A Comprehensive Guide to CPU Scheduling Algorithms: Principles, Problems, and Solutions

Section 1: Foundational Concepts in CPU Scheduling

Introduction to Process Scheduling

In the context of a multiprogramming operating system, the CPU is a fundamental resource that must be shared among multiple concurrently active processes. CPU scheduling is the core mechanism by which the operating system selects a process from the ready queue and allocates the CPU to it for execution.[1] This decision-making process is indispensable for maximizing CPU utilization, achieving high system throughput, and providing a responsive and fair experience for users in time-sharing environments.[1]

The management of this process involves two key components: the **Process Scheduler** (or short-term scheduler) and the **Dispatcher**. The scheduler is responsible for selecting the next process to run from the pool of available processes in the ready queue, based on a specific algorithm.[1] Once a process is selected, the

Dispatcher takes over. The Dispatcher is the module that gives control of the CPU to the process selected by the scheduler. This involves switching the context from the old process to the new one, switching to user mode, and jumping to the proper location in the user program to restart that program.[1] The time taken for the dispatcher to stop one process and start another running is known as dispatch latency. Efficient scheduling and dispatching are central to the overall performance of an operating system, forming a critical part of the Process Management duties outlined in typical Master of Computer Applications (MCA) syllabi.[4]

Key Performance Metrics (The Criteria for "Good" Scheduling)

To evaluate and compare the effectiveness of different CPU scheduling algorithms, a set of standard performance metrics is used. The primary goal of any scheduling algorithm is to optimize these metrics according to the specific needs of the system (e.g., batch processing vs. interactive use).[3]

The essential terminologies are defined as follows [2]:

- **Arrival Time (AT):** The time at which a process enters the ready queue, becoming eligible for execution.
- **Burst Time (BT):** Also known as CPU Time or Execution Time, this is the total time required by a process for execution on the CPU.
- **Completion Time (CT):** The time at which a process finishes its execution and exits the system.
- **Turnaround Time (TAT):** This metric represents the total time a process spends in the system, from its arrival to its completion. It is a measure of the total time taken to execute a particular process. The objective is generally to minimize this value.[3] The formula is:
  $TAT = CT - AT$
- **Waiting Time (WT):** This metric represents the total time a process spends waiting in the ready queue, not executing on the CPU or performing I/O. The scheduling algorithm has a direct impact on the waiting time of processes.[2] Minimizing waiting time is a key objective of most scheduling algorithms. The formula is:
  $WT = TAT - BT$
- **Response Time (RT):** In an interactive system, this is the time from when a request is submitted until the first response is produced, not the time it takes to output the final result. It is the time it takes to start responding, not the time it takes to finish. For a process, it is the time from its arrival until it is first allocated the CPU.[3] The formula is:
  $RT = \text{Time of First CPU Allocation} - AT$

The relationship $TAT = WT + BT$ is fundamental.[8] For any given process, its Burst Time (BT) is a fixed property. Therefore, any improvement in Turnaround Time (TAT) is achieved solely by reducing its Waiting Time (WT). This makes the Average Waiting Time (AWT) the most direct and crucial measure of a scheduling algorithm's efficiency in managing the ready queue and, by extension, overall system performance.

The Core Dichotomy: Preemptive vs. Non-Preemptive Scheduling

CPU scheduling algorithms are broadly classified into two categories based on whether a running process can be interrupted by the scheduler.[1]

- **Non-Preemptive Scheduling:** In this mode, once the CPU has been allocated to a process, it cannot be forcibly taken away. The process retains control of the CPU until it either completes its CPU burst and terminates, or it voluntarily relinquishes the CPU by switching to the waiting state (e.g., to perform an I/O operation).[7] The scheduler only makes a decision when the running process gives up the CPU. This approach is simpler to implement and has lower overhead as it involves fewer context switches. However, it is inflexible and can lead to poor response times and the potential for starvation, where short processes get stuck behind a long-running process.[10] Examples include First-Come, First-Served (FCFS) and the non-preemptive versions of Shortest-Job-First (SJF) and Priority scheduling.[10]
- **Preemptive Scheduling:** In this mode, the CPU can be taken away from a currently running process by the operating system and allocated to another process. This preemption can be triggered by various events, such as the arrival of a new process with a higher priority, or a timer interrupt indicating that the current process has exhausted its allocated time slice.[1] This approach is more complex and incurs higher overhead due to more frequent context switching. However, it provides better responsiveness and fairness, making it suitable for time-sharing and interactive systems.[10] Examples include Round Robin (RR), Shortest Remaining Time First (SRTF), and the preemptive version of Priority scheduling.[10]

The Ready Queue is the central data structure where these scheduling decisions are enacted. A Gantt chart is a visual representation of the *outcome* of these decisions over time. However, to truly understand the dynamic logic of an algorithm, one must trace the state of the Ready Queue at each decision point (e.g., process arrival, completion, or preemption). This trace reveals *why* a particular process was chosen over others at a specific moment, providing a much deeper insight than the final schedule alone.

Section 2: First-Come, First-Served (FCFS) Scheduling

Algorithm Principles and Characteristics

First-Come, First-Served (FCFS) is the most straightforward CPU scheduling algorithm. As its name implies, the process that requests the CPU first is allocated the CPU first.[8] The implementation is managed with a simple First-In, First-Out (FIFO) queue data structure.[7] When a process enters the ready queue, its Process Control Block (PCB) is linked onto the tail of the queue. When the CPU becomes free, it is allocated to the process at the head of the queue. FCFS is a strictly non-preemptive scheduling algorithm; once a process has the CPU, it runs to completion or until it blocks for I/O.[8]

While simple and inherently starvation-free, its performance is generally poor for time-sharing systems.[7]

Analysis and Critical Flaws

The primary weakness of FCFS is that its performance is highly sensitive to the arrival order of processes, often resulting in a high average waiting time.[8] This leads to a significant problem known as the

Convoy Effect.

The Convoy Effect occurs when a process with a very long CPU burst occupies the CPU, forcing all other processes, including those with very short CPU bursts, to wait. This creates a "convoy" of processes backed up behind the long one. The result is poor utilization of both CPU and I/O devices. While the long process runs, the shorter, I/O-bound processes are stuck waiting in the ready queue, leaving I/O devices idle. When the long process finally releases the CPU, the short processes execute quickly and move to I/O, potentially leaving the CPU idle while they queue for I/O devices. This cycle of inefficient resource utilization makes FCFS a poor choice for most modern systems.[11]

The flaws of FCFS serve as the primary motivation for developing more sophisticated algorithms like SJF, Priority, and Round Robin, which incorporate additional information beyond simple arrival time to make more intelligent scheduling decisions.

Solved Problems

Problem 1: The Convoy Effect (Simultaneous Arrival)

Consider a set of processes that arrive at time 0.
Process Specification

Process ID	Arrival Time (AT)	Burst Time (BT)
P1	0	24
P2	0	3
P3	0	3

Solution: Scenario A (Order: P1, P2, P3)

Ready Queue and Execution Trace

- **Time 0:** All processes P1, P2, P3 arrive. Ready Queue: [P1, P2, P3]. P1 is at the head and is dispatched.
- **Time 24:** P1 completes. Ready Queue: [P2, P3]. P2 is dispatched.
- **Time 27:** P2 completes. Ready Queue: [P3]. P3 is dispatched.
- **Time 30:** P3 completes. Ready Queue is empty.

Gantt Chart

```
| P1 | P2 | P3 |
0        24    27    30
```

Performance Calculation

Process ID	CT	TAT (CT - AT)	WT (TAT - BT)
P1	24	24 - 0 = 24	24 - 24 = 0
P2	27	27 - 0 = 27	27 - 3 = 24
P3	30	30 - 0 = 30	30 - 3 = 27

- **Average Waiting Time (AWT):** (0+24+27)/3=17.0 ms
- **Average Turnaround Time (ATT):** (24+27+30)/3=27.0 ms

Solution: Scenario B (Order: P2, P3, P1)

Ready Queue and Execution Trace

- **Time 0:** All processes arrive. Assume arrival order is P2, P3, P1. Ready Queue: [P2, P3, P1]. P2 is dispatched.
- **Time 3:** P2 completes. Ready Queue: [P3, P1]. P3 is dispatched.
- **Time 6:** P3 completes. Ready Queue: [P1]. P1 is dispatched.
- **Time 30:** P1 completes. Ready Queue is empty.

Gantt Chart

```
| P2 | P3 | P1 |
0     3     6        30
```

Performance Calculation

Process ID	CT	TAT (CT - AT)	WT (TAT - BT)

P2	3	3 - 0 = 3	3 - 3 = 0
P3	6	6 - 0 = 6	6 - 3 = 3
P1	30	30 - 0 = 30	30 - 24 = 6

- **Average Waiting Time (AWT):** (0+3+6)/3=3.0 ms
- **Average Turnaround Time (ATT):** (3+6+30)/3=13.0 ms

Analysis: This problem starkly illustrates the convoy effect. A simple change in arrival order reduced the AWT from 17.0 ms to 3.0 ms, demonstrating the algorithm's high performance variance.[12]

Problem 2: Staggered Arrival

Consider the following set of processes.
Process Specification

Process ID	Arrival Time (AT)	Burst Time (BT)
P1	0	8
P2	1	4
P3	2	9
P4	3	5

Solution

Ready Queue and Execution Trace
The Ready Queue trace shows the state of the queue at each event (arrival or completion).

Time	Event	Ready Queue Contents	Notes
0	P1 Arrives	[P1]	P1 is dispatched.

1	P2 Arrives	[P1(running)], [P2]	P1 continues (non-preemptive).
2	P3 Arrives	[P1(running)], [P2, P3]	P1 continues.
3	P4 Arrives	[P1(running)], [P2, P3, P4]	P1 continues.
8	P1 Completes	[P2, P3, P4]	P2 is dispatched (FCFS order).
12	P2 Completes	[P3, P4]	P3 is dispatched.
21	P3 Completes	[P4]	P4 is dispatched.
26	P4 Completes	``	Queue is empty.

