Linus Torvalds says “Don’t use ZFS”—but doesn’t seem to understand it

Enlarge / Linus Torvalds is eminently qualified to discuss issues with license compatibility and kernel policy. However, this does not mean he’s equally qualified to discuss individual projects in project-specific context.

reader comments

74 with 55 posters participating, including story author

Share this story

Last Monday in the “Moderated Discussions” forum at realworldtech.com, Linus Torvalds—founding developer and current supreme maintainer of the Linux kernel—answered a user’s question about a year-old kernel maintenance controversy that heavily impacted the ZFS on Linux project. After answering the user’s actual question, Torvalds went on to make inaccurate and damaging claims about the ZFS filesystem itself.

Given the massive weight automatically given Torvalds’ words due to his status as founding developer and chief maintainer of the Linux kernel, we feel it’s a good idea to explain both the controversial kernel change itself, and Torvalds’ comments about both the change in question and the ZFS filesystem.

The original January 2019 controversy, explained

In January 2019, kernel developer Greg Kroah-Hartman decided to disable exporting certain kernel symbols to non-GPL loadable kernel modules.

For those whose heads are spinning, kernel symbol exports expose internal information about the kernel state to loadable kernel modules. The particular symbol being discussed here, _kernel_fpu_, tracks the state of the processor’s Floating Point Unit. Without access to that symbol, external kernel modules that access the FPU directly—as ZFS does—must implement state preservation code of their own. State preservation, whether in-kernel or native to kernel modules, makes sure that the original state of the FPU is restored before control is released to other kernel code that may be dependent on the values they last saw in the FPU’s registers.

The technical impact of refusing to continue exporting the

_kernel_fpu_ symbol is not to prevent modules from accessing the FPU directly—it only prevents them from using the kernel's own state-management facilities to preserve and restore state. Removing access to that symbol therefore requires module developers to reinvent their own state-preservation code individually. This increases the likelihood of catastrophic error within the kernel itself, since improperly restored state could cause a later kernel operation to crash.

Kroah-Hartman's decision to stop exporting the symbol to non-GPL kernel modules appeared to be driven largely by spite, as borne out by his own comment regarding the change: "my tolerance for ZFS is pretty non-existent." Normally, ZFS—on any platform, including the BSDs—uses SSE/AVX SIMD vector optimization to speed up certain operations. Without access to the _kernel_fpu_ symbol, ZFS developers were initially forced to disable the SIMD optimizations entirely, with fairly significant real-world performance degradation.

Although Kroah-Hartman's change initially spawned a lot of drama and uncertainty, the long-term impact on the Linux ZFS community was fairly minimal. The breaking change only affected bleeding-edge kernels that few ZFS users were using in production, and in July 2019 new, in-module state management code was committed to the ZFS on Linux source tree.

"We don't break users"

Torvalds' position in last Monday's forum post starts out reasonable and well-informed—after all, he's Linus Torvalds, discussing the Linux kernel. He notes that the famous kernel mantra "we don't break users" is "literally about user-space applications"—and so it does not apply to Kroah-Hartman's decision to stop exporting kernel symbols to non-GPL kernel modules. By definition, if you're looking for a kernel symbol, you aren't a user-space application. The line being drawn here is a very bright and functional one: Torvalds is saying that

if you want to run in kernel space, you need to keep up with kernel development.

From there, Torvalds branches out into license concerns, another topic on which he's accurate and reasonable. "Honestly, there is no way I can merge any of the ZFS efforts until I get an official letter from Oracle," he writes. "Other people think it can be OK to merge ZFS code into the kernel and that the module interface makes it OK, and that's *their* decision. But considering Oracle's litigious nature, and the questions over licensing, there's no way I can feel safe in ever doing so."

He goes on to discuss the legally flimsy nature of the kernel module "shim" that the ZFS on Linux project (along with other non-GPL and non-weak-permissive projects, such as Nvidia's proprietary graphics drivers) use. There's some question as to whether they constitute a reasonable defense *now*—since nobody has challenged any project for using an AGPL shim for 20 years and running—but in purely logical terms, there isn't much question that the shims don't accomplish much. The real function of an AGPL kernel module shim isn't to sanction touching the kernel with non-GPL code, it's to protect the proprietary code on the far side of the shim from being forcibly published in the event of a GPL enforcement lawsuit victory.

So far, so good, but then Torvalds dips into his own impressions of ZFS itself, both as a project and a filesystem. This is where things go badly off the rails, as Torvalds states, "Don't use ZFS. It's that simple. It was always more of a buzzword than anything else, I feel… [the] benchmarks I've seen do not make ZFS look all that great. And as far as I can tell, it has no real maintenance behind it any more…"

“It was always more of a buzzword than anything else”

This jaw-dropping statement makes me wonder whether Torvalds has ever actually used or seriously investigated ZFS. Keep in mind, he’s not merely making this statement about ZFS *now*, he’s making it about ZFS for the last 15 years—and is relegating everything from atomic snapshots to rapid replication to on-disk compression to per-block checksumming to automatic data repair *and more* to the status of “just buzzwords.”

