
The Completely Fair Scheduler

Thomas Gleixner

Linuxfoundation Japan Symposium
July 2008

Linux scheduler history

Linux 1.0 simple linked list of runnable processes.

Scheduler decision $O(N)$

Linux 2.0 SMP support

Linux 2.5 $O(1)$ scheduler

Linux 2.6.23 CFS

O(1) Scheduler

- Priority sorted array of lists
- O(1) insertion and selection time
- Nice values are converted to priorities

O(1) Scheduler

Realtime Priority 99
Realtime Priority 98

Realtime Priority 2
Realtime Priority 1
Nice -20
Nice -19

Nice +1
Nice 0
Nice -1

Nice +18
Nice +19

O(1) Scheduler

- Fairness is hard to achieve
- Interactive tasks are treated special via heuristics
- In kernel auto renicing to avoid starvation of higher nice levels
- CPU share is not easy to calculate

Staircase scheduler

- Developed by Con Kolivas et al
- Interactivity focussed in the first place
- Fairness algorithm on top of the $O(1)$ scheduler design

CFS Scheduler

- Inspired by Con Kolivas scheduler work
- Complete new design written by Ingo Molnar
- Free of heuristics
- Fairness algorithm is simple math
- Extendible framework, which makes it easy to introduce new scheduler algorithms or even a pluggable scheduler implementation

O(1) vs. CFS Scheduler

O(1) Scheduler

- One priority array for RT and non RT tasks
- One decision algorithm based on the position in the priority array
- Fairness based on heuristics

