1.Explain the role of time quantum with respect to Round robin scheduling.

The time quantum is a fixed amount of time that is assigned to each process in round robin scheduling. Round robin scheduling is a CPU scheduling algorithm that is designed for time-sharing systems. It works by allocating the CPU to each process in a circular order for a time quantum, and preempting the process if it does not finish within that time. [The preempted process is then moved to the end of the ready queue and waits for its next turn1](https://www.arpapress.com/Volumes/Vol5Issue1/IJRRAS_5_1_09.pdf)[2](https://en.wikipedia.org/wiki/Round-robin_scheduling).

**Role of Time Quantum**

The time quantum plays an important role in round robin scheduling, as it affects the performance and fairness of the algorithm. The time quantum should be chosen carefully, as it has the following implications:

* If the time quantum is too large, the algorithm becomes similar to first-come first-served (FCFS) scheduling, which may result in poor response time and low CPU utilization. This is because some processes may have to wait for a long time before getting the CPU, while other processes may finish quickly and leave the CPU idle[1](https://www.arpapress.com/Volumes/Vol5Issue1/IJRRAS_5_1_09.pdf)[2](https://en.wikipedia.org/wiki/Round-robin_scheduling)[3](https://ieeexplore.ieee.org/document/4741092).
* If the time quantum is too small, the algorithm becomes similar to processor sharing, which may result in high context switching overhead and low throughput. This is because the processes may spend more time switching between the CPU and the ready queue than executing on the CPU[1](https://www.arpapress.com/Volumes/Vol5Issue1/IJRRAS_5_1_09.pdf)[2](https://en.wikipedia.org/wiki/Round-robin_scheduling)[3](https://ieeexplore.ieee.org/document/4741092).
* If the time quantum is optimal, the algorithm achieves a balance between response time and throughput, as well as fairness among processes. The optimal time quantum depends on various factors, such as the number and characteristics of processes, the system load, and the hardware configuration[1](https://www.arpapress.com/Volumes/Vol5Issue1/IJRRAS_5_1_09.pdf)[2](https://en.wikipedia.org/wiki/Round-robin_scheduling)[3](https://ieeexplore.ieee.org/document/4741092).

**Finding Optimal Time Quantum**

There is no general formula or method to find the optimal time quantum for round robin scheduling, as it may vary depending on different scenarios and objectives. However, some heuristics or guidelines can be used to estimate a reasonable time quantum, such as:

* The time quantum should be larger than the context switch time, otherwise the context switch overhead will dominate the CPU utilization[1](https://www.arpapress.com/Volumes/Vol5Issue1/IJRRAS_5_1_09.pdf)[2](https://en.wikipedia.org/wiki/Round-robin_scheduling)[3](https://ieeexplore.ieee.org/document/4741092).
* [The time quantum should be smaller than the average burst time of processes, otherwise the response time will be degraded1](https://www.arpapress.com/Volumes/Vol5Issue1/IJRRAS_5_1_09.pdf)[2](https://en.wikipedia.org/wiki/Round-robin_scheduling)[3](https://ieeexplore.ieee.org/document/4741092).
* [The time quantum should be proportional to the priority or importance of processes, such that higher priority processes get more CPU time than lower priority processes1](https://www.arpapress.com/Volumes/Vol5Issue1/IJRRAS_5_1_09.pdf)[2](https://en.wikipedia.org/wiki/Round-robin_scheduling)[3](https://ieeexplore.ieee.org/document/4741092).
* The time quantum should be adaptive or dynamic, such that it can adjust to changing system conditions and process requirements[1](https://www.arpapress.com/Volumes/Vol5Issue1/IJRRAS_5_1_09.pdf)[2](https://en.wikipedia.org/wiki/Round-robin_scheduling)[3](https://ieeexplore.ieee.org/document/4741092).

**Example**

Here is an example of how round robin scheduling works with a given time quantum. Consider five processes P1, P2, P3, P4 and P5 with their arrival times and burst times as follows:

| **Process** | **Arrival Time** | **Burst Time** |
| --- | --- | --- |
| P1 | 0 | 10 |
| P2 | 0 | 29 |
| P3 | 0 | 3 |
| P4 | 0 | 7 |
| P5 | 0 | 12 |

Assume that the context switch time is negligible and that the time quantum is 10 units. The Gantt chart for round robin scheduling with this example is shown below:

|P1|P2|P3|P4|P5|P1|P2|P4|P5|P2|

0 10 20 23 30 42 52 62 69 81 90

Copy

The waiting time and turnaround time for each process are calculated as follows:

| **Process** | **Waiting Time** | **Turnaround Time** |
| --- | --- | --- |
| P1 | 32 | 42 |
| P2 | 41 | 70 |
| P3 | 20 | 23 |
| P4 | 23 | 30 |
| P5 | 30 | 42 |

The average waiting time and average turnaround time are calculated as follows:

Average waiting time = (32 + 41 + 20 + 23 + 30) / 5 = 29.2 units

Average turnaround time = (42 + 70 + 23 + 30 + 42) / 5 = 41.4 units

2.What is the problem encountered in priority scheduling? Explain the solution it ?

