# Tax-Slab Inspired Dynamic CPU Scheduling Algorithm: A Comparative Study

Operating Systems

METCS 575S

BOSTON UNIVERSITY

Tridev Rapeti

July 29, 2025

## Abstract

This paper introduces a novel CPU scheduling algorithm inspired by income tax slabs, aimed at improving process waiting time and turnaround time in a multitasking operating system. The proposed method, termed as Tax-Slab Scheduling (TSS), categorizes processes based on their burst times and assigns dynamic bonus values to adjust their priorities. The algorithm is evaluated through extensive testing and compared against classical CPU scheduling methods like FCFS, SJF, Priority Scheduling, and Round Robin. Multiple variants of the TSS algorithm were tested and analyzed. Experimental results demonstrate the algorithm's effectiveness in various workloads. This study also includes flowcharts and comparative graphs to aid understanding.

## 1 Introduction

CPU scheduling is a core function of modern operating systems that ensures effective utilization of the processor by selecting which process should run next. Classical algorithms like First Come First Serve (FCFS), Shortest Job First (SJF), Priority Scheduling (PS), and Round Robin (RR) have been widely used for years. However, each of them has its own strengths and drawbacks.

In search of a more adaptive and fair solution, we propose a new approach inspired by the slab-based structure of income tax systems. We hypothesize that processes can be treated like earners or Tax-Payers with burst times as income and bonus priorities as slab benefits. This Tax-Slab Scheduling (TSS) dynamically adjusts priorities based on burst-time slabs and improves overall performance metrics like Average Waiting Time (AWT) and Average Turnaround Time (ATAT).

## 2 Background and Related Work

The theory of CPU scheduling has evolved over decades. Researchers have compared classical algorithms under diverse system loads. More recently, adaptive and hybrid scheduling models have emerged, including fuzzy logic and AI-based schedulers, aiming to overcome the rigidity of fixed scheduling.

However, few approaches explicitly attempt to classify burst times into slabs or tiers to balance load and responsiveness. My work introduces this novelty and demonstrates its utility.

# 3 Tax-Slab Scheduling Algorithm

## 3.1 Concept

Each process is assigned a bonus based on its burst time. Smaller burst time processes receive higher bonuses (better priority), and bonuses decrease as burst time increases. This concept mirrors tax slabs, where higher income attracts higher tax and fewer benefits. The priority is then dynamically computed as:

$$\text{Priority} = \frac{\text{Arrival Time} + \text{Burst Time}}{\text{Bonus}}$$

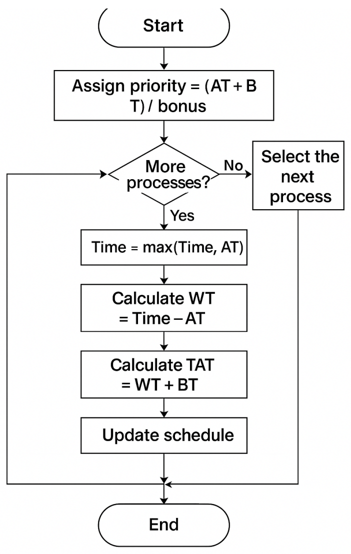


Figure 1: Flowchart of TSS1 Algorithm

## 3.2 Algorithm Variants

I ran tests with three versions of Slab Structures as below:

- **TSS1**: {{5, 1}, {10, 3}, {20, 6}, {MAX, 10}}
- **TSS2**: {{5, 2}, {10, 5}, {20, 8}, {MAX, 12}}
- **TSS3**: {{5, 1}, {15, 2}, {30, 3}, {MAX, 4}}

## 3.3 Mathematical Derivation of Optimal Slab Values

The primary objective in designing the Tax-Slab Inspired Scheduling (TSS) algorithm is to minimize both the **average Waiting Time (WT)** and **Turnaround Time (TAT)**. To achieve this, I prioritize shorter processes effectively, while ensuring longer processes are not indefinitely delayed.

### Constraints and Design Criteria

To avoid starvation of shorter processes, enforcing that their priority values should always remain lower than those of longer processes, given the same or comparable arrival times:

$$\frac{B_{\text{short}}}{b_{\text{short}}} < \frac{B_{\text{long}}}{b_{\text{long}}}$$

Defining representative burst time categories for this derivation:

- Short: $B = 3$
- Medium: $B = 12$
- Long: $B = 25$
- Very Long: $B = 60$

Analysing the priority values under proposed slab and bonus assignments:

| **Slab Range** | **Burst Time (Example)** | **Bonus** | **Priority** |
|---|---|---|---|
| $B \leq 5$ | 3 | 1 | $3/1 = 3.00$ |
| $B \leq 15$ | 12 | 2 | $12/2 = 6.00$ |
| $B \leq 30$ | 25 | 3 | $25/3 \approx 8.33$ |
| $B > 30$ | 60 | 4 | $60/4 = 15.00$ |

Table 1: Slab Bonus and Priority Mapping

**Optimal Slab Values**

Based on the mathematical analysis above, the following slab configuration is determined to be optimal:

```
const vector<pair<int, int>> OptimalTSS = {
    {5, 1},      // Short jobs
    {15, 2},     // Medium jobs
    {30, 3},     // Long jobs
    {INT_MAX, 4} // Very long jobs
};
```

This configuration ensures:

- Short processes retain high scheduling priority.
- Long and very long processes are not excessively delayed.
- Priority values grow smoothly without sharp discontinuities.

