**UNIT-2:**

1. **Process Management in OS**

* A Program does nothing unless its instructions are executed by a CPU. A program in execution is called a process. In order to accomplish its task, process needs the computer resources.
* There may exist more than one process in the system which may require the same resource at the same time. Therefore, the operating system has to manage all the processes and the resources in a convenient and efficient way.
* Some resources may need to be executed by one process at one time to maintain the consistency otherwise the system can become inconsistent and deadlock may occur.
* The operating system is responsible for the following activities in connection with Process Management

1. Scheduling processes and threads on the CPUs.
2. Creating and deleting both user and system processes.
3. Suspending and resuming processes.
4. Providing mechanisms for process synchronization.
5. Providing mechanisms for process communication.

# Attributes of a process

🡺The Attributes of the process are used by the Operating System to create the process control block (PCB) for each of them.

🡺This is also called context of the process.

🡺Attributes which are stored in the PCB are described below.

### Process ID

### 🡺When a process is created, a unique id is assigned to the process which is used for unique identification of the process in the system.

### Program counter

🡺A program counter stores the address of the last instruction of the process on which the process was suspended. The CPU uses this address when the execution of this process is resumed.

### Process State

🡺The Process, from its creation to the completion, goes through various states which are new, ready, running and waiting. We will discuss about them later in detail.

### Priority

🡺Every process has its own priority. The process with the highest priority among the processes gets the CPU first. This is also stored on the process control block.

### General Purpose Registers

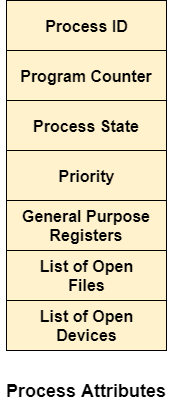
🡺Every process has its own set of registers which are used to hold the data which is generated during the execution of the process.

### List of open files

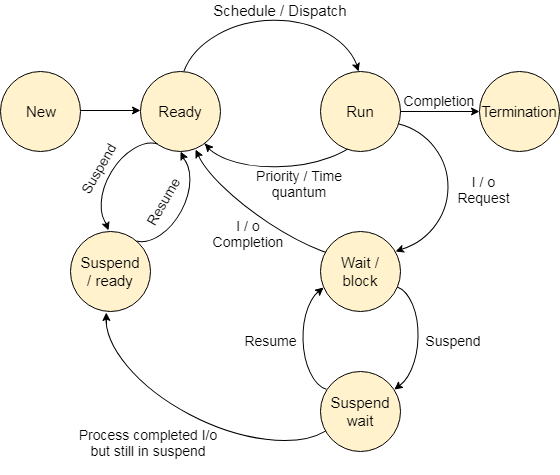
🡺During the Execution, Every process uses some files which need to be present in the main memory. OS also maintains a list of open files in the PCB.

### List of open devices

🡺OS also maintain the list of all open devices which are used during the execution of the process.



1. **State Diagram**



* The process, from its creation to completion, passes through various states.
* The minimum number of states is five.

### New

🡺A program which is going to be picked up by the OS into the main memory is called a new process.

### Ready

🡺Whenever a process is created, it directly enters in the ready state, in which, it waits for the CPU to be assigned. The OS picks the new processes from the secondary memory and put all of them in the main memory.

🡺The processes which are ready for the execution and reside in the main memory are called ready state processes. There can be many processes present in the ready state.

### Running

🡺One of the processes from the ready state will be chosen by the OS depending upon the scheduling algorithm. Hence, if we have only one CPU in our system, the number of running processes for a particular time will always be one. If we have n processors in the system then we can have n processes running simultaneously.

### Block or wait

🡺From the Running state, a process can make the transition to the block or wait state depending upon the scheduling algorithm or the intrinsic behavior of the process.

🡺When a process waits for a certain resource to be assigned or for the input from the user then the OS move this process to the block or wait state and assigns the CPU to the other processes.

### Completion or termination

🡺When a process finishes its execution, it comes in the termination state. All the context of the process (Process Control Block) will also be deleted the process will be terminated by the Operating system.

### Suspend ready

🡺A process in the ready state, which is moved to secondary memory from the main memory due to lack of the resources (mainly primary memory) is called in the suspend ready state.

🡺If the main memory is full and a higher priority process comes for the execution then the OS have to make the room for the process in the main memory by throwing the lower priority process out into the secondary memory. The suspend ready processes remain in the secondary memory until the main memory gets available.

