Tópicos em Sistemas Operacionais

Escalonadores para o Linux

Islene Calciolari Garcia

Instituto de Computação - Unicamp

Segundo Semestre de 2013

Sumário

Políticas de escalonamento

Devem lidar com questões conflitantes:

- Bom tempo de resposta (baixa latência)
- Máxima utilização do sistema (alta vazão)

Qual é a fatia de tempo ideal?

- grandes: piora a interatividade
- pequenas: muitas trocas de contexto
- processos I/O-bound ou CPU-bound?

Prioridades

Normalmente, espera-se que

- Processos de mais alta prioridade rodem antes dos de mais baixa prioridade
- Processos de mesma prioridade rodem em política *roundin-robin*
- Processos de mais alta prioridade recebam fatias de tempo maiores

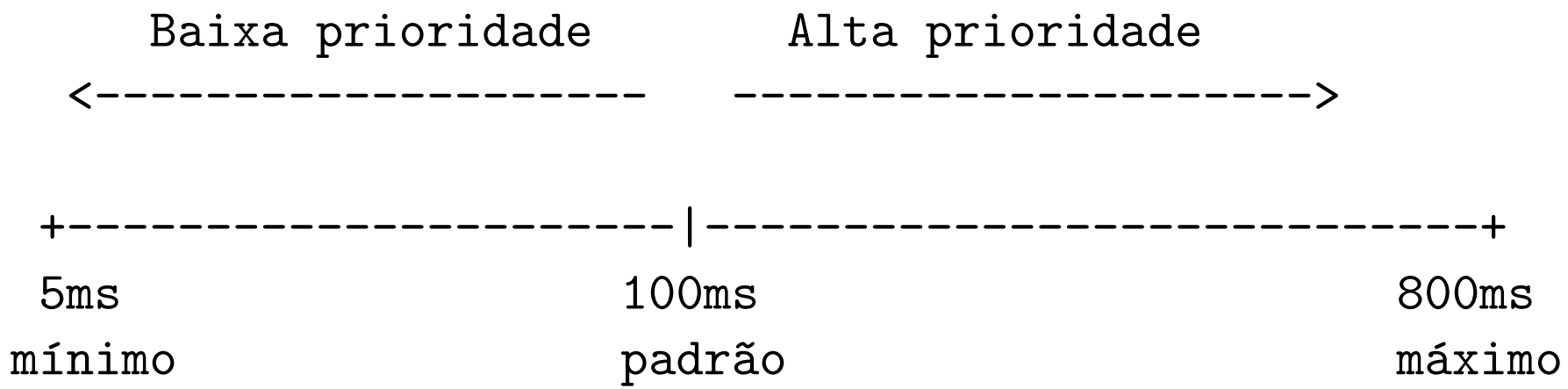
Linux:

- Favorece processos I/O-bound
- Não abandona processos CPU-bound

Linux implementa dois tipos de prioridade

- Estática (nice)
  - -20..19
  - $>$ valor, $<$ prioridade
  - $<$ valor, $>$ prioridade, $+$ CPU
  - `ps -el`
- Dinâmica
  - 0..99
  - $>$ valor, $>$ prioridade
  - favorece threads interativas

Fatias de tempo no linux



- Nice: 19→ 20 ⇒ timeslice: 10ms → 5ms
- Nice: 0→ 1 ⇒ timeslice: 10ms → 5ms

Até o kernel 2.4

- $O(n)$, não escalava bem
- apenas uma run-queue que era percorrida considerando-se prioridades
- suporte ruim para vários processadores
- enquanto uma CPU escolhia o processo as outras precisavam esperar
- cold cache quando um processo era escalonado para outro processador

