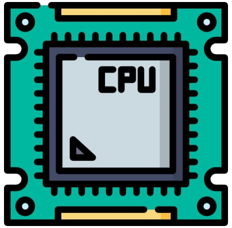
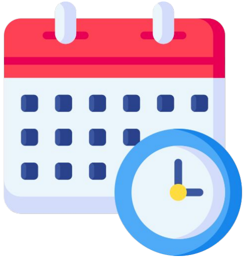
# CPU Scheduling: CFS and EEVDF

By: James Young

# CPU Scheduling

- CPU schedulers allocates CPU time to various processes and threads
- Main goals are to optimize system performance, ensure fairness, and minimize latency
- Two prominent Linux CPU scheduling algorithms include Completely Fair Scheduler (CFS) and Earliest Eligible Virtual Deadline First (EEVDF)



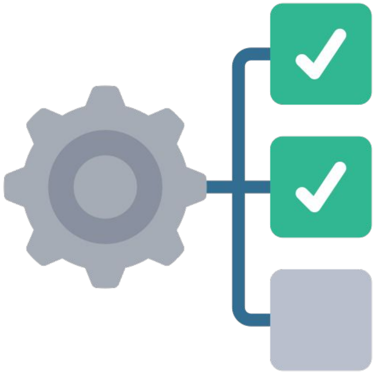



# CFS Overview

- Before CFS, Linux used CPU schedulers such as O(n) scheduler and O(1) scheduler

- Problem: Older CPU schedulers had problem of being either inefficient, having high complexity, or lacked fairness

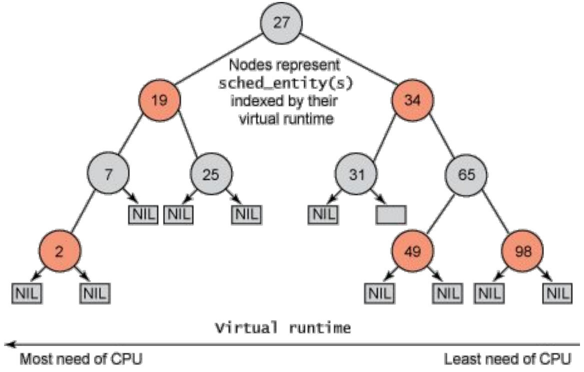
- Solution: CFS introduced into Linux in kernel 2.6, where it focuses on fairness whilst having efficient operations



# CFS Algorithm Implementation

- CFS algorithm uses **virtual runtime** (vruntime) and **red-black tree**
- For fairness, CFS computes vruntime for each process and chooses the smallest vruntime to run
- For efficiency, CFS stores the processes in a red-black tree, where each node is a process with a value vruntime

$$vruntime = (actual\ runtime) \cdot \frac{1024}{1 + nice\ value}$$




Example of a CFS Red-Black Tree

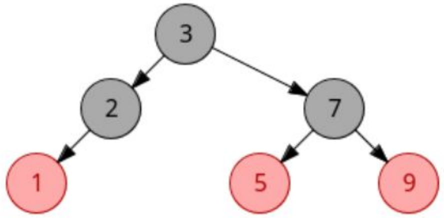
# CFS Red-Black Tree

- Red-Black Tree Properties:

    a. Root Property: The root is always black.
    b. Red Property: Red nodes cannot have red children, preventing consecutive red nodes.
    c. Black Depth Property: Every path from a node to its descendant leaves must have the same number of black nodes.
    d. Leaf Nodes: All null leaf nodes are black.

- Above properties keep tree balanced, meaning efficient operations ($O(\log n)$ time complexity)

- Red-Black tree structure also ensures that leftmost node will be smallest virtual runtime value, so CFS chooses leftmost node to run




Example of a Red-Black Tree

# CFS Example

Store 4 Processes (Figure 1):

- Process A: vruntime = 1
- Process B: vruntime = 3
- Process C: vruntime = 5
- Process D: vruntime = 7

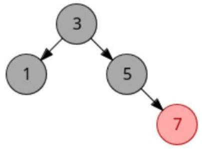


Figure 1

Insert Process E (Figure 2):

- Process E: vruntime = 2

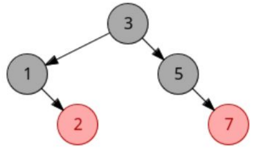


Figure 2

Run Process (Figure 3):

