gets the CPU is called Response Time.

# CPU Scheduling

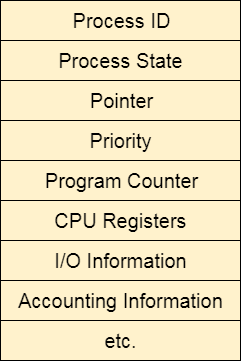
**In the uniprogrammming systems** like MS DOS, when a process waits for any I/O operation to be done, the CPU remains idol. This is an overhead since it wastes the time and causes the problem of starvation. However, In Multiprogramming systems, the CPU doesn't remain idle during the waiting time of the Process and it starts executing other processes. Operating System has to define which process the CPU will be given.

**In Multiprogramming systems**, the Operating system schedules the processes on the CPU to have the maximum utilization of it and this procedure is called **CPU scheduling**. The Operating System uses various scheduling algorithm to schedule the processes.

This is a task of the short term scheduler to schedule the CPU for the number of processes present in the Job Pool. Whenever the running process requests some IO operation then the short term scheduler saves the current context of the process (also called PCB) and changes its state from running to waiting. During the time, process is in waiting state; the Short term scheduler picks another process from the ready queue and assigns the CPU to this process. This procedure is called **context switching**.

## What is saved in the Process Control Block?

The Operating system maintains a process control block during the lifetime of the process. The Process control block is deleted when the process is terminated or killed. There is the following information which is saved in the process control block and is changing with the state of the process.



## Why do we need Scheduling?

In Multiprogramming, if the long term scheduler picks more I/O bound processes then most of the time, the CPU remains idol. The task of Operating system is to optimize the utilization of resources.

If most of the running processes change their state from running to waiting then there may always be a possibility of deadlock in the system. Hence to reduce this overhead, the OS needs to schedule the jobs to get the optimal utilization of CPU and to avoid the possibility to deadlock.

Scheduling Algorithms in OS (Operating System)

There are various algorithms which are used by the Operating System to schedule the processes on the processor in an efficient way.

The Purpose of a Scheduling algorithm

1. Maximum CPU utilization
2. Fare allocation of CPU
3. Maximum throughput
4. Minimum turnaround time
5. Minimum waiting time
6. Minimum response time

There are the following algorithms which can be used to schedule the jobs.

1. First Come First Serve

It is the simplest algorithm to implement. The process with the minimal arrival time will get the CPU first. The lesser the arrival time, the sooner will the process gets the CPU. It is the non-preemptive type of scheduling.

2. Round Robin

In the Round Robin scheduling algorithm, the OS defines a time quantum (slice). All the processes will get executed in the cyclic way. Each of the process will get the CPU for a small amount of time (called time quantum) and then get back to the ready queue to wait for its next turn. It is a preemptive type of scheduling.

### 3. Shortest Job First

The job with the shortest burst time will get the CPU first. The lesser the burst time, the sooner will the process get the CPU. It is the non-preemptive type of scheduling.

### 4. Shortest remaining time first

It is the preemptive form of SJF. In this algorithm, the OS schedules the Job according to the remaining time of the execution.

### 5. Priority based scheduling

In this algorithm, the priority will be assigned to each of the processes. The higher the priority, the sooner will the process get the CPU. If the priority of the two processes is same then they will be scheduled according to their arrival time.

### 6. Highest Response Ratio Next

In this scheduling Algorithm, the process with highest response ratio will be scheduled next. This reduces the starvation in the system.

First Come First Serve CPU Process Scheduling in Operating Systems

In this tutorial, we are going to learn an important concept in CPU Process Scheduling Algorithms. The important concept name is First Come First Serve. This is the basic algorithm which every student must learn to understand all the basics of CPU Process Scheduling Algorithms.

First Come First Serve paves the way for understanding of other algorithms. This algorithm may have many disadvantages. But these disadvantages created very new and efficient algorithms. So, it is our responsibility to learn about First Come First Serve CPU Process Scheduling Algorithms.