### Suspend wait

🡺Instead of removing the process from the ready queue, it's better to remove the blocked process which is waiting for some resources in the main memory. Since it is already waiting for some resource to get available hence it is better if it waits in the secondary memory and make room for the higher priority process. These processes complete their execution once the main memory gets available and their wait is finished.

* Operations on the Process

### 1. Creation

Once the process is created, it will be ready and come into the ready queue (main memory) and will be ready for the execution.

### 2. Scheduling

Out of the many processes present in the ready queue, the Operating system chooses one process and start executing it. Selecting the process which is to be executed next, is known as scheduling.

### 3. Execution

Once the process is scheduled for the execution, the processor starts executing it. Process may come to the blocked or wait state during the execution then in that case the processor starts executing the other processes.

### 4. Deletion/killing

Once the purpose of the process gets over then the OS will kill the process. The Context of the process (PCB) will be deleted and the process gets terminated by the Operating system.

# Process Scheduling in OS (Operating System)

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

### 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.

1. 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.

### 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.

# Process control block:

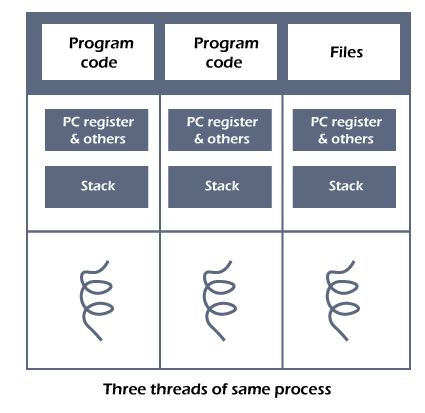
* The Process Control Block is a collection of process-related data and information.
* While the specifics vary based on the operating system, a typical PCB has the following components:
  1. **Process ID(PID):** A distinct Process ID (PID) on the PCB serves as the process's identifier within the operating system. The operating system uses this ID to keep track of, manage, and differentiate among processes.
  2. **Process State:** The state of the process, such as running, waiting, ready, or terminated, is indicated. The operating system makes use of this data to schedule and manage operations.
  3. **Program Counter(PC):** The program counter value, which indicates the address of the following instruction to be performed in the process, is stored on the PCB. The program counter is saved in the PCB of the running process during context switches and then restored to let execution continue where it left off.
  4. **CPU registers:** Looks at how the process's associated CPU registers are now working. Examples include ***stack pointers***, general-purpose registers, and ***program status flags***. Processes can continue operating uninterrupted during context changes by saving and restoring register values.
  5. **Memory Management Information:** Includes the process's memory allocation information, such as the ***base and limit registers or page tables***. This information allows the operating system to manage the process's memory requirements appropriately.
  6. **Priority:**Some operating systems provide a priority value to each process to decide the order in which processes receive CPU time. The PCB may have a priority field that determines the process's priority level, allowing the scheduler to distribute CPU resources appropriately.
  7. **Parent Process ID(PPID):**The PID of the parent process that gave rise to the present process. This data is important for process management and tracking process linkages, particularly in scenarios requiring process hierarchy or process tree architectures.
  8. **I/O status:** The PCB maintains information about I/O devices and data related to the process. Open ***file descriptors, I/O buffers***, and pending I/O requests are all included. Storing this information enables the operating system to manage I/O operations and efficiently handle input/output requests.
  9. **Accounting information:** Keeps track of the process's resource utilization data, such as ***CPU time, memory usage, and I/O activities***. This data aids in performance evaluation and resource allocation choices.
  10. **Inter-Process Communication (IPC) information:** If a process communicates with other processes, the PCB may contain fields or pointers to communication channels, message queues, shared memory regions, or synchronization primitives. This allows processes to communicate and share data successfully.

# The Process Control Block is a fundamental data structure for the process management capability of the operating system. It enables the operating system to switch between processes, allocate resources, manage interruptions, and maintain the general stability and performance of the system. The PCB allows the operating system to retain control over process execution and guarantee optimal resource utilization inside the system by storing and maintaining critical process-related information.

# 

# Thread:

* A thread is a single sequential flow of execution of tasks of a process so it is also known as thread of execution or thread of control.
* There is a way of thread execution inside the process of any operating system. Apart from this, there can be more than one thread inside a process. Each thread of the same process makes use of a separate program counter and a stack of activation records and control blocks. Thread is often referred to as a lightweight process.