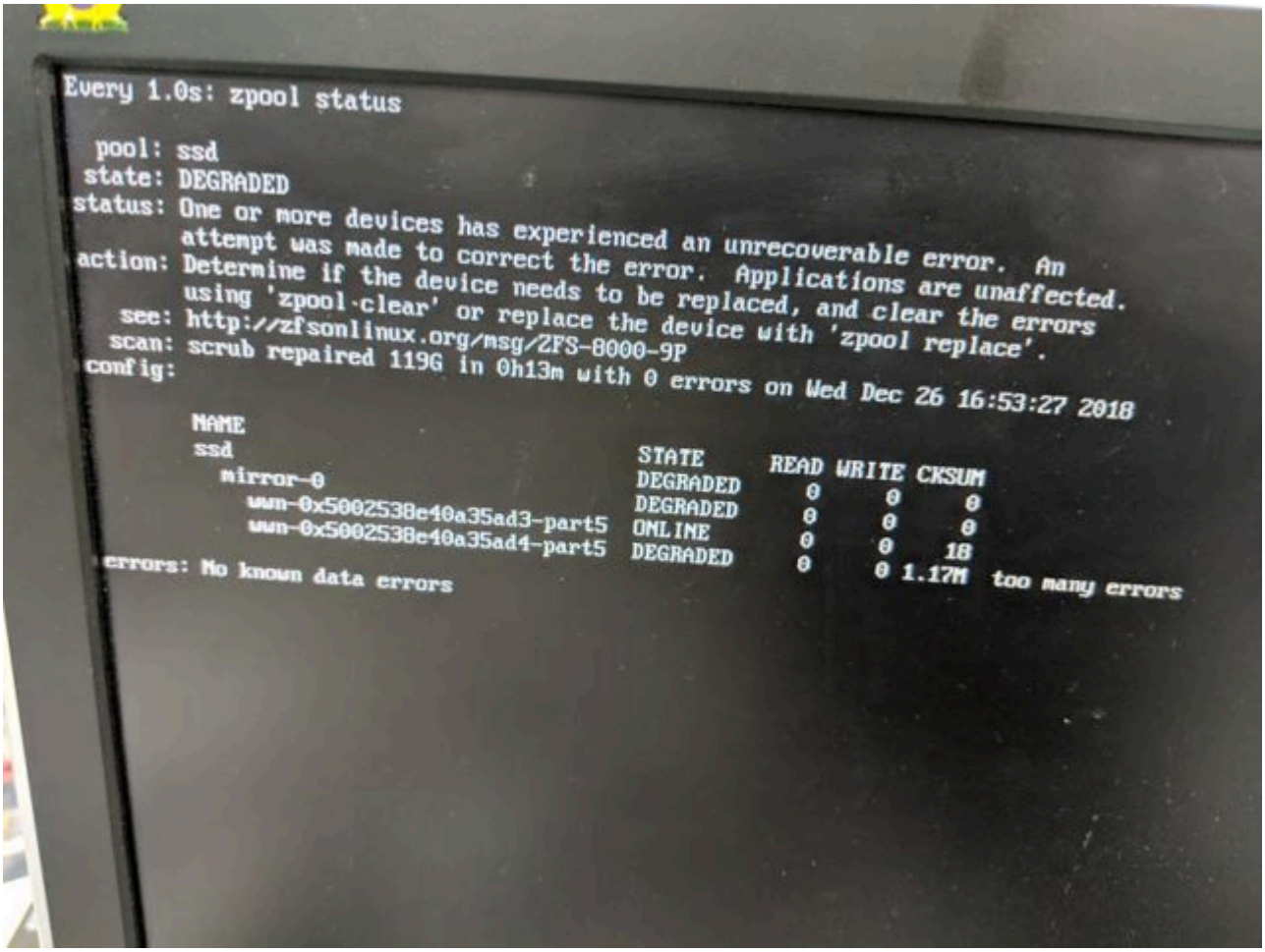
There’s only one other widely available filesystem that even takes a respectable stab at providing most of those features, and that’s btrfs—which was not available for the first several years of ZFS’ general availability. In fact, btrfs still isn’t really stable enough for production use, unless you nerf all the features that make it interesting in the first place.

ZFS’ per-block checksumming and automatic data repair has prevented data loss in my own real-world use *many* times, including this particularly egregious case of a SATA controller gone rabid. A standard RAID1 mirror would have cheerfully returned that 119GB of bad data with no warning whatsoever, but ZFS’ live checksumming and error detection mitigated the whole thing to the point of never having to so much as touch a backup.

Meanwhile, atomic snapshots make it possible to keep a full block-for-block identical copy of storage at a point in time with negligible performance overhead and minimal storage overhead—and replication of those snapshots is typically hundreds or thousands of times faster (and more reliable) than non-filesystem-integrated solutions like rsync.

It’s possible to not have a personal need for ZFS. But to write it off as “more of a buzzword than anything else” seems

to expose massive ignorance on the subject.



Enlarge / Yes, that’s more than one MILLION blocks that returned bad data on one disk in the mirror—and another 18 on the other disk, just for good measure. “No known data errors.”
Jim Salter
https://arstechnica.com/?p=1642331