A Study on CFS and EEVDF CPU Scheduling

James Young

Abstract

This research paper presents a comprehensive analysis of two prominent CPU scheduling algorithms: the Completely Fair Scheduler (CFS) and the Enhanced Earliest Virtual Deadline First (EEVDF). The paper will provide an overview of both algorithms by looking at their implementations and their theoretical framework. Following this, I will analyze both algorithms by evaluating their performance and conduct a comparative analysis to determine the strengths and weaknesses of CFS and EEVDF.

Introduction

CPU scheduling algorithms play a critical role in modern operating systems by managing how processes gain access to the CPU. This management is vital for ensuring efficient execution and optimal resource utilization, which directly affects the overall performance of the system. By determining the order in which processes are executed, these algorithms have a significant impact on key performance metrics such as response time, throughput, and the fairness of resource allocation among competing processes [1]. Among the various scheduling strategies, two prominent algorithms are CFS and EEVDF, both of which are widely implemented in Linux systems. This paper aims to provide a comprehensive overview of the implementation details of these two algorithms and will conduct an analysis and comparison of both algorithms.

CFS Algorithm

CFS is built on the principle of fairness, where each runnable task should receive an equal share of the CPU over time. It seeks to minimize the wait time for each process by ensuring that no single process monopolizes the CPU. Whereas CPU scheduling algorithms before CFS typically used run queues to determine the task to execute next, CFS employs a red-black tree to manage runnable tasks [2]. A red-black tree is a self-balancing binary search tree that maintains sorted data. Each node has a color (either red or black) and follows these key properties [3]:

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

These properties keep the tree balanced, resulting in efficient O(log*n*) time complexity for operations [2]. It uses a virtual runtime (also known as vruntime) to determine balance, where a virtual runtime is a metric that tracks the amount of CPU time a process has received, adjusted for its priority. Figure 1 shows the formula used to compute virtual runtime in CFS, where the nice value is an integer that determines the priority and the actual runtime is the process has spent executing on the CPU [3]:

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

Figure 1: Virtual Runtime Calculation

In the CFS red-black tree, each node represents a process that is organized according to the value of the virtual runtime of each process [2]. Figure 1 shows an example of a CFS red-black tree with 4 processes stored.

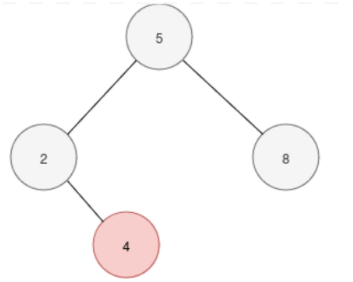

Figure 2: Example of CFS Red-Black Tree

The process with the smallest virtual runtime is located at the leftmost position in the tree, which allows CFS to quickly identify which process should be scheduled next. In the case of Figure 2, the node with virtual runtime 2 is the leftmost node and will be executed. Figure 3 shows the re-balanced red-black tree after node 2 is removed and Figure 4 shows an example of inserting a node process with virtual runtime 3.

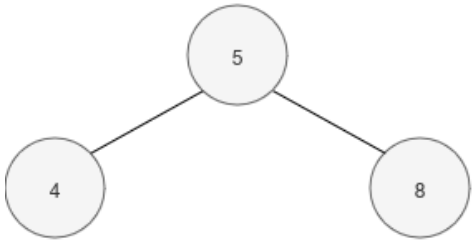

Figure 3: Example of CFS Red-Black Tree
After Removal of Node 2

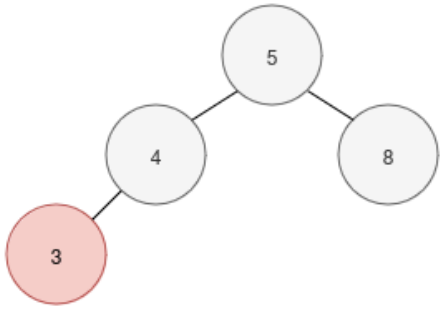

Figure 4: Example of CFS Red-Black Tree
After Insertion of Node 3

By choosing the leftmost node (smallest virtual runtime), CFS ensures that all processes receive CPU time in proportion to their needs, preventing any single process from monopolizing the CPU to ensure fairness whilst doing it with efficient time complexity.

EEVDF Algorithm

EEVDF, like CFS, aims to provide fairness to processes. It does so by calculating lag, where lag is the difference between a process' virtual runtime and its actual running time [4]. This means that when lag is positive, the CPU owes a process time and when lag is negative, it has over-provisioned time for the task. To ensure fairness, a process is only eligible if the lag is zero or positive [5]. Figure 5 shows a high-level formula for calculating lag.

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

Figure 5: EEVDF Lag Calculation

EEVDF also aims to reduce lag and latency, where it uses the concept of virtual deadline to do so. Virtual deadlines are computed by taking the sum of eligible time and time splice scaled by the weight of a task [5]. Figure 6 shows a simplified equation to demonstrate virtual deadline calculation. In the equation, time splice refers to the amount of CPU time allocated to a task, eligible time refers to the amount of time it has been ready for execution, and weight is the nice value [4].

