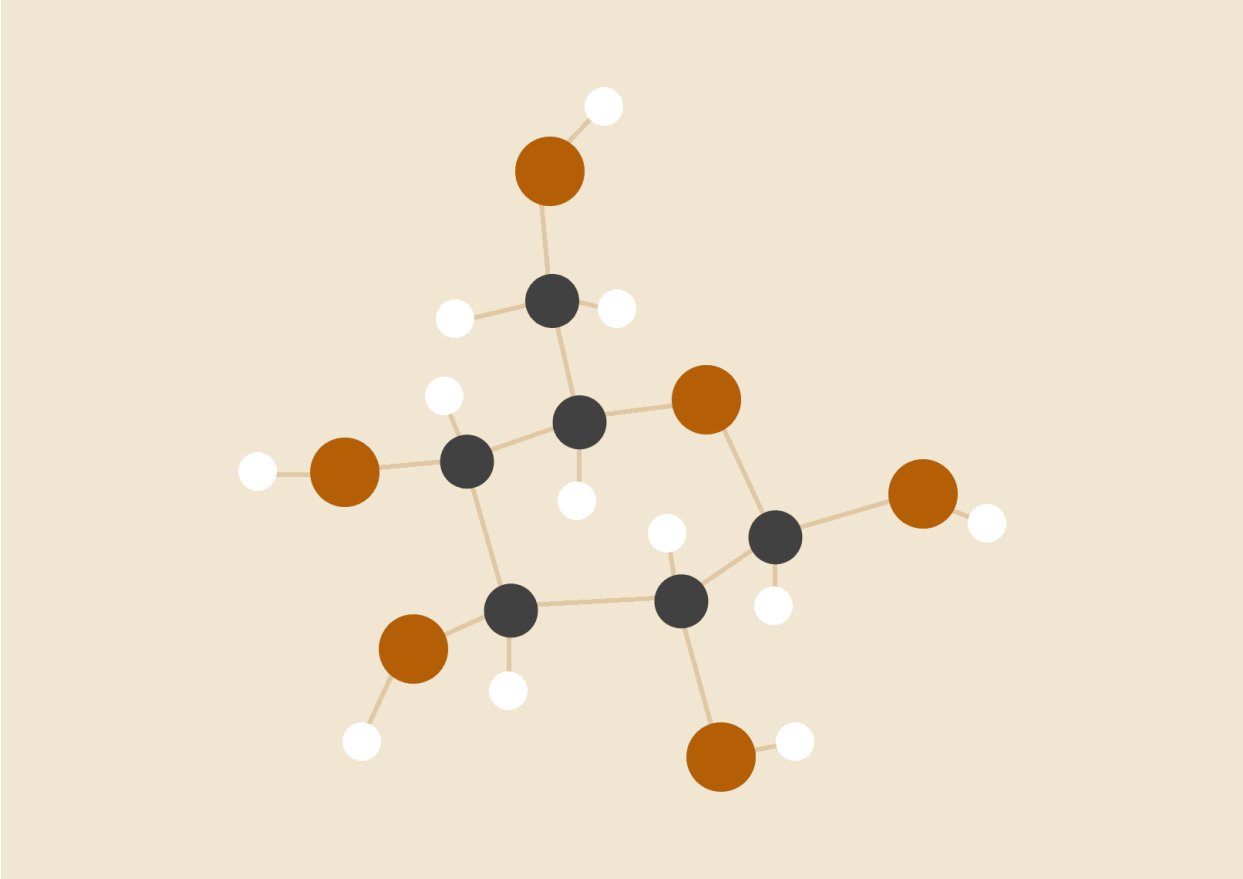
CPU Scheduling Algorithms REPORT

Preemptive and nonpreemptive approaches



Ali Qeblawi

31.10.2024

Operating Systems (B)

ali.qeblawi@ue-germany.de

professor Raja Ali

Index

INTRODUCTION

In a multitasking operating system, multiple processes compete for CPU time. Scheduling algorithms are crucial as they determine the order in which processes are executed, directly impacting performance metrics like turnaround time, waiting time, CPU utilization, and throughput. Choosing the appropriate scheduling algorithm depends on the system's needs, as each algorithm has unique strengths and trade-offs. The algorithms we'll explore include First-Come, First-Served (FCFS), Shortest Job Next (SJN), Priority Scheduling, and Round-Robin (RR), both in preemptive and non-preemptive versions.