Gantt Chart

```
| P1 | P2 | P3 | P4 |
0    8    12        21   26
```

Performance Calculation

Process ID	CT	TAT (CT - AT)	WT (TAT - BT)
P1	8	8 - 0 = 8	8 - 8 = 0
P2	12	12 - 1 = 11	11 - 4 = 7
P3	21	21 - 2 = 19	19 - 9 = 10
P4	26	26 - 3 = 23	23 - 5 = 18

- **Average Waiting Time (AWT):** (0+7+10+18)/4=8.75 ms

- **Average Turnaround Time (ATT):** (8+11+19+23)/4=15.25 ms

Analysis: This problem demonstrates how FCFS handles processes arriving at different times. The scheduling order is strictly determined by their arrival times, regardless of their burst times.[7]

Problem 3: Convoy Effect with Staggered Arrival

Consider the following set of processes.
Process Specification

Process ID	Arrival Time (AT)	Burst Time (BT)
P1	0	10
P2	1	29
P3	2	3
P4	3	7

Solution

Ready Queue and Execution Trace

Time	Event	Ready Queue Contents	Notes
0	P1 Arrives	[P1]	P1 is dispatched.
1	P2 Arrives	[P1(running)], [P2]	
2	P3 Arrives	[P1(running)], [P2, P3]	
3	P4 Arrives	[P1(running)], [P2, P3, P4]	

10	P1 Completes	[P2, P3, P4]	P2 is dispatched next.
39	P2 Completes	[P3, P4]	P3 is dispatched next.
42	P3 Completes	[P4]	P4 is dispatched next.
49	P4 Completes	``	

Gantt Chart

```
| P1 | P2 | P3 | P4 |
0      10          39   42      49
```

Performance Calculation

Process ID	CT	TAT (CT - AT)	WT (TAT - BT)
P1	10	10 - 0 = 10	10 - 10 = 0
P2	39	39 - 1 = 38	38 - 29 = 9
P3	42	42 - 2 = 40	40 - 3 = 37
P4	49	49 - 3 = 46	46 - 7 = 39

- **Average Waiting Time (AWT):** (0+9+37+39)/4=21.25 ms
- **Average Turnaround Time (ATT):** (10+38+40+46)/4=33.5 ms

Analysis: Even with staggered arrivals, the long process P2 causes significant waiting times for the shorter processes P3 and P4, again highlighting the convoy effect.[13]

Problem 4: Mixed Burst Times (Simultaneous Arrival)

Consider the following set of processes arriving at time 0.
Process Specification

Process ID	Arrival Time (AT)	Burst Time (BT)
P1	0	3
P2	0	6
P3	0	4
P4	0	2

Solution

Ready Queue and Execution Trace

- **Time 0:** All processes arrive. Ready Queue: [P1, P2, P3, P4]. P1 is dispatched.
- **Time 3:** P1 completes. Ready Queue: [P2, P3, P4]. P2 is dispatched.
- **Time 9:** P2 completes. Ready Queue: [P3, P4]. P3 is dispatched.
- **Time 13:** P3 completes. Ready Queue: [P4]. P4 is dispatched.
- **Time 15:** P4 completes.

Gantt Chart

```
| P1 | P2 | P3 | P4 |
0   3      9     13   15
```

Performance Calculation

Process ID	CT	TAT (CT - AT)	WT (TAT - BT)
P1	3	3 - 0 = 3	3 - 3 = 0
P2	9	9 - 0 = 9	9 - 6 = 3
P3	13	13 - 0 = 13	13 - 4 = 9

P4	15	15 - 0 = 15	15 - 2 = 13

- **Average Waiting Time (AWT):** (0+3+9+13)/4=6.25 ms
- **Average Turnaround Time (ATT):** (3+9+13+15)/4=10.0 ms

Analysis: A straightforward FCFS example with simultaneous arrivals.[13]

Problem 5: Staggered Arrival with Floating Point Times

Consider the following set of processes.
Process Specification

Process ID	Arrival Time (AT)	Burst Time (BT)
P1	0.0	8
P2	0.4	4
P3	1.0	1

Solution

Ready Queue and Execution Trace

Time	Event	Ready Queue Contents	Notes
0.0	P1 Arrives	[P1]	P1 is dispatched.
0.4	P2 Arrives	[P1(running)], [P2]	
1.0	P3 Arrives	[P1(running)], [P2, P3]	
8.0	P1 Completes	[P2, P3]	P2 is dispatched.
12.0	P2 Completes	[P3]	P3 is dispatched.

13.0	P3 Completes	``	

Gantt Chart

```
| P1 | P2 | P3 |
0.0         8.0       12.0   13.0
```

Performance Calculation

Process ID	CT	TAT (CT - AT)	WT (TAT - BT)
P1	8.0	8.0 - 0.0 = 8.0	8.0 - 8 = 0.0
P2	12.0	12.0 - 0.4 = 11.6	11.6 - 4 = 7.6
P3	13.0	13.0 - 1.0 = 12.0	12.0 - 1 = 11.0

- **Average Waiting Time (AWT):** (0.0+7.6+11.0)/3≈6.2 ms
- **Average Turnaround Time (ATT):** (8.0+11.6+12.0)/3≈10.53 ms

Analysis: This problem illustrates that the FCFS logic remains the same even with non-integer arrival times.[13]

Section 3: Shortest Job First (SJF) Scheduling

3.1 Non-Preemptive SJF

Algorithm Principles

The Shortest Job First (SJF) scheduling algorithm associates with each process the length of its next CPU burst. When the CPU is available, it is assigned to the process that has the smallest next CPU burst.[6] In the non-preemptive version, once a process begins execution, it runs until its CPU burst is complete. It cannot be interrupted by the arrival of a new process with a shorter burst time.[16] If two or more processes have the same shortest burst time, the tie is typically broken using FCFS scheduling.[7]

Analysis

SJF is provably optimal, as it yields the minimum possible average waiting time for a given set of processes.[6] This makes it an important benchmark for scheduling algorithms. However, its primary and significant drawback is the difficulty of knowing the length of the next CPU burst in advance, especially for interactive systems.[6] While this can be estimated using techniques like a weighted average of previous burst times, it is not precise.[6] This impracticality limits its use, though it is suitable for long-term scheduling in batch systems where job runtimes might be specified by the user.[15]

Another potential issue is **starvation**. If there is a continuous stream of short processes arriving, a process with a long CPU burst might be delayed indefinitely.[6] This can be mitigated using a technique called aging, where the priority of a waiting process is increased over time.[6]

Solved Problems

Problem 1: Simultaneous Arrival

Consider the following processes arriving at time 0.

Process Specification

Process ID	Arrival Time (AT)	Burst Time (BT)
P1	0	6
P2	0	8
P3	0	7
P4	0	3

Solution

Ready Queue and Execution Trace

- **Time 0:** All processes arrive. Ready Queue: [P1(6), P2(8), P3(7), P4(3)]. P4 has the shortest burst time and is dispatched.
- **Time 3:** P4 completes. Ready Queue: [P1(6), P2(8), P3(7)]. P1 has the shortest burst time and is dispatched.
- **Time 9:** P1 completes. Ready Queue: [P2(8), P3(7)]. P3 has the shortest burst time and is dispatched.
- **Time 16:** P3 completes. Ready Queue: [P2(8)]. P2 is dispatched.
- **Time 24:** P2 completes.

Gantt Chart

```
| P4 | P1 | P3 | P2 |
0   3       9       16      24
```

Performance Calculation

Process ID	CT	TAT (CT - AT)	WT (TAT - BT)
P1	9	9 - 0 = 9	9 - 6 = 3

P2	24	24 - 0 = 24	24 - 8 = 16
P3	16	16 - 0 = 16	16 - 7 = 9
P4	3	3 - 0 = 3	3 - 3 = 0

- **Average Waiting Time (AWT):** (3+16+9+0)/4=7.0 ms
- **Average Turnaround Time (ATT):** (9+24+16+3)/4=13.0 ms

Analysis: This problem shows the basic principle of SJF. Compared to FCFS for the same set (AWT would be (0+6+14+21)/4 = 10.25 ms), SJF provides a significant improvement in average waiting time.

Problem 2: Staggered Arrival

Consider the following set of processes.
Process Specification

Process ID	Arrival Time (AT)	Burst Time (BT)
P1	0	6
P2	2	8
P3	4	3
P4	5	4

Solution

Ready Queue and Execution Trace
The scheduler makes a decision only when the CPU is free.

Time	Event	Ready Queue Contents	Notes
0	P1 Arrives	[P1(6)]	P1 is the only process, so it is

			dispatched.
2	P2 Arrives	[P1(running)], [P2(8)]	P1 continues (non-preemptive).
4	P3 Arrives	[P1(running)], [P2(8), P3(3)]	P1 continues.
5	P4 Arrives	[P1(running)], [P2(8), P3(3), P4(4)]	P1 continues.
6	P1 Completes	[P2(8), P3(3), P4(4)]	CPU is free. Compare BTs of processes in queue. P3 is shortest. P3 is dispatched.
9	P3 Completes	[P2(8), P4(4)]	CPU is free. P4 is shortest. P4 is dispatched.
13	P4 Completes	[P2(8)]	CPU is free. P2 is dispatched.
21	P2 Completes	``	

Gantt Chart

```
| P1 | P3 | P4 | P2 |
0        6     9     13        21
```

Performance Calculation

Process ID	CT	TAT (CT - AT)	WT (TAT - BT)
P1	6	6 - 0 = 6	6 - 6 = 0

P2	21	21 - 2 = 19	19 - 8 = 11
P3	9	9 - 4 = 5	5 - 3 = 2
P4	13	13 - 5 = 8	8 - 4 = 4

- **Average Waiting Time (AWT):** (0+11+2+4)/4=4.25 ms
- **Average Turnaround Time (ATT):** (6+19+5+8)/4=9.5 ms

Analysis: This problem illustrates a key aspect of non-preemptive SJF. At time 6, when P1 completes, the scheduler must choose from all processes that have arrived by that time (P2, P3, and P4). It doesn't matter that P2 arrived before P3 and P4; the decision is based solely on the shortest burst time among the available processes.

Problem 3: Tie-Breaking with FCFS

Consider the following set of processes arriving at the same time.
Process Specification

Process ID	Arrival Time (AT)	Burst Time (BT)
P1	0	6
P2	0	10
P3	0	4
P4	0	6

Solution

Ready Queue and Execution Trace

- **Time 0:** All processes arrive. Ready Queue: [P1(6), P2(10), P3(4), P4(6)]. P3 has the shortest BT (4) and is dispatched.
- **Time 4:** P3 completes. Ready Queue: [P1(6), P2(10), P4(6)]. P1 and P4 have the same shortest BT (6). Since P1 arrived before P4 (or has a lower process ID), P1 is chosen (FCFS tie-break). P1 is dispatched.

- **Time 10:** P1 completes. Ready Queue: [P2(10), P4(6)]. P4 has the shorter BT (6) and is dispatched.
- **Time 16:** P4 completes. Ready Queue: [P2(10)]. P2 is dispatched.
- **Time 26:** P2 completes.