Important Abbreviations

1. CPU - - - > Central Processing Unit
2. FCFS - - - > First Come First Serve
3. AT - - - > Arrival Time
4. BT - - - > Burst Time
5. WT - - - > Waiting Time
6. TAT - - - > Turn Around Time
7. CT - - - > Completion Time
8. FIFO - - - > First In First Out

First Come First Serve

First Come First Serve CPU Scheduling Algorithm shortly known as FCFS is the first algorithm of CPU Process Scheduling Algorithm. In First Come First Serve Algorithm what we do is to allow the process to execute in linear manner.

This means that whichever process enters process enters the ready queue first is executed first. This shows that First Come First Serve Algorithm follows First In First Out (FIFO) principle.

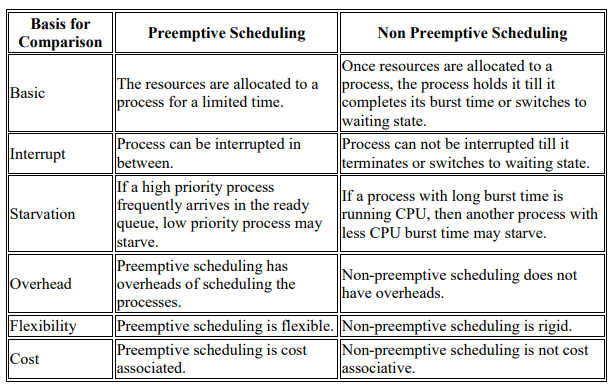
The First Come First Serve Algorithm can be executed in Pre Emptive and Non Pre Emptive manner. Before, going into examples, let us understand what is Pre Emptive and Non Pre Emptive Approach in CPU Process Scheduling.

## Pre Emptive Approach

In this instance of Pre Emptive Process Scheduling, the OS allots the resources to a Process for a predetermined period of time. The process transitions from running state to ready state or from waiting state to ready state during resource allocation. This switching happens because the CPU may assign other processes precedence and substitute the currently active process for the higher priority process.

## Non Pre Emptive Approach

In this case of Non Pre Emptive Process Scheduling, the resource cannot be withdrawn from a process before the process has finished running. When a running process finishes and transitions to the waiting state, resources are switched.



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.
* Waiting time - How much time processes spend in the ready queue waiting their turn to get on the CPU.
* Response time - The time taken in an interactive program from the issuance of a command to the commence of a response to that command.

Types of Scheduling Algorithm (a) First Come First Serve (FCFS)

In FCFS Scheduling

• The process which arrives first in the ready queue is firstly assigned the CPU.

• In case of a tie, process with smaller process id is executed first.

• It is always non-preemptive in nature.

• 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.

Advantages-

• It is simple and easy to understand.

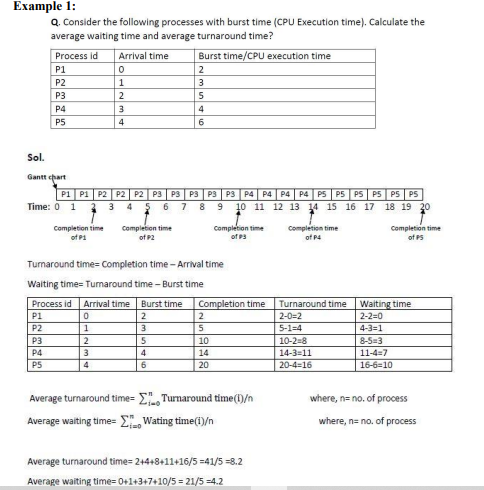
• It can be easily implemented using queue data structure.

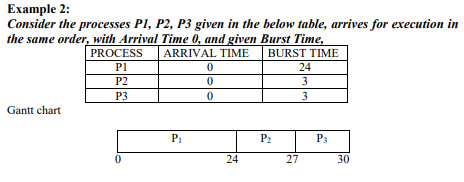
• It does not lead to starvation.