* The process can be split down into so many threads. **For example**, in a browser, many tabs can be viewed as threads. MS Word uses many threads - formatting text from one thread, processing input from another thread, etc.

## Need of Thread:

* It takes far less time to create a new thread in an existing process than to create a new process.
* Threads can share the common data, they do not need to use Inter- Process communication.
* Context switching is faster when working with threads.
* It takes less time to terminate a thread than a process.

## Types of Threads

1. Kernel level thread.
2. User-level thread.

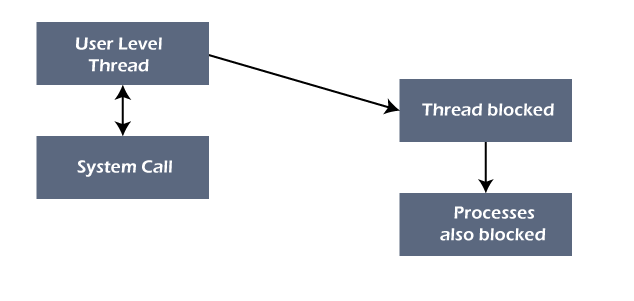
### User-level thread

* The [operating system](https://www.tpointtech.com/operating-system) does not recognize the user-level thread. User threads can be easily implemented and it is implemented by the user. If a user performs a user-level thread blocking operation, the whole process is blocked. The kernel level thread does not know nothing about the user level thread. The kernel-level thread manages user-level threads as if they are single-threaded processes?examples: [Java](https://www.tpointtech.com/java-tutorial) thread, POSIX threads, etc.
* **Advantages of User-level threads**

1. The user threads can be easily implemented than the kernel thread.
2. User-level threads can be applied to such types of operating systems that do not support threads at the kernel-level.
3. It is faster and efficient.
4. Context switch time is shorter than the kernel-level threads.
5. It does not require modifications of the operating system.
6. User-level threads representation is very simple. The register, PC, stack, and mini thread control blocks are stored in the address space of the user-level process.
7. It is simple to create, switch, and synchronize threads without the intervention of the process.

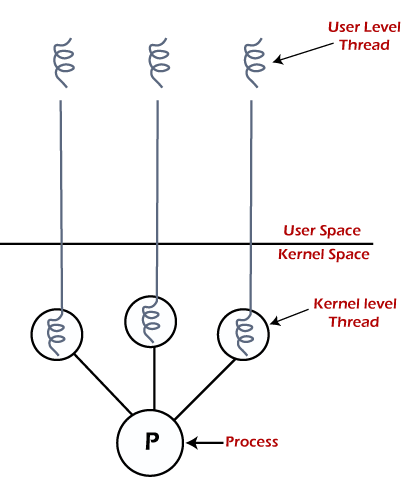
* **Disadvantages of User-level threads**

1. User-level threads lack coordination between the thread and the kernel.
2. If a thread causes a page fault, the entire process is blocked.



### Kernel level thread

* The kernel thread recognizes the operating system. There is a thread control block and process control block in the system for each thread and process in the kernel-level thread.
* The kernel-level thread is implemented by the operating system. The kernel knows about all the threads and manages them. The kernel-level thread offers a system call to create and manage the threads from user-space.
* The implementation of kernel threads is more difficult than the user thread. Context switch time is longer in the kernel thread. If a kernel thread performs a blocking operation, the Banky thread execution can continue.
* Example: Window Solaris.



* **Advantages of Kernel-level threads**

1. The kernel-level thread is fully aware of all threads.
2. The scheduler may decide to spend more CPU time in the process of threads being large numerical.
3. The kernel-level thread is good for those applications that block the frequency.

* **Disadvantages of Kernel-level threads**

1. The kernel thread manages and schedules all threads.
2. The implementation of kernel threads is difficult than the user thread.
3. The kernel-level thread is slower than user-level threads.

## Components of Threads

Any thread has the following components.