Gantt Chart

```
| P3 | P1 | P4 | P2 |
0    4     10     16      26
```

Performance Calculation

Process ID	CT	TAT (CT - AT)	WT (TAT - BT)
P1	10	10 - 0 = 10	10 - 6 = 4
P2	26	26 - 0 = 26	26 - 10 = 16
P3	4	4 - 0 = 4	4 - 4 = 0
P4	16	16 - 0 = 16	16 - 6 = 10

- **Average Waiting Time (AWT):** (4+16+0+10)/4=7.5 ms
- **Average Turnaround Time (ATT):** (10+26+4+16)/4=14.0 ms

Analysis: This problem explicitly demonstrates the FCFS tie-breaking rule, which is a standard convention when multiple processes share the same shortest burst time.[16]

Problem 4: Complex Staggered Arrival

Consider the following set of processes.
Process Specification

Process ID	Arrival Time (AT)	Burst Time (BT)

P1	3	1
P2	1	4
P3	4	2
P4	0	6
P5	2	3

Solution

Ready Queue and Execution Trace

Time	Event	Ready Queue Contents	Notes
0	P4 Arrives	[P4(6)]	P4 is dispatched.
1	P2 Arrives	[P4(running)], [P2(4)]	
2	P5 Arrives	[P4(running)], [P2(4), P5(3)]	
3	P1 Arrives	[P4(running)], [P2(4), P5(3), P1(1)]	
4	P3 Arrives	[P4(running)], [P2(4), P5(3), P1(1), P3(2)]	
6	P4 Completes	[P2(4), P5(3), P1(1), P3(2)]	CPU free. Compare BTs. P1 is shortest. P1 is dispatched.
7	P1 Completes	[P2(4), P5(3), P3(2)]	CPU free. P3 is shortest. P3 is dispatched.
9	P3 Completes	[P2(4), P5(3)]	CPU free. P5 is shortest. P5 is

			dispatched.
12	P5 Completes	[P2(4)]	CPU free. P2 is dispatched.
16	P2 Completes	``	

Gantt Chart

```
| P4 | P1 | P3 | P5 | P2 |
0        6    7    9    12      16
```

Performance Calculation

Process ID	CT	TAT (CT - AT)	WT (TAT - BT)
P1	7	7 - 3 = 4	4 - 1 = 3
P2	16	16 - 1 = 15	15 - 4 = 11
P3	9	9 - 4 = 5	5 - 2 = 3
P4	6	6 - 0 = 6	6 - 6 = 0
P5	12	12 - 2 = 10	10 - 3 = 7

- **Average Waiting Time (AWT):** (3+11+3+0+7)/5=4.8 units
- **Average Turnaround Time (ATT):** (4+15+5+6+10)/5=8.0 units

Analysis: A comprehensive example showing how the ready queue accumulates processes while a long job (P4) runs, and how the scheduler then works through the queued processes based on the SJF policy.[17]

Problem 5: Another Staggered Arrival Example

Consider the following set of processes.
Process Specification

Process ID	Arrival Time (AT)	Burst Time (BT)
P1	0	7
P2	2	4
P3	4	1
P4	5	4

Solution

Ready Queue and Execution Trace

Time	Event	Ready Queue Contents	Notes
0	P1 Arrives	[P1(7)]	P1 dispatched.
2	P2 Arrives	[P1(running)], [P2(4)]	
4	P3 Arrives	[P1(running)], [P2(4), P3(1)]	
5	P4 Arrives	[P1(running)], [P2(4), P3(1), P4(4)]	
7	P1 Completes	[P2(4), P3(1), P4(4)]	CPU free. P3 is shortest. P3 dispatched.
8	P3 Completes	[P2(4), P4(4)]	CPU free. P2 and P4 tie. P2 chosen (FCFS tie-break).
12	P2 Completes	[P4(4)]	CPU free. P4

			dispatched.
16	P4 Completes	``	

Gantt Chart

```
| P1 | P3 | P2 | P4 |
0          7    8       12      16
```

Performance Calculation

Process ID	CT	TAT (CT - AT)	WT (TAT - BT)
P1	7	7 - 0 = 7	7 - 7 = 0
P2	12	12 - 2 = 10	10 - 4 = 6
P3	8	8 - 4 = 4	4 - 1 = 3
P4	16	16 - 5 = 11	11 - 4 = 7

- **Average Waiting Time (AWT):** (0+6+3+7)/4=4.0 ms
- **Average Turnaround Time (ATT):** (7+10+4+11)/4=8.0 ms

Analysis: This problem reinforces the handling of staggered arrivals and the FCFS tie-breaking rule.

3.2 Preemptive SJF (Shortest Remaining Time First - SRTF)

Algorithm Principles

Shortest Remaining Time First (SRTF) is the preemptive version of SJF.[17] The scheduler allocates the CPU to the process with the smallest amount of time remaining until completion. A key difference from non-preemptive SJF is that the scheduler re-evaluates its decision whenever a new process arrives in the ready queue. If the newly arrived process has a CPU burst time that is less than the

remaining time of the currently executing process, the scheduler will preempt the current process and start executing the new, shorter process.[19]

Analysis

Like its non-preemptive counterpart, SRTF is optimal in terms of minimizing average waiting time.[17] By always running the job that can finish the quickest, it ensures that shorter processes are serviced rapidly, which improves system responsiveness.[19] However, it shares the same weaknesses: the practical difficulty of predicting future burst times and the potential for starvation of long processes if a continuous stream of shorter jobs arrives.[17] Furthermore, the preemptive nature of SRTF can lead to higher overhead due to more frequent context switching compared to non-preemptive SJF.[18]

The transition from non-preemptive SJF to SRTF highlights a fundamental trade-off in scheduler design: the benefit of constantly making the optimal choice versus the overhead cost of that constant re-evaluation. SRTF's willingness to preempt demonstrates a commitment to optimality at every time unit, removing the "lock-in" of a non-preemptive decision. When a new process arrives, SRTF effectively asks, "Given this new information, is my current decision still the absolute best one?" This aggressive re-evaluation is the essence of its preemptive strategy.

Solved Problems

Problem 1: Basic Preemption

Consider the following set of processes.
Process Specification

Process ID	Arrival Time (AT)	Burst Time (BT)
P1	0	8
P2	1	4
P3	2	9
P4	3	5

Solution

Ready Queue and Execution Trace
This trace must be followed unit-by-unit to capture the logic of SRTF.

Time	Running Process (Rem. Time)	Event	Ready Queue (with Rem. Time)	Notes
0	P1 (8)	P1 Arrives	``	P1 is the only process, starts execution.
1	P1 (7)	P2 Arrives	[P2(4)]	P2's BT (4) is < P1's rem. time (7). P1 is preempted.
1	P2 (4)	-	[P1(7)]	P2 starts execution.
2	P2 (3)	P3 Arrives	[P1(7), P3(9)]	P2's rem. time (3) is shortest. P2 continues.
3	P2 (2)	P4 Arrives	[P1(7), P3(9), P4(5)]	P2's rem. time (2) is shortest.

				P2 continues.
5	-	P2 Completes	[P1(7), P3(9), P4(5)]	CPU free. Compare rem. times. P4 is shortest.
5	P4 (5)	-	[P1(7), P3(9)]	P4 starts execution.
10	-	P4 Completes	[P1(7), P3(9)]	CPU free. P1 is shortest.
10	P1 (7)	-	[P3(9)]	P1 resumes execution.
17	-	P1 Completes	[P3(9)]	CPU free. P3 starts.
17	P3 (9)	-	``	P3 starts execution.
26	-	P3 Completes	``	All processes finished.

Gantt Chart

```
|P1| P2 | P4 | P1 | P3 |
0  1       5       10        17        26
```

Performance Calculation

Process ID	CT	TAT (CT - AT)	WT (TAT - BT)
P1	17	17 - 0 = 17	17 - 8 = 9

P2	5	5 - 1 = 4	4 - 4 = 0
P3	26	26 - 2 = 24	24 - 9 = 15
P4	10	10 - 3 = 7	7 - 5 = 2

- **Average Waiting Time (AWT):** (9+0+15+2)/4=6.5 ms
- **Average Turnaround Time (ATT):** (17+4+24+7)/4=13.0 ms

Analysis: This problem from [12] (SRTF part) clearly shows the preemptive nature. P1 is preempted by the shorter job P2. Later, when P2 finishes, P4 is chosen over the original P1 because its total burst time (5) is less than P1's remaining time (7).

Problem 2: Multiple Preemptions

Consider the following set of processes.
Process Specification

Process ID	Arrival Time (AT)	Burst Time (BT)
P1	0	6
P2	1	3
P3	2	7
P4	3	2

Solution

Ready Queue and Execution Trace

Time	Running Process (Rem. Time)	Event	Ready Queue (with Rem. Time)	Notes
0	P1 (6)	P1 Arrives	``	P1 starts.

1	P1 (5)	P2 Arrives	[P2(3)]	P2's BT (3) < P1's rem. time (5). Preempt P1.
1	P2 (3)	-	[P1(5)]	P2 starts.
2	P2 (2)	P3 Arrives	[P1(5), P3(7)]	P2's rem. time (2) is shortest. P2 continues.
3	P2 (1)	P4 Arrives	[P1(5), P3(7), P4(2)]	P2's rem. time (1) is shortest. P2 continues.
4	-	P2 Completes	[P1(5), P3(7), P4(2)]	CPU free. P4 is shortest.
4	P4 (2)	-	[P1(5), P3(7)]	P4 starts.
6	-	P4 Completes	[P1(5), P3(7)]	CPU free. P1 is shortest.
6	P1 (5)	-	[P3(7)]	P1 resumes.
11	-	P1 Completes	[P3(7)]	CPU free. P3 starts.
11	P3 (7)	-	``	P3 starts.
18	-	P3 Completes	``	

Gantt Chart

```
|P1| P2 | P4 | P1 | P3 |
0  1      4   6       11      18
```

Performance Calculation

Process ID	CT	TAT (CT - AT)	WT (TAT - BT)
P1	11	11 - 0 = 11	11 - 6 = 5
P2	4	4 - 1 = 3	3 - 3 = 0
P3	18	18 - 2 = 16	16 - 7 = 9
P4	6	6 - 3 = 3	3 - 2 = 1

- **Average Waiting Time (AWT):** (5+0+9+1)/4=3.75 ms
- **Average Turnaround Time (ATT):** (11+3+16+3)/4=8.25 ms

Analysis: This example shows that a process (P2) can run to completion amidst arrivals if it consistently maintains the shortest remaining time.