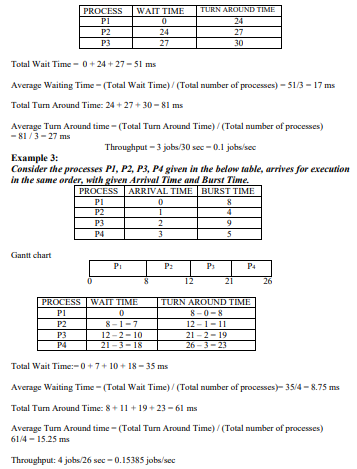
Disadvantages-

• It does not consider the priority or burst time of the processes.

• It suffers from convoy effect i.e. processes with higher burst time arrived before the processes with smaller burst time.







(b) Shortest Job First (SJF)

• Process which have the shortest burst time are scheduled first.

• If two processes have the same bust time, then FCFS is used to break the tie.

• This is a non-pre-emptive, 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.

• Pre-emptive mode of Shortest Job First is called as Shortest Remaining Time First (SRTF).

Advantages-

• SRTF is optimal and guarantees the minimum average waiting time.

• It provides a standard for other algorithms since no other algorithm performs better than it.

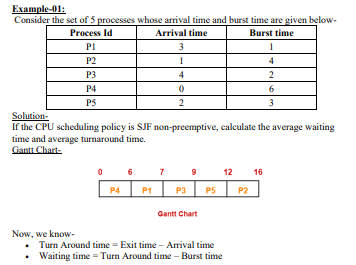
Disadvantages-

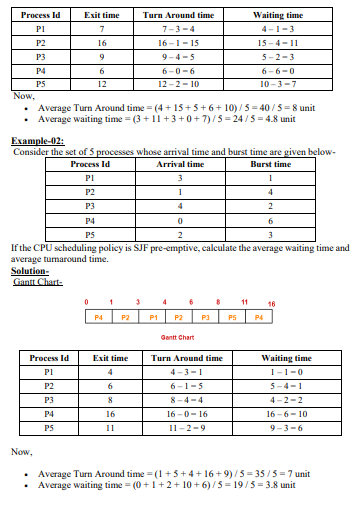
• It can not be implemented practically since burst time of the processes can not be known in advance.

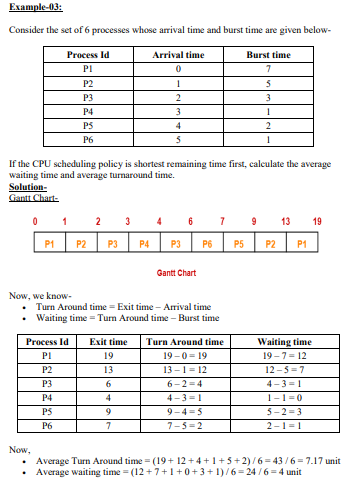
• It leads to starvation for processes with larger burst time.

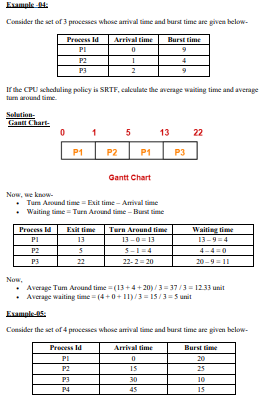
• Priorities can not be set for the processes.