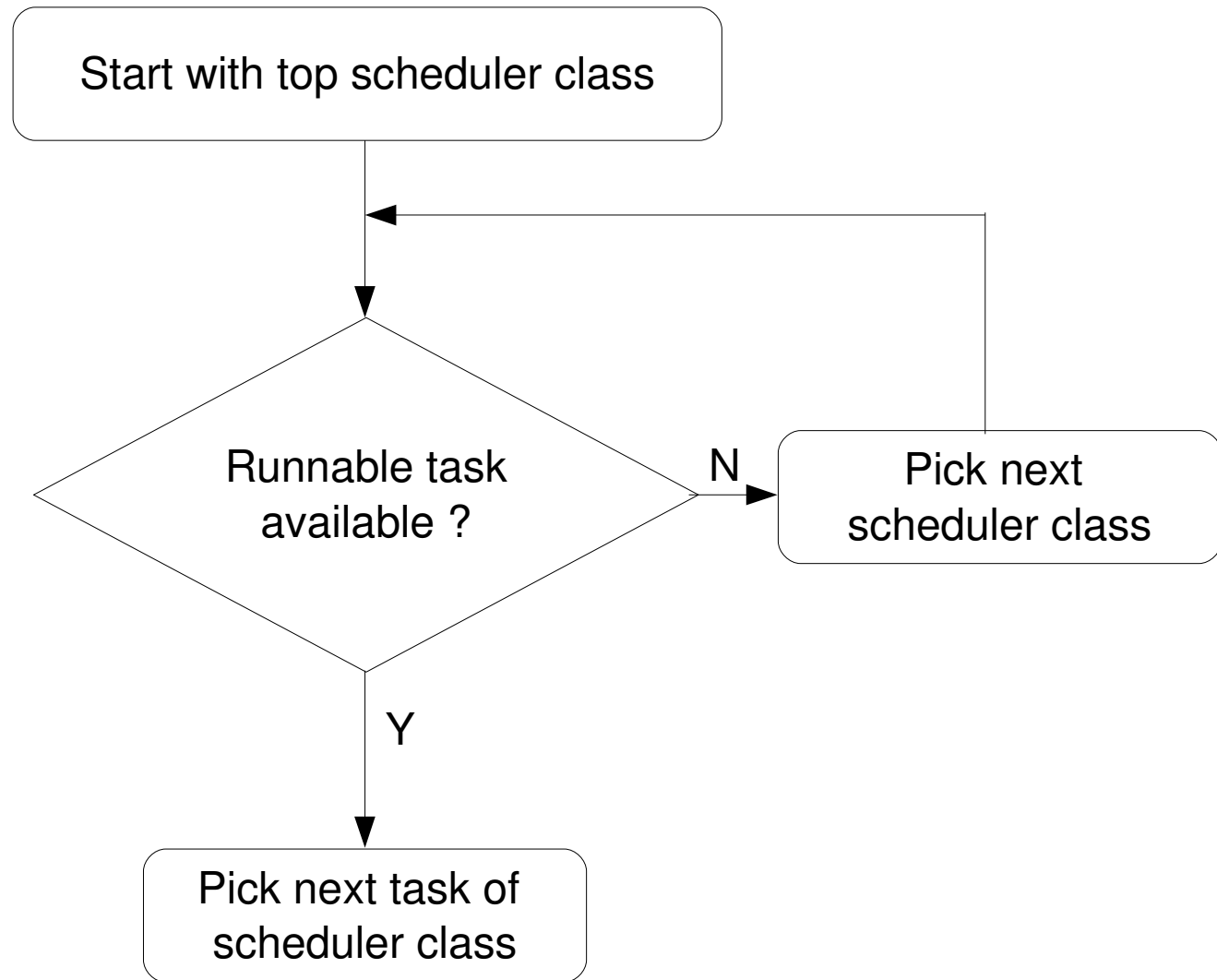
CFS Scheduler

- Seperate queue mechanisms and scheduler decision functions for RT and non RT tasks (scheduler classes)
- Fairness based on pure math

CFS Scheduler classes

- Extensible hierarchical set of scheduler classes
- `rt_sched_class`
 - Handles `SCHED_FIFO/RR` tasks
 - $O(1)$ priority array
- `fair_sched_class`
 - Handles `SCHED_OTHER` tasks
 - $O(\log(N))$ red black tree
- `idle_sched_class`
 - Handles idle task

CFS Scheduler decision



CFS fair_sched_class

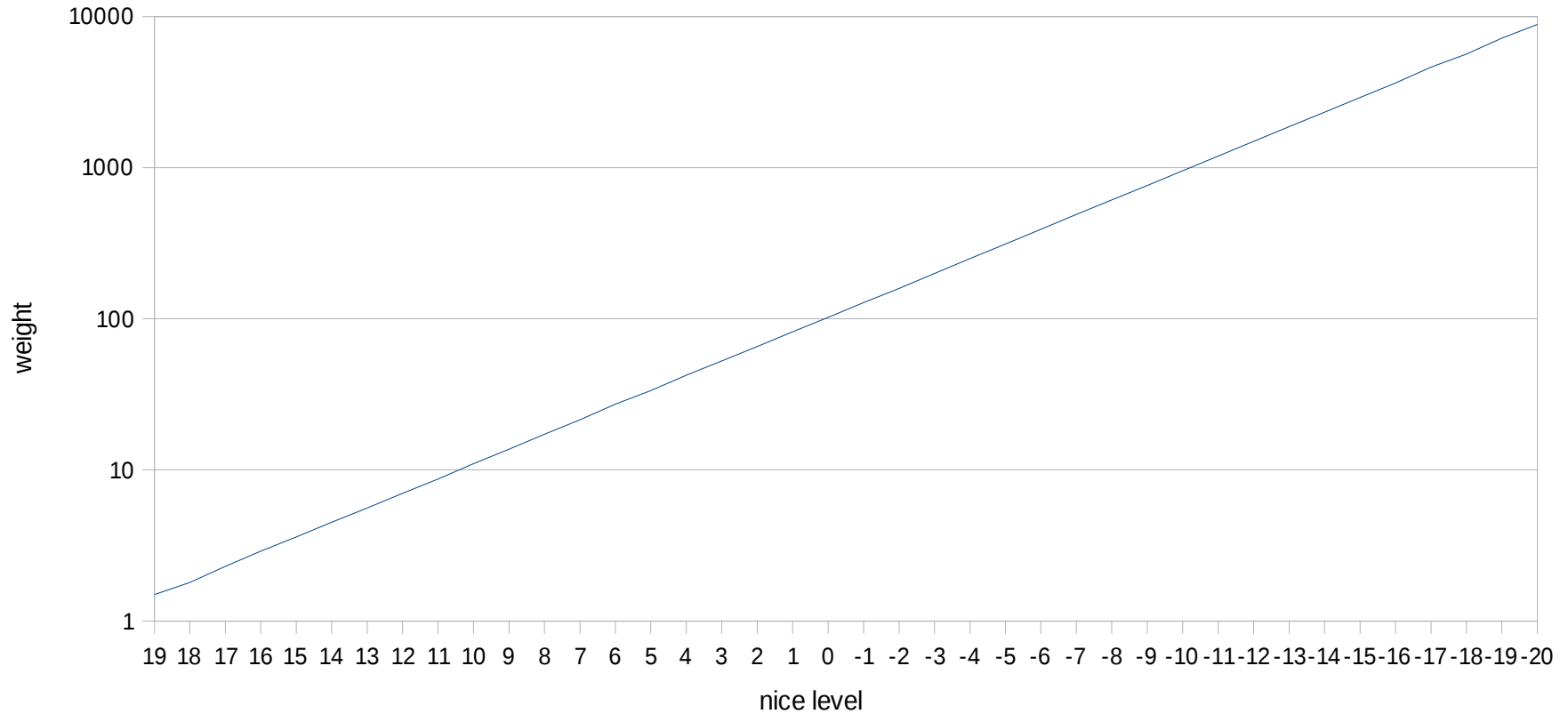
- Similar to “fair queuing” for packet networks
- Red black tree for task management keeps a virtual timeline of tasks to schedule
- Scheduler decision is $O(1)$
- Reinsertion of a task is $O(\log(N))$