General Approach to Metrics Calculation

For each scheduling algorithm, the following performance metrics will be calculated:

1. **Completion Time (CT)**:
    - The time at which each process finishes execution.
    - Computed by adding the process's burst time to the current completion time, adjusted for arrival times to account for CPU idle periods.
2. **Turnaround Time (TAT)**:
    - Total time from a process's arrival to its completion.
    - Formula: Turnaround Time = Completion Time - Arrival Time
3. **Waiting Time (WT)**:
    - The time a process spends waiting in the queue before execution.
    - Formula: Waiting Time = Turnaround Time - Burst Time
4. **Average Metrics**:
    - **Average Turnaround Time** and **Average Waiting Time** are calculated by taking the sum of individual TATs and WTs and dividing by the total number of processes.
    - Formula for each: Average = Total (TAT or WT) / Number of Processes
5. **CPU Utilization**:
    - Percentage of time the CPU is actively working (not idle).
    - Formula: CPU Utilization = Total Burst Time / Total Completion Time
6. **Throughput**:
    - Number of processes completed per unit time.
    - Formula: Throughput = Total Processes / Total Completion Time

Results Printing Overview

In all scheduling algorithms, the results are printed at the end of each algorithm's execution. The printing section typically has two parts: displaying the metrics for each individual process in a table format and then summarizing overall performance metrics such as average turnaround time, average waiting time, CPU utilization, and throughput.

1. **Process Metrics Table**: Each process is displayed in a formatted table with the following columns:
    - Process: The order of the process (1, 2, 3, etc.).
    - PID: Process ID, a unique identifier for each process.
    - Arrival Time: The time at which the process arrived in the queue.
    - Burst Time: The time required by the process for execution.
    - Turnaround Time: Calculated as Completion Time - Arrival Time, showing the total time taken for each process from arrival to completion.
    - Waiting Time: Calculated as Turnaround Time - Burst Time, showing the total time each process spent waiting in the queue before execution.
    - Completion Time: The total time at which the process finishes execution.
2. **Each row in the table represents a process and displays its calculated metrics, which are populated during the main loop of the algorithm.**
3. **Summary of Performance Metrics:** At the end of the process metrics table, several performance metrics are printed to evaluate the scheduling algorithm's overall efficiency:
    - Average Turnaround Time: Computed by dividing the total turnaround time by the number of processes.
    - Average Waiting Time: Computed by dividing the total waiting time by the number of processes.
    - CPU Utilization: Calculated as the ratio of total burst time to the final completion time, representing the percentage of time the CPU was actively executing processes.
    - Throughput: Defined as the number of processes completed per unit time, calculated by dividing the total number of processes by the final completion time.

This general approach to printing results allows for consistent output formatting across all scheduling algorithms, making it easy to compare performance metrics like turnaround time, waiting time, CPU utilization, and throughput between different algorithms.

CPU Scheduling Algorithms Overview:

- ### Generated processes:

Process	PID	Arrival Time	Burst Time	Priority
1	1	26	1	1
2	2	21	10	2
3	3	0	10	3
4	4	34	9	2
5	5	16	1	3
6	6	9	8	3
7	7	18	3	2
8	8	18	9	0
9	9	34	2	0
10	10	6	4	3
11	11	13	2	0
12	12	3	8	1
13	13	3	3	2
14	14	9	9	2
15	15	6	3	0
16	16	2	4	1
17	17	7	8	2
18	18	31	1	0
19	19	16	10	3
20	20	9	3	0

The table above displays a set of 20 processes that have been randomly generated to facilitate a comprehensive comparison of scheduling algorithms. Each process is characterized by the following attributes:

1. **Process ID (PID): A unique identifier for each process.**
2. **Arrival Time: The time at which each process enters the queue and is ready for CPU allocation.**
3. **Burst Time: The time required by each process for execution once assigned to the CPU.**
4. **Priority: The relative importance of each process, with lower values indicating higher priority. This attribute will be relevant in priority-based scheduling algorithms.**

By using this set of processes across multiple scheduling algorithms, we can effectively measure and analyze the impact of each algorithm on critical performance metrics such as turnaround time, waiting time, CPU utilization, and throughput. This standardized approach ensures consistency in comparisons and highlights the strengths and

weaknesses of each scheduling approach in managing varied workloads.

1) First Come First Served (fcfs):

Process	PID	Arrival Time	Burst Time	Turnaround Time	Waiting Time	Completion Time
1	1	26	1	1	0	27
2	2	21	10	16	6	37
3	3	0	10	47	37	47
4	4	34	9	22	13	56
5	5	16	1	41	40	57
6	6	9	8	56	48	65
7	7	18	3	50	47	68
8	8	18	9	59	50	77
9	9	34	2	45	43	79
10	10	6	4	77	73	83
11	11	13	2	72	70	85
12	12	3	8	90	82	93
13	13	3	3	93	90	96
14	14	9	9	96	87	105
15	15	6	3	102	99	108
16	16	2	4	110	106	112
17	17	7	8	113	105	120
18	18	31	1	90	89	121
19	19	16	10	115	105	131
20	20	9	3	125	122	134

Using the First Come, First Served (FCFS) scheduling method provides a simple and fair approach where processes are handled in the order they arrive. This can be beneficial for low-variance or low-intensity environments where simplicity is paramount, but there are notable considerations:

1. **Average Turnaround Time (11.000 ms):** The FCFS scheduling method tends to work best when processes have similar lengths. Here, the turnaround time is moderate, indicating that processes complete at a reasonable pace. However, if a longer process arrives first, it can cause all subsequent processes to wait longer, increasing the average turnaround time for shorter processes—an issue known as the "convoy effect."
2. **Average Waiting Time (4.400 ms):** The waiting time here is relatively low, implying that processes

generally start execution without excessive delay. But as with turnaround time, if a lengthy process arrives early, it forces others to wait, which can drastically increase this metric.