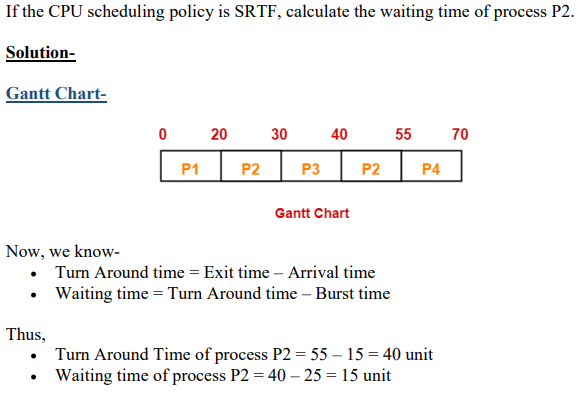
• Processes with larger burst time have poor response time











Round Robin Scheduling

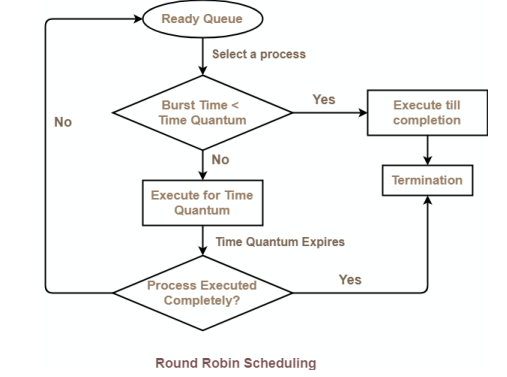
• CPU is assigned to the process on the basis of FCFS for a fixed amount of time.

• This fixed amount of time is called as time quantum or time slice.

• After the time quantum expires, the running process is preempted and sent to the ready queue.

• Then, the processor is assigned to the next arrived process.

• It is always preemptive in nature.



Advantages-

• It gives the best performance in terms of average response time.

• It is best suited for time sharing system, client server architecture and interactive system. Disadvantages-

• It leads to starvation for processes with larger burst time as they have to repeat the cycle many times.

• Its performance heavily depends on time quantum.

• Priorities can not be set for the processes.

With decreasing value of time quantum,

• Number of context switch increases

• Response time decreases

• Chances of starvation decreases Thus, smaller value of time quantum is better in terms of response time. With increasing value of time quantum,

• Number of context switch decreases

• Response time increases

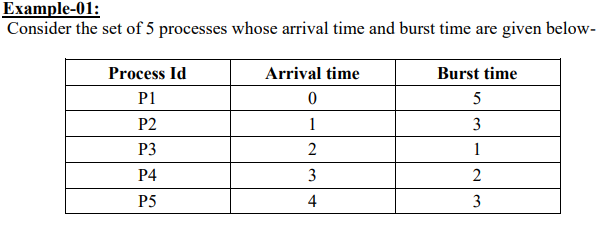
• Chances of starvation increases Thus, higher value of time quantum is better in terms of number of context switch.

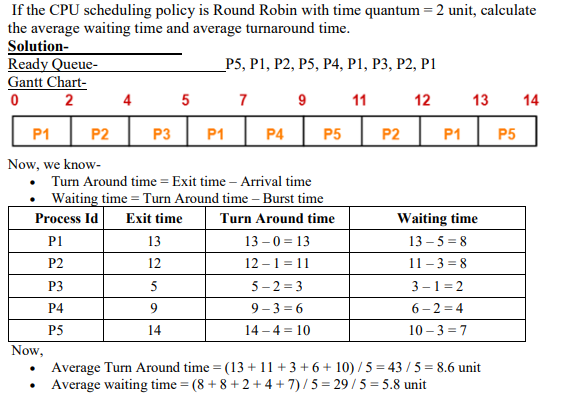
• With increasing value of time quantum, Round Robin Scheduling tends to become FCFS Scheduling.