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

Figure 6: Virtual Runtime Calculation

With this formulation, processes with shorter time slices will have smaller virtual deadlines and will hence be executed first, so long as it is eligible (lag is positive or zero). Since tasks with shorter time slices tend to be latency sensitive, therefore executing them first helps reduce lag and latency [4]. Therefore, this combination of virtual deadline and lag calculation ensures that EEVDF is fair whilst being responsive.

EEVDF and CFS Comparison and Performance

CFS was first introduced into the Linux kernel in version 2.6 due to its fairness and efficiency on multi-tasking CPU [6]. However, one key issue with CFS is handling latency. CFS does not include an explicit way for processes to express their latency requirements [4]. To improve latency, a method was needed to ensure that certain processes can access the CPU quickly without compromising fairness. Therefore, due to EEVDF's implementation in reducing latency, it has eventually replaced CFS in kernel 6.6 [7].

To analyze the performance of both CPU schedulers, I ran benchmarks using Phoronix Test Suite tool [8]. The benchmarks used were Hackbench, Nginx, and 7zip. Hackbench is a popular tool used for testing kernel schedulers [9], whilst the other benchmarks were included for testing on a real-world application. Figure 7 shows the system information used to run the tests, where I compiled and used kernel 6.5 for CFS and kernel 6.6 for EEVDF. Figure 8 shows the testing results.

Phoronix Test Suite 10.8.4	Phoronix Test Suite 10.8.4
AMD Ryzen 7 4800H @ 2.90GHz (8 Cores / 16 Threads)	Processor
LENOVO LNVNB161216 (FSCN18WW BIOS)	Motherboard
AMD Renoir/Cezanne	Chipset
16GB	Memory
1024GB SK hynix HFS001TD9TNI-L2A0B	Disk
NVIDIA GeForce RTX 2060 Mobile	Graphics
NVIDIA TU106 HD Audio	Audio
Realtek RTL8111/8168/8211/8411 + Intel Wi-Fi 6 AX200	Network
Arch Linux	OS
6.5.0CFS (x86_64) 6.6.0EEVDF (x86_64)	Kernels
GCC 14.2.1 20240910	Compiler
ext4	File-System