CFS fair_sched_class

- nanoseconds based accounting (independent of HZ and jiffies)
- Task with the longest wait time in the r/b tree is selected next
- Nice levels are not depending on the timeslice
- Nice levels are multiplicative
- Interactive tasks sleep time is honoured

CFS nice levels

runtime weight vs. nice level



$$\text{weight}(n) = 1.25 * \text{weight}(n-1)$$

CFS and high resolution timers

- On high resolution timer systems fine granular scheduling with an hrtimer instead of the tick
- Scheduler timer also can be used for:
 - Fine grained Round Robin time slices
 - Runtime watchdog for SCHED_FIFO tasks

Group scheduling

- Makes use of the hierarchical modular design
- Groups tasks per UID or user defined control groups
- Exact CPU time share for each group
- Fairness across users
- Can limit the total CPU time available for RT tasks

Work in Progress

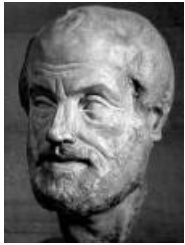
- EDF (Early Deadline First) scheduling
 - Basic implementation available
 - Priority Inheritance support for EDF is not yet solved

Authors

- Main author: Ingo Molnar
- Contributors:

Dmitry Adamushko, Dhaval Giani, Gautham Shenoy, Gregory Haskins, Hiroshi Shimamoto, Mike Galbraith, Mike Travis, Oleg Nesterov, Peter Williams, Peter Zijlstra, Srivatsa Vaddagiri, Steven Rostedt and many others

Troughout history, the advancement of universal knowledge has rested on the contributions of any that facilitate a continuing series of inspired innovations and discoveries.



"The whole is greater than the sum of its parts."

Aristotle (384 BC - 322 BC)



"If I have seen further it is by standing on the shoulders of giants."

Sir Isaac Newton (1642 - 1727)



"What is impossible for the individual, many can do."

Friedrich Wilhelm Raiffeisen (1818 - 1888)

In formal computer science advances
are made by standing on the shoulders
of giants.

Linux has proved that if there are
enough of you, you can advance just
as far by standing on each others toes.

LKML 2007
