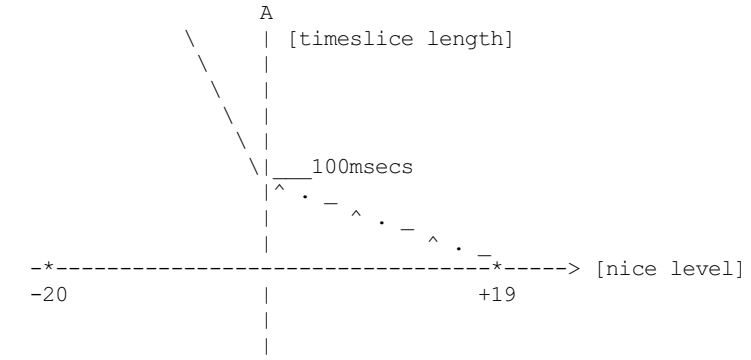
Scheduler Nice Design

This document explains the thinking about the revamped and streamlined nice-levels implementation in the new Linux scheduler.

Nice levels were always pretty weak under Linux and people continuously pestered us to make nice +19 tasks use up much less CPU time.

Unfortunately that was not that easy to implement under the old scheduler, (otherwise we'd have done it long ago) because nice level support was historically coupled to timeslice length, and timeslice units were driven by the HZ tick, so the smallest timeslice was 1/HZ.

In the O(1) scheduler (in 2003) we changed negative nice levels to be much stronger than they were before in 2.4 (and people were happy about that change), and we also intentionally calibrated the linear timeslice rule so that nice +19 level would be _exactly_ 1 jiffy. To better understand it, the timeslice graph went like this (cheesy ASCII art alert!):



So that if someone wanted to really renice tasks, +19 would give a much bigger hit than the normal linear rule would do. (The solution of changing the ABI to extend priorities was discarded early on.)

This approach worked to some degree for some time, but later on with HZ=1000 it caused 1 jiffy to be 1 msec, which meant 0.1% CPU usage which we felt to be a bit excessive. Excessive _not_ because it's too small of a CPU utilization, but because it causes too frequent (once per millisec) rescheduling. (and would thus trash the cache, etc. Remember, this was long ago when hardware was weaker and caches were smaller, and people were running number crunching apps at nice +19.)

So for HZ=1000 we changed nice +19 to 5msecs, because that felt like the right minimal granularity - and this translates to 5% CPU utilization. But the fundamental HZ-sensitive property for nice+19 still remained, and we never got a single complaint about nice +19 being too _weak_ in terms of CPU utilization, we only got complaints about it (still) being too _strong_ :-)

To sum it up: we always wanted to make nice levels more consistent, but within the constraints of HZ and jiffies and their nasty design level coupling to timeslices and granularity it was not really viable.

The second (less frequent but still periodically occurring) complaint about Linux's nice level support was its asymmetry around the origin (which you can see demonstrated in the picture above), or more accurately: the fact that nice level behavior depended on the _absolute_ nice level as well, while the nice API itself is fundamentally "relative":

```
int nice(int inc);

asmlinkage long sys_nice(int increment)
```

(the first one is the glibc API, the second one is the syscall API.) Note that the 'inc' is relative to the current nice level. Tools like bash's "nice" command mirror this relative API.

With the old scheduler, if you for example started a niced task with +1 and another task with +2, the CPU split between the two tasks would depend on the nice level of the parent shell - if it was at nice -10 the CPU split was different than if it was at +5 or +10.

A third complaint against Linux's nice level support was that negative nice levels were not 'punchy enough', so lots of people had to resort to run audio (and other multimedia) apps under RT priorities such as SCHED_FIFO. But this caused other problems: SCHED_FIFO is not starvation proof, and a buggy SCHED_FIFO app can also lock up the system for good.

The new scheduler in v2.6.23 addresses all three types of complaints:

To address the first complaint (of nice levels being not "punchy" enough), the scheduler was decoupled from 'time slice' and HZ concepts (and granularity was made a separate concept from nice levels) and thus it was possible to implement better and more consistent nice +19 support: with the new scheduler nice +19 tasks get a HZ-independent 1.5%, instead of the variable 3%-5%-9% range they got in the old scheduler.

To address the second complaint (of nice levels not being consistent), the new scheduler makes nice(1) have the same CPU utilization effect on tasks, regardless of their absolute nice levels. So on the new scheduler, running a nice +10 and a nice 11 task has the same CPU utilization "split" between them as running a nice -5 and a nice -4 task. (one will get 55% of the CPU, the other 45%.) That is why nice levels were changed to be "multiplicative" (or exponential) - that way it does not matter which nice level you start out from, the 'relative result' will always be the same.

The third complaint (of negative nice levels not being "punchy" enough and forcing audio apps to run under the more dangerous SCHED_FIFO scheduling policy) is addressed by the new scheduler almost automatically: stronger negative nice levels are an automatic side-effect of the recalibrated dynamic range of nice levels.