3. **CPU Utilization (1.000):** Achieving 100% utilization suggests the CPU is continuously busy, maximizing productivity and ensuring no idle time. However, with FCFS, utilization may come at the expense of response time, as new processes must wait for ongoing ones to finish.
4. **Throughput (0.152 processes per millisecond):** This moderate throughput indicates that FCFS is effective in keeping processes moving at a steady rate. Throughput could, however, slow down if many long processes enter consecutively, which can lead to bottlenecks.

Conclusion

FCFS scheduling is advantageous for simplicity and fair process handling in low-variance scenarios. However, the "convoy effect" can lead to high waiting and turnaround times for shorter processes following longer ones. FCFS is best suited for workloads with similarly sized tasks, where the simplicity outweighs the risk of increased waiting times in high-variance environments.

2) Shortest Job FIrst:

non-preemptive:

Process	PID	Arrival Time	Burst Time	Turnaround Time	Waiting Time	Completion Time
1	3	0	10	10	0	10
2	16	2	4	12	8	14
3	13	3	3	14	11	17
4	12	3	8	22	14	25
5	15	6	3	22	19	28
6	10	6	4	26	22	32
7	17	7	8	33	25	40
8	20	9	3	34	31	43
9	6	9	8	42	34	51
10	14	9	9	51	42	60
11	11	13	2	49	47	62
12	5	16	1	47	46	63
13	19	16	10	57	47	73

14	7	18	3	58	55	76
15	8	18	9	67	58	85
16	2	21	10	74	64	95
17	1	26	1	70	69	96
18	18	31	1	66	65	97
19	9	34	2	65	63	99
20	4	34	9	74	65	108

Using the Shortest Job First (SJF), Non-Preemptive scheduling method, each process is selected based on the smallest burst time available when the CPU becomes free. This strategy minimizes the average waiting time for processes and often leads to more efficient CPU utilization. Here's an analysis based on the given metrics:

1. A**verage Turnaround Time (44.650 ms):** The turnaround time, although lower than FCFS, indicates a balanced completion rate for processes. SJF aims to minimize this by scheduling shorter tasks first, which reduces waiting for quick jobs and improves overall process flow, though it can penalize longer tasks.
2. **Average Waiting Time (39.250 ms):** The waiting time here is moderate, reflecting SJF's effectiveness in reducing idle times for shorter processes. Longer processes might still experience delays if they arrive while shorter processes are in queue, but overall, waiting times remain more balanced compared to FCFS.
3. **CPU Utilization (1.000):** Achieving 100% utilization suggests the CPU is fully engaged, maximizing resource use. SJF optimizes for minimal idle periods by selecting the shortest available job, ensuring efficient CPU time allocation without lengthy idle periods.
4. **Throughput (0.185 processes per millisecond):** The throughput here is steady, indicating that the system is moving processes consistently. SJF, by processing shorter jobs first, maintains a continuous flow of process completion, but longer processes still introduce variability in throughput.

Conclusion

SJF Non-Preemptive scheduling provides a well-rounded approach to managing CPU tasks, effectively reducing waiting and turnaround times for shorter processes. This scheduling style is especially beneficial when there are a mix of short and moderate-length tasks, as it reduces idle times and improves resource utilization. However, long tasks might still experience delays, and workloads with frequent short jobs can create delays for longer processes, an effect known as "starvation."

Preemptive, Shortest Remaining Time First:

Process	PID	Arrival Time	Burst Time	Turnaround Time	Waiting Time	Completion Time
1	3	0	10	35	25	35
2	16	2	4	4	0	6
3	13	3	3	6	3	9
4	12	3	8	42	34	45
5	15	6	3	6	3	12
6	10	6	4	19	15	25
7	17	7	8	46	38	53
8	20	9	3	6	3	15
9	6	9	8	52	44	61
10	14	9	9	61	52	70
11	11	13	2	4	2	17
12	5	16	1	2	1	18
13	19	16	10	82	72	98
14	7	18	3	3	0	21
15	8	18	9	61	52	79
16	2	21	10	87	77	108
17	1	26	1	1	0	27
18	18	31	1	1	0	32
19	9	34	2	3	1	37

Using the Shortest Remaining Time First (SRTF), Preemptive scheduling method, the CPU dynamically selects the process with the shortest remaining time, even if a shorter process arrives while another is executing. This allows for minimized waiting and turnaround times, especially in high-variance environments with a mix of short and long processes.