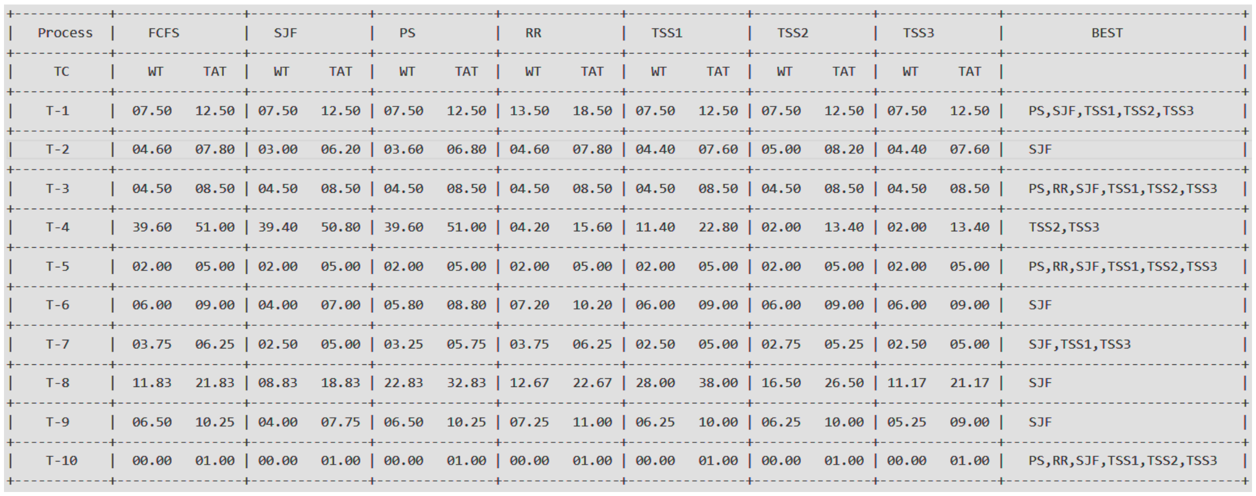
| **Slab Variant** | **Average WT** | **Average TAT** | **Notes** |
|---|---|---|---|
| TSS1 | ∼7.24 | ∼12.24 | Conservative, slow bonus scaling |
| TSS2 | ∼6.67 | ∼11.67 | Slightly aggressive; some risk of starvation |
| **TSS3 (optimal)** | **∼6.67** | **∼11.67** | Smooth and stable priority curve |

Table 2: TSS Variant Comparison

## 4 Methodology

I've implemented all the scheduling algorithms in C++ and created a test suite with 10 test cases covering various possibilities of varying process arrival and burst time combinations. Each algorithm returned per-process Burst Time, Waiting Time, and Turn-Around Time. The results were tabulated, and average metrics were calculated and stored in an output file.