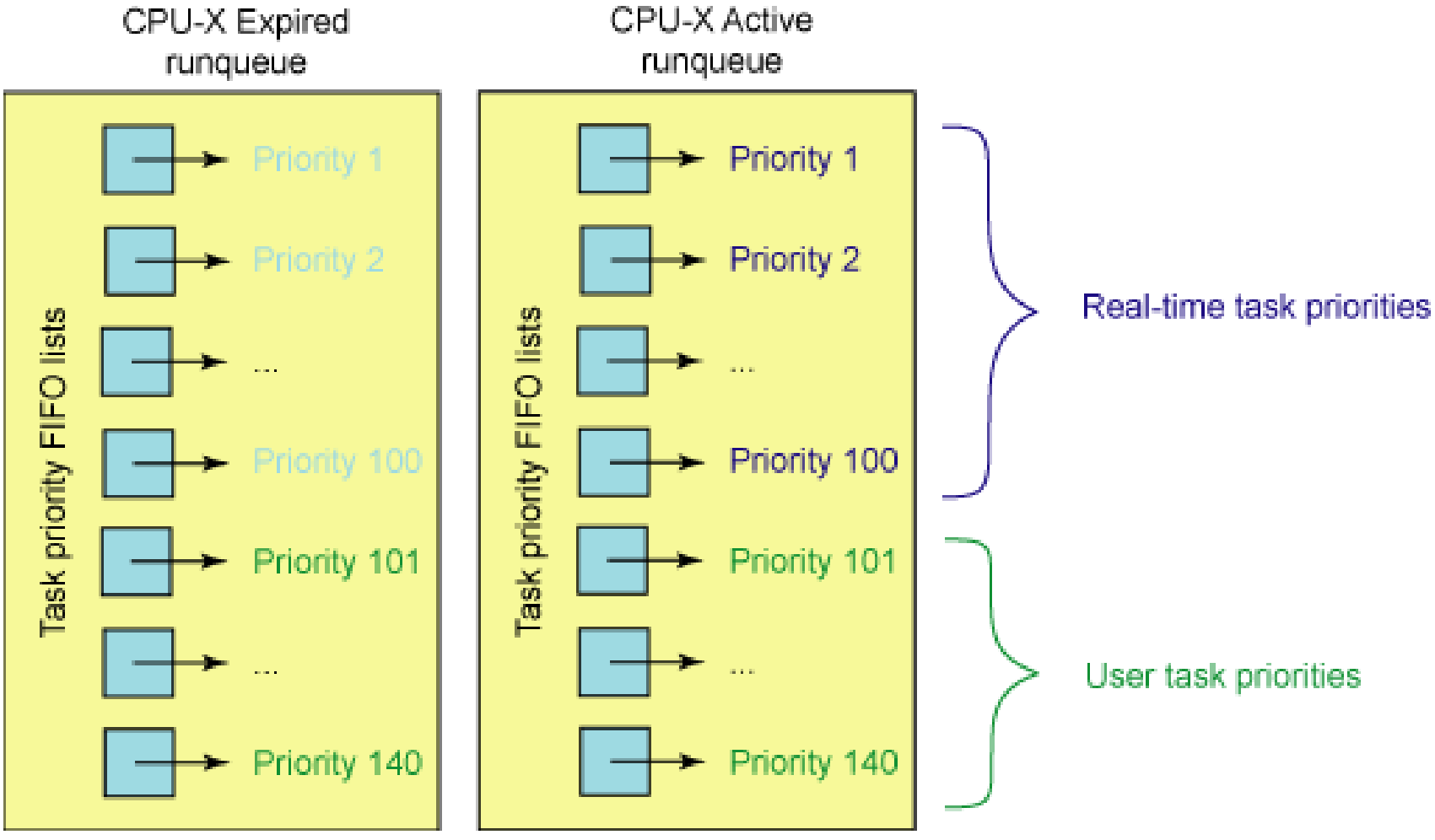
Kernel 2.5 até 2.6.22

- $O(1)$
- 140 listas de prioridades
- uma run-queue por processador
- processos migram apenas para balanceamento das run-queues
- bom desempenho para servidores, mas não tão bom para desktops
- prioridade interativa difícil de ser compreendida
- by Ingo Molnar, mantenedor

Como o $O(1)$ é garantido?