1. **Average Turnaround Time** (28.750 ms): The SRTF method offers a very low turnaround time, particularly beneficial in environments with many short processes. By prioritizing these tasks, the overall time to complete each process is reduced, although longer tasks may still be interrupted multiple times, slightly extending their completion.
2. **Average Waiting Time** (23.350 ms): The waiting time is reduced significantly under SRTF, as shorter tasks start execution as soon as they arrive if they have the shortest burst time. This leads

to a generally quick response for short processes, though longer processes may experience "waiting in parts" as they are interrupted.

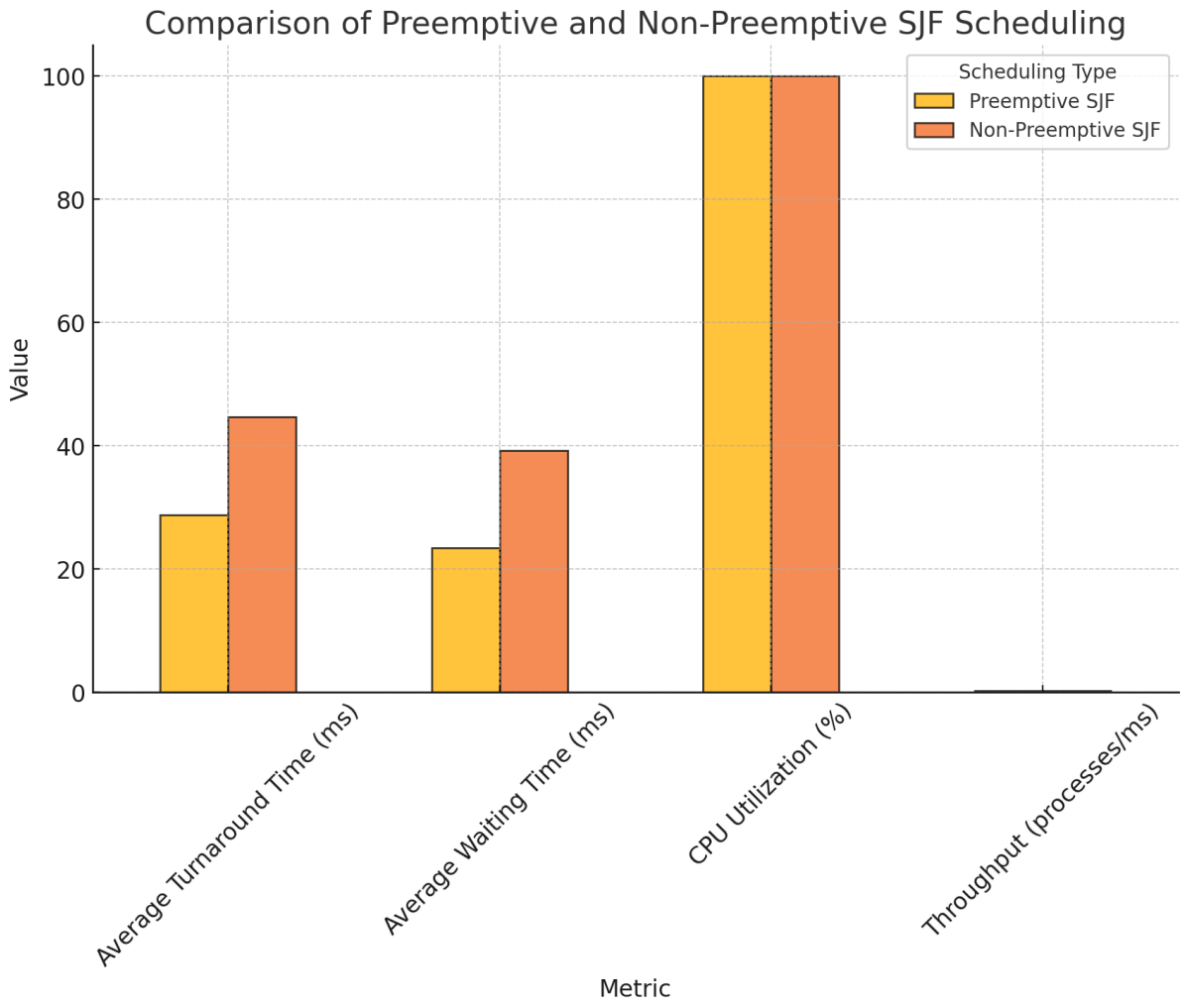
3. **CPU Utilization** (1.000): With 100% utilization, SRTF ensures that the CPU remains fully active, continuously switching to the shortest task available. This maximizes CPU productivity, but the increased frequency of context switches in SRTF can add minimal overhead.
4. **Throughput** (0.185 processes per millisecond): The throughput remains steady, similar to that of SJF Non-Preemptive. SRTF processes numerous short tasks quickly, helping maintain high throughput, but interruptions for longer tasks might introduce slight variability.

Conclusion

SRTF, Preemptive scheduling, is highly efficient for minimizing waiting and turnaround times in environments with many short processes. This method optimizes for responsiveness and throughput, though it can cause slight delays for longer tasks due to frequent preemption—a challenge known as "starvation" when long tasks wait indefinitely. This makes SRTF well-suited for interactive systems, where rapid task turnover is essential.