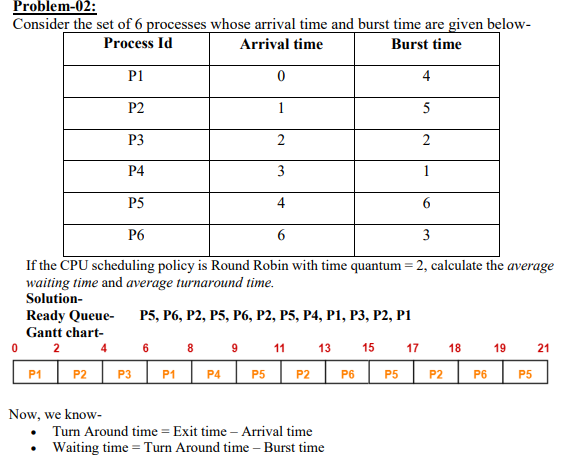
• When time quantum tends to infinity, Round Robin Scheduling becomes FCFS Scheduling.

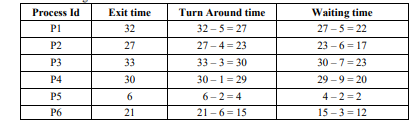
• The performance of Round Robin scheduling heavily depends on the value of time quantum.

* The value of time quantum should be such that it is neither too big nor too small.

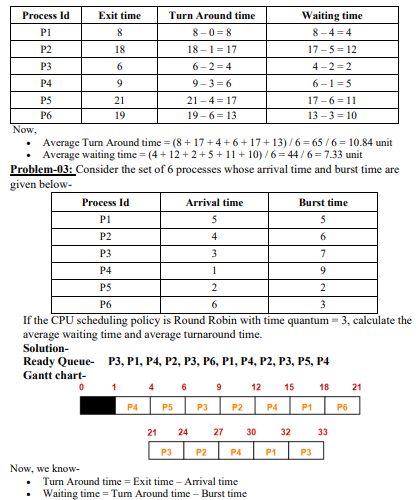










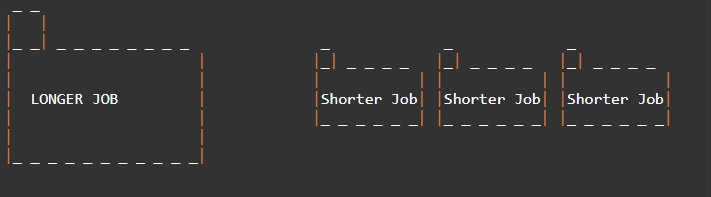


## Convoy Effect In First Come First Serve (FCFS )

Convoy Effect is a phenomenon which occurs in the Scheduling Algorithm named First Come First Serve (FCFS). The First Come First Serve Scheduling Algorithm occurs in a way of non preemptive way.

The Non preemptive way means that if a process or job is started execution, then the operating system must complete its process or job. Until, the process or job is zero the new or next process or job does not start its execution. The definition of Non Preemptive Scheduling in terms of Operating System means that the Central Processing Unit (CPU) will be completely dedicated till the end of the process or job started first and the new process or job is executed only after finishing of the older process or job.

There may be a few cases, which might cause the Central Processing Unit (CPU) to allot a too much time. This is because in the First Come First Serve Scheduling Algorithm Non Preemptive Approach, the Processes or the jobs are chosen in serial order. Due, to this shorter jobs or processes behind the larger processes or jobs takes too much time to complete its execution. Due, to this the Waiting Time, Turn Around Time, Completion Time is very high.



So, here as the first process is large or completion time is too high, then this Convoy effect in the First Come First Serve Algorithm is occurred.

Let us assume that Longer Job takes infinite time to complete. Then, the remaining processes have to wait for the same infinite time. Due to this Convoy Effect created by the Longer Job the Starvation of the waiting processes increases very rapidly. This is the biggest disadvantage of FCFS CPU Process Scheduling.

Characteristics of FCFS CPU Process Scheduling

The characteristics of FCFS CPU Process Scheduling are:

1. Implementation is simple.
2. Does not cause any causalities while using
3. It adopts a non pre emptive and pre emptive strategy.
4. It runs each procedure in the order that they are received.
5. Arrival time is used as a selection criterion for procedures.

