

Completely Fair Scheduler (CFS)

Pedroso

26 de agosto de 2025

Completely Fair Scheduler (CFS)

- ▶ Introduzido no kernel Linux 2.6.23 como o escalonador padrão.
- ▶ O CFS é um escalonador preemptivo de tempo compartilhado.
- ▶ Objetivo principal é fornecer justiça no compartilhamento da CPU entre as tarefas.
- ▶ O CFS é ideal para sistemas de uso geral (servidores, desktops, ambientes multiusuário), mas não serve para tempo real rígido porque não garante deadlines nem baixa latência determinística.
- ▶ O kernel Linux oferece outros escalonadores para lidar com tempo real¹:
 - ▶ SCHED_FIFO: fila de prioridade fixa, sem quantum (executa até bloquear ou terminar).
 - ▶ SCHED_RR: round-robin com prioridades fixas, maior prioridade preempta os de menor prioridade.
 - ▶ SCHED_DEADLINE: baseado em Earliest Deadline First (EDF), projetado especificamente para aplicações de tempo real.

¹Ver <http://www.eletrica.ufpr.br/pedroso/2025/TE355/ls-sched.sh>

Tempo de Execução Virtual (vruntime)

- ▶ O CFS não usa *quantum* de tempo fixos. Ele usa o **vruntime** para rastrear o tempo de CPU de cada tarefa.
- ▶ É a métrica central do escalonador. Uma tarefa com um **vruntime** menor está *atrasada* e tem maior prioridade para ser executada.
- ▶ O CFS sempre escolhe a tarefa com o **menor vruntime** para executar.
- ▶ O **vruntime** de uma tarefa aumenta enquanto ela está efetivamente em execução na CPU.

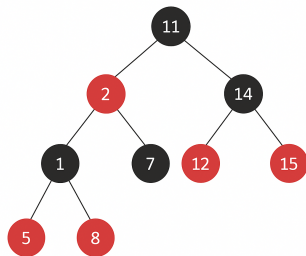
Sumário: Como o CFS Funciona

- 1 O CFS, diferentemente de algoritmos anteriores, **quantum** de tempo fixo.
- 2 Todas as tarefas executáveis são mantidas em uma **Árvore Rubro-Negra** (rbtree) ordenada por tempo.
- 3 Todas as tarefas executáveis na rbtree são classificadas pela sua chave **virtual runtime (vruntime)**.
- 4 A tarefa com o **menor vruntime** é a próxima a ser escalonada pela CPU.
- 5 O quantum de tempo de cada processo é calculado dinamicamente, levando em conta seu **peso** (derivado de sua prioridade base) e a fração da **latência alvo** (target_latency).

Estrutura de Dados: A Fila de Execução

- ▶ O CFS não utiliza filas de execução tradicionais.
- ▶ Todas as tarefas executáveis são organizadas em uma **Árvore Rubro-Negra** (Red-Black Tree).
- ▶ A árvore é ordenada pelo **vruntime**. O nó mais à esquerda da árvore sempre aponta para a tarefa com o menor **vruntime**.
- ▶ Vantagens:
 - ▶ Busca rápida ($O(\log n)$) pela próxima tarefa a ser executada.
 - ▶ Inserção e remoção eficientes.
 - ▶ Garante que o tempo para escalonamento da próxima tarefa seja muito rápido.

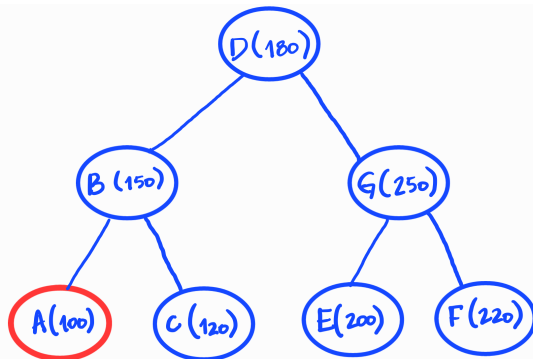
A Árvore Rubro-Negra



- ▶ Elementos com peso maior à direita e peso menor à esquerda.
- ▶ Todo nó na árvore recebe uma cor: vermelho ou preto.
- ▶ A raiz deve ser sempre preto.
- ▶ Nós vermelhos não podem ter filhos vermelhos.
- ▶ Todas as folhas nulas são pretas.
- ▶ As red-black trees usam essas regras de coloração e rebalanceamento (através de operações de rotação e recoloração) para garantir que a árvore nunca fique muito desequilibrada.