Problem 3: No Preemption After Resuming

Consider the following set of processes.
Process Specification

Process ID	Arrival Time (AT)	Burst Time (BT)
P1	0	8
P2	4	10
P3	4	3
P4	10	4

Solution

Ready Queue and Execution Trace

Time	Running Process (Rem. Time)	Event	Ready Queue (with Rem. Time)	Notes
0	P1 (8)	P1 Arrives	``	P1 starts.
4	P1 (4)	P2, P3 Arrive	[P2(10), P3(3)]	P3's BT (3) < P1's rem. time (4). Preempt P1.
4	P3 (3)	-	[P1(4), P2(10)]	P3 starts.
7	-	P3 Completes	[P1(4), P2(10)]	CPU free. P1 is shortest.
7	P1 (4)	-	[P2(10)]	P1 resumes.
10	P1 (1)	P4 Arrives	[P2(10), P4(4)]	P1's rem. time (1) is shortest. P1 continues.
11	-	P1 Completes	[P2(10), P4(4)]	CPU free. P4 is shortest.
11	P4 (4)	-	[P2(10)]	P4 starts.
15	-	P4 Completes	[P2(10)]	CPU free. P2 starts.
15	P2 (10)	-	``	P2 starts.
25	-	P2 Completes	``	

Gantt Chart

```
| P1 | P3 | P1 | P4 | P2 |
0       4    7       11    15        25
```

Performance Calculation

Process ID	CT	TAT (CT - AT)	WT (TAT - BT)
P1	11	11 - 0 = 11	11 - 8 = 3
P2	25	25 - 4 = 21	21 - 10 = 11
P3	7	7 - 4 = 3	3 - 3 = 0
P4	15	15 - 10 = 5	5 - 4 = 1

- **Average Waiting Time (AWT):** (3+11+0+1)/4=3.75 ms
- **Average Turnaround Time (ATT):** (11+21+3+5)/4=10.0 ms

Analysis: A key point is at time 10. When P4 arrives, P1 is running. Even though P4 has a shorter *total* burst time (4) than P2, the scheduler's comparison is between P4's burst time (4) and P1's *remaining* time (1). Since P1's remaining time is shorter, it is not preempted.[16]

Problem 4: A Different Scenario

Consider the set of 5 processes whose arrival time and burst time are given below.
Process Specification

Process ID	Arrival Time (AT)	Burst Time (BT)
P1	3	1
P2	1	4
P3	4	2
P4	0	6
P5	2	3

Solution

Ready Queue and Execution Trace

Time	Running Process (Rem. Time)	Event	Ready Queue (with Rem. Time)	Notes
0	P4 (6)	P4 Arrives	``	P4 starts.
1	P4 (5)	P2 Arrives	[P2(4)]	P2's BT (4) < P4's rem. time (5). Preempt P4.
1	P2 (4)	-	[P4(5)]	P2 starts.
2	P2 (3)	P5 Arrives	[P4(5), P5(3)]	Tie between P2 and P5. P2 continues.
3	P2 (2)	P1 Arrives	[P4(5), P5(3), P1(1)]	P1's BT (1) is shortest. Preempt P2.
3	P1 (1)	-	[P4(5), P2(2), P5(3)]	P1 starts.
4	-	P1 Completes, P3 Arrives	[P4(5), P2(2), P5(3), P3(2)]	CPU free. P2 and P3 tie for shortest. P2 chosen (FCFS tie-break).
4	P2 (2)	-	[P4(5), P5(3), P3(2)]	P2 resumes.
6	-	P2 Completes	[P4(5), P5(3), P3(2)]	CPU free. P3 is shortest.
6	P3 (2)	-	[P4(5), P5(3)]	P3 starts.
8	-	P3 Completes	[P4(5), P5(3)]	CPU free. P5 is

				shortest.
8	P5 (3)	-	[P4(5)]	P5 starts.
11	-	P5 Completes	[P4(5)]	CPU free. P4 starts.
11	P4 (5)	-	``	P4 resumes.
16	-	P4 Completes	``	

Gantt Chart

```
|P4|P2|P1| P2 | P3 | P5 | P4 |
0  1  3  4    6    8   11       16
```

Performance Calculation

Process ID	CT	TAT (CT - AT)	WT (TAT - BT)
P1	4	4 - 3 = 1	1 - 1 = 0
P2	6	6 - 1 = 5	5 - 4 = 1
P3	8	8 - 4 = 4	4 - 2 = 2
P4	16	16 - 0 = 16	16 - 6 = 10
P5	11	11 - 2 = 9	9 - 3 = 6

- **Average Waiting Time (AWT):** (0+1+2+10+6)/5=3.8 units
- **Average Turnaround Time (ATT):** (1+5+4+16+9)/5=7.0 units

Analysis: This complex example from [17] demonstrates multiple preemptions and tie-breaking scenarios, providing a thorough test of the SRTF algorithm's logic.

Problem 5: Simultaneous Arrival

Consider the following set of processes arriving at time 0.
Process Specification

Process ID	Arrival Time (AT)	Burst Time (BT)
P1	0	6
P2	0	8
P3	0	5

Solution

Ready Queue and Execution Trace
Since all processes arrive at the same time, this behaves identically to non-preemptive SJF because no new arrivals can trigger a preemption.

- **Time 0:** All processes arrive. Ready Queue: [P1(6), P2(8), P3(5)]. P3 is shortest and is dispatched.
- **Time 5:** P3 completes. Ready Queue: [P1(6), P2(8)]. P1 is shortest and is dispatched.
- **Time 11:** P1 completes. Ready Queue: [P2(8)]. P2 is dispatched.
- **Time 19:** P2 completes.

Gantt Chart

```
| P3 | P1 | P2 |
0      5      11      19
```

Performance Calculation

Process ID	CT	TAT (CT - AT)	WT (TAT - BT)

P1	11	11 - 0 = 11	11 - 6 = 5
P2	19	19 - 0 = 19	19 - 8 = 11
P3	5	5 - 0 = 5	5 - 5 = 0

- **Average Waiting Time (AWT):** (5+11+0)/3≈5.33 ms
- **Average Turnaround Time (ATT):** (11+19+5)/3≈11.67 ms

Analysis: This problem shows that when all processes are available from the start, SRTF provides no additional benefit over non-preemptive SJF, as there are no new arrivals to cause preemption.[19]

Section 4: Priority Scheduling

Priority scheduling introduces an external, policy-based metric to guide scheduling decisions. This marks a significant shift from algorithms like FCFS or SJF, which rely on intrinsic process characteristics (arrival time, burst time). By assigning a priority to each process, the operating system can enforce system-level goals, such as ensuring critical system processes are serviced before user applications.[21] This configurability is a fundamental step toward more sophisticated and manageable operating systems.

4.1 Non-Preemptive Priority Scheduling

Algorithm Principles

In non-preemptive priority scheduling, a priority value (typically an integer) is associated with each process. The CPU is allocated to the process with the highest priority. If multiple processes share the same highest priority, the tie is broken using the FCFS algorithm.[21] Once a process is allocated the CPU, it runs to completion or until it blocks, without being interrupted by the arrival of a higher-priority process.[21]

Note: Priority schemes can vary. For the following problems, we will assume a lower number indicates a higher priority.

Solved Problems

Problem 1: Simultaneous Arrival

Consider the following processes arriving at time 0.
Process Specification

Process ID	Arrival Time (AT)	Burst Time (BT)	Priority
P1	0	6	2
P2	0	10	1
P3	0	4	3
P4	0	6	2

Solution

Ready Queue and Execution Trace

- **Time 0:** All processes arrive. Ready Queue: [P1(P:2), P2(P:1), P3(P:3), P4(P:2)]. P2 has the highest priority (1) and is dispatched.
- **Time 10:** P2 completes. Ready Queue: [P1(P:2), P3(P:3), P4(P:2)]. P1 and P4 have the same highest priority (2). P1 is chosen via FCFS tie-break. P1 is dispatched.
- **Time 16:** P1 completes. Ready Queue: [P3(P:3), P4(P:2)]. P4 has the highest priority (2) and is dispatched.
- **Time 22:** P4 completes. Ready Queue: [P3(P:3)]. P3 is dispatched.
- **Time 26:** P3 completes.

Gantt Chart

```
| P2 | P1 | P4 | P3 |
0         10      16      22    26
```

Performance Calculation

Process ID	CT	TAT (CT - AT)	WT (TAT - BT)
P1	16	16 - 0 = 16	16 - 6 = 10
P2	10	10 - 0 = 10	10 - 10 = 0
P3	26	26 - 0 = 26	26 - 4 = 22
P4	22	22 - 0 = 22	22 - 6 = 16

- **Average Waiting Time (AWT):** (10+0+22+16)/4=12.0 ms
- **Average Turnaround Time (ATT):** (16+10+26+22)/4=18.5 ms

Analysis: This problem demonstrates the core logic of non-preemptive priority scheduling, including the FCFS tie-breaking rule.[23]

Problem 2: Staggered Arrival

Consider the following set of processes.
Process Specification

Process ID	Arrival Time (AT)	Burst Time (BT)	Priority
P1	0	4	2
P2	1	2	1

P3	2	6	3

Solution

Ready Queue and Execution Trace

Time	Event	Ready Queue Contents	Notes
0	P1 Arrives	[P1(P:2)]	P1 is the only process, dispatched.
1	P2 Arrives	[P1(running)], [P2(P:1)]	P1 continues (non-preemptive).
2	P3 Arrives	[P1(running)], [P2(P:1), P3(P:3)]	P1 continues.
4	P1 Completes	[P2(P:1), P3(P:3)]	CPU free. P2 has highest priority. P2 dispatched.
6	P2 Completes	[P3(P:3)]	CPU free. P3 dispatched.
12	P3 Completes	``	

Gantt Chart

```
| P1 | P2 | P3 |
0        4     6            12
```

Performance Calculation

Process ID	CT	TAT (CT - AT)	WT (TAT - BT)

P1	4	4 - 0 = 4	4 - 4 = 0
P2	6	6 - 1 = 5	5 - 2 = 3
P3	12	12 - 2 = 10	10 - 6 = 4

- **Average Waiting Time (AWT):** (0+3+4)/3=2.33 ms
- **Average Turnaround Time (ATT):** (4+5+10)/3=6.33 ms

Analysis: At time 4, when P1 finishes, the scheduler chooses P2 over P3 because P2 has a higher priority, even though P3 was also in the ready queue.[21]

Problem 3: Complex Staggered Arrival

Consider the following set of processes.
Process Specification

Process ID	Arrival Time (AT)	Burst Time (BT)	Priority
P1	0	8	4
P2	2	6	1
P3	2	1	2
P4	1	9	2
P5	3	3	3

Solution

Ready Queue and Execution Trace

Time	Event	Ready Queue Contents	Notes
0	P1 Arrives	[P1(P:4)]	P1 is dispatched.

1	P4 Arrives	[P1(running)], [P4(P:2)]	
2	P2, P3 Arrive	[P1(running)], [P4(P:2), P2(P:1), P3(P:2)]	
3	P5 Arrives	[P1(running)], [P4(P:2), P2(P:1), P3(P:2), P5(P:3)]	
8	P1 Completes	[P4(P:2), P2(P:1), P3(P:2), P5(P:3)]	CPU free. P2 has highest priority (1). P2 dispatched.
14	P2 Completes	[P4(P:2), P3(P:2), P5(P:3)]	CPU free. P4 and P3 tie (P:2). P4 chosen (FCFS tie-break).
23	P4 Completes	[P3(P:2), P5(P:3)]	CPU free. P3 has highest priority. P3 dispatched.
24	P3 Completes	[P5(P:3)]	CPU free. P5 dispatched.
27	P5 Completes	``	

Gantt Chart

```
| P1 | P2 | P4 | P3 | P5 |
0          8        14          23   24   27
```

Performance Calculation

Process ID	CT	TAT (CT - AT)	WT (TAT - BT)

P1	8	8 - 0 = 8	8 - 8 = 0
P2	14	14 - 2 = 12	12 - 6 = 6
P3	24	24 - 2 = 22	22 - 1 = 21
P4	23	23 - 1 = 22	22 - 9 = 13
P5	27	27 - 3 = 24	24 - 3 = 21

- **Average Waiting Time (AWT):** (0+6+21+13+21)/5=12.2 ms
- **Average Turnaround Time (ATT):** (8+12+22+22+24)/5=17.6 ms

Analysis: This problem from [38] shows a complex scenario where the scheduler must select from a full ready queue at time 8, demonstrating both priority selection and FCFS tie-breaking.

4.2 Preemptive Priority Scheduling

Algorithm Principles

In the preemptive version of priority scheduling, the scheduler's logic is more aggressive. When a new process arrives in the ready queue, its priority is immediately compared to the priority of the currently running process. If the new process has a higher priority, the currently running process is preempted and returned to the ready queue, and the new process is allocated the CPU.[21] This ensures that the CPU is always executing the highest-priority process that is ready to run.

Solved Problems

Problem 1: Basic Preemption

Consider the following set of processes.
Process Specification

Process ID	Arrival Time (AT)	Burst Time (BT)	Priority
P1	0	4	2
P2	1	3	1
P3	2	1	3
P4	3	5	4

Note: For this problem, a higher number indicates a higher priority.

Solution

Ready Queue and Execution Trace

Time	Running Process (Rem. Time, P)	Event	Ready Queue (Rem. Time, P)	Notes
0	P1 (4, P:2)	P1 Arrives	``	P1 starts.
1	P1 (3, P:2)	P2 Arrives	[P2(3, P:1)]	P1's priority (2) > P2's (1). P1 continues.
2	P1 (2, P:2)	P3 Arrives	[P2(3, P:1), P3(1, P:3)]	P3's priority (3) > P1's (2). Preempt P1.
2	P3 (1, P:3)	-	[P1(2, P:2), P2(3, P:1)]	P3 starts.
3	-	P3 Completes, P4 Arrives	[P1(2, P:2), P2(3, P:1), P4(5, P:4)]	CPU free. P4 has highest priority (4).

3	P4 (5, P:4)	-	[P1(2, P:2), P2(3, P:1)]	P4 starts.
8	-	P4 Completes	[P1(2, P:2), P2(3, P:1)]	CPU free. P1 has highest priority (2).
8	P1 (2, P:2)	-	[P2(3, P:1)]	P1 resumes.
10	-	P1 Completes	[P2(3, P:1)]	CPU free. P2 starts.
10	P2 (3, P:1)	-	``	P2 starts.
13	-	P2 Completes	``	

Gantt Chart

```
| P1 | P3 | P4 | P1 | P2 |
0        2   3         8     10     13
```

Performance Calculation

Process ID	CT	TAT (CT - AT)	WT (TAT - BT)
P1	10	10 - 0 = 10	10 - 4 = 6
P2	13	13 - 1 = 12	12 - 3 = 9
P3	3	3 - 2 = 1	1 - 1 = 0
P4	8	8 - 3 = 5	5 - 5 = 0

- **Average Waiting Time (AWT):** (6+9+0+0)/4=3.75 ms
- **Average Turnaround Time (ATT):** (10+12+1+5)/4=7.0 ms

Analysis: This problem from [37] and [37] (with priority scheme reversed for clarity) shows P1 being preempted by P3 upon its arrival due to higher priority.

Problem 2: Complex Preemption Scenario

Consider the following set of processes. (Lower number = higher priority)
Process Specification

Process ID	Arrival Time (AT)	Burst Time (BT)	Priority
P1	0	9	5
P2	1	4	3
P3	2	5	1
P4	3	7	2
P5	4	3	4

Solution

Ready Queue and Execution Trace

Time	Running Process (Rem. Time, P)	Event	Ready Queue (Rem. Time, P)	Notes
0	P1 (9, P:5)	P1 Arrives	``	P1 starts.
1	P1 (8, P:5)	P2 Arrives	[P2(4, P:3)]	P2's priority (3) > P1's (5). Preempt P1.
1	P2 (4, P:3)	-	[P1(8, P:5)]	P2 starts.
2	P2 (3, P:3)	P3 Arrives	[P1(8, P:5), P3(5, P:1)]	P3's priority (1) > P2's (3).

				Preempt P2.
2	P3 (5, P:1)	-	[P1(8, P:5), P2(3, P:3)]	P3 starts.
3	P3 (4, P:1)	P4 Arrives	[P1(8, P:5), P2(3, P:3), P4(7, P:2)]	P3's priority (1) is highest. P3 continues.
4	P3 (3, P:1)	P5 Arrives	[P1(8, P:5), P2(3, P:3), P4(7, P:2), P5(3, P:4)]	P3's priority (1) is highest. P3 continues.
7	-	P3 Completes	[P1(8, P:5), P2(3, P:3), P4(7, P:2), P5(3, P:4)]	CPU free. P4 has highest priority (2).
7	P4 (7, P:2)	-	[P1(8, P:5), P2(3, P:3), P5(3, P:4)]	P4 starts.
14	-	P4 Completes	[P1(8, P:5), P2(3, P:3), P5(3, P:4)]	CPU free. P2 has highest priority (3).
14	P2 (3, P:3)	-	[P1(8, P:5), P5(3, P:4)]	P2 resumes.
17	-	P2 Completes	[P1(8, P:5), P5(3, P:4)]	CPU free. P5 has highest priority (4).
17	P5 (3, P:4)	-	[P1(8, P:5)]	P5 starts.
20	-	P5 Completes	[P1(8, P:5)]	CPU free. P1 resumes.
20	P1 (8, P:5)	-	``	P1 resumes.
28	-	P1 Completes	``	

Gantt Chart

```
|P1|P2| P3 | P4 | P2 | P5 | P1 |
0  1  2          7            14      17   20           28
```

Performance Calculation

Process ID	CT	TAT (CT - AT)	WT (TAT - BT)
P1	28	28 - 0 = 28	28 - 9 = 19
P2	17	17 - 1 = 16	16 - 4 = 12
P3	7	7 - 2 = 5	5 - 5 = 0
P4	14	14 - 3 = 11	11 - 7 = 4
P5	20	20 - 4 = 16	16 - 3 = 13

- **Average Waiting Time (AWT):** (19+12+0+4+13)/5=9.6 ms
- **Average Turnaround Time (ATT):** (28+16+5+11+16)/5=15.2 ms

Analysis: This detailed problem from [22] illustrates the dynamic nature of the ready queue and the constant re-evaluation performed by the preemptive priority scheduler.

Problem 3: Another Preemption Scenario

Consider the following set of processes. (Lower number = higher priority)
Process Specification

Process ID	Arrival Time (AT)	Burst Time (BT)	Priority
P1	0	8	3

P2	1	2	4
P3	2	6	4
P4	3	4	6
P5	5	2	10
P6	6	5	2

Solution

Ready Queue and Execution Trace

Time	Running Process (Rem. Time, P)	Event	Ready Queue (Rem. Time, P)	Notes
0	P1 (8, P:3)	P1 Arrives	``	P1 starts.
1	P1 (7, P:3)	P2 Arrives	[P2(2, P:4)]	P1's priority (3) is higher. P1 continues.
2	P1 (6, P:3)	P3 Arrives	[P2(2, P:4), P3(6, P:4)]	P1's priority (3) is higher. P1 continues.
3	P1 (5, P:3)	P4 Arrives	[P2(2, P:4), P3(6, P:4), P4(4, P:6)]	P1's priority (3) is higher. P1 continues.
5	P1 (3, P:3)	P5 Arrives	[P2(2, P:4), P3(6, P:4), P4(4, P:6), P5(2, P:10)]	P1's priority (3) is higher. P1 continues.
6	P1 (2, P:3)	P6 Arrives	[P2(2, P:4), P3(6, P:4), P4(4, P:6), P5(2, P:10), P6(5, P:2)]	P6's priority (2) is highest. Preempt P1.