Advantages of FCFS CPU Process Scheduling

The advantages of FCFS CPU Process Scheduling are:

1. In order to allocate processes, it uses the First In First Out queue.
2. The FCFS CPU Scheduling Process is straight forward and easy to implement.
3. In the FCFS situation pre emptive scheduling, there is no chance of process starving.
4. As there is no consideration of process priority, it is an equitable algorithm.

Disadvantages of FCFS CPU Process Scheduling

The disadvantages of FCFS CPU Process Scheduling are:

* FCFS CPU Scheduling Algorithm has Long Waiting Time
* FCFS CPU Scheduling favors CPU over Input or Output operations
* In FCFS there is a chance of occurrence of Convoy Effect
* Because FCFS is so straight forward, it often isn't very effective. Extended waiting periods go hand in hand with this. All other orders are left idle if the CPU is busy processing one time-consuming order.

Problems in the First Come First Serve CPU Scheduling Algorithm

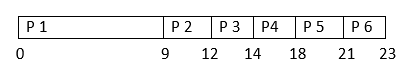
Example

1. S. No      Process ID      Process Name      Arrival Time      Burst Time
2. \_ \_ \_     \_ \_ \_ \_ \_ \_     \_ \_ \_ \_ \_ \_ \_     \_ \_ \_ \_ \_ \_ \_     \_ \_ \_ \_ \_ \_ \_
3. 1           P 1            A                  0                 9
4. 2           P 2            B                  1                 3
5. 3           P 3            C                  1                 2
6. 4           P 4            D                  1                 4
7. 5           P 5            E                  2                 3
8. 6           P 6            F                  3                 2

Non Pre Emptive Approach

Now, let us solve this problem with the help of the Scheduling Algorithm named First Come First Serve in a Non Preemptive Approach.

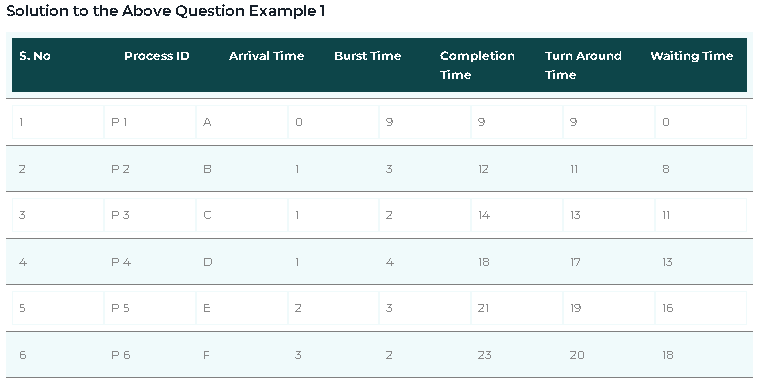
Gantt chart for the above Example 1 is:



Turn Around Time = Completion Time - Arrival Time

Waiting Time = Turn Around Time - Burst Time

### Solution to the Above Question Example 1



**The Average Completion Time is:**

Average CT = ( 9 + 12 + 14 + 18 + 21 + 23 ) / 6

Average CT = 97 / 6

Average CT = 16.16667

**The Average Waiting Time is:**

Average WT = ( 0 + 8 + 11 + 13 + 16 + 18 ) /6

Average WT = 66 / 6

Average WT = 11

**The Average Turn Around Time is:**

Average TAT = ( 9 + 11 + 13 + 17 + 19 +20 ) / 6

Average TAT = 89 / 6

Average TAT = 14.83334

This is how the FCFS is solved in Non Pre Emptive Approach.

Now, let us understand how they can be solved in Pre Emptive Approach

### Pre Emptive Approach

Now, let us solve this problem with the help of the Scheduling Algorithm named First Come First Serve in a Pre Emptive Approach.

In Pre Emptive Approach we search for the best process which is available

Gantt chart for the above Example 1 is:

FCFS Scheduling Algorithms in OS (Operating System)