Exemplo: A Árvore Rubro-Negra

Árvore de Tarefas Executáveis



Análise do Exemplo:

- ▶ A árvore contém 7 tarefas.
- ▶ A tarefa com o menor **vruntime** (100) é a **Tarefa A**.
- ▶ A Tarefa A é o nó mais à esquerda e será a próxima a ser escalonada.
- ▶ Após a execução da Tarefa A, seu **vruntime** será atualizado, e a árvore será re-balanceada para manter a ordem.

- ▶ **Prioridade Base (nice):** No Linux, as prioridades dos processos são representadas por valores **nice**, que variam de -20 (maior prioridade) a +20 (menor prioridade).
- ▶ O **nice** não determina diretamente a prioridade de execução, mas sim o *peso* da tarefa.
- ▶ O CFS não utiliza prioridades dinâmicas.
- ▶ **Valores nice e o vruntime:**
 - ▶ **nice < 0 (negativo):** Tarefas com **nice** negativo têm o **vruntime** que cresce mais lentamente, fazendo com que sejam escalonadas com mais frequência.
 - ▶ **nice > 0 (positivo):** Tarefas com **nice** positivo têm o **vruntime** que cresce mais rapidamente, diminuindo sua chance de serem escalonadas.

Cálculo do Virtual Runtime (vruntime)

- ▶ O **vruntime** é a métrica central do CFS. Ele é o tempo de CPU real que um processo utilizou, ajustado pela sua prioridade base.
- ▶ O **vruntime** é medido em *nanosegundos*.
- ▶ O **peso** do processo é derivado do seu valor de prioridade base (*nice*).
- ▶ Quando um processo executa na CPU por um tempo Δt , seu **vruntime** é atualizado:

$$\text{vruntime}_{\text{novo}} = \text{vruntime}_{\text{atual}} + \Delta t \times \left(\frac{\text{Peso Padrão}}{\text{Peso do Processo}} \right)$$

- ▶ **Peso Padrão:** É o peso de uma tarefa com *nice*=0.
- ▶ **Peso do Processo:** Corresponde ao peso da tarefa, conforme seu valor *nice*.

Entendendo a Fórmula

- ▶ Tarefas com **nice=0** (peso padrão) têm seu **vruntime** crescendo na mesma velocidade que o tempo real.
- ▶ Tarefas com **nice < 0** (alta prioridade) têm um peso maior. O denominador da fórmula aumenta, e o **vruntime** cresce mais devagar.

Cálculo do Virtual Runtime (vruntime): Exemplo

- ▶ A ideia do CFS é que cada incremento de 1 no valor de nice deve reduzir a fatia de CPU da tarefa em aproximadamente 10%.
- ▶ O nice=0 é considerado o *padrão* e recebe o valor 1024. ($2^{10} = 1024$, o que facilita operações inteiras, evitando operações de ponto flutuante).
- ▶ A fórmula usada para conversão de prioridade base e peso é:

$$\text{peso}(\text{nice}) = \text{peso}(0) \times (1.25)^{\text{nice}}$$

- ▶ Suponha dois processos:
 - ▶ P1 (nice=0, peso=1024)
 - ▶ P2 (nice=5, peso $\approx$ 335)
- ▶ Ambos executam 10ms de CPU: $\Delta_{\text{exec}} = 10\text{ms}$.
 - ▶ P1: $\text{vruntime} = \text{vruntime} + 10\text{ms} * (1024/1024) = 10\text{ms}$
 - ▶ P2: $\text{vruntime} = \text{vruntime} + 10\text{ms} * (1024/335) \approx 30.6\text{ms}$
- ▶ Finalmente, para evitar o crescimento indefinido de vruntime, todos os vruntime são normalizados considerando o menor valor registrado.

Quantum de Tempo no CFS

- ▶ **Abordagem Diferente:** Diferentemente de escalonadores anteriores, o CFS não usa **quantum** de tempo fixo.
- ▶ O CFS calcula o quantum de tempo de cada processo de forma **dinâmica**, garantindo uma distribuição justa do tempo de CPU.
- ▶ A meta do escalonador é dar a cada processo um **quantum** justo, considerando seu peso e a **latência alvo** do sistema.
- ▶ A **latência alvo** (`target_latency`) é o tempo mínimo para que todas as tarefas executáveis recebam ao menos uma chance de rodar na CPU. O valor padrão é 20ms.