The problem encountered in priority scheduling is indefinite blocking or starvation. This is a situation where a low-priority process may never get the CPU if there are always higher-priority processes arriving and requesting the CPU. This can lead to poor performance and unfairness for the low-priority process[1](https://www.geeksforgeeks.org/program-for-priority-cpu-scheduling-set-1/)[2](https://www.guru99.com/priority-scheduling-program.html).

The solution to the problem of indefinite blocking or starvation is aging. Aging is a technique of gradually increasing the priority of processes that wait in the system for a long period of time. [This way, the low-priority process will eventually get a chance to execute on the CPU when its priority becomes higher than the other processes1](https://www.geeksforgeeks.org/program-for-priority-cpu-scheduling-set-1/)[2](https://www.guru99.com/priority-scheduling-program.html)[3](https://www.easycodingzone.com/2021/06/c-program-of-priority-scheduling.html).

## Example

Here is an example of how aging can prevent starvation in priority scheduling. Consider four processes P1, P2, P3 and P4 with their arrival times, burst times and priorities as follows:

| **Process** | **Arrival Time** | **Burst Time** | **Priority** |
| --- | --- | --- | --- |
| P1 | 0 | 10 | 3 |
| P2 | 0 | 1 | 1 |
| P3 | 0 | 2 | 4 |
| P4 | 1 | 1 | 5 |

Assume that the context switch time is negligible and that the priority scheduling is preemptive. The Gantt chart for priority scheduling without aging is shown below:

|P2|P1|P3|P4|

0 1 11 13 14

Copy

The waiting time and turnaround time for each process are calculated as follows:

| **Process** | **Waiting Time** | **Turnaround Time** |
| --- | --- | --- |
| P1 | 1 | 11 |
| P2 | 0 | 1 |
| P3 | 11 | 13 |
| P4 | 12 | 13 |

The average waiting time and average turnaround time are calculated as follows:

Average waiting time = (1 + 0 + 11 + 12) / 4 = 6 units

Average turnaround time = (11 + 1 + 13 + 13) / 4 = 9.5 units

Copy

Note that process P4 has to wait for a long time before getting the CPU, even though it has a very short burst time. This is because it has the lowest priority among the processes.

Now, assume that the aging factor is 1, which means that the priority of a process increases by 1 for every unit of time it waits in the ready queue. The Gantt chart for priority scheduling with aging is shown below:

|P2|P1|P4|P3|

0 1 11 12 14

Copy

The waiting time and turnaround time for each process are calculated as follows:

| **Process** | **Waiting Time** | **Turnaround Time** |
| --- | --- | --- |
| P1 | 1 | 11 |
| P2 | 0 | 1 |
| P3 | 12 | 14 |
| P4 | 11 | 12 |

The average waiting time and average turnaround time are calculated as follows:

Average waiting time = (1 + 0 + 12 + 11) / 4 = 6 units

Average turnaround time = (11 + 1 + 14 + 12) / 4 = 9.5 units

Copy

Note that process P4 gets the CPU earlier than before, as its priority increases from 5 to 6 after waiting for one unit of time. This way, aging prevents starvation for process P4.

3.Differentiate between Preemptive and Non-Preemptive scheduling. Discuss it with an example using shortest job first scheduling.

[Preemptive and non-preemptive scheduling are two types of CPU scheduling algorithms that differ in how they handle the running process when a new process arrives or becomes ready1](https://www.guru99.com/shortest-job-first-sjf-scheduling.html)[2](https://www.geeksforgeeks.org/preemptive-and-non-preemptive-scheduling/).

## Preemptive Scheduling

In preemptive scheduling, the running process can be interrupted and moved to the ready queue if a new process arrives with a higher priority or a shorter burst time. The interrupted process will resume its execution when it gets another chance to use the CPU. Preemptive scheduling allows more responsiveness and fairness, but it also incurs more overhead due to context switching and synchronization[1](https://www.guru99.com/shortest-job-first-sjf-scheduling.html)[2](https://www.geeksforgeeks.org/preemptive-and-non-preemptive-scheduling/).

## Non-Preemptive Scheduling

In non-preemptive scheduling, the running process cannot be interrupted and will continue to use the CPU until it finishes or blocks. The new process will have to wait in the ready queue until the running process releases the CPU. Non-preemptive scheduling reduces the overhead of context switching and synchronization, but it also may cause poor response time and starvation for some processes[1](https://www.guru99.com/shortest-job-first-sjf-scheduling.html)[2](https://www.geeksforgeeks.org/preemptive-and-non-preemptive-scheduling/).