1. Program counter
2. Register set
3. Stack space

## Benefits of Threads

* **Enhanced throughput of the system:** When the process is split into many threads, and each thread is treated as a job, the number of jobs done in the unit time increases. That is why the throughput of the system also increases.
* **Effective Utilization of Multiprocessor system:** When you have more than one thread in one process, you can schedule more than one thread in more than one processor.
* **Faster context switch:** The context switching period between threads is less than the process context switching. The process context switch means more overhead for the CPU.
* **Responsiveness:** When the process is split into several threads, and when a thread completes its execution, that process can be responded to as soon as possible.
* **Communication:** Multiple-thread communication is simple because the threads share the same address space, while in process, we adopt just a few exclusive communication strategies for communication between two processes.
* **Resource sharing:** Resources can be shared between all threads within a process, such as code, data, and files. Note: The stack and register cannot be shared between threads. There is a stack and register for each thread.

# Process VS thread:

# 

# 

# 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.
* 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.

* CPU scheduler?
* A **CPU Scheduler** is a key component of the operating system that decides **which process will use the CPU next** when there are multiple processes ready to execute.
* The CPU scheduler **selects a process** from the **ready queue** and **allocates the CPU** to it.
* In a multitasking system, many processes may be waiting to use the CPU. The scheduler ensures:
  1. Efficient CPU utilization
  2. Fairness (no process is starved)
  3. Quick response time for interactive systems

# Scheduling criteria:

## Importance of Selecting the Right CPU Scheduling Algorithm for Specific Situations:

1. Selecting the right CPU scheduling algorithm is crucial for optimizing system performance & ensuring the efficient utilization of resources. Different scheduling algorithms have different strengths & weaknesses, & they are suited for different types of workloads & system requirements.
2. For example, in a system with a mix of CPU-bound & I/O-bound processes, a scheduling algorithm that prioritizes I/O-bound processes, such as the Shortest Job Next (SJN) algorithm, can help reduce the average waiting time & improve overall system performance. On the other hand, in a system with mostly CPU-bound processes, a scheduling algorithm that focuses on maximizing CPU utilization, such as the Round Robin (RR) algorithm, may be more appropriate.
3. Moreover, the choice of scheduling algorithm can also impact the user experience. In an interactive system, such as a desktop computer, a scheduling algorithm that provides good response times, such as the Priority Scheduling algorithm, is important for ensuring a smooth user experience. In contrast, in a batch processing system, such as a scientific computing cluster, a scheduling algorithm that optimizes throughput, such as the First-Come, First-Served (FCFS) algorithm, may be more suitable.
4. Therefore, it is essential to carefully consider the specific requirements & characteristics of the system & the workload when selecting a CPU scheduling algorithm. Choosing the right algorithm can significantly impact system performance, resource utilization, & user satisfaction.

## Factors Influencing CPU Scheduling Algorithms

1. **Process Priority**: Some processes may have higher priority than others based on their importance or urgency. Priority-based scheduling algorithms, such as Priority Scheduling, take process priority into account when making scheduling decisions.
2. **Process Type**: Processes can be classified as CPU-bound or I/O-bound. CPU-bound processes spend most of their time performing computations, while I/O-bound processes spend most of their time waiting for I/O operations to complete. Scheduling algorithms may treat these process types differently to optimize system performance.
3. **System Load**: The number & characteristics of processes in the ready queue can affect the performance of scheduling algorithms. Some algorithms may perform better under heavy load, while others may be more suitable for light load conditions.
4. **Context Switching:** Context switching occurs when the CPU switches from one process to another. It involves saving the state of the current process & loading the state of the next process. Frequent context switching can overhead & impact system performance. Scheduling algorithms that minimize context switching, such as the Shortest Remaining Time First (SRTF) algorithm, can help reduce this overhead.
5. **Preemption:** Preemptive scheduling algorithms allow the CPU to be taken away from a running process & allocated to another process based on certain criteria, such as priority or shortest remaining time. Non-preemptive algorithms, on the other hand, allow a process to run until it voluntarily releases the CPU or terminates. The choice between preemptive & non-preemptive scheduling depends on the specific requirements of the system.
6. **Fairness:** Scheduling algorithms should ensure fair allocation of CPU time among processes. Fairness can be achieved through techniques such as time-slicing, where each process is given a fixed amount of CPU time in a cyclic manner, as in the Round Robin algorithm.