Cálculo do Quantum

- ▶ Cada processo recebe um **quantum** tempo equivalente a $1/N$ da **latência alvo**, onde N é o número de processos executáveis.
- ▶ Este quantum é **ajustado com base na prioridade base** do processo.
- ▶ **O efeito da prioridade base:**
 - ▶ Um valor de alta prioridade (por exemplo, '-20') faz com que o processo receba um quantum **relativamente maior**.
 - ▶ Um valor de baixa prioridade (por exemplo, '+19') faz com que o processo receba um quantum **relativamente menor**.
- ▶ O quantum é calculado da seguinte forma:

$$\text{quantum}_i = \text{target_latency} \times \frac{\text{peso}_i}{\sum_{j=1}^N \text{peso}_j}$$

- ▶ Assim, processos de maior prioridade (maior peso) ganham fatias maiores, mas todos têm direito a rodar dentro do `target_latency`.

Exemplo de Cálculo do Quantum no CFS

- ▶ Configuração:

- ▶ $\text{target_latency} = 12 \text{ ms}$
- ▶ 3 tarefas com pesos:
 - ▶ $P_1 (\text{nice} = 0 \rightarrow \text{peso} = 1024)$
 - ▶ $P_2 (\text{nice} = 5 \rightarrow \text{peso} = 335)$
 - ▶ $P_3 (\text{nice} = 0 \rightarrow \text{peso} = 1024)$

- ▶ soma de pesos = $1024 + 335 + 1024 = 2383$

- ▶ **Quantum:**

- ▶ $P_1 : 12 \times \frac{1024}{2383} \approx 5.15 \text{ ms}$
- ▶ $P_2 : 12 \times \frac{335}{2383} \approx 1.69 \text{ ms}$
- ▶ $P_3 : 12 \times \frac{1024}{2383} \approx 5.15 \text{ ms}$

Resumo e Vantagens do CFS

- ▶ **Simplicidade:** A filosofia de *justiça* é mais simples para implementar do que os escalonadores anteriores.
- ▶ **Escalabilidade:** A Árvore Rubro-Negra garante bom desempenho mesmo com um grande número de processos.
- ▶ **Fairness:** O runtime garante que nenhuma tarefa seja negligenciada.
- ▶ **Flexibilidade:** O uso dos valores **nice** permite que o usuário influencie o comportamento de escalonamento de forma intuitiva.

Round Robin vs. Completely Fair Scheduler (CFS)

Round Robin com Prioridades

- ▶ Usa **quantum** de tempo fixo.
- ▶ Processos de mesma prioridade se revezam em ordem cíclica (Round Robin).
- ▶ A **prioridade dinâmica** determina qual processo será escalonado (múltiplas filas).
- ▶ Incremento da prioridade dinâmica para processos *iobound*.
- ▶ Decremento da prioridade dinâmica para processos *cpubound*.
- ▶ **Prioridade base** representa maior valor de prioridade dinâmica que pode ser atribuída a um processo.
- ▶ Pode ser ineficiente em termos de uso de CPU devido a preempções frequentes.
- ▶ Usado no Windows, macOS, AIX.

Completely Fair Scheduler (CFS)

- ▶ Não usa **quantum** de tempo fixo.
- ▶ Baseado no **tempo de execução virtual (vruntime)**.
- ▶ As prioridades base (valores *nice*) influenciam o **crescimento do vruntime**.
- ▶ Usa uma **Árvore Rubro-Negra** para organizar os processos, sempre escolhendo o que está mais *atrasado*.
- ▶ Escalável e *justo* em sistemas com grande número de tarefas, de acordo com a prioridade.
- ▶ Não é necessário identificar processos *iobound* e *cpubound*.
- ▶ Usado no Linux. É importante reforçar que o Linux tem ocupado o espaço de sistemas como AIX, HPUX, Irix, Solaris, Ultrix, entre outros, no ambiente corporativo como servidor.

- ▶ IBM Developer. "Inside the Linux 2.6 Completely Fair Scheduler". Disponível em: <https://developer.ibm.com/tutorials/1-completely-fair-scheduler/>
- ▶ The Linux Kernel. "Design of the CFS". Disponível em: <https://docs.kernel.org/scheduler/sched-design-CFS.html>
- ▶ Bajwa, N. "Process Scheduling: Completely Fair Scheduler (CFS) — Part I". Disponível em: https://medium.com/@noman_bajwa/process-scheduling-completely-fair-scheduler-cfs-part-i-f
- ▶ Jean-Pierre Lozi, Baptiste Lepers, Justin Funston, Fabien Gaud, Vivien Quéma P, *The Linux Scheduler: a Decade of Wasted Cores*. **EuroSys '16, 11th European Conference on Computer Systems (2016)**