- CFS chooses lowest vruntime, so leftmost node (Process A)
- After deletion of node, tree rebalances

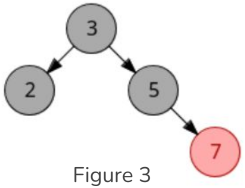


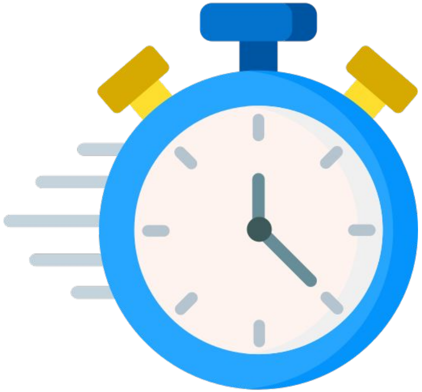
Figure 3

# EEVDF Overview

- CFS Problem: Does not explicitly consider latency requirements for processes

- Relies instead on nice values for priority, which does not adequately address varying latency needs

- Solution: EEVDF, which uses concepts of **lag** and **virtual deadline** to remain fair whilst minimizing latency, replaced CFS in kernel 6.6



# EEVDF Algorithm: Lag

- To ensure fairness, EEVDF calculates lag to determine a process eligibility

- Lag is basically the difference between the time that process should have gotten and how much it actually got

- If a process has positive or zero lag value, it is eligible to be run and negative lag value means it is not eligible

- Tasks with high lag are prioritized, aim to even out lag values across and avoid excessive delays

$$lag = Expected\ Execution\ Time - Actual\ Execution\ Time$$

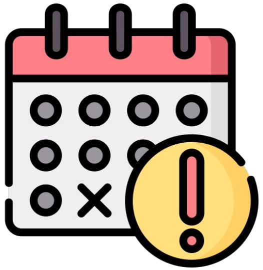
# EEVDF Algorithm: Virtual Deadline

- To minimize latency, EEVDF calculates virtual deadline, which takes into account a process’ urgency and expected execution time

- Ensures that processes with the highest need for CPU time are scheduled promptly, quicker access for latency-sensitive processes

- Done by choosing the smallest virtual deadline (Earliest Deadline) task among the Eligible Tasks

- Stores processes in Red-Black tree (similar to CFS), with virtual deadline being the node value