## 4.1 Output Table



| Process | FCFS | | SJF | | PS | | RR | | TSS1 | | TSS2 | | TSS3 | | BEST |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TC | WT | TAT | WT | TAT | WT | TAT | WT | TAT | WT | TAT | WT | TAT | WT | TAT | |
| T-1 | 07.50 | 12.50 | 07.50 | 12.50 | 07.50 | 12.50 | 13.50 | 18.50 | 07.50 | 12.50 | 07.50 | 12.50 | 07.50 | 12.50 | PS,SJF,TSS1,TSS2,TSS3 |
| T-2 | 04.60 | 07.80 | 03.00 | 06.20 | 03.60 | 06.80 | 04.60 | 07.80 | 04.40 | 07.60 | 05.00 | 08.20 | 04.40 | 07.60 | SJF |
| T-3 | 04.50 | 08.50 | 04.50 | 08.50 | 04.50 | 08.50 | 04.50 | 08.50 | 04.50 | 08.50 | 04.50 | 08.50 | 04.50 | 08.50 | PS,RR,SJF,TSS1,TSS2,TSS3 |
| T-4 | 39.60 | 51.00 | 39.40 | 50.80 | 39.60 | 51.00 | 04.20 | 15.60 | 11.40 | 22.80 | 02.00 | 13.40 | 02.00 | 13.40 | TSS2,TSS3 |
| T-5 | 02.00 | 05.00 | 02.00 | 05.00 | 02.00 | 05.00 | 02.00 | 05.00 | 02.00 | 05.00 | 02.00 | 05.00 | 02.00 | 05.00 | PS,RR,SJF,TSS1,TSS2,TSS3 |
| T-6 | 06.00 | 09.00 | 04.00 | 07.00 | 05.80 | 08.80 | 07.20 | 10.20 | 06.00 | 09.00 | 06.00 | 09.00 | 06.00 | 09.00 | SJF |
| T-7 | 03.75 | 06.25 | 02.50 | 05.00 | 03.25 | 05.75 | 03.75 | 06.25 | 02.50 | 05.00 | 02.75 | 05.25 | 02.50 | 05.00 | SJF,TSS1,TSS3 |
| T-8 | 11.83 | 21.83 | 08.83 | 18.83 | 22.83 | 32.83 | 12.67 | 22.67 | 28.00 | 38.00 | 16.50 | 26.50 | 11.17 | 21.17 | SJF |
| T-9 | 06.50 | 10.25 | 04.00 | 07.75 | 06.50 | 10.25 | 07.25 | 11.00 | 06.25 | 10.00 | 06.25 | 10.00 | 05.25 | 09.00 | SJF |
| T-10 | 00.00 | 01.00 | 00.00 | 01.00 | 00.00 | 01.00 | 00.00 | 01.00 | 00.00 | 01.00 | 00.00 | 01.00 | 00.00 | 01.00 | PS,RR,SJF,TSS1,TSS2,TSS3 |

Figure 2: A consolidated Output Table of the Experiments conducted

## 4.2 Graphical Analysis

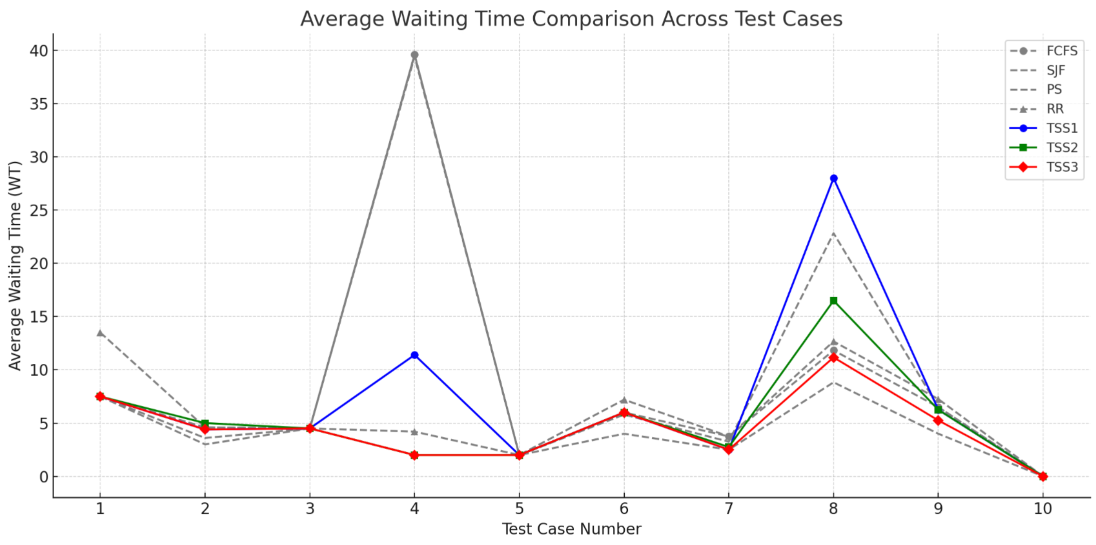


Figure 3: Average Waiting Time across all Algorithms and Test Cases

## 5 Results and Discussion

My experimental results across 10 test cases reveal the following insights:

- Classical algorithms like SJF performed well in predictable, short-burst scenarios.
- RR showed high waiting times in tightly clustered process arrival patterns.
- All TSS variants performed competitively or better in mixed and edge workloads.
- TSS2 and TSS3 had lower average WT and TAT in several high-load test cases.

The design of slabs influences the results significantly. For example, TSS3 with flatter slab structure performed well when long burst processes existed.

## 6 Conclusion

The Tax-Slab Scheduling algorithm introduces a dynamic and flexible way to prioritize processes based on burst-time slabs. Our study shows that it holds competitive ground against classical CPU schedulers and may outperform them in several contexts. Future enhancements could involve dynamic slab adjustment using real-time feedback or combining TSS with other adaptive schedulers.

## References

1. Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating System Concepts* (10th ed.). Wiley.

2. Agrawal, R., & Kaur, M. (2015). Comparative Analysis of CPU Scheduling Algorithms. *International Journal of Computer Applications*, 111(3).

3. Rajesh, D., & Lakshmi, M. (2017). Adaptive Scheduling Algorithm using Hybrid Approach. *Procedia Computer Science*, 115, 660–667.