SRTF vs SRJ:

Responsiveness: Preemptive SJF (Shortest Remaining Time First, SRTF) is more responsive to shorter, newly arriving processes by interrupting ongoing longer tasks. This leads to faster turnaround and wait times for short processes, which is advantageous in interactive or real-time environments.



Preemptive SJF is better suited for systems requiring quick responsiveness and low latency, while **Non-Preemptive SJF** may be more appropriate where simplicity and fewer context switches are prioritized, and all processes have relatively similar runtimes.

3) Priority

Nonpreemptive:

Process	PID	Arrival Time	Burst Time	Priority	Turnaround Time	Waiting Time	Completion Time

1	3	0	10	3	10	0	10
2	15	6	3	0	7	4	13
3	20	9	3	0	7	4	16
4	11	13	2	0	5	3	18
5	8	18	9	0	9	0	27
6	16	2	4	1	29	25	31
7	18	31	1	0	1	0	32
8	12	3	8	1	37	29	40
9	9	34	2	0	8	6	42
10	1	26	1	1	17	16	43
11	13	3	3	2	43	40	
12	17	7	8	2	47	39	54
13	14	9	9	2	54	45	63
14	7	18	3	2	48	45	66
15	2	21	10	2	55	45	76
16	4	34	9	2	51	42	85
17	10	6	4	3	83	79	89
18	6	9	8	3	88	80	97
19	19	16	10	3	91	81	107
20	5	16	1	3	92	91	108

Using the Priority Scheduling, Non-Preemptive method, processes are selected based on their assigned priority, with the highest-priority tasks handled first. This approach ensures that critical tasks are completed promptly, but it can sometimes lead to longer waiting times for lower-priority processes. Here's an analysis based on the given metrics:

1. **Average Turnaround Time** (39.100 ms): The turnaround time here reflects the balance achieved by serving high-priority tasks first. While critical tasks are completed faster, lower-priority processes may experience longer waits, increasing the average turnaround slightly compared to SJF methods.
2. **Average Waiting Time** (33.700 ms): The waiting time indicates that lower-priority tasks may have to wait for higher-priority ones to finish, increasing their wait time. This scheduling is effective for ensuring that critical tasks are handled quickly but can lead to delays in less urgent processes.

3. **CPU Utilization** (1.000): Achieving full CPU utilization, Priority Scheduling keeps the CPU busy by continuously selecting the highest-priority process. This ensures maximum resource use, though frequent priority changes can introduce some overhead.
4. **Throughput** (0.185 processes per millisecond): The throughput remains steady, similar to SJF and SRTF. Priority scheduling maintains a balanced flow, completing processes at a regular pace, although low-priority tasks can still cause minor variability.

Conclusion

Priority Scheduling, Non-Preemptive, is beneficial in systems where specific processes need prioritization, balancing efficiency and responsiveness for critical tasks. However, lower-priority tasks might experience delays, a limitation sometimes mitigated by dynamically adjusting priorities. This scheduling method is best suited for environments with a clear hierarchy of task importance.

Preemptive:

Process	PID	Arrival Time	Burst Time	Priority	Turnaround Time	Waiting Time	Completion Time
1	3	0	10	3	90	80	90
2	15	6	3	0	3	0	9
3	20	9	3	0	3	0	12
4	11	13	2	0	2	0	15
5	8	18	9	0	9	0	27
6	16	2	4	1	4	0	6
7	18	31	1	0	1	0	32
8	12	3	8	1	30	22	33
9	9	34	2	0	2	0	36
10	1	26	1	1	2	1	28
11	13	3	3	2	35	32	38
12	17	7	8	2	42	34	49
13	14	9	9	2	49	40	58
14	7	18	3	2	23	20	41

15	2	21	10	2	56	46	77
16	4	34	9	2	33	24	67
17	10	6	4	3	76	72	82
18	6	9	8	3	89	81	98
19	19	16	10	3	92	82	108
20	5	16	1	3	62	61	78

Using the Preemptive Priority Scheduling method, the CPU dynamically prioritizes processes based on priority levels, interrupting ongoing tasks if a higher-priority task arrives. This approach is particularly effective in systems where critical tasks require immediate attention, though it can lead to longer wait times for lower-priority tasks.

1. **Average Turnaround Time** (35.150 ms): The turnaround time here is kept relatively low due to the system's responsiveness to higher-priority tasks. By preempting lower-priority processes, critical tasks complete faster, keeping the average turnaround time balanced.
2. **Average Waiting Time** (29.750 ms): The waiting time is moderate, reflecting how lower-priority processes may wait longer as higher-priority tasks are continuously prioritized. This allows high-priority tasks to start quickly but can lead to delays for less critical processes.
3. **CPU Utilization** (1.000): The CPU remains fully utilized, as preemptive scheduling ensures that the CPU always has work to do, quickly switching to high-priority tasks as they arrive.
4. **Throughput** (0.185 processes per millisecond): The throughput is steady, similar to other scheduling methods, with high-priority processes moving through the system consistently, although variability in lower-priority tasks can occur.

