| **Preemptive Scheduling** | |
| --- | --- |
| A processor can be preempted to execute the different processes in the middle of any current process execution. | |
| **Round-Robin (RR) Algorithm** | **Shortest Remaining Time First (SRTF) Algorithm** |
| Each process in the ready state gets the CPU for a fixed time in a cycle way. | The process with the least burst time remaining is executed first.  The preemptive version of Shortest Job Next (SJN) algorithm. |
| **Advantages:**   * It doesn’t face the issues of starvation or convoy effect. * All the jobs get a fair allocation of CPU. * It deals with all process without any priority * If you know the total number of processes on the run queue, then you can also assume the worst-case response time for the same process. * This scheduling method does not depend upon burst time. That’s why it is easily implementable on the system. * Once a process is executed for a specific set of the period, the process is preempted, and another process executes for that given time period. * Allows OS to use the Context switching method to save states of preempted processes. * It gives the best performance in terms of average response time. | **Advantages:**   * Processes with a shorter burst time are executed quickly. * The system overhead is minimal since the system only needs to make a choice when a process completes its execution, or a new process is added to the queue. * Since it's a preemptive algorithm, whenever a new process is added to the queue, it just has to compare the presently executing process and the new one. |
| **Disadvantages:**   * If the slicing time of the OS is low, the processor output will be reduced. * This method spends more time on context switching * Its performance heavily depends on the time   quantum.   * Priorities cannot be set for the processes. * Round-robin scheduling doesn’t give special priority to more important tasks. * Decreases comprehension * Lower quantum results in higher context switching overhead in the system. * Finding a correct time quantum is a quite difficult task in this system. | **Disadvantages:**   * The context switch is done a lot more times significantly and consumes the CPU's important time for handling. This amounts to its handling time and decreases its benefit of quick handling. * It has the potential for process starvation since it always selects the shortest jobs first. * If the shorter process is continuously added, the longer processes may be held off indefinitely. |

| **Non-preemptive Scheduling** | | |
| --- | --- | --- |
| Once the processor starts its execution, it must finish it before executing the other. It can’t be paused in the middle. | | |
| **First Come, First Served (FCFS)** | **Shortest job first (SJF)** | **Priority-Based** |
| Efficiently and automatically execute queued tasks, processes and requests by the order of their arrival | Selects the waiting process with the smallest execution time to execute next. | Scheduling processes that is based on its given priority. |
| **Advantages:**   * Simplicity, Orders are completed in the sequence that they are placed * Scheduling can be quickly and easily implemented into any scheduling system that your organization possesses, with minimal time, effort, and expenses. * Orders are processed in the exact order that they are placed. | **Advantages:**   * Used for long term scheduling. * Reduces the average waiting time over FCFS. * Gives the lowest average waiting time for a specific set of processes. * Appropriate for the jobs running in batch, where run times are known in advance. * Probably optimal with regard to average turnaround time. | **Advantages:**   * Easy to use. * Processes with higher priority execute first which saves time. * The importance of each process is precisely defined. * A good algorithm for applications with fluctuating time and resource requirements. |
| **Disadvantages:**   * It doesn’t release the CPU until it finishes executing. * May cause starvation if the first job has the longest burst time. * High Average Waiting Time. * Short processes at the back of the queue have to wait for the long processes to finish execution. * Not ideal for time-sharing systems. * Inefficient (lower device utilization). | **Disadvantages:**   * Job completion time must be known earlier, but it is hard to predict. * It is often used in a batch system for long term scheduling. * Can’t be implemented for CPU scheduling for the short term. * Requires knowledge of how long a process or job will run. * It leads to starvation that does not reduce average turnaround time. * It is hard to know the length of the upcoming CPU request. | **Disadvantages:**   * We can lose all the low-priority processes if the system crashes. * This process can cause starvation if high-priority processes take too much CPU time. The lower priority process can also be postponed for an indefinite time. * There is a chance that a process can’t run even when it is ready as some other process is running currently. |