$$virtual\ deadline = eligible\ time + \left(\frac{time\ slice}{weight}\right)$$

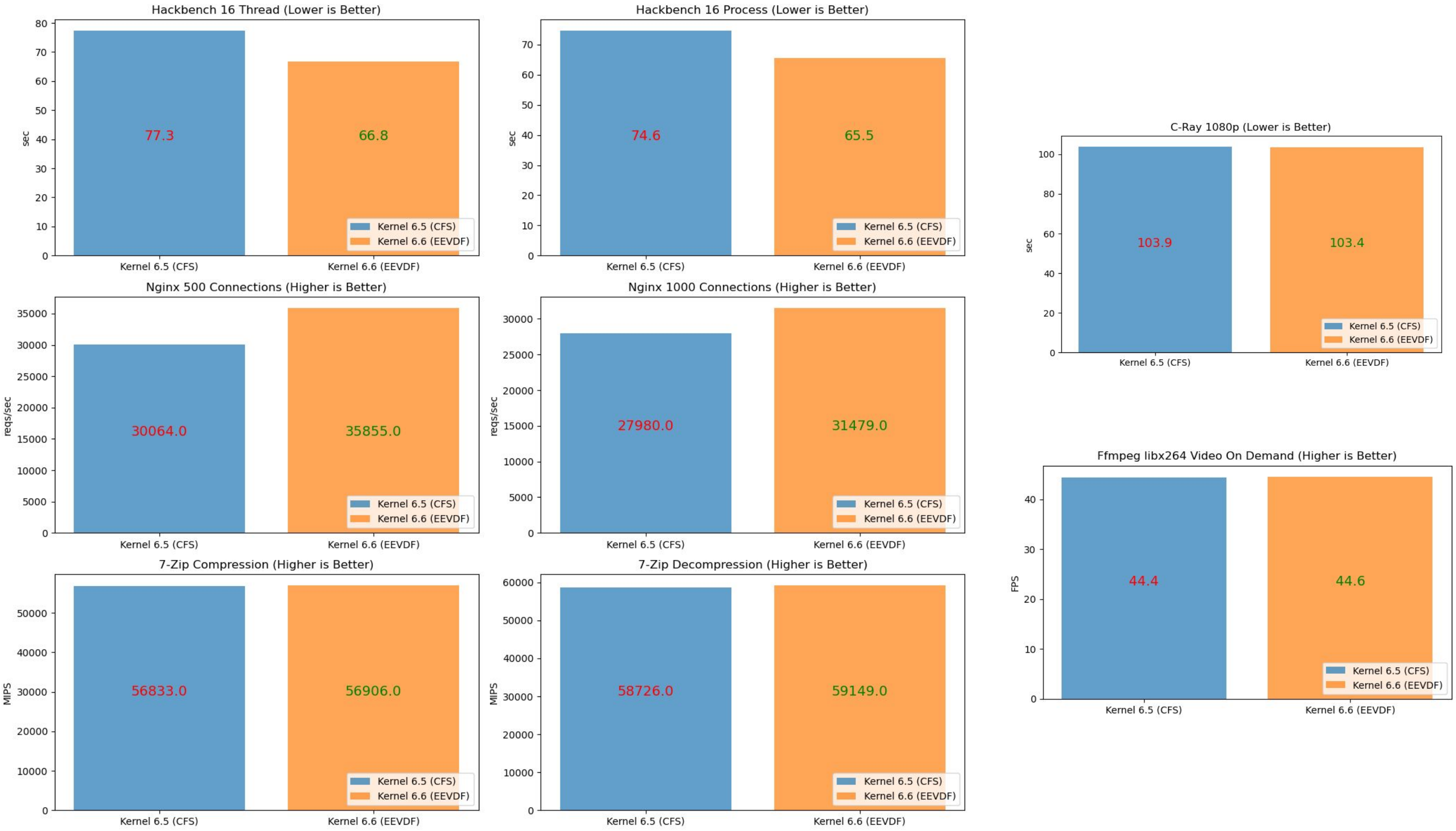
# CFS vs EEVDF Performance Benchmark

- Ran benchmarks to compare performance, used Phoronix Test Suite for running benchmarks

- Scheduler-based benchmark: Hackbench (benchmark tool for kernel schedule)

- Real-world Application benchmark: Nginx, 7-zip, C-Ray, FFmpeg

Phoronix Test Suite



Hackbench 16 Thread (Lower is Better)
sec
77.3
66.8
Hackbench 16 Process (Lower is Better)
sec
74.6
65.5
C-Ray 1080p (Lower is Better)
sec
103.9
103.4
Nginx 500 Connections (Higher is Better)
reqs/sec
30064.0
35855.0
Nginx 1000 Connections (Higher is Better)
reqs/sec
27980.0
31479.0
Ffmpeg libx264 Video On Demand (Higher is Better)
FPS
44.4
44.6
7-Zip Compression (Higher is Better)
MIPS
56833.0
56906.0
7-Zip Decompression (Higher is Better)
MIPS
58726.0
59149.0
Kernel 6.5 (CFS)
Kernel 6.6 (EEVDF)

# References


- N. Goel and R. B. Garg, A Comparative Study of CPU Scheduling Algorithms. 2013. [Online]. Available: https://arxiv.org/abs/1307.4165
- M. Jones, “Inside the Linux 2.6 Completely Fair Scheduler,” IBM developer, https://developer.ibm.com/tutorials/l-completely-fair-scheduler/ (accessed Nov. 5, 2024).
- P. Patil, S. Dhotre, and R. Jamale, “A Survey on Fairness and Performance Analysis of Completely Fair Scheduler in Linux Kernel,” International Journal of Control Theory and Applications, vol. 9, no. 44, 2016.
- J. Corbet, “An EEVDF CPU scheduler for linux,” LWN.net, https://lwn.net/Articles/925371/ (accessed Nov. 8, 2024).
- Stoica, I., & Abdel-WahabDepartment, H. (1995). Eligible Virtual Deadline First : A Flexibleand Accurate Mechanism for Proportional ShareResource Allocation. https://api.semanticscholar.org/CorpusID:7263527
- “CFS Scheduler,” The Linux Kernel Documentation, https://docs.kernel.org/scheduler/sched-design-CFS.html (accessed Nov. 8, 2024).
- “EEVDF Scheduler,” The Linux Kernel Documentation, https://docs.kernel.org/scheduler/sched-eevdf.html (accessed Nov. 8, 2024).