# Context switching:

* The Context switching is a technique or method used by the operating system to switch a process from one state to another to execute its function using CPUs in the system.
* When switching perform in the system, it stores the old running process's status in the form of registers and assigns the [CPU](https://www.tpointtech.com/cpu-full-form) to a new process to execute its tasks. While a new process is running in the system, the previous process must wait in a ready queue. The execution of the old process starts at that point where another process stopped it.
* The need for Context switching:

1. The switching of one process to another process is not directly in the system. A context switching helps the operating system that switches between the multiple processes to use the CPU's resource to accomplish its tasks and store its context. We can resume the service of the process at the same point later. If we do not store the currently running process's data or context, the stored data may be lost while switching between processes.
2. If a high priority process falls into the ready queue, the currently running process will be shut down or stopped by a high priority process to complete its tasks in the system.
3. If any running process requires I/O resources in the system, the current process will be switched by another process to use the CPUs. And when the I/O requirement is met, the old process goes into a ready state to wait for its execution in the CPU. Context switching stores the state of the process to resume its tasks in an operating system. Otherwise, the process needs to restart its execution from the initials level.
4. If any interrupts occur while running a process in the operating system, the process status is saved as registers using context switching. After resolving the interrupts, the process switches from a wait state to a ready state to resume its execution at the same point later, where the operating system interrupted occurs.
5. A context switching allows a single CPU to handle multiple process requests simultaneously without the need for any additional processors.

### Context switching triggers

1. Interrupts
2. Multitasking
3. Kernel/User switch

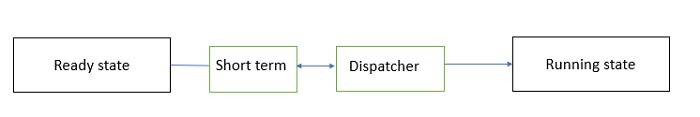
**Interrupts**: A CPU requests for the data to read from a disk, and if there are any interrupts, the context switching automatic switches a part of the hardware that requires less time to handle the interrupts.

**Multitasking**: A context switching is the characteristic of multitasking that allows the process to be switched from the CPU so that another process can be run. When switching the process, the old state is saved to resume the process's execution at the same point in the system.

**Kernel/User Switch**: It is used in the operating systems when switching between the user mode, and the kernel/user mode is performed.

# Dispatcher:

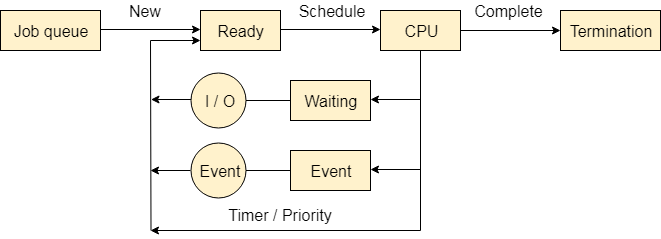
* The dispatcher is done after the scheduler. It gives control of the CPU to the process selected by the short-term scheduler. After selecting the process, the dispatcher gives CPU to it.
* Functions
  1. Switching context.
  2. Switching to user mode.
* The dispatcher is given below −



|  |  |  |
| --- | --- | --- |
| BASIS | SCHEDULER | DISPATCHER |
| Definition | Decides which process should be executed next. | Gives control of the CPU to the selected process. |
| Function | Selects a process from the ready queue. | Performs the context switching to start execution. |
| Main role | Process selection for execution. | Switching the CPU to the selected process. |
| Involves | Scheduling algorithms (FCFS, SJF, Round Robin, etc.). | Context switch, switching to user mode, and jumping to process location. |
| Types | Logical decision-maker. | Physical executor of the decision. |
| Time taken | Usually negligible compared to dispatcher. | Time taken is known as dispatch latency. |
| Execution frequency | Occurs periodically based on process state changes | Invoked each time a new process is dispatched. |
| Dependency | Works independently before dispatcher takes action. | Depends on the scheduler’s decision. |
| Complexity | More complex due to algorithm selection and fairness. | Less complex, deals with the mechanics of process switching. |
| Example task | Choosing the next process based on priority or time slice. | Saving context of old process and loading context of new process. |

# 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.



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.

### 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.

### 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.

# Types of CPU scheduling:

# Categories of Process Scheduling

# Non-preemptive Scheduling

# Non-preemptive scheduling algorithms refer to the class of CPU scheduling technique where once a process is allocated the CPU, it holds the CPU till the process gets terminated or is pushed to the waiting state. No process is interrupted until it runs to completion. The scheduler allocates another process to the CPU only after the currently allocated process terminates and relinquishes control on the CPU.