Figure 7: System Used for Benchmarks

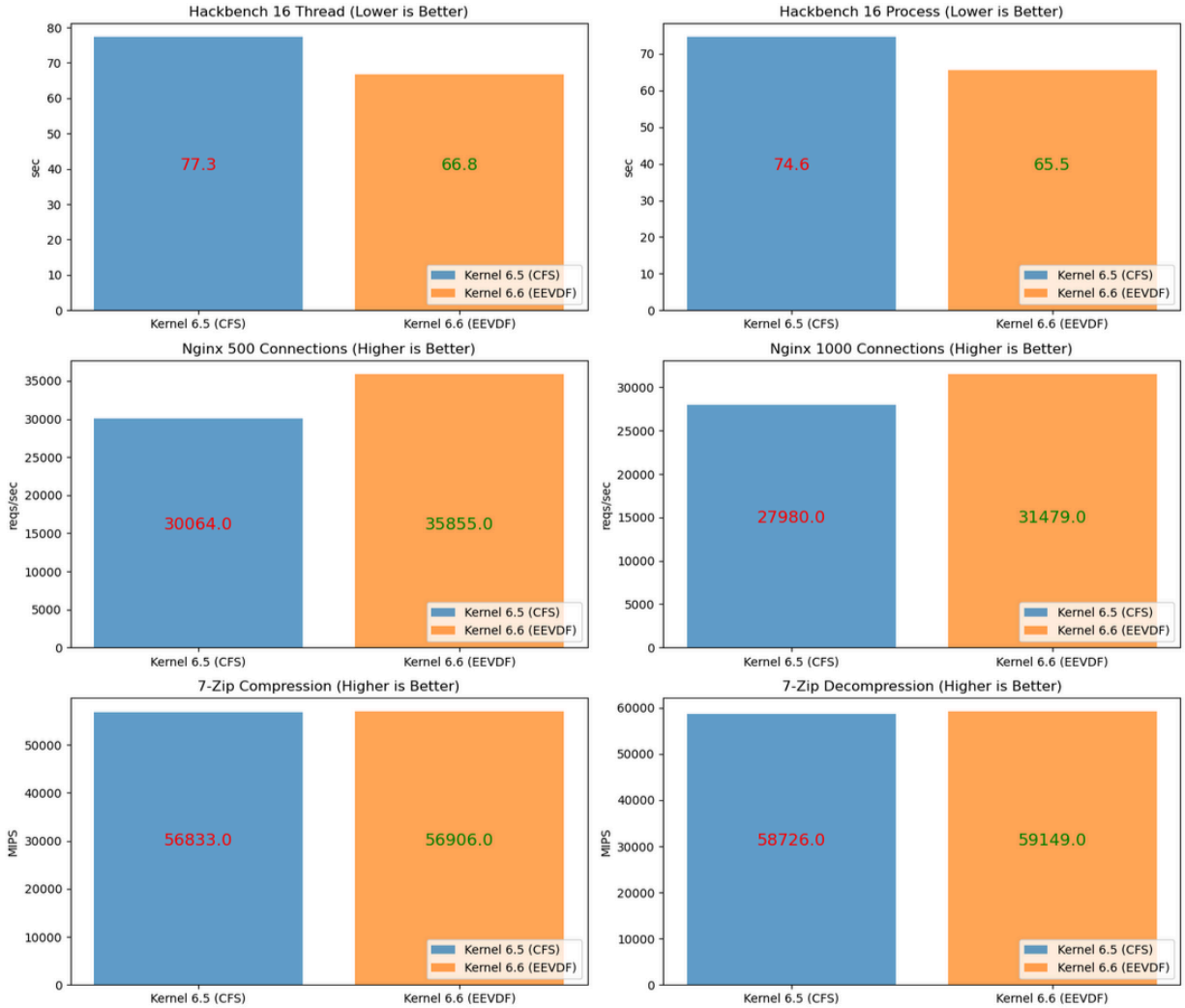

Figure 8: Benchmark Results

As the results indicate, EEVDF's algorithm can provide real-world advantages over CFS. It is worth noting that the performance differences can also be due to other kernel patches from 6.5 to 6.6. However, it is more likely that the performance gain is from the change in CPU scheduler as this was the major feature and change of the kernel patch 6.6 [10].

The improvements of EEVDF may be more noticeable on higher core count processors. For example, the lead developer of Phoronix Test Suite found larger improvements in kernel 6.6 when compared to 6.5 tested on AMD EPYC 9754, which he attributed the performance improvements to EEVDF [11]. Figure 9 shows an example of one of his tests with TiDB.

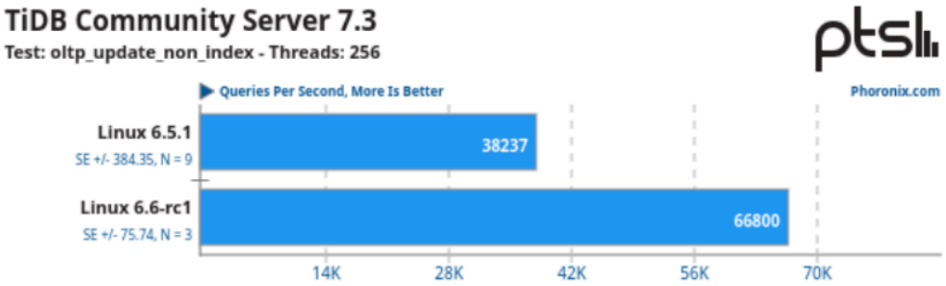

Figure 9: TiDB Benchmark with AMD EPYC 9754

Conclusion

In conclusion, CFS and EEVDF are both prominent CPU scheduling algorithms that have been utilized in modern Linux environments. CFS uses the concept of virtual runtime and red-black tree to provide fairness to scheduling, whilst EEVDF uses lag and virtual deadlines to provide fairness whilst minimizing latency. Due to EEVDF's better ability to meet latency requirements it has since replaced CFS as the CPU scheduler in the Linux kernel starting from version 6.6.

References

[1] N. Goel and R. B. Garg, A Comparative Study of CPU Scheduling Algorithms. 2013. [Online]. Available: https://arxiv.org/abs/1307.4165

[2] 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).

[3] 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.

[4] J. Corbet, "An EEVDF CPU scheduler for linux," LWN.net, https://lwn.net/Articles/925371/ (accessed Nov. 8, 2024).

[5] Stoica, I., & Abdel-WahabDepartment, H. (1995). Eligible Virtual Deadline First : A Flexibleand Accurate Mechanism for Proportional ShareResource Allocation. https://api.semanticscholar.org/CorpusID:7263527

[6] "CFS Scheduler," The Linux Kernel Documentation, https://docs.kernel.org/scheduler/sched-design-CFS.html (accessed Nov. 8, 2024).

[7] "EEVDF Scheduler," The Linux Kernel Documentation, https://docs.kernel.org/scheduler/sched-eevdf.html (accessed Nov. 8, 2024).

[8] Phoronix Media. Phoronix test suite – Linux testing & benchmarking platform, automated testing, open-source benchmarking. http://www.phoronix-test-suite.com/.

[9] Gouicem, R., Carver, D., Lozi, J.-P., Sopena, J., Lepers, B., Zwaenepoel, W., Palix, N., Lawall, J., & Muller, G. (2020). Fewer Cores, More Hertz: Leveraging High-Frequency Cores in the OS Scheduler for Improved Application Performance. 2020 USENIX Annual Technical Conference (USENIX ATC 20), 435–448. https://www.usenix.org/conference/atc20/presentation/gouicern

[10] J. Corbet, "The 6.6 kernel has been released," LWN.net, https://lwn.net/Articles/925371/ (accessed Nov. 8, 2024).

[11] M. Larabel, "Linux 6.6 delivers some impressive gains for AMD EPYC 9754 'Bergamo' server performance," Phoronix, https://www.phoronix.com/review/linux-66-bergamo/4 (accessed Nov. 11, 2024).