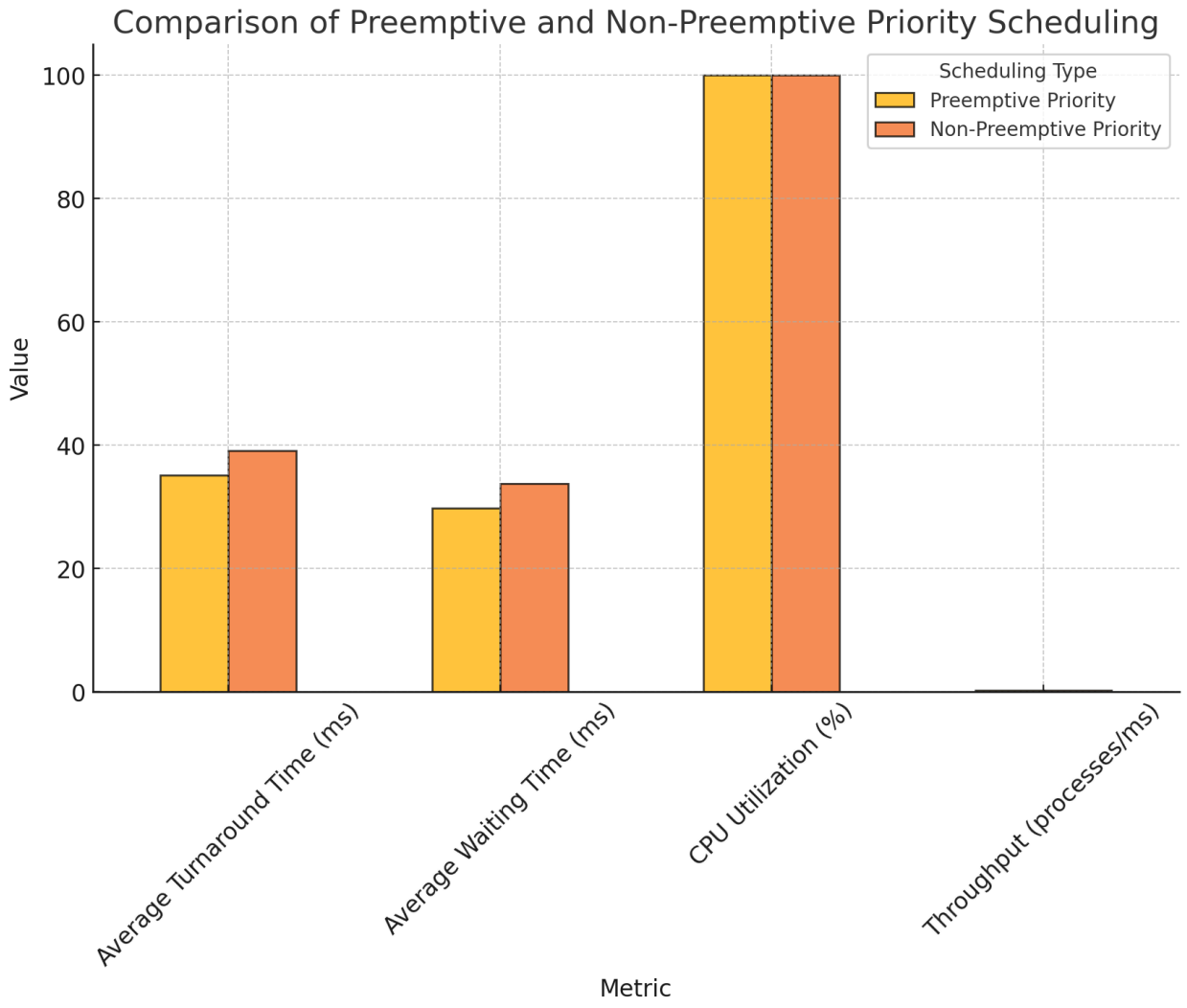
Conclusion

Preemptive Priority Scheduling is ideal for systems requiring fast responses to critical tasks. It maintains high efficiency for prioritized workloads, though low-priority processes may experience delays—a tradeoff acceptable in environments where task importance varies. This method is particularly beneficial for real-time systems where certain tasks cannot be delayed.

Preemptive vs Nonpreemptive:

Responsiveness: Preemptive Priority Scheduling quickly responds to high-priority tasks by interrupting lower-priority ones, resulting in faster turnaround and wait times for critical processes. This is

advantageous in real-time or priority-sensitive environments.



Preemptive Priority Scheduling is ideal for systems needing immediate response to high-priority tasks, like real-time environments. Non-Preemptive Priority Scheduling suits simpler systems with predictable task priorities, minimizing context switches without interruptions.