# Example of Non-preemptive Scheduling

# First-Come-First-Serve (FCFS) Scheduling

# Shortest Job First (SJF) Scheduling

# Priority Non-preemptive Scheduling

# Highest Response Ratio Next (HRRN) Scheduling

# Advantages of Non-preemptive Scheduling

# Non-preemptive scheduling methods are simpler to design and implement.

# These techniques need lesser resources and have lower overhead.

# They require lesser number of context switches. So, the amount of time expended for scheduling is very less in comparison to execution of the processes.

# They are deterministic in nature and the scheduling outputs are easily predictable.

# Disadvantages of Non-preemptive Scheduling

# Non-preemptive scheduling cannot respond to dynamically changing system.

# It may result in deadlock if there are processes in the system which are holding some resources which waiting for other resources that are held by other processes.

# If the executing process erroneously enter infinite loop, it may cause system crash since no other process can interrupt the execution.

# Preemptive Scheduling

# Preemptive scheduling algorithms fall under the category of process scheduling technique in which a running process can be interrupted by another process and sent to the ready queue even when it has not completed its entire execution in CPU.

# An executing process may be pre-empted under the following situations –

# The first scenario is when the process has reached the limit of the time slot allotted to it. If the burst time of the process is greater than CPU cycle, it is placed back into the ready queue and will execute in the next chance.

# The second scenario is when a higher priority process enters the system. The process state of the executing process is saved in the process table and the higher priority process starts to execute.

# Another scenario is when a short process enters the system which uses Shortest Remaining Time Next scheduling algorithm. If a longer process is executing, it is pre-empted out to let the shorter process execute.

# Example of Preemptive Scheduling

# Round Robin (RR) Scheduling

# Shortest Remaining Time Next (SJF Preemptive)

# Priority Preemptive Scheduling

# Advantages of Preemptive Scheduling

# No process can monopolize the CPU.

# Fair scheduling can be applied.

# Preemptive scheduling improves the average response time.

# They allow reconsideration of scheduling decision after each pre-emption, thus making the scheduling decision more adaptive to change.

# Deadlocks can be avoided by preempting processes.

# Disadvantages of Preemptive Scheduling

# Preemptive scheduling require frequent context switching which is time-consuming.

# This requires more memory since all context information for each process needs to be stored and updated frequently.

# Often a low priority process or a long process has to wait for long time due to influx of many higher priority or shorter processes in the system.

# The processes must be appropriately coded so that they can be switched at all times without information loss. This leads to complex processes.

# 

1. CPU scheduling algorithms:
   1. FCFS
   2. SJF
   3. RR
   4. PRIORITY
   5. MULTILEVEL QUEUE
   6. MULTILEVEL FEEDBACK QUEUE
2. MULTILEVEL FEEDBACK QUEUE

* The **Multilevel Feedback Queue (MLFQ)** is an advanced **CPU scheduling algorithm** that uses **multiple queues** with **different priority levels** and allows processes to **move between queues** based on their behavior and requirements.
* **How it Works:**

1. **Multiple Queues:**

* Each queue has its own **scheduling algorithm** (e.g., higher priority queue may use **Round Robin**, while lower ones use **FCFS**).

1. **Feedback Mechanism:**

* If a process **uses too much CPU time**, it gets **moved to a lower-priority queue**.
* If a process **waits too long**, it may be moved to a **higher-priority queue** (to prevent starvation).

1. **Dynamic Behavior:**

* Unlike fixed-priority scheduling, MLFQ **adapts** to the behavior of processes (I/O-bound vs CPU-bound).

### **Advantages of MLFQ:**

1. **Highly Flexible:**

* Adapts to various process types (interactive, batch, background).

1. **Prevents Starvation:**

* Using techniques like **aging** or priority boosts, it ensures **fairness**.

1. **Efficient Resource Use:**

* Prioritizes short or I/O-bound jobs, improving **system responsiveness**.

1. **Combines Best Features:**

* Merges benefits of **Round Robin**, **FCFS**, and **priority scheduling**.

### **Disadvantages of MLFQ:**

1. **Complex Implementation:**

* Managing multiple queues, time quanta, and feedback rules adds **complexity**.

1. **Tuning is Difficult:**

* Needs **fine-tuning** of number of queues, time slices, and movement rules for best performance.

1. **Can Still Have Starvation (If Misconfigured):**

* Without proper aging, **lower-priority queues** may suffer.
* Examples are: