# CPU Scheduling Algorithms I

Week 5

SDB

Autumn 2025

# Agenda: The Art of CPU Scheduling

**Deciding Who Gets the CPU and When**

1. **Scheduling Criteria & Goals** *(What do we optimize for?)*
2. **FCFS: First-Come, First-Served** *(Simplicity, but with a catch)*
3. **SJF: Shortest Job First** *(Optimality vs. Practicality)*
4. **Round Robin (RR)** *(Fairness in Time-Sharing)*
5. **Algorithm Comparison & Evaluation** *(Understanding the Trade-offs)*
6. **Summary & Discussion**

## Agenda: The Art of CPU Scheduling

**Deciding Who Gets the CPU and When**

**Think Ahead: Resource Allocation**

In a multi-process environment, many processes are ready to run, but there's only one CPU (for a single-core system). How does the OS decide which process runs next? What are the implications of different decision-making strategies on system performance and user experience?

## Scheduling Criteria: What Makes a "Good" Schedule?

- **CPU Utilization:** Keep the CPU as busy as possible (range 0-100%).
- **Throughput:** Number of processes completed per unit time. High throughput means more work done.
- **CPU Burst Time:** The amount of time a process needs to execute on the CPU before it either terminates or performs an I/O operation. This is a **key characteristic of a process** and a crucial factor for many scheduling algorithms (e.g., Shortest-Job-First).
- **Turnaround Time (TAT):** Total time from process arrival to completion (Waiting Time + Execution Time). Minimize this.
- **Waiting Time (WT):** Total time a process spends in the Ready Queue, waiting for the CPU. Minimize this.
- **Response Time (RT):** Time from a request submitted until the first response is produced (for interactive systems). Minimize this for user satisfaction.
- **Fairness:** Each process gets a fair share of the CPU over time. Prevents starvation.

## Scheduling Criteria: What Makes a "Good" Schedule?

- **CPU Utilization:** Keep the CPU as busy as possible (range 0-100%).
- **Throughput:** Number of processes completed per unit time. High throughput means more work done.
- **CPU Burst Time:** The amount of time a process needs to execute on the CPU before it either terminates or performs an I/O operation. This is a **key characteristic of a process** and a crucial factor for many scheduling algorithms (e.g., Shortest-Job-First).
- **Turnaround Time (TAT):** Total time from process arrival to completion (Waiting Time + Execution Time). Minimize this.
- **Waiting Time (WT):** Total time a process spends in the Ready Queue, waiting for the CPU. Minimize this.
- **Response Time (RT):** Time from a request submitted until the first response is produced (for interactive systems). Minimize this for user satisfaction.
- **Fairness:** Each process gets a fair share of the CPU over time. Prevents starvation.

**The Trade-off Challenge**

It is often impossible to optimize all criteria simultaneously. For example, optimizing average turnaround time might lead to poor response time for some processes. OS designers must choose which criteria to prioritize based on system goals.

## Additional CPU Scheduling Considerations

Beyond the primary goals and metrics, a good scheduler must also account for these practical considerations and parameters.

- **Scheduler Overhead:** The amount of time and resources (CPU cycles, memory) the scheduler itself consumes to make a decision and perform a context switch. A good scheduler should be fast and efficient, minimizing overhead so that more CPU time can be spent on executing user processes.
- **Starvation:** A specific problem that arises from a lack of fairness. It occurs when a low-priority process is continuously prevented from gaining access to the CPU by higher-priority processes, causing it to never finish execution. It's a key consideration when designing priority-based scheduling algorithms.
- **Priority:** This is a numerical or categorical value assigned to a process that the scheduler uses as a decision-making factor. It's not a performance metric to be optimized, but a crucial input parameter for many scheduling policies, such as Priority Scheduling and Multilevel Feedback Queue.
- **Latency:** A broader term for delay. While "Response Time" specifically measures the delay for interactive tasks, other forms of latency, like I/O latency, are also important, especially in real-time or embedded systems. A good scheduler tries to minimize latency where it matters most.

## FCFS: First-Come, First-Served (FIFO Scheduling)

**Concept:** Processes are scheduled in the exact order they arrive in the Ready Queue, similar to a queue at a bank.

**Characteristics:**

- **Non-preemptive:** Once a process starts executing, it runs to completion (or blocks for I/O) without interruption.
- **Simple and easy to implement:** Relies on a simple FIFO queue.

## FCFS: First-Come, First-Served (FIFO Scheduling)

**Concept:** Processes are scheduled in the exact order they arrive in the Ready Queue, similar to a queue at a bank.

**Characteristics:**

- **Non-preemptive:** Once a process starts executing, it runs to completion (or blocks for I/O) without interruption.
- **Simple and easy to implement:** Relies on a simple FIFO queue.

**Pros & Cons:**

- **Pros:** Simple, fair in a temporal sense (first in, first out).
- **Cons:**
  - **High Average Waiting Time:** Especially if a long process arrives before several short ones.
  - **Convoy Effect:** A short process gets stuck behind a long process, leading to low CPU utilization and poor performance for the short process. This scenario is most common when a long CPU-bound process is followed by multiple I/O-bound processes.

## FCFS Example & Performance Analysis

**Processes (Arrival Time, Burst Time):**

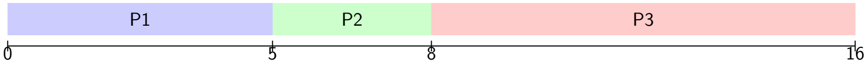
- P1: (0, 5) *(Arrives at 0, needs 5 units CPU)*
- P2: (1, 3) *(Arrives at 1, needs 3 units CPU)*
- P3: (2, 8) *(Arrives at 2, needs 8 units CPU)*

## FCFS Example & Performance Analysis

**Processes (Arrival Time, Burst Time):**

- P1: (0, 5)   *(Arrives at 0, needs 5 units CPU)*
- P2: (1, 3)   *(Arrives at 1, needs 3 units CPU)*
- P3: (2, 8)   *(Arrives at 2, needs 8 units CPU)*

**Gantt Chart (Execution Order: P1 $\rightarrow$ P2 $\rightarrow$ P3):**

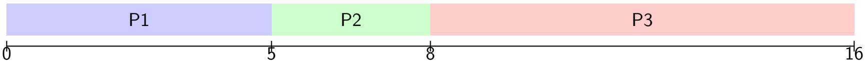


## FCFS Example & Performance Analysis

**Processes (Arrival Time, Burst Time):**

- P1: (0, 5) *(Arrives at 0, needs 5 units CPU)*
- P2: (1, 3) *(Arrives at 1, needs 3 units CPU)*
- P3: (2, 8) *(Arrives at 2, needs 8 units CPU)*

**Gantt Chart (Execution Order: P1 $\rightarrow$ P2 $\rightarrow$ P3):**



**Performance Metrics Calculation:**

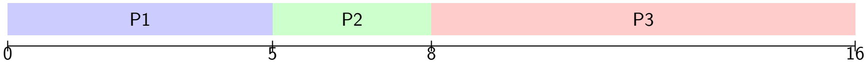
| Process | Arrival Time | Burst Time | Completion Time | Turnaround Time (TAT) | Waiting Time (WT) |
|---|---|---|---|---|---|
| P1 | 0 | 5 | 5 | 5 - 0 = **5** | 5 - 5 = **0** |
| P2 | 1 | 3 | 8 | 8 - 1 = **7** | 7 - 3 = **4** |
| P3 | 2 | 8 | 16 | 16 - 2 = **14** | 14 - 8 = **6** |

## FCFS Example & Performance Analysis

**Processes (Arrival Time, Burst Time):**

- P1: (0, 5) *(Arrives at 0, needs 5 units CPU)*
- P2: (1, 3) *(Arrives at 1, needs 3 units CPU)*
- P3: (2, 8) *(Arrives at 2, needs 8 units CPU)*

**Gantt Chart (Execution Order: P1 → P2 → P3):**



**Performance Metrics Calculation:**

| Process | Arrival Time | Burst Time | Completion Time | Turnaround Time (TAT) | Waiting Time (WT) |
|---|---|---|---|---|---|
| P1 | 0 | 5 | 5 | 5 - 0 = **5** | 5 - 5 = **0** |
| P2 | 1 | 3 | 8 | 8 - 1 = **7** | 7 - 3 = **4** |
| P3 | 2 | 8 | 16 | 16 - 2 = **14** | 14 - 8 = **6** |

**Average Turnaround Time:** $(5 + 7 + 14)/3 =$ **8.67** units **Average Waiting Time:** $(0 + 4 + 6)/3 =$ **3.33** units

## SJF: Shortest Job First (Optimizing Waiting Time)

**Concept:** The process with the shortest CPU burst time is selected for execution next.

**Types of SJF:**

- **Non-preemptive SJF:** Once the CPU is allocated to a process, it cannot be preempted until its CPU burst is completed.
- **Preemptive SJF (Shortest-Remaining-Time-First - SRTF):** If a new process arrives with a CPU burst time less than the **remaining** time of the currently executing process, the CPU is preempted.

## SJF: Shortest Job First (Optimizing Waiting Time)

**Concept:** The process with the shortest CPU burst time is selected for execution next.

**Types of SJF:**

- **Non-preemptive SJF:** Once the CPU is allocated to a process, it cannot be preempted until its CPU burst is completed.
- **Preemptive SJF (Shortest-Remaining-Time-First - SRTF):** If a new process arrives with a CPU burst time less than the **remaining** time of the currently executing process, the CPU is preempted.

**Pros & Cons:**

- **Pros:** **Optimal** – Gives the minimum average waiting time for a given set of processes.
- **Cons:**
  - **Impractical:** Requires knowing the exact future CPU burst time in advance, which is generally not possible. (Can be estimated using exponential averaging of past bursts).
  - **Starvation:** Long processes might never get to execute if there's a continuous stream of short processes.

## SJF Example (Non-Preemptive)

**Processes (Arrival Time, Burst Time):**

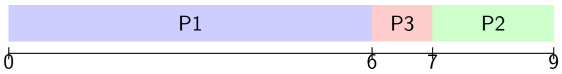
- P1: (0, 6)
- P2: (2, 2)
- P3: (4, 1)

## SJF Example (Non-Preemptive)

**Processes (Arrival Time, Burst Time):**

- P1: (0, 6)
- P2: (2, 2)
- P3: (4, 1)

**Gantt Chart (Execution Order: P1 → P3 → P2):**

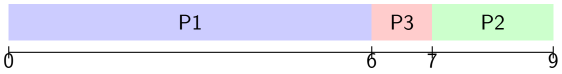


## SJF Example (Non-Preemptive)

**Processes (Arrival Time, Burst Time):**

- P1: (0, 6)
- P2: (2, 2)
- P3: (4, 1)

**Gantt Chart (Execution Order: P1 → P3 → P2):**



**Performance Metrics Calculation:**

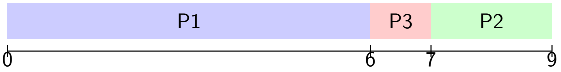
| Process | Arrival Time | Burst Time | Completion Time | Turnaround Time (TAT) | Waiting Time (WT) |
|---|---|---|---|---|---|
| P1 | 0 | 6 | 6 | 6 - 0 = **6** | 6 - 6 = **0** |
| P2 | 2 | 2 | 9 | 9 - 2 = **7** | 7 - 2 = **5** |
| P3 | 4 | 1 | 7 | 7 - 4 = **3** | 3 - 1 = **2** |

## SJF Example (Non-Preemptive)

**Processes (Arrival Time, Burst Time):**

- P1: (0, 6)
- P2: (2, 2)
- P3: (4, 1)

**Gantt Chart (Execution Order: P1 → P3 → P2):**



**Performance Metrics Calculation:**

| **Process** | **Arrival Time** | **Burst Time** | **Completion Time** | **Turnaround Time (TAT)** | **Waiting Time (WT)** |
|---|---|---|---|---|---|
| P1 | 0 | 6 | 6 | 6 - 0 = **6** | 6 - 6 = **0** |
| P2 | 2 | 2 | 9 | 9 - 2 = **7** | 7 - 2 = **5** |
| P3 | 4 | 1 | 7 | 7 - 4 = **3** | 3 - 1 = **2** |

**Average Turnaround Time:** $(6 + 7 + 3)/3 =$ **5.33** units **Average Waiting Time:** $(0 + 5 + 2)/3 =$ **2.33** units

## SJF Example (Preemptive - SRTF)

**Processes (Arrival Time, Burst Time):**

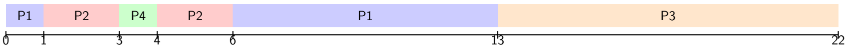
- P1: (0, 8)
- P2: (1, 4)
- P3: (2, 9)
- P4: (3, 1)

## SJF Example (Preemptive - SRTF)

**Processes (Arrival Time, Burst Time):**

- P1: (0, 8)
- P2: (1, 4)
- P3: (2, 9)
- P4: (3, 1)

**Gantt Chart (Execution Order):**

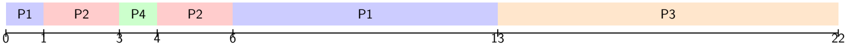


# SJF Example (Preemptive - SRTF)

**Processes (Arrival Time, Burst Time):**

- P1: (0, 8)
- P2: (1, 4)
- P3: (2, 9)
- P4: (3, 1)

**Gantt Chart (Execution Order):**



**Performance Metrics Calculation:**

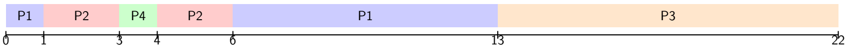
| Process | Arrival Time | Burst Time | Completion Time | Turnaround Time (TAT) | Waiting Time (WT) |
|---|---|---|---|---|---|
| P1 | 0 | 8 | 13 | 13 - 0 = **13** | 13 - 8 = **5** |
| P2 | 1 | 4 | 6 | 6 - 1 = **5** | 5 - 4 = **1** |
| P3 | 2 | 9 | 22 | 22 - 2 = **20** | 20 - 9 = **11** |
| P4 | 3 | 1 | 4 | 4 - 3 = **1** | 1 - 1 = **0** |

# SJF Example (Preemptive - SRTF)

**Processes (Arrival Time, Burst Time):**

- P1: (0, 8)
- P2: (1, 4)
- P3: (2, 9)
- P4: (3, 1)

**Gantt Chart (Execution Order):**



**Performance Metrics Calculation:**

| Process | Arrival Time | Burst Time | Completion Time | Turnaround Time (TAT) | Waiting Time (WT) |
|---|---|---|---|---|---|
| P1 | 0 | 8 | 13 | 13 - 0 = **13** | 13 - 8 = **5** |
| P2 | 1 | 4 | 6 | 6 - 1 = **5** | 5 - 4 = **1** |
| P3 | 2 | 9 | 22 | 22 - 2 = **20** | 20 - 9 = **11** |
| P4 | 3 | 1 | 4 | 4 - 3 = **1** | 1 - 1 = **0** |

**Average Turnaround Time:** $(13 + 5 + 20 + 1)/4 = 39/4 =$ **9.75** units **Average Waiting Time:** $(5 + 1 + 11 + 0)/4 = 17/4 =$ **4.25** units

## Round Robin (RR) Scheduling

**Concept:** Each process gets a small unit of CPU time, called a **time quantum (or time slice)**, typically 10 to 100 milliseconds. When a process's quantum expires, it is preempted and added to the end of the Ready Queue.

**Characteristics:**

- **Preemptive:** Yes, based on time quantum.
- **Fair:** Ensures that no process waits indefinitely (no starvation).
- Suitable for **time-sharing systems** where responsiveness is crucial.

## Round Robin (RR) Scheduling

**Concept:** Each process gets a small unit of CPU time, called a **time quantum (or time slice)**, typically 10 to 100 milliseconds. When a process's quantum expires, it is preempted and added to the end of the Ready Queue.

**Characteristics:**

- **Preemptive:** Yes, based on time quantum.
- **Fair:** Ensures that no process waits indefinitely (no starvation).
- Suitable for **time-sharing systems** where responsiveness is crucial.

**Pros & Cons:**

- **Pros:** Good response time, fair allocation of CPU, no starvation.
- **Cons:**
  - **Context Switch Overhead:** Frequent context switches increase overhead (CPU is not doing useful work during switches).
  - **Performance depends heavily on Quantum Size:**
    - **Large Quantum:** Approaches FCFS, poor response for short jobs.
    - **Small Quantum:** Increases context switch overhead, may reduce overall throughput. An optimal quantum is typically found between 80% of CPU bursts.

## RR Example (Quantum = 2)

**Processes (Arrival Time, Burst Time):**

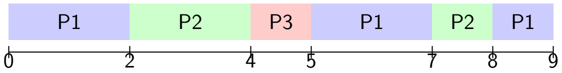
- P1: (0, 5)
- P2: (1, 3)
- P3: (2, 1)

## RR Example (Quantum = 2)

**Processes (Arrival Time, Burst Time):**

- P1: (0, 5)
- P2: (1, 3)
- P3: (2, 1)

**Gantt Chart (Quantum = 2):**

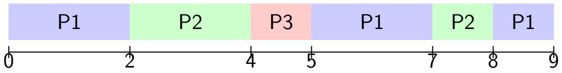


## RR Example (Quantum = 2)

**Processes (Arrival Time, Burst Time):**

- P1: (0, 5)
- P2: (1, 3)
- P3: (2, 1)

**Gantt Chart (Quantum = 2):**



**Performance Metrics Calculation:**

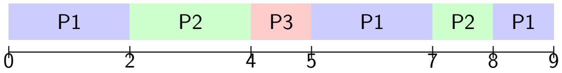
| Process | Arrival Time | Burst Time | Completion Time | Turnaround Time (TAT) | Waiting Time (WT) | Response Time (RT) |
|---|---|---|---|---|---|---|
| P1 | 0 | 5 | 9 | 9 - 0 = **9** | 9 - 5 = **4** | 0 - 0 = **0** |
| P2 | 1 | 3 | 8 | 8 - 1 = **7** | 7 - 3 = **4** | 2 - 1 = **1** |
| P3 | 2 | 1 | 5 | 5 - 2 = **3** | 3 - 1 = **2** | 4 - 2 = **2** |

## RR Example (Quantum = 2)

**Processes (Arrival Time, Burst Time):**

- P1: (0, 5)
- P2: (1, 3)
- P3: (2, 1)

**Gantt Chart (Quantum = 2):**



**Performance Metrics Calculation:**

| Process | Arrival Time | Burst Time | Completion Time | Turnaround Time (TAT) | Waiting Time (WT) | Response Time (RT) |
|---|---|---|---|---|---|---|
| P1 | 0 | 5 | 9 | 9 - 0 = **9** | 9 - 5 = **4** | 0 - 0 = **0** |
| P2 | 1 | 3 | 8 | 8 - 1 = **7** | 7 - 3 = **4** | 2 - 1 = **1** |
| P3 | 2 | 1 | 5 | 5 - 2 = **3** | 3 - 1 = **2** | 4 - 2 = **2** |

**Average Turnaround Time:** $(9 + 7 + 3)/3 =$ **6.33** units **Average Waiting Time:** $(4 + 4 + 2)/3 =$ **3.33** units **Average Response Time:** $(0 + 1 + 2)/3 =$ **1.0** units

## Comparison & Evaluation of Basic Scheduling Algorithms

| **Algorithm** | **Type** | **Starvation?** | **Optimal for** | **Real-World Use Case** | **Quantum Dependent?** |
|---|---|---|---|---|---|
| FCFS | Non-preemptive | No | Simplicity | Batch systems, simple job queues | No |
| SJF | Non-preemptive | Yes (long jobs) | Average Waiting Time | Batch systems (if burst known), specialized tasks | No |
| SRTF | Preemptive | Yes (long jobs) | Average Waiting Time | Idealized scenario, some real-time systems (with priority) | No |
| RR | Preemptive | No | Response Time / Fairness | Interactive systems, time-sharing OS (desktops, servers) | Yes (crucial) |

## Comparison & Evaluation of Basic Scheduling Algorithms

| **Algorithm** | **Type** | **Starvation?** | **Optimal for** | **Real-World Use Case** | **Quantum Dependent?** |
|---|---|---|---|---|---|
| FCFS | Non-preemptive | No | Simplicity | Batch systems, simple job queues | No |
| SJF | Non-preemptive | Yes (long jobs) | Average Waiting Time | Batch systems (if burst known), specialized tasks | No |
| SRTF | Preemptive | Yes (long jobs) | Average Waiting Time | Idealized scenario, some real-time systems (with priority) | No |
| RR | Preemptive | No | Response Time / Fairness | Interactive systems, time-sharing OS (desktops, servers) | Yes (crucial) |

**Key Takeaways from Comparison:**

- No single algorithm is universally "best"; choice depends on system goals.
- Preemptive algorithms generally provide better responsiveness but incur context switch overhead.
- Practical algorithms often combine elements of these basic strategies.

## Key Takeaways

- CPU scheduling is essential for managing concurrent processes and balancing various system performance **criteria** (e.g., CPU utilization, throughput, turnaround, waiting, response time, fairness).
- **FCFS** is simple but can suffer from the convoy effect and high average waiting times.
- **SJF** (including **SRTF**) is theoretically optimal for average waiting time but impractical due to the need for future burst time prediction and prone to starvation.
- **Round Robin** provides fairness and good response time for interactive systems but introduces context switch overhead, highly dependent on the time quantum.
- No single scheduling algorithm is perfect; understanding their **trade-offs** is crucial for OS design.

## Key Takeaways

- CPU scheduling is essential for managing concurrent processes and balancing various system performance **criteria** (e.g., CPU utilization, throughput, turnaround, waiting, response time, fairness).
- **FCFS** is simple but can suffer from the convoy effect and high average waiting times.
- **SJF** (including **SRTF**) is theoretically optimal for average waiting time but impractical due to the need for future burst time prediction and prone to starvation.
- **Round Robin** provides fairness and good response time for interactive systems but introduces context switch overhead, highly dependent on the time quantum.
- No single scheduling algorithm is perfect; understanding their **trade-offs** is crucial for OS design.

**Reflection Prompt**

Imagine you are designing a scheduling algorithm for a new smartphone OS. Which of the criteria discussed would be most important, and which algorithm (or a combination thereof) would you lean towards? Why?

## Next Week Preview: Advanced Scheduling & Beyond

**Building on the Basics:**

- **Priority Scheduling:** Understanding how different priorities affect execution, including preemption and non-preemption based on priority.
- **Multilevel Queue Scheduling:** Designing complex schedulers with different queues for different process types.
- **Multilevel Feedback Queue Scheduling:** Addressing starvation and dynamically adjusting priorities.
- **Aging:** A technique to prevent starvation.
- Real-World Schedulers (e.g., Linux CFS, Windows Scheduler).

## Next Week Preview: Advanced Scheduling & Beyond

**Building on the Basics:**

- **Priority Scheduling:** Understanding how different priorities affect execution, including preemption and non-preemption based on priority.
- **Multilevel Queue Scheduling:** Designing complex schedulers with different queues for different process types.
- **Multilevel Feedback Queue Scheduling:** Addressing starvation and dynamically adjusting priorities.
- **Aging:** A technique to prevent starvation.
- Real-World Schedulers (e.g., Linux CFS, Windows Scheduler).

**Prep Tip for Next Session**

Review the concepts of preemptive vs. non-preemptive scheduling. Think about why simple priority scheduling might lead to starvation and how that problem could be solved.

## Outline

# Quick Quiz: Understanding Scheduling Concepts I

**Test Your Conceptual Knowledge:**

1. **True/False:** SJF always produces the shortest average response time. Explain why.
2. **Define:** What is the "Convoy Effect" in FCFS scheduling?
3. **Scenario:** A CPU-bound process runs for 10 seconds, then an I/O-bound process needs 1 second of CPU and 9 seconds of I/O.
   - Which algorithm would give the best response time to the I/O-bound process?
   - Which algorithm would result in the highest waiting time for the I/O-bound process if it arrived just after the CPU-bound one started?
4. **Trade-off:** In Round Robin, what happens if the time quantum is set to an extremely large value (e.g., 1000 seconds)? What about an extremely small value (e.g., 1 microsecond)?

## Quick Quiz: Understanding Scheduling Concepts II

**Think & Discuss**

Formulate your answers before reviewing the solutions or discussing with peers.

## Exercise: CPU Scheduling Calculations (Gantt & Metrics) I

**Instructions:** For each scenario, draw the Gantt chart and calculate the Turnaround Time, Waiting Time, and average for all processes. Assume context switch time is negligible.

**Processes:**

- P1: Arrival=0, Burst=8
- P2: Arrival=1, Burst=4
- P3: Arrival=2, Burst=9
- P4: Arrival=3, Burst=5

1. **FCFS Scheduling:** Draw the Gantt chart and calculate average TAT and WT.

2. **Non-Preemptive SJF Scheduling:** Draw the Gantt chart and calculate average TAT and WT.

## Exercise: CPU Scheduling Calculations (Gantt & Metrics) II

3. **Preemptive SJF (SRTF) Scheduling:** Draw the Gantt chart and calculate average TAT and WT. *(Hint: Track remaining burst times carefully!)*

4. **Round Robin Scheduling (Quantum = 4):** Draw the Gantt chart and calculate average TAT, WT, and Response Time.