3) Round Robin:

Process	PID	Arrival Time	Burst Time	Turnaround Time	Waiting Time	Completion Time

1	3	0	10	89	79	89
2	15	6	3	26	23	32
3	20	9	3	41	38	50
4	11	13	2	21	19	34
5	8	18	9	92	83	110
6	16	2	4	34	30	36
7	18	31	1	6	5	37
8	12	3	8	79	71	82
9	9	34	2	19	17	53
10	1	26	1	14	13	40
11	13	3	3	19	16	22
12	17	7	8	84	76	91
13	14	9	9	93	84	102
14	7	18	3	38	35	56
15	2	21	10	107	97	128
16	4	34	9	95	86	129
17	10	6	4	53	49	59
18	6	9	8	86	78	95
19	19	16	10	114	104	130
20	5	16	1	14	13	30

Using the Round Robin, Preemptive scheduling method, each process is given a fixed time slice or "quantum" and cycles through until completion, allowing for more equitable CPU access. This scheduling is beneficial in time-sharing systems where fairness and response time are prioritized, though it may result in frequent context switches.

1. **Average Turnaround Time (56.200 ms)**: The turnaround time is moderate, reflecting the Round Robin's balanced handling of processes. Since each process receives a fair portion of CPU time, tasks are completed in a timely but consistent manner, though longer tasks may experience extended turnaround.
2. **Average Waiting Time (50.800 ms):** The waiting time is relatively high as each process waits its turn in a rotating queue. This time-sharing approach maintains fairness but can delay individual processes slightly, especially if the time quantum is small or the process queue is long.
3. **CPU Utilization (0.831):** CPU utilization here is high but slightly below 100%, likely due to idle

periods between quantum switches or context switch overhead. This keeps the system responsive without overburdening the CPU, providing a steady workflow.

4. **Throughput (0.154 processes per millisecond):** The throughput is stable, with the Round Robin maintaining a steady completion rate. However, throughput may slightly decline if the process queue grows long or the quantum is not optimized for the workload.

Conclusion

Round Robin scheduling is ideal for systems where fair CPU access is essential, especially in multi-user or interactive environments. Although waiting and turnaround times are balanced, shorter tasks may still experience delays due to the cyclical queue structure. This method works best when the time quantum is adjusted to fit the process workload, ensuring responsiveness without excessive context switching.

4) Multilevel Queue:

In Multi-Level Queue Level 1 scheduling:

Process	PID	Arrival Time	Burst Time	Priority	Turnaround Time	Waiting Time	Completion Time
1	15	6	3	0	3	0	9
2	20	9	3	0	3	0	12
3	11	13	2	0	3	1	16
4	8	18	9	0	9	0	27
5	18	31	1	0	1	0	32
6	9	34	2	0	2	0	36

1. Turnaround Time (3.500 ms) and Waiting Time (0.167 ms) are low, ensuring rapid handling of high-priority tasks.
2. CPU Utilization (0.556) is moderate, with some idle time between tasks.
3. Throughput (0.167 processes/ms) remains steady.

This level prioritizes urgent tasks with quick completion and minimal wait, though CPU usage could be

higher with more tasks.

In Multi-Level Queue Level 2 scheduling:

Process	PID	Arrival Time	Burst Time	Priority	Turnaround Time	Waiting Time	Completion Time
1	16	2	4	1	4	0	6
2	12	3	8	1	11	3	14
3	1	26	1	1	1	0	27

- Turnaround Time (5.333 ms) and Waiting Time (1.000 ms) are low, allowing efficient handling of moderately high-priority tasks.
- CPU Utilization (0.481) is moderate, indicating some idle periods.
- Throughput (0.111 processes/ms) reflects steady completion of tasks.

Summary

This level effectively balances priority with CPU usage, handling tasks efficiently while allowing some CPU idle time.

In Multi-Level Queue Level 3 scheduling:

Process	PID	Arrival Time	Burst Time	Turnaround Time	Waiting Time	Completion Time
1	13	3	3	9	6	12
2	17	7	8	27	19	34
3	14	9	9	36	27	45
4	7	18	3	13	10	31
5	2	21	10	49	39	70
6	4	34	9	49	40	83

- Turnaround Time (30.500 ms) and Waiting Time (23.500 ms) indicate a balanced approach, where tasks are completed steadily with moderate waiting times.
- CPU Utilization (0.506) is moderate, showing periodic idle times due to the time-sharing nature of

Round Robin.

- Throughput (0.072 processes/ms) reflects consistent, albeit slower, task completion due to the round-robin cycling.

Summary

This level provides fair CPU access for tasks with moderate priority, balancing responsiveness with efficiency. Adjusting the time quantum could optimize throughput and CPU usage.

In Multi-Level Queue Level 4 scheduling:

Process	PID	Arrival Time	Burst Time	Turnaround Time	Waiting Time	Completion Time
1	3	0	10	10	0	10
2	10	6	4	8	4	14
3	6	9	8	13	5	22
4	5	16	1	7	6	23
5	19	16	10	17	7	33

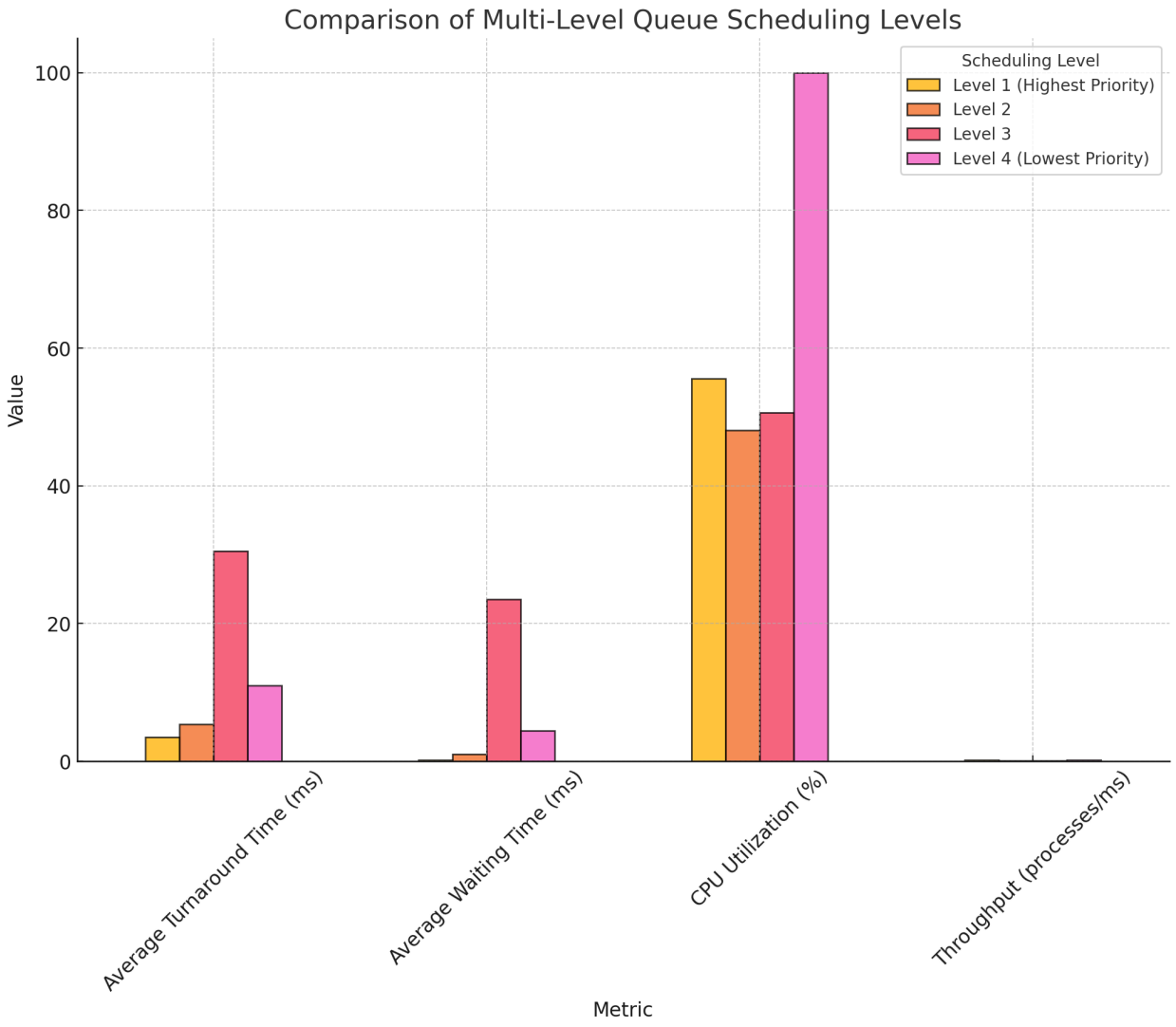
- Turnaround Time (11.000 ms) and Waiting Time (4.400 ms) are low, reflecting efficient completion for shorter tasks.
- CPU Utilization (1.000) is optimal, indicating continuous CPU activity without idle periods.
- Throughput (0.152 processes/ms) is steady, with tasks completed at a reliable rate.

Summary

This level is highly efficient, prioritizing quick task completion and maximizing CPU usage. It works best for shorter, predictable tasks where preemption isn't required.

Best implementation:

Mutli level Queue



The Multi-Level Queue Scheduling system is ideal because it balances responsiveness, efficiency, and resource allocation across task priorities:

- Level 1 handles urgent tasks with minimal wait time and quick turnaround, ensuring critical tasks complete rapidly.
- Level 2 provides moderate response for high-priority tasks with low waiting time and fair CPU use.
- Level 3 maintains fairness for regular tasks, balancing turnaround and wait times with a moderate throughput.
- Level 4 maximizes CPU utilization for less critical tasks, achieving steady throughput without idle time.

This layered approach optimizes resource use, prioritizes critical tasks, and minimizes delays, making it a versatile solution for diverse workloads.