## Example

Here is an example of how preemptive and non-preemptive scheduling work with shortest job first (SJF) algorithm. SJF is an algorithm that selects the process with the shortest burst time for the next execution. It can be either preemptive or non-preemptive[1](https://www.guru99.com/shortest-job-first-sjf-scheduling.html)[2](https://www.geeksforgeeks.org/preemptive-and-non-preemptive-scheduling/).

Consider four processes P1, P2, P3 and P4 with their arrival times and burst times as follows:

| **Process** | **Arrival Time** | **Burst Time** |
| --- | --- | --- |
| P1 | 0 | 8 |
| P2 | 1 | 4 |
| P3 | 2 | 9 |
| P4 | 3 | 5 |

Assume that the context switch time is negligible.

### Non-Preemptive SJF

In non-preemptive SJF, the running process will not be interrupted by a new process. The Gantt chart for non-preemptive SJF is shown below:

|P1|P2|P4|P3|

0 8 12 17 26

Copy

The waiting time and turnaround time for each process are calculated as follows:

| **Process** | **Waiting Time** | **Turnaround Time** |
| --- | --- | --- |
| P1 | 0 | 8 |
| P2 | 7 | 11 |
| P3 | 15 | 24 |
| P4 | 9 | 14 |

The average waiting time and average turnaround time are calculated as follows:

Average waiting time = (0 + 7 + 15 + 9) / 4 = 7.75 units

Average turnaround time = (8 + 11 + 24 + 14) / 4 = 14.25 units

Copy

### Preemptive SJF

In preemptive SJF, the running process can be interrupted by a new process with a shorter burst time. The Gantt chart for preemptive SJF is shown below:

|P1|P2|P4|P1|P3|

0 1 5 10 18 27

Copy

The waiting time and turnaround time for each process are calculated as follows:

| **Process** | **Waiting Time** | **Turnaround Time** |
| --- | --- | --- |
| P1 | 7 | 15 |
| P2 | 0 | 4 |
| P3 | 16 | 25 |
| P4 | 2 | 7 |

The average waiting time and average turnaround time are calculated as follows:

Average waiting time = (7 + 0 + 16 + 2) / 4 = 6.25 units

Average turnaround time = (15 + 4 + 25 +7) /4 =12.75 units

4.List out the CPU scheduling criteria and explain it.

CPU scheduling criteria are the measures or metrics that are used to evaluate and compare the performance of different CPU scheduling algorithms. CPU scheduling algorithms try to optimize some of these criteria, while minimizing others. [The choice of a particular algorithm depends on the objectives and requirements of the system1](https://www.geeksforgeeks.org/cpu-scheduling-criteria/)[2](https://www.geeksforgeeks.org/cpu-scheduling-in-operating-systems/).

**CPU Scheduling Criteria**

Some of the common CPU scheduling criteria are:

* CPU utilization: The main objective of any CPU scheduling algorithm is to keep the CPU as busy as possible. CPU utilization is the percentage of time that the CPU is executing processes, rather than being idle. Higher CPU utilization means better use of the system resources and lower overhead. [Theoretically, CPU utilization can range from 0 to 100 percent, but in practice, it varies depending on the system load and the type of processes1](https://www.geeksforgeeks.org/cpu-scheduling-criteria/)[2](https://www.geeksforgeeks.org/cpu-scheduling-in-operating-systems/)[3](https://www.guru99.com/cpu-scheduling-algorithms.html).
* Throughput: A measure of the work done by the CPU is the number of processes that complete their execution per unit of time. This is called throughput. Higher throughput means higher productivity and efficiency of the system. However, throughput may vary depending on the length or duration of the processes. For example, if there are many short processes, the throughput will be high, but if there are many long processes, the throughput will be low[1](https://www.geeksforgeeks.org/cpu-scheduling-criteria/)[2](https://www.geeksforgeeks.org/cpu-scheduling-in-operating-systems/)[3](https://www.guru99.com/cpu-scheduling-algorithms.html).
* Turnaround time: For a particular process, an important criterion is how long it takes to execute that process. The time elapsed from the time of submission of a process to the time of completion is known as the turnaround time. Turnaround time includes the waiting time, the execution time and the I/O time of a process. Lower turnaround time means faster completion and better performance for a process[1](https://www.geeksforgeeks.org/cpu-scheduling-criteria/)[2](https://www.geeksforgeeks.org/cpu-scheduling-in-operating-systems/)[3](https://www.guru99.com/cpu-scheduling-algorithms.html).
* Waiting time: A scheduling algorithm does not affect the time required to complete a process once it starts execution. It only affects the waiting time of a process, i.e., the time spent by a process waiting in the ready queue for its turn to use the CPU. Lower waiting time means less delay and better responsiveness for a process[1](https://www.geeksforgeeks.org/cpu-scheduling-criteria/)[2](https://www.geeksforgeeks.org/cpu-scheduling-in-operating-systems/)[3](https://www.guru99.com/cpu-scheduling-algorithms.html).
* Response time: In an interactive system, turnaround time may not be the best criterion to measure user satisfaction. A process may produce some output fairly early and continue computing new results while previous results are being output to the user. Thus another criterion is the time taken from submission of a request by a process until the first response is produced. This measure is called response time. Lower response time means better interactivity and user experience for a process[1](https://www.geeksforgeeks.org/cpu-scheduling-criteria/)[2](https://www.geeksforgeeks.org/cpu-scheduling-in-operating-systems/)[3](https://www.guru99.com/cpu-scheduling-algorithms.html).
* Completion time: The completion time is the time when a process stops executing, which means that the process has completed its burst time and is completely executed. Lower completion time means that a process has finished its work earlier and can free up its resources for other processes[4](http://www.facweb.iitkgp.ac.in/~isg/OS/SLIDES/ch6-CPU_Scheduling.pdf).
* Priority: If the operating system assigns priorities to processes, based on some criteria such as memory requirements, I/O requirements, user preference, etc., then the scheduling algorithm should favor the higher-priority processes over lower-priority processes. This way, more important or urgent processes can get more CPU attention and service than less important or urgent processes[1](https://www.geeksforgeeks.org/cpu-scheduling-criteria/)[2](https://www.geeksforgeeks.org/cpu-scheduling-in-operating-systems/)[3](https://www.guru99.com/cpu-scheduling-algorithms.html).
* Predictability: A given process should always run in about the same amount of time under similar system conditions. This means that there should not be much variation or fluctuation in the scheduling behavior or performance metrics for a process. Higher predictability means more reliability and consistency for a process[5](https://quescol.com/operating-system/cpu-scheduling-algorithms-criteria).

5.Write short notes on Multilevel feedback queue scheduling.

Multilevel feedback queue scheduling is a CPU scheduling algorithm that allows processes to move between multiple queues based on their behavior and needs. It is an extension of multilevel queue scheduling, which assigns processes to different queues based on their characteristics, such as priority, memory requirements, I/O requirements, etc. [However, unlike multilevel queue scheduling, multilevel feedback queue scheduling does not keep processes permanently assigned to a queue, but rather adjusts their priority and queue placement dynamically1](https://www.geeksforgeeks.org/multilevel-feedback-queue-scheduling-mlfq-cpu-scheduling/)[2](https://en.wikipedia.org/wiki/Multilevel_feedback_queue).

**Characteristics of Multilevel Feedback Queue Scheduling**

Some of the characteristics of multilevel feedback queue scheduling are:

* Multiple queues: Similar to multilevel queue scheduling, multilevel feedback queue scheduling divides processes into multiple queues based on their priority levels. However, unlike multilevel queue scheduling, processes can move between queues based on their behavior and needs.
* Priorities adjusted dynamically: The priority of a process can be adjusted dynamically based on its behavior, such as how much CPU time it has used or how often it has been blocked. Higher-priority processes are given more CPU time and lower-priority processes are given less.
* Time-slicing: Each queue is assigned a time quantum or time slice, which determines how much CPU time a process in that queue is allowed to use before it is preempted and moved to a lower priority queue.
* Feedback mechanism: Multilevel feedback queue scheduling uses a feedback mechanism to adjust the priority and queue placement of a process based on its behavior over time. For example, if a process in a lower-priority queue uses up its time slice, it may be moved to a higher-priority queue to ensure it gets more CPU time. Similarly, if a process in a higher-priority queue blocks for I/O frequently, it may be moved to a lower-priority queue to give other processes a chance to use the CPU.
* Preemption: Preemption is allowed in multilevel feedback queue scheduling, meaning that a higher-priority process can preempt a lower-priority process to ensure it gets the CPU time it needs.

**Advantages of Multilevel Feedback Queue Scheduling**

Some of the advantages of multilevel feedback queue scheduling are:

* It is more flexible and adaptable than multilevel queue scheduling, as it allows processes to move between queues according to their changing needs and behavior.
* It prevents starvation by moving processes that wait too long in lower-priority queues to higher-priority queues.
* It favors short and interactive processes over long and CPU-bound processes, as they get higher priority and more CPU time.
* It can handle different types of processes with different characteristics and requirements.

**Disadvantages of Multilevel Feedback Queue Scheduling**

Some of the disadvantages of multilevel feedback queue scheduling are:

* It is more complex and difficult to implement than multilevel queue scheduling, as it requires more parameters and rules to define the queues and the movement of processes between them.
* It produces more overhead due to frequent context switches and priority adjustments.
* It may cause some processes to oscillate between queues if they have varying CPU and I/O bursts.