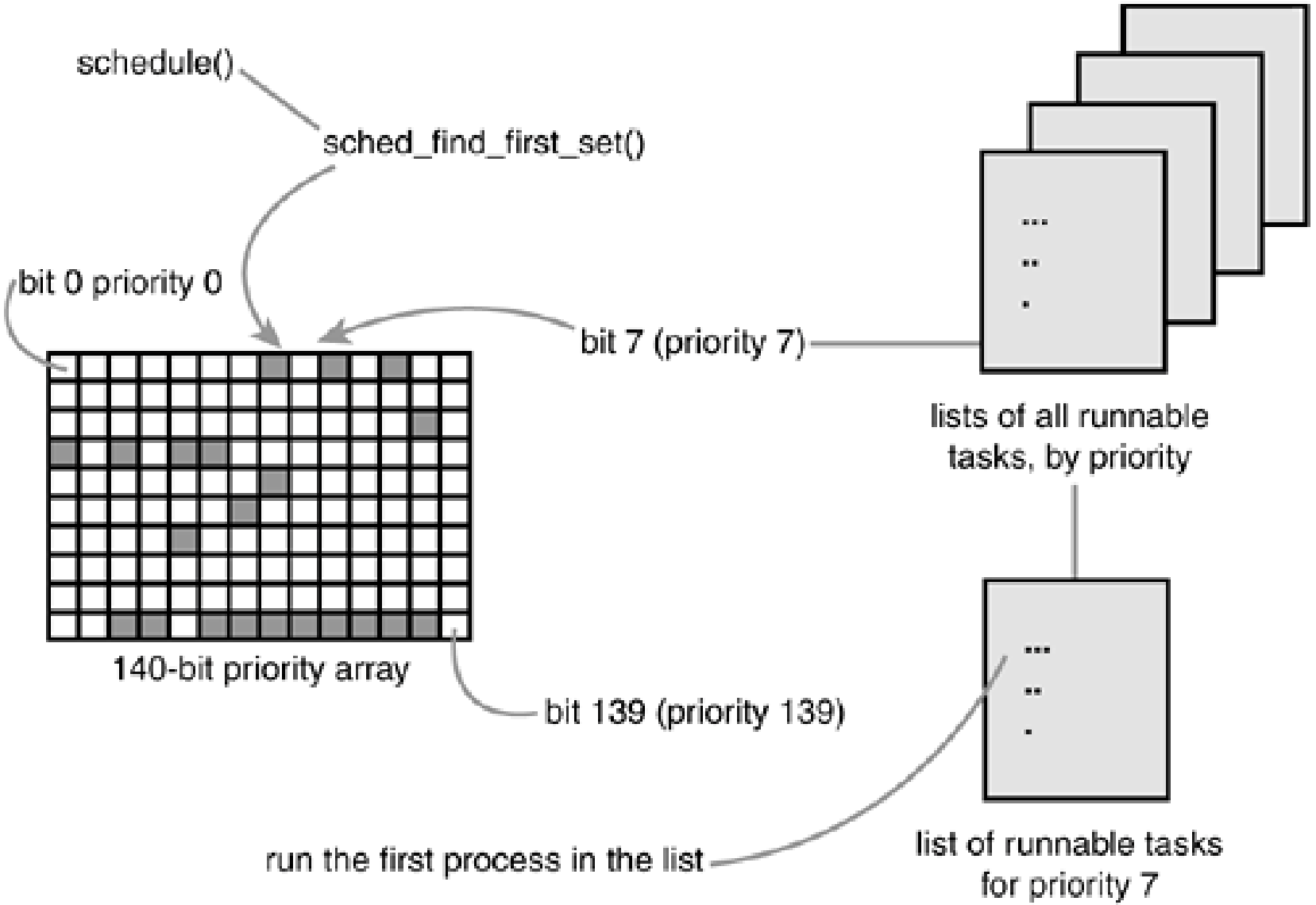
- Número de prioridades é estático = 140
- Bit map: 5 palavras de 32 bits = 160 bits
- busca pelo primeiro bit 1, que indicará a posição na tabela em que temos o processo com maior prioridade
- temos duas tabelas: uma de processos ativos e outra de processos que já consumiram sua fatia de tempo