**Calculation Tip**

For each process, remember:
Completion Time = Start Time + Burst Time (or last burst segment)
Turnaround Time = Completion Time - Arrival Time
Waiting Time = Turnaround Time - Burst Time
Response Time = First CPU Start Time - Arrival Time

# Advanced Topics & Important Details I

**Deepening Your Knowledge of CPU Scheduling**

- **Multiprocessor Scheduling:**
  - **Load Sharing:** Distributing processes among multiple CPUs.
  - **Processor Affinity:** Trying to keep a process on the same CPU to leverage cache locality.
  - **Symmetric Multiprocessing (SMP) vs. Asymmetric Multiprocessing (AMP) Schedulers.**
- **Real-Time Scheduling:**
  - **Hard Real-Time:** Guarantees completion by a deadline (e.g., flight control).
  - **Soft Real-Time:** Prioritizes real-time tasks but doesn't guarantee deadlines (e.g., multimedia).
  - Common algorithms: Rate Monotonic (RM), Earliest Deadline First (EDF).
- **Fair-Share Scheduling:**

## Advanced Topics & Important Details II

- Ensuring that users or groups of processes get a fair share of CPU time, rather than just individual processes.

- **Lottery Scheduling:**
  - A probabilistic approach where processes are given "tickets," and a random ticket is drawn to pick the next process. Can ensure fairness with good response time.

- **Dynamic Priority Adjustments (Aging):**
  - How OSes prevent starvation in priority-based systems by gradually increasing the priority of long-waiting processes.

- **Estimating Next CPU Burst (Exponential Averaging):**
  - The practical technique used by OSes to predict the next CPU burst for SJF-like behavior, based on a weighted average of past bursts.

- **Scheduler Data Structures:**

## Advanced Topics & Important Details III

- How ready queues are implemented (e.g., priority queues, linked lists of process control blocks) for efficient selection.

- **Real-World Scheduler Examples:**
  - **Linux CFS (Completely Fair Scheduler):** A very popular and sophisticated scheduler that aims for fairness in CPU allocation.
  - **Windows NT/Vista/7/10 Schedulers:** Priority-based, preemptive schedulers with 32 priority levels.
  - **MacOS Grand Central Dispatch:** A task-based concurrency framework that manages threads and queues.

## Categorizing CPU Scheduling Techniques I

**Understanding the Diverse Strategies Employed by Operating Systems**

### I. By Preemption Type

- **Non-Preemptive Scheduling:**
  - Once granted, CPU is held until burst completion or voluntary yield.
  - *Examples:* FCFS, Non-Preemptive SJF, *Non-Preemptive Priority*.
- **Preemptive Scheduling:**
  - CPU can be forcibly taken away (preempted) by an interrupt or higher-priority arrival.
  - *Examples:* Round Robin (RR), Shortest-Remaining-Time-First (SRTF), *Preemptive Priority*.

## Categorizing CPU Scheduling Techniques II

**II. By Optimization Goal / Strategy**

- **Batch-Oriented Scheduling:**
  - Focus: Maximizing throughput, minimizing average turnaround time.
  - *Examples:* FCFS, SJF.
- **Interactive / Time-Sharing Scheduling:**
  - Focus: Good response time, fairness among users.
  - *Examples:* Round Robin, Priority Scheduling (*e.g., often used in Desktop OS like Windows, macOS*).
- **Real-Time Scheduling:**
  - Focus: Meeting strict deadlines for critical tasks.
  - *Examples:* Rate Monotonic (RM), Earliest Deadline First (EDF) (*for Embedded Systems, Industrial Control*).

## Categorizing CPU Scheduling Techniques III

### III. By System Complexity / Specialized Needs

- **Multi-Level Queue & Feedback Schedulers:**
  - Processes organized into different queues, potentially moving between them based on behavior/priority.
  - *Example:* Many general-purpose OS schedulers use principles of feedback queues.
- **Multiprocessor Schedulers:**
  - Manages scheduling across multiple CPU cores.
  - *Concepts:* Load Balancing, Processor Affinity (*common in Server OS like Linux, Windows Server*).
- **Fair-Share Scheduling:**
  - Guarantees a minimum CPU share to users or groups, not just individual processes.
  - *Example:* Implemented in various forms, including parts of *Linux CFS (Completely Fair Scheduler)*.
- **Energy-Aware Schedulers:**
  - Prioritizes power efficiency alongside performance.
  - *Common in Mobile OS (e.g., Android, iOS) and Laptops*.