# Zack's Kernel News

**Chronicler Zack Brown reports on the latest news, views, dilemmas, and developments within the Linux kernel community.** *By Zack Brown*

## Benchmarking Futexes

Hitoshi Mitake forked some code that had been originally written by Darren Hart and Michel Lespinasse and produced a new performance-testing subsystem that he submitted as a patch, to be added to the `perf bench` code in the kernel. Hitoshi's subsystem was specifically oriented toward benchmarking futexes, a low-level locking mechanism.

The whole reason for kernel locks is to allow multiple users to access the same hardware resources on a given system. A resource is locked and released so quickly and so frequently that the system appears to be performing multiple user tasks at the same time. But locks also take CPU time away from the processes they're regulating. With so many locks closing and opening at all times, there's tremendous value to making the locking code itself as efficient as possible.

Darren, who was still maintaining his `futextest` test suite, on which Hitoshi's code was based, had some concerns. He didn't mind seeing his code get merged into perf, because that would necessarily give it a much wider audience, more testing, and all the usual benefits of code that's accepted into the kernel, but he said he was still actively modifying `futextest` and didn't want to have a plethora of similar-but-different tests appearing in perf. He also thought there were definitely some parts of `futextest` that would not be a good fit to be added to the perf code in the kernel.

Hitoshi said he'd only intended to port the code he'd already taken, but he felt that other parts of `futexttest` would be valuable to include in perf's `tools/` directory because they were useful on their own and were good examples of futex usage. He also felt that by including the code in the kernel, it would produce a statically linked binary, which would be better for embedded systems folks.

Darren was open to this, and Ingo Molnár also gave his OK, but he pointed out that if embedded systems people wanted a smaller binary, it would be better to include the functionality directly into the perf binary and just allow perf to be built with a user-selected group of features, to make it as small as possible.

For this, Arnaldo Carvalho de Melo suggested a standard-format

```
make -C tools/perf menuconfig
```

command, to configure perf in the same way that the kernel itself was configured. Everyone seemed to be on board with this, and Hitoshi said he'd get right on a fresh patch – modulo requirements of his day job.

## Memory Management For Swapless Systems

Sometimes an idea goes nowhere, but it's still interesting to think about.

John Stultz wanted to make some improvements to anonymous memory management on swapless systems. As he pointed out, anonymous memory was tracked in the kernel on two lists: one for active pages, and one for inactive pages. The way things stood, it would be straightforward to migrate inactive pages into swap if need be.

John's issue was that on swapless systems, the distinction between the active and inactive list became more vague, and the usefulness of both became more murky. The way it worked currently, active pages would sometimes migrate to the inactive list, but because there was no swap space to go to from there, they'd just sit pointlessly on the inactive list.

John's idea was to just skip that whole process. On systems without swap, active pages would just stay in the active list, and never migrate. That seemed like a much more simple and straightforward approach to him, and he posted a patch to accomplish it.

Minchan Kim replied that he'd tried to submit a similar patch long before, but that Rik van Riel had rejected his code. According to Minchan, Rik had said that the idea could work fine for truly swapless systems, but that a system could appear to be swapless, just because it didn't have any swap enabled; in which case, if Minchan's (or John's) patch were in place, all of a sudden

**ZACK BROWN**

The Linux kernel mailing list comprises the core of Linux development activities. Traffic volumes are immense, often reaching 10,000 messages in a week, and keeping up to date with the entire scope of development is a virtually impossible task for one person. One of the few brave souls to take on this task is **Zack Brown**.

the logic would be broken if that system decided to enable swap. So, that was that. But, it's really cool that people like John are exploring the kernel, learning its intricacies, and looking for ways to smooth it out and make it better.

## Leap Seconds

Before the June 30th "leap second," Richard Cochran submitted some code to help the kernel deal with it correctly. According to Richard, Posix UTC was simply a flawed design that could never be implemented usefully. Apparently, at the time when the Posix standard was being devised, computer clocks weren't accurate enough for the relevant issues to be obvious to the standards folks.

After correcting an early flawed definition of Posix UTC leap years, the standard has still suffered from problems that have become more and more significant as computer clocks have become more accurate. To accommodate Posix, the Unix implementations of the world have had to become more complex. The ideal solution, according to some, would be a better standard that takes into account modern timekeeping issues; however, that would apparently involve a massive, world-wide transition effort, not to mention backward compatibility issues.

The easier solution, as Richard proposed, would be to treat UTC time as a feature that the kernel could provide if requested. This is actually a sort of standard way Linux gets around Posix compliance issues in general – it provides the Posix interface as required, but has a whole different set of interfaces for people who *really* want to get things done the right way. I believe some aspects of Linux process threading get around Posix this way as well.

In Richard's patch, the kernel would implement its own internal timekeeping, correctly handling leap seconds, leap minutes, leap days, leap years, leap centuries, leap millennia, leap eternities, and so on. Then, for any process that requested UTC time, the kernel would just apply the offsets it had recorded and produce the UTC time on demand. This way, the kernel would not have to monitor for leap seconds continuously, or set up timers, or use other awkward constructions.

One interesting topic that came up during the discussion is the way Google handled the leap second issue on its servers. According to a company blog entry, they implemented a "smearing" function, whereby the kernel would just lengthen or contract its time reporting by very tiny increments, until the leap second had been accomplished. In this way, applications would continue to function, and the time adjustment would occur when needed.

The problem with this approach, as the blog entry affirms, is that it results in inaccurate time reporting. For Google's purposes this inaccuracy was unimportant, but for a general-purpose operating system like Linux, there's a greater need to really nail down accurate time reporting.

So, although some kernel developers advocated the Google approach as a relatively elegant workaround, it wasn't taken seriously as something that might actually be adopted into the kernel.

The discussion got pretty technical at a certain point, mainly between John Stultz and Richard, as they hammered out the details of Richard's implementation and what it would accomplish. I personally love these Y2K-esque issues when they come up. They're so crazy. The deeper you dig, the crazier they seem, and the more amazing the developers seem for being able to fathom all the nuances. ■■■