Tabelas de prioridade



Inside the Linux Scheduler

Tabelas de prioridade



2004: Rotating Staircase Scheduler

- by Con Kolivas, médico anestesista
- Código enxuto (200 versus 498 linhas de código)
- Ideias mais claras, mais fácil de se entender
- Out-of-tree
- Contribuição não foi aceita

Interactivity, what is it?

Con Kolivas, Mon Jul 11 17:29:21 2005

There has been a lot of talk about what makes up a nice feeling desktop under linux. It comes down to two different but intimately related parameters which are not well defined. We often use the terms responsiveness and interactivity in the same sentence, but I'd like to separate the two. As there is no formal definition I prefer to define them as such:

Responsiveness: *The rate at which your workloads can proceed under different load conditions.*

Interactivity: *The scheduling latency and jitter present in tasks where the user would notice a palpable deterioration under different load conditions.*

Responsiveness would allow you to continue using your machine without too much interruption to your work, whereas interactivity would allow you to play audio or video without any dropouts, or drag a gui window across the screen and have it render smoothly across the screen without jerks.

2007: Rotating Staircase Deadline Scheduler (RSDL)

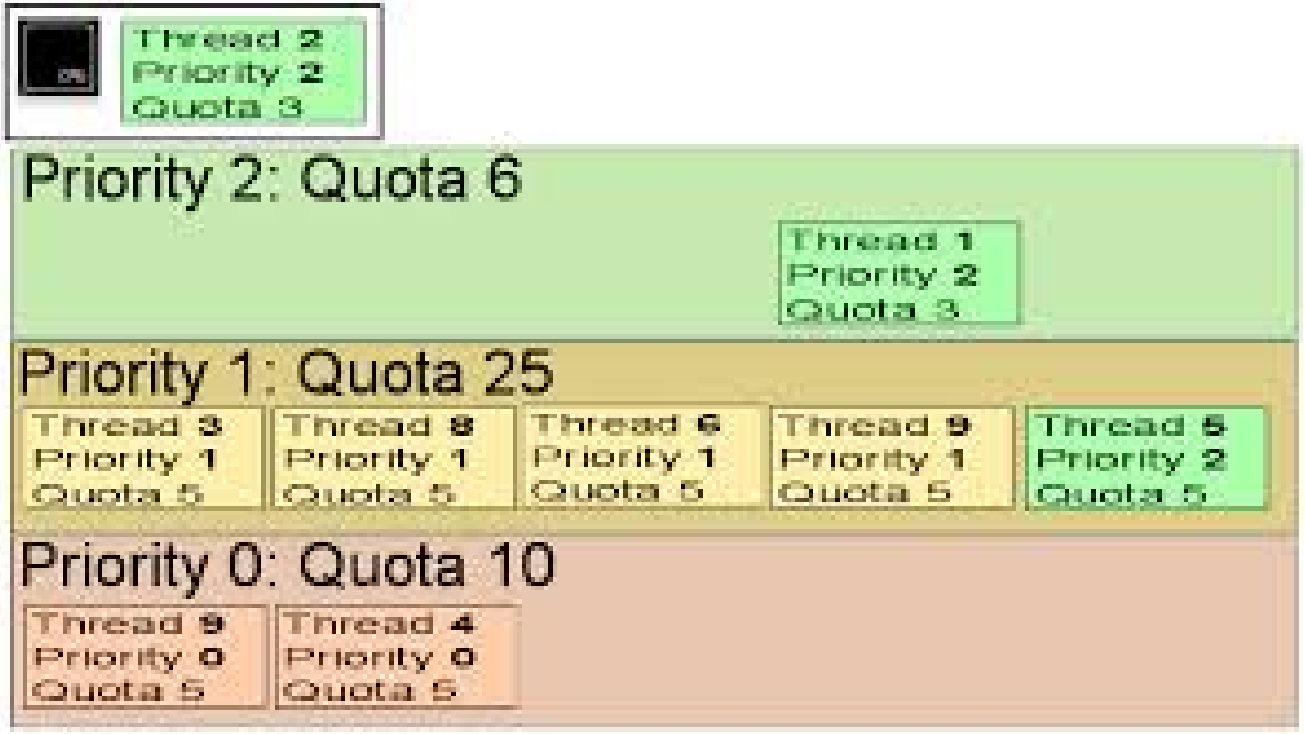
Con Kolivas

A starvation free, strict fairness O(1) scalable design with interactivity as good as the above restrictions can provide. There is no interactivity estimator, no sleep/run measurements and only simple fixed accounting. The design has strict enough a design and accounting