6	P6 (5, P:2)	-	[P1(2, P:3), P2(2, P:4), P3(6, P:4), P4(4, P:6), P5(2, P:10)]	P6 starts.
11	-	P6 Completes	[P1(2, P:3), P2(2, P:4), P3(6, P:4), P4(4, P:6), P5(2, P:10)]	CPU free. P1 has highest priority (3).
11	P1 (2, P:3)	-	[P2(2, P:4), P3(6, P:4), P4(4, P:6), P5(2, P:10)]	P1 resumes.
13	-	P1 Completes	[P2(2, P:4), P3(6, P:4), P4(4, P:6), P5(2, P:10)]	CPU free. P2/P3 tie. P2 chosen (FCFS).
13	P2 (2, P:4)	-	[P3(6, P:4), P4(4, P:6), P5(2, P:10)]	P2 starts.
15	-	P2 Completes	[P3(6, P:4), P4(4, P:6), P5(2, P:10)]	CPU free. P3 has highest priority.
15	P3 (6, P:4)	-	[P4(4, P:6), P5(2, P:10)]	P3 starts.
21	-	P3 Completes	[P4(4, P:6), P5(2, P:10)]	CPU free. P4 has highest priority.
21	P4 (4, P:6)	-	[P5(2, P:10)]	P4 starts.
25	-	P4 Completes	[P5(2, P:10)]	CPU free. P5 starts.
25	P5 (2, P:10)	-	``	P5 starts.
27	-	P5 Completes	``	

Gantt Chart

```
| P1 | P6 | P1 | P2 | P3 | P4 | P5 |
0          6        11   13   15        21       25   27
```

Performance Calculation

Process ID	CT	TAT (CT - AT)	WT (TAT - BT)
P1	13	13 - 0 = 13	13 - 8 = 5
P2	15	15 - 1 = 14	14 - 2 = 12
P3	21	21 - 2 = 19	19 - 6 = 13
P4	25	25 - 3 = 22	22 - 4 = 18
P5	27	27 - 5 = 22	22 - 2 = 20
P6	11	11 - 6 = 5	5 - 5 = 0

- **Average Waiting Time (AWT):** $(5+12+13+18+20+0)/6 \approx 11.33$ ms
- **Average Turnaround Time (ATT):** $(13+14+19+22+22+5)/6 \approx 15.83$ ms

4.3 Critical Issues and Solutions

A major problem inherent in priority scheduling algorithms is the potential for **indefinite blocking**, or **starvation**.[21] In a heavily loaded system, if there is a continuous stream of high-priority processes, a low-priority process may be forced to wait indefinitely, never getting a chance to run.[23]

The most common solution to this problem is **aging**.[12] Aging is a technique of gradually increasing the priority of processes that have been waiting in the system for a long time. For example, a process's priority could be incremented every few seconds it spends in the ready queue. Eventually, the low-priority process will "age" enough to have its priority raised to a level where it can be scheduled for execution, thus guaranteeing it will not starve.[22] The introduction of aging marks a crucial step toward dynamic scheduling, where the scheduler alters a process's parameters over time to correct a systemic flaw like starvation. This feedback mechanism is a core principle fully realized in more advanced algorithms like the Multilevel Feedback Queue.

Section 5: Round Robin (RR) Scheduling

Algorithm Principles

Round Robin (RR) scheduling is designed specifically for time-sharing and interactive systems, where providing good response time is a primary goal.[9] It is a preemptive algorithm that operates similarly to FCFS but adds a timer-based preemption mechanism. The ready queue is treated as a circular FIFO queue.[25] Each process is allocated a small, fixed unit of CPU time called a

time quantum or **time slice**, which is generally in the range of 10 to 100 milliseconds.[9]

The scheduler dispatches the process at the head of the ready queue. If the process completes its CPU burst before the time quantum expires, it relinquishes the CPU voluntarily. If it is still running when the quantum expires, a timer interrupt occurs, and the scheduler preempts the process, placing it at the tail of the ready queue.[26] The scheduler then proceeds to execute the next process at the head of the queue. This ensures that every process gets a fair share of the CPU, preventing starvation.[9]

Analysis: The Critical Role of Time Quantum (q)

The performance of the RR algorithm is critically dependent on the size of the time quantum, q.[25] The choice of

q represents a fundamental trade-off between system responsiveness and CPU efficiency.

- **If q is very large:** The likelihood of a time quantum expiring before a process completes its burst becomes very low. In the limit, as q approaches infinity, RR scheduling degenerates into FCFS scheduling.[25]
- **If q is very small:** Responsiveness improves, as processes are switched frequently. However, the overhead from frequent context switching can become excessive. If the quantum is smaller than the context-switch time, the system will spend more time switching between processes than doing useful work, leading to a significant drop in CPU efficiency and overall throughput.[25]

Therefore, the time quantum q acts as a crucial tuning parameter. It must be large enough relative to the context-switch time to minimize overhead, but small enough to provide good response time for interactive users.[25] This parameterized nature is a key feature of many practical, real-world schedulers.

Solved Problems

Problem 1: Impact of Varying Time Quantum

Consider the following set of processes arriving at time 0.
Process Specification

Process ID	Arrival Time (AT)	Burst Time (BT)
P1	0	10
P2	0	5
P3	0	8

Solution: Scenario A (Time Quantum, q = 2)

Ready Queue and Execution Trace

Time	Running Process (Rem. Time)	Event	Ready Queue State
0	P1 (10)	-	[P2, P3]
2	P2 (5)	P1 preempted	[P3, P1(8)]
4	P3 (8)	P2 preempted	[P1(8), P2(3)]
6	P1 (8)	P3 preempted	[P2(3), P3(6)]
8	P2 (3)	P1 preempted	[P3(6), P1(6)]
10	P3 (6)	P2 preempted	[P1(6), P2(1)]
12	P1 (6)	P3 preempted	[P2(1), P3(4)]
14	P2 (1)	P1 preempted	[P3(4), P1(4)]
15	P3 (4)	P2 completes	[P1(4), P3(4)]
17	P1 (4)	P3 preempted	[P3(2), P1(4)]
19	P3 (2)	P1 preempted	[P1(2), P3(2)]
21	P1 (2)	P3 completes	[P1(2)]
23	-	P1 completes	``

Gantt Chart

```
|P1|P2|P3|P1|P2|P3|P1|P2|P3|P1|P3|P1|
0  2  4  6  8 10 12 14 15 17 19 21 23
```

Performance Calculation

Process ID	CT	TAT (CT - AT)	WT (TAT - BT)
P1	23	23 - 0 = 23	23 - 10 = 13
P2	15	15 - 0 = 15	15 - 5 = 10
P3	21	21 - 0 = 21	21 - 8 = 13

- **Average Waiting Time (AWT):** (13+10+13)/3=12.0 ms
- **Average Turnaround Time (ATT):** (23+15+21)/3≈19.67 ms

Solution: Scenario B (Time Quantum, q = 5)

Ready Queue and Execution Trace

Time	Running Process (Rem. Time)	Event	Ready Queue State
0	P1 (10)	-	[P2, P3]
5	P2 (5)	P1 preempted	[P3, P1(5)]
10	P3 (8)	P2 completes	[P1(5)]
15	P1 (5)	P3 preempted	[P3(3)]
20	P3 (3)	P1 completes	``
23	-	P3 completes	``

Gantt Chart

```
| P1 | P2 | P3 | P1 | P3 |
0    5    10   15   20   23
```

Performance Calculation

Process ID	CT	TAT (CT - AT)	WT (TAT - BT)
P1	20	20 - 0 = 20	20 - 10 = 10
P2	10	10 - 0 = 10	10 - 5 = 5
P3	23	23 - 0 = 23	23 - 8 = 15

- **Average Waiting Time (AWT):** $(10+5+15)/3=10.0$ ms
- **Average Turnaround Time (ATT):** $(20+10+23)/3\approx17.67$ ms

Analysis: Increasing the time quantum from 2 to 5 reduced the number of context switches, leading to a lower AWT. This demonstrates the direct impact of q on performance.[25]

Problem 2: Staggered Arrival

Consider the following set of processes with a time quantum q = 2.
Process Specification

Process ID	Arrival Time (AT)	Burst Time (BT)
P1	0	5
P2	4	2
P3	5	4

Solution

Ready Queue and Execution Trace

Time	Running Process (Rem. Time)	Event	Ready Queue State
0	P1 (5)	P1 Arrives	``
2	P1 (3)	Quantum Exp.	[P1(3)]
4	P1 (1)	Quantum Exp., P2 Arrives	[P2(2), P1(1)]
6	P1 (1)	P2 Completes, P3 Arrives	[P1(1), P3(4)]
7	P3 (4)	P1 Completes	[P3(4)]
9	P3 (2)	Quantum Exp.	[P3(2)]
11	-	P3 Completes	``

Gantt Chart

```
| P1 | P2 |P1| P3 |
0      4    6  7      11
```

Performance Calculation

Process ID	CT	TAT (CT - AT)	WT (TAT - BT)
P1	7	7 - 0 = 7	7 - 5 = 2
P2	6	6 - 4 = 2	2 - 2 = 0
P3	11	11 - 5 = 6	6 - 4 = 2

- **Average Waiting Time (AWT):** (2+0+2)/3≈1.33 ms
- **Average Turnaround Time (ATT):** (7+2+6)/3=5.0 ms

Analysis: This problem from [26] shows how the ready queue is managed with staggered arrivals. At time 4, P2 arrives and is placed in the queue. P1's quantum expires, so it is placed at the tail of the queue, resulting in the order [P2, P1].

Problem 3: Simultaneous Arrival (Complex)

Consider the following processes arriving at time 0 with a time quantum q = 5.
Process Specification

Process ID	Arrival Time (AT)	Burst Time (BT)
P1	0	21
P2	0	3
P3	0	6
P4	0	2

Solution

Ready Queue and Execution Trace

Time	Running Process (Rem. Time)	Event	Ready Queue State
0	P1 (21)	-	[P2, P3, P4]
5	P2 (3)	P1 preempted	[P3, P4, P1(16)]
8	P3 (6)	P2 completes	[P4, P1(16)]
13	P4 (2)	P3 preempted	[P1(16), P3(1)]
15	P1 (16)	P4 completes	[P3(1)]
20	P3 (1)	P1 preempted	[P1(11)]

21	P1 (11)	P3 completes	``
26	P1 (6)	P1 preempted	[P1(6)]
31	P1 (1)	P1 preempted	[P1(1)]
32	-	P1 completes	``

Gantt Chart

```
| P1 | P2 | P3 |P4| P1 |P3| P1 |
0    5    8    13 15   20 21         32
```

Performance Calculation

Process ID	CT	TAT (CT - AT)	WT (TAT - BT)
P1	32	32 - 0 = 32	32 - 21 = 11
P2	8	8 - 0 = 8	8 - 3 = 5
P3	21	21 - 0 = 21	21 - 6 = 15
P4	15	15 - 0 = 15	15 - 2 = 13

- **Average Waiting Time (AWT):** (11+5+15+13)/4=11.0 ms
- **Average Turnaround Time (ATT):** (32+8+21+15)/4=19.0 ms

Analysis: A comprehensive example from [9] showing multiple cycles through the ready queue.

Problem 4: Staggered Arrival (from NPTEL)

Consider the following processes with a time quantum q = 15.
Process Specification

Process ID	Arrival Time (AT)	Burst Time (BT)
P0	0	80
P1	10	20
P2	10	10
P3	80	20
P4	85	50

Solution

Ready Queue and Execution Trace
A simplified trace showing the queue state at the start of each quantum.

Time	Running Process	Ready Queue at start of quantum	Notes
0	P0	``	P1, P2 arrive during this quantum.
15	P1	[P2, P0(65)]	
30	P2	[P0(65), P1(5)]	
40	P0	[P1(5)]	P2 completes early.
55	P1	[P0(50)]	
60	P0	``	P1 completes early. P3 arrives at t=80, P4 at t=85.
75	P3	[P0(35), P4(50)]	
90	P4	[P0(35), P3(5)]	

105	P0	[P3(5), P4(35)]	
120	P3	[P4(35), P0(20)]	
125	P4	[P0(20)]	P3 completes early.
140	P0	[P4(20)]	
155	P4	[P0(5)]	
170	P0	[P4(5)]	
175	P4	``	P0 completes early.
180	-		P4 completes early.

Gantt Chart
(A detailed Gantt chart would be very long. The sequence is P0, P1, P2, P0, P1, P0, P3, P4, P0, P3, P4, P0, P4, P0, P4)
Completion Times: P0=175, P1=60, P2=40, P3=125, P4=180.

Performance Calculation

Process ID	CT	TAT (CT - AT)	WT (TAT - BT)
P0	175	175 - 0 = 175	175 - 80 = 95
P1	60	60 - 10 = 50	50 - 20 = 30
P2	40	40 - 10 = 30	30 - 10 = 20
P3	125	125 - 80 = 45	45 - 20 = 25
P4	180	180 - 85 = 95	95 - 50 = 45

- **Average Waiting Time (AWT):** (95+30+20+25+45)/5=43.0 sec
- **Average Turnaround Time (ATT):** (175+50+30+45+95)/5=79.0 sec

Analysis: This problem from [28] (with corrected calculations) shows RR with a large quantum and long processes, illustrating how waiting times can accumulate significantly.

Section 6: Multilevel Queue Scheduling (MLQ)

Principles

Multilevel Queue (MLQ) scheduling addresses the limitation that a single scheduling algorithm is often insufficient for a system with a diverse mix of processes. MLQ partitions the ready queue into several separate queues.[29] Processes are permanently assigned to one queue upon entry, typically based on some static property such as process type (e.g., system vs. user), memory size, or a fixed priority level.[29]

Each queue has its own scheduling algorithm, tailored to the characteristics of the processes it holds. For example, a high-priority **foreground queue** for interactive processes might use Round Robin (RR) for good response time, while a low-priority **background queue** for batch processes might use First-Come, First-Served (FCFS) for simplicity.[29]

Scheduling must also occur *between* the queues. This is commonly implemented as a fixed-priority preemptive scheme. Queues are given absolute priority over lower-priority queues. For instance, no process in the background queue can run unless the foreground queue is empty. If a foreground process arrives while a background process is running, the background process is preempted.[29]

Analysis

The main advantage of MLQ is its flexibility in applying different scheduling policies to different classes of processes. However, this static assignment of processes to queues is also its main drawback. There is little to no mobility between queues. This can lead to **starvation** for processes in lower-priority queues if the higher-priority queues are continuously supplied with new processes.[29]

Solved Problem

Consider a system with 3 queues and the following processes.
Queue Structure

Queue ID	Priority	Scheduling Algorithm	Time Quantum (q)
Q1	1 (Highest)	Round Robin (RR)	2
Q2	2	Shortest Remaining Time First (SRTF)	N/A
Q3	3 (Lowest)	First-Come, First-Served (FCFS)	N/A

Process Specification

Process ID	Arrival Time (AT)	Burst Time (BT)	Assigned Queue
P1	7	2	Q1
P2	4	6	Q2
P3	4	4	Q2
P4	2	3	Q3
P5	0	12	Q3

Solution

Multilevel Ready Queue and Execution Trace
The scheduler always services Q1 first, then Q2, then Q3. A process in a lower-priority queue is preempted by any arrival in a higher-priority queue.

Time	Running	Event	Q1 (RR,	Q2 (SRTF)	Q3 (FCFS)	Notes

	Process		q=2)			
0	P5	P5 Arrives	``	``	[P5]	Only Q3 has a process. P5 starts.
2	P5	P4 Arrives	``	``	[P5(running), P4]	P5 continues (FCFS in Q3).
4	P3	P2, P3 Arrive	``	[P2(6), P3(4)]	[P5(rem 8), P4]	P2/P3 arrive in Q2, preempting P5. In Q2 (SRTF), P3 is shorter than P2. P3 starts.
7	P1	P1 Arrives	[P1(2)]	[P2(6), P3(rem 1)]	[P5(rem 8), P4]	P1 arrives in Q1, preempting P3. P1 starts.
9	P3	P1 Completes	``	[P2(6), P3(rem 1)]	[P5(rem 8), P4]	Q1 is empty. Scheduler looks at Q2. P3 has shortest rem. time. P3 resumes.
10	P2	P3 Completes	``	[P2(6)]	[P5(rem 8), P4]	Q1 empty. In Q2, only P2 remains. P2 starts.

16	P5	P2 Completes	``	``	[P5(rem 8), P4]	Q1, Q2 empty. Scheduler looks at Q3. P5 was first. P5 resumes.
24	P4	P5 Completes	``	``	[P4]	In Q3, only P4 remains. P4 starts.
27	-	P4 Completes	``	``	``	All processes finished.

Gantt Chart

```
| P5 | P3 | P1 |P3| P2 | P5 | P4 |
0        4       7    9  10        16             24     27
```

Performance Calculation

Process ID	CT	TAT (CT - AT)	WT (TAT - BT)
P1	9	9 - 7 = 2	2 - 2 = 0
P2	16	16 - 4 = 12	12 - 6 = 6
P3	10	10 - 4 = 6	6 - 4 = 2
P4	27	27 - 2 = 25	25 - 3 = 22
P5	24	24 - 0 = 24	24 - 12 = 12

- **Average Waiting Time (AWT):** (0+6+2+22+12)/5=8.4 ms
- **Average Turnaround Time (ATT):** (2+12+6+25+24)/5=13.8 ms

Analysis: This detailed example from [29] showcases the complex interactions in an MLQ system, particularly the preemption of lower-priority processes (P5 by P3, P3 by P1) and the application of different scheduling rules (SRTF in Q2, FCFS in Q3) once higher queues are empty.

Section 7: Multilevel Feedback Queue Scheduling (MLFQ)

Principles

Multilevel Feedback Queue (MLFQ) scheduling is one of the most general and sophisticated CPU scheduling algorithms. It was designed to achieve the best of all worlds: provide low response time for interactive jobs while also ensuring long-running CPU-bound jobs get processed, all without requiring prior knowledge of a process's burst time.[34]

Unlike MLQ, where processes are permanently assigned to a queue, MLFQ allows processes to **move between queues**.[31] The scheduler learns about a process's behavior over time and adjusts its priority accordingly. This "feedback" mechanism is the algorithm's defining feature.[33]

Rules of Operation

While specific implementations vary, most MLFQ schedulers follow a common set of rules [34]:

1. **Priority Rule:** A process with higher priority (in a higher queue) runs before any process with lower priority. If multiple processes are in the same queue, they are typically scheduled using Round Robin.[33]

2. **Entry Rule:** When a new process enters the system, it is placed in the highest-priority queue.[34] The scheduler initially assumes all jobs are short and interactive.
3. **Demotion Rule:** If a process uses its entire time quantum at a given priority level without blocking for I/O, its priority is reduced, and it is moved down to the next lower queue. This rule penalizes CPU-bound jobs, which prove themselves to be "long" jobs by consuming their full time slice.[34] To prevent gaming the scheduler, a robust implementation tracks the total CPU time a process has used at a given level, demoting it once its total allotment is consumed, regardless of how many times it has been scheduled.[36]
4. **Promotion Rule (Aging):** To prevent starvation of long-running jobs that have been demoted to low-priority queues, a promotion mechanism is used. After a certain time period S, all jobs in the system are moved back to the highest-priority queue. This "priority boost" ensures that all jobs, including long-running ones, get a chance to run.[35]

By dynamically adjusting priorities based on observed behavior, MLFQ effectively approximates SRTF. It gives high priority to short jobs (which complete quickly in the top queues) and low priority to long jobs (which filter down to the bottom queues), achieving excellent responsiveness and preventing starvation without needing an "oracle" to predict burst times.[34]

Solved Problem

Consider an MLFQ system with the following structure.
Queue Structure

Queue ID	Priority	Scheduling Algorithm	Time Quantum (q)
Q0	1 (Highest)	Round Robin (RR)	8
Q1	2	Round Robin (RR)	16
Q2	3 (Lowest)	First-Come, First-Served (FCFS)	N/A

A process is demoted if it uses its full quantum. A new process enters at Q0.

Process Specification

Process ID	Arrival Time (AT)	Burst Time (BT)
P1	0	20
P2	5	6
P3	12	30

Solution

Multilevel Feedback Queue and Execution Trace

Time	Running Process	Event	Q0 (RR, q=8)	Q1 (RR, q=16)	Q2 (FCFS)	Notes
0	P1	P1 Arrives	[P1]	``	``	P1 starts in Q0.
5	P1	P2 Arrives	[P1(running)], [P2]	``	``	P2 enters Q0.
8	P2	P1 uses full q=8	[P2]	[P1(rem 12)]	``	P1 is demoted to Q1. P2 from Q0 runs.
12	P2	P3 Arrives	[P2(running)], [P3]	[P1(rem 12)]	``	P3 enters Q0.
14	P3	P2 completes	[P3]	[P1(rem 12)]	``	P2 finishes early, stays at Q0 priority (but is gone). P3 from Q0

						runs.
22	P1	P3 uses full q=8	``	[P1(rem 12), P3(rem 22)]	``	P3 is demoted to Q1. Q0 is empty. P1 from Q1 runs.
38	P3	P1 uses full q=16	``	[P3(rem 22)]	[P1(rem -4 -> 0)]	P1's BT was 20, ran 8 in Q0, so 12 remain. It uses all 12 (less than q=16) and completes. P3 from Q1 runs.
54	P3	P3 uses full q=16	``	``	[P3(rem 6)]	P3 is demoted to Q2. Q0, Q1 empty. P3 from Q2 runs (FCFS).
60	-	P3 completes	``	``	``	All processes finished.

Gantt Chart

```
| P1 | P2 | P3 | P1 | P3 | P3 |
0          8       14         22              38              54       60
```

Performance Calculation

Process ID	CT	TAT (CT - AT)	WT (TAT - BT)
P1	38	38 - 0 = 38	38 - 20 = 18
P2	14	14 - 5 = 9	9 - 6 = 3
P3	60	60 - 12 = 48	48 - 30 = 18

- **Average Waiting Time (AWT):** (18+3+18)/3=13.0 ms
- **Average Turnaround Time (ATT):** (38+9+48)/3=31.67 ms

Analysis: This problem, based on the structure from [31] and [31], illustrates the core feedback mechanism. P1 and P3, being long jobs, are demoted. P2, a short job, completes quickly at a high priority. The scheduler successfully separates jobs by their behavior without prior knowledge.