that task behaviour can be modelled and maximum scheduling latencies can be predicted by the virtual deadline mechanism that manages runqueues. The prime concern in this design is to maintain fairness at all costs determined by nice level, yet to maintain as good interactivity as can be allowed within the constraints of strict fairness.

2007: Rotating Staircase Deadline Scheduler (RSDL)

- Lista de prioridades ativas
- Processos têm uma cota para rodar em uma determinada prioridade
- Quando esta cota termina, devem passar para um nível com prioridade mais baixa
- A fila de prioridade também tem cota
- Quando a cota termina, todos os processos são rebaixados
- Limite de tempo para um processo de baixa prioridade rodar

Tabelas de prioridade



www.cse421.net

Out-of-tree and maintainership

Do NOT fall into the trap of adding more and more stuff to an out-of-tree project. It just makes it harder and harder to get it merged. There are many examples of this.– Andrew Morton

The fact is, maintainership does _not_ mean ownership. It means that you should be _responsible_ for the code, and you get credit for it, but if problems happen you do NOT "own" it. Not at all. –Linus Torvalds

Histórico

http://www.linux-kongress.org/2010/slides/KN.lk2010_jon_corbet_fail.pdf

- 2007-03-04: First post
- 2007-03-05: Linus amenable to merging
- 2007-03-19: Linus gets irritated
- 2007-04-13: Molnar posts CFS
- 2007-07-10: CFS merged for 2.6.23
- 2007-07-25 Con leaves the kernel community

So, I've had enough. I'm out of here forever. I want to leave before I get so disgruntled that I end up using windows. – Con Kolivas

Completely Fair Scheduler

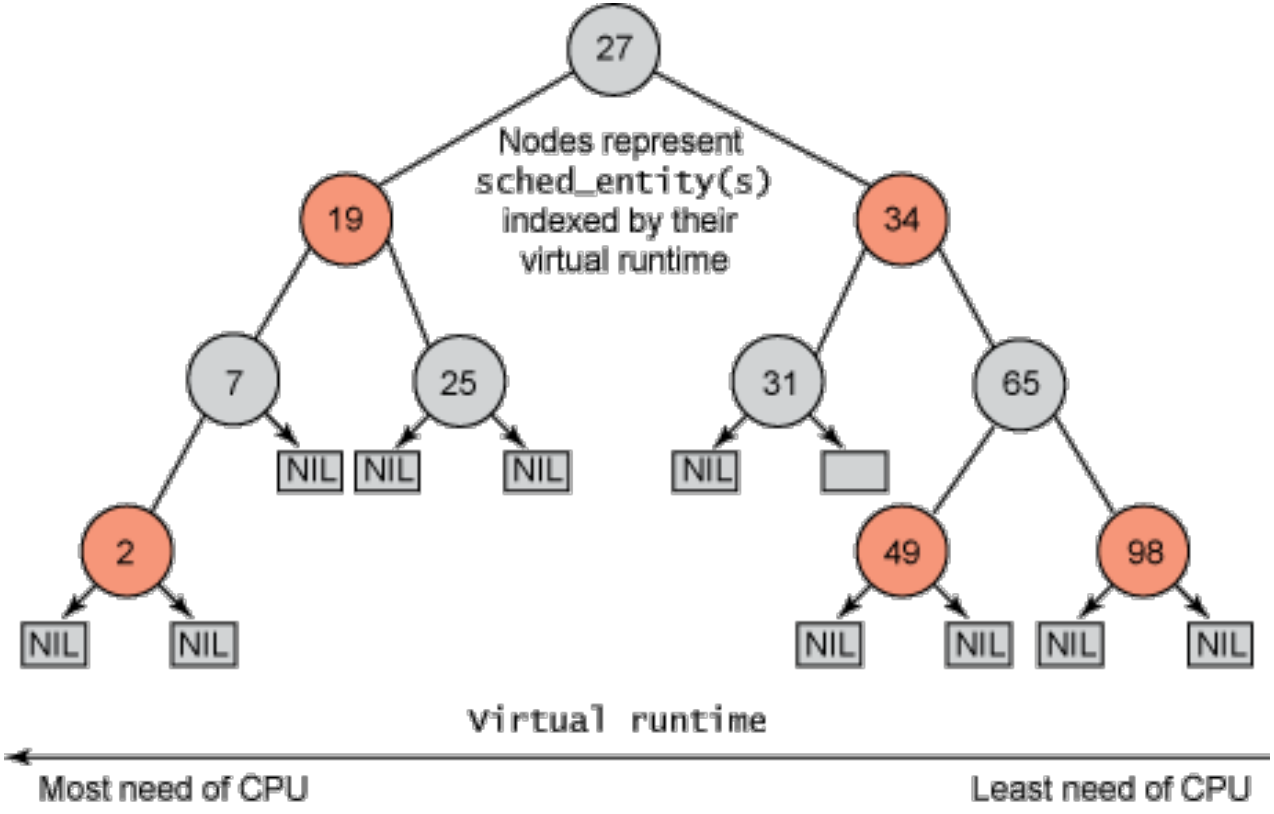
- Em uma CPU multitask perfeita
  - n processos poderiam rodar ao mesmo tempo
  - cada um receberia $1/n$ do poder de processamento
- Em uma CPU real
  - um processo roda e os outros esperam
- Primeira versão foi escrita em 62 horas

CFS: Implementação

- p->wait_runtime: tempo que a thread deve rodar para alcançar a justiça na distribuição
- em um ambiente ideal, p->wait_runtime seria sempre zero
- thread escolhida tem sempre o maior valor de p->wait_runtime
- enquanto uma thread está rodando p->wait_runtime é decrementada

  p->wait_runtime = p->wait_runtime - time_running
- a thread sofre preempção quando atinge o menor p->wait_runtime

CFS: Árvore Rubro-Negra



Inside the Linux 2.6 Completely Fair Scheduler

CFS

- Virtual time: tempo corre mais devagar do que no mundo real
- Velocidade é inversamente proporcional ao número de processos

Um cartoon inspirou Con Kolivas

Con Kolivas Introduces New BFS Scheduler, Linux Magazine

2009: BFS

BFS is the Brain Fuck Scheduler. It was designed to be forward looking only, make the most of lower spec machines, and not scale to massive hardware. ie it is a desktop orientated scheduler, with extremely low latencies for excellent interactivity by design rather than "calculated", with rigid fairness, nice priority distribution and extreme scalability within normal load levels. —Con Kolivas

Referências

- Inside the Linux 2.6 Completely Fair Scheduler, Tim Jones
- Inside the Linux Scheduler, Tim Jones