Section 8: Comparative Analysis and Strategic Selection

The Grand Comparison Table

The choice of a CPU scheduling algorithm involves a trade-off between conflicting objectives such as minimizing waiting time, maximizing throughput, and ensuring responsiveness. The following table synthesizes the characteristics of the discussed algorithms to provide a high-level comparison, which is a valuable tool for study and strategic analysis.

Algorithm	CPU Scheduling Type	Avg. Waiting Time	Avg. Turnaround Time	Response Time	Starvation Risk	Implementation Complexity

FCFS	Non-Preemptive	Poor/High	Poor/High	Poor/High	No	Very Low
SJF (Non-P)	Non-Preemptive	Optimal/Low	Optimal/Low	Can be high	Yes (without aging)	Low (if BT known)
SRTF (Preemptive)	Preemptive	Optimal/Low	Optimal/Low	Good/Low	Yes (without aging)	Moderate
Priority (Non-P)	Non-Preemptive	Variable	Variable	Variable	High (without aging)	Low
Priority (P)	Preemptive	Variable	Variable	Good/Low	High (without aging)	Moderate
Round Robin (RR)	Preemptive	High	High	Very Low	No	Low
MLQ	Preemptive (inter-queue)	Variable	Variable	Variable	Yes (for low queues)	High
MLFQ	Preemptive	Good/Low	Good/Low	Very Low	No (with aging)	Very High

Strategic Selection

The optimal scheduling algorithm is highly dependent on the target system's requirements.[5]

- **Batch Systems:** For systems where jobs are run in batches and user interaction is not a concern, throughput and turnaround time are key.
    - **FCFS** is simple to implement but its performance is unreliable.[8]

    - **SJF** is optimal for minimizing AWT and is a strong choice if burst times can be reasonably estimated or are provided by the user, which is more common in batch environments.[14]
- **Interactive Systems:** For general-purpose and time-sharing systems, responsiveness is the most critical metric. Users expect the system to respond quickly to their commands.
    - **Round Robin (RR)** is a classic choice, as it is designed specifically to minimize response time by giving all processes a fair and frequent share of the CPU.[9]
    - **Multilevel Feedback Queue (MLFQ)** is often considered the best choice for this environment. It provides excellent response time for interactive tasks while still allowing CPU-bound background tasks to make progress, adapting dynamically to the workload.[34]
- **Real-Time Systems:** These systems have strict deadlines that must be met. The predictability of the scheduler is more important than its average performance.
    - **Preemptive Priority Scheduling** is essential. Critical real-time tasks are assigned the highest priorities to ensure they can preempt any other process and meet their deadlines.[21]

In conclusion, while simple algorithms like FCFS and SJF provide important theoretical foundations, modern, general-purpose operating systems typically employ more sophisticated, parameterized schedulers like MLFQ. These advanced algorithms combine the principles of priority, preemption, and feedback to dynamically balance the competing goals of responsiveness, fairness, and efficiency.

Works cited

1. Operating System Tutorial - GeeksforGeeks, accessed on August 2, 2025, https://www.geeksforgeeks.org/operating-systems/operating-systems/
2. CPU Scheduling in Operating Systems - GeeksforGeeks, accessed on August 2, 2025, https://www.geeksforgeeks.org/operating-systems/cpu-scheduling-in-operating-systems/
3. CPU Scheduling in Operating System - Scaler Topics, accessed on August 2, 2025, https://www.scaler.com/topics/operating-system/cpu-scheduling/
4. Mca Os Syllabus | PDF | Operating System | Process (Computing) - Scribd, accessed on August 2, 2025, https://www.scribd.com/document/37407684/Mca-Os-Syllabus
5. COURSE STRUCTURE FOR MCA, accessed on August 2, 2025, https://makautwb.ac.in/syllabus/MCA_New_Syllabus.pdf
6. Program for Shortest Job First (or SJF) CPU Scheduling | Set 1 (Non ..., accessed on August 2, 2025,

https://www.geeksforgeeks.org/dsa/program-for-shortest-job-first-or-sjf-cpu-scheduling-set-1-non-preemptive/
7. Unit IV – CPU Scheduling and Algorithm, accessed on August 2, 2025, https://aissmspoly.org.in/wp-content/uploads/2020/01/CPU-Scheduling-and-Algorithm-.pdf
8. FCFS - First Come First Serve CPU Scheduling - GeeksforGeeks, accessed on August 2, 2025, https://www.geeksforgeeks.org/dsa/first-come-first-serve-cpu-scheduling-non-preemptive/
9. Round Robin Scheduling Algorithm | Studytonight, accessed on August 2, 2025, https://www.studytonight.com/operating-system/round-robin-scheduling
10. Preemptive and Non-Preemptive Scheduling - GeeksforGeeks, accessed on August 2, 2025, https://www.geeksforgeeks.org/operating-systems/preemptive-and-non-preemptive-scheduling/
11. Scheduling Algorithms In OS (Operating System) Explained +Examples - Unstop, accessed on August 2, 2025, https://unstop.com/blog/scheduling-algorithms-in-operating-system
12. Operating Systems: CPU Scheduling, accessed on August 2, 2025, https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/5_CPU_Scheduling.html
13. First Come First Serve Problems | PDF | Computing | Software ..., accessed on August 2, 2025, https://www.scribd.com/document/659722874/First-Come-First-Serve-Problems
14. Shortest Job First or SJF CPU Scheduling - GeeksforGeeks, accessed on August 2, 2025, https://www.geeksforgeeks.org/operating-systems/shortest-job-first-or-sjf-cpu-scheduling/
15. SJF Process and Examples | MyCareerwise, accessed on August 2, 2025, https://mycareerwise.com/content/sjf-process-and-examples/content/exam/gate/computer-science
16. Shortest Job First Scheduling in Operating Systems - Tutorialspoint, accessed on August 2, 2025, https://www.tutorialspoint.com/operating_system/os_shortest_job_first_scheduling.htm
17. SJF Scheduling - SRTF - CPU Scheduling | PDF - Scribd, accessed on August 2, 2025, https://www.scribd.com/document/524597323/7-SJF-Scheduling
18. Introduction to shortest remaining time first (SRTF) algorithm - Educative.io, accessed on August 2, 2025, https://www.educative.io/answers/introduction-to-shortest-remaining-time-first-srtf-algorithm
19. Shortest Remaining Time First (Preemptive SJF) Scheduling Algorithm - GeeksforGeeks, accessed on August 2, 2025, https://www.geeksforgeeks.org/dsa/shortest-remaining-time-first-preemptive-sjf-scheduling-algorithm/

20. SRTF Process and Examples - MyCareerwise, accessed on August 2, 2025, https://mycareerwise.com/content/srtf-process-and-examples/content/exam/gate/computer-science
21. Priority Scheduling in Operating System - GeeksforGeeks, accessed on August 2, 2025, https://www.geeksforgeeks.org/operating-systems/priority-scheduling-in-operating-system/
22. Priority Scheduling | Hexainclude, accessed on August 2, 2025, http://www.hexainclude.com/priority-scheduling/
23. Priority Scheduling Algorithm in Operating Systems - Tutorialspoint, accessed on August 2, 2025, https://www.tutorialspoint.com/operating_system/os_priority_scheduling_algorithm.htm
24. Preemptive Priority CPU Scheduling Algorithm - GeeksforGeeks, accessed on August 2, 2025, https://www.geeksforgeeks.org/operating-systems/preemptive-priority-cpu-scheduling-algortithm/
25. Round Robin Process and Examples | MyCareerwise, accessed on August 2, 2025, https://mycareerwise.com/content/round-robin-process-and-examples/content/exam/gate/computer-science
26. Round Robin Scheduling in Operating System - GeeksforGeeks, accessed on August 2, 2025, https://www.geeksforgeeks.org/operating-systems/round-robin-scheduling-in-operating-system/
27. Round-robin scheduling - Wikipedia, accessed on August 2, 2025, https://en.wikipedia.org/wiki/Round-robin_scheduling
28. Worked out Examples - Nptel, accessed on August 2, 2025, https://archive.nptel.ac.in/content/storage2/courses/106108101/pdf/Worked_Out_Problems/Mod_3.pdf
29. Multilevel Queue Scheduling in Operating Systems - Tutorialspoint, accessed on August 2, 2025, https://www.tutorialspoint.com/operating_system/os_multilevel_queue_scheduling.htm
30. Multilevel Queue (MLQ) CPU Scheduling - GeeksforGeeks, accessed on August 2, 2025, https://www.geeksforgeeks.org/operating-systems/multilevel-queue-mlq-cpu-scheduling/
31. Chapter 5: CPU Scheduling - FSU Computer Science, accessed on August 2, 2025, https://www.cs.fsu.edu/~zwang/files/cop4610/Fall2016/chapter5.pdf
32. Multilevel Queue Scheduling Algorithm - Studytonight, accessed on August 2, 2025, https://www.studytonight.com/operating-system/multilevel-queue-scheduling
33. Unit-II Scheduling: Part-II, accessed on August 2, 2025, https://techiefood4u.wordpress.com/wp-content/uploads/2020/02/unit-ii-scheduling-part-2.pdf

34. Scheduling: The Multi-Level Feedback Queue - cs.wisc.edu, accessed on August 2, 2025, https://pages.cs.wisc.edu/~remzi/OSTEP/cpu-sched-mlfq.pdf
35. Multilevel Feedback Queue Scheduling (MLFQ) CPU Scheduling ..., accessed on August 2, 2025, https://www.geeksforgeeks.org/operating-systems/multilevel-feedback-queue-scheduling-mlfq-cpu-scheduling/
36. Multilevel Feedback Queue - Medium, accessed on August 2, 2025, https://medium.com/@francescofranco_39234/multilevel-feedback-queue-3ae862436a95
37. Preemptive Priority CPU Scheduling Algorithm - Tutorialspoint, accessed on August 2, 2025, https://www.tutorialspoint.com/preemptive-priority-cpu-scheduling-algorithm
38. CPU Scheduling Exercises Problem 2 Solutions, accessed on August 2, 2025, https://resources.saylor.org/wwwresources/archived/site/wp-content/uploads/2011/11/CS401-CPU-Scheduling-Exercise-Problem-2-Solution-FINAL.pdf