Zack's Kernel News

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

By Zack Brown

Status of OverlayFS and Union Filesystems in General

Recently, Miklos Szeredi requested that OverlayFS be included in the main kernel tree. OverlayFS allows two directory trees to appear as one. Two files with the same path on each tree would appear to occupy the same directory in the overlayed filesystem. The project has been in existence for several years, but this time Linus Torvalds replied, "Yes, I think we should just do it. It's in use, it's pretty small, and the other alternatives are worse. Let's just plan on getting this thing done with."

Al Viro said he'd start reviewing the code, but he also suggested that if they were going to merge a union filesystem such as OverlayFS, they might as well consider merging other similar projects, such as Unionmount and Aufs. Unionmount in particular, he said, had been getting some good work lately from David Howells.

Meanwhile, Sedat Dilek jumped for joy at seeing OverlayFS close to acceptance. Al also replied again with his initial review. He'd identified some security issues and other technical problems, and he went back and forth with Miklos about them. The two at first didn't see eye-to-eye about how to fix the issues, or even whether a given issue was really a problem.

At one point, George Spelvin offered his, admittedly, somewhat hacky solution to one of Al's problems. The whole thing boiled down to the way OverlayFS or any union filesystem would behave under the full range of possible uses. Regarding George's particular suggestion, Al walked through the convoluted process necessary to remove a directory [1] and replied, "I'm sorry, but this is insane."

Elsewhere, in an entirely different thread, Sedat asked about the status of David's Unionmount project. David replied, "It's being reengineered again to take account of VFS changes that went in in the last merge window."

He added, "It's a maze of twisty locking problems – some of which also apply to things like overlayfs:-(".

The discussion in both threads ended there. It appears everyone, including Linus, is ready to see union filesystems like OverlayFS in the kernel. But no one, including Al Viro and the maintainers of the various union filesystem projects, are able to solve satisfactorily the technical problems that remain. At the moment, none of the projects seem close to getting past Al's laser-beam code reviews, and until that happens, I'm certain none of them will be merged.

Astonishing Tux3 Performance Claims

There seems to be some suspicion between certain kernel developers and Tux3 developers. Tux3 is a versioning filesystem that's been in development since 2008. Recently, Daniel Phillips, the project leader, posted some benchmarks that showed Tux3 outperforming tmpFS. As he put it, "To put this in perspective, we normally regard tmpfs as unbeatable because it is just a thin shim between the standard VFS mechanisms that every filesystem must use, and the swap device."

Dave Chinner took a look at Daniel's numbers and found some issues that he felt indicated a deliberate attempt to mislead people. In particular, he pointed out that the Tux3 benchmark didn't include any "flush" operations – the Tux3 front end was off-loading all of its work to a back end that could take all the time it needed to complete the job. The front end would never block, and so it could simply race through the benchmark and exit. Dave said, "You've carefully crafted the benchmark to demonstrate a best case workload for the tux3 architecture, then carefully not measured the overhead of the work tux3 has offloaded, and then not disclosed any of this in the hope that all people will look at is the headline."

Hirofumi Ogawa, one of the Tux3 developers, responded, saying `fsync()` had not yet been implemented, and the benchmarks were intended to show comparisons between just the parts of the code that had already been written.

Daniel also responded to Dave's post, saying, "I should indeed have noted that 'modified dbench' was used for this benchmark, thus amplifying Tux3's advantage in delete performance. This literary oversight does not make the results any less interesting: we beat Tmpfs on that particular load. Beating tmpfs at anything is worthy of note."

Regarding the specific issue Dave had raised about off-loading 100% of Tux3's

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**.

work, Daniel said, "Yes, that is the entire point of our front/back design: reduce application latency for buffered filesystem transactions."

Theodore Ts'o pointed out that one couldn't simply ignore the `fsync()` data and expect a meaningful benchmark result. As he put it, "Since `fsync()` is defined as not returning until the data written to the file descriptor is flushed out to stable storage – so it is guaranteed to be seen after a system crash – it means that the foreground application must not continue until the data is written by Tux3's back-end." He added, "any advantage of decoupling the front/back end is nullified, since `fsync()` requires a temporal coupling."

Daniel replied that when they optimized fsync, he expects "… Tux3 to perform competitively, because our delta commit scheme does manage the job with a minimal number of block writes …" [2].

Elsewhere in the thread, Dave remarked on his real concern. He said, "I don't care how fast tux3 is – I care about being able to reproduce other people's results. Hence if you are going to report benchmark results comparing filesystems then you need to tell everyone exactly what you've tweaked and why, from the hardware all the way up to the benchmark config."

The discussion trailed out around there, but some kernel folks also seemed to feel that Daniel's approach was too marketing-oriented, trying to make big announcements at the expense of clarifying the real progress made.

Dealing with Empty Symlinks

Back in January, Pádraig Brady noticed that Linux didn't allow users to create symlinks that pointed to non-existent files. He asked why this was, because POSIX specified that it should be allowed, and other operating systems supported it. There was no discussion at the time, but he recently followed up again, asking if this was going to be fixed.

Part of the idea was that symlinks could be valuable just to store data in their name alone, without utilizing their traditional purpose of linking to other files.

But Al Viro thought this was "utterly pointless," especially considering that the behavior would end up being operating-system-dependent anyway. He said, "blanket refusal to traverse such beasts is a legitimate option."

Eric Blake replied that the real point was not whether creating an empty symlink should be allowed in Linux – it was the way Linux should behave when it encountered an empty symlink during path resolution.

After all, even if Linux didn't allow empty symlinks to be created, other operating systems did, and the filesystems containing those symlinks could be mounted under Linux. It would make sense to handle those cases correctly. Eric remarked:

"I personally don't care whether you fix the Linux kernel `symlink()` to allow empty symlinks, or successfully argue for a bug fix against POSIX to permit the existing Linux `symlink()` behavior. I'd love to see Linux obtain POSIX certification someday, and either of those two courses of action would get us closer. Meanwhile, I know there are enough other issues in the kernel … that it will be a long time before we ever get a POSIX certification of a Linux system."

Pavel Machek started exploring the extent of the issue under Linux, trying to identify which tools would break when encountering empty symlinks and how bad a break it would be, but the discussion ended at that point, with no clear resolution on a course of action, or even it was worth doing anything about the situation.

Linus Torvalds is notoriously disdainful of compliance for compliance's sake. If there's no cost to it, he's not opposed, but if there are valid technical reasons to implement something in a non-compliant way, he'll choose that over compliance every time, and he makes no secret of his contempt for certain parts of the POSIX standard.

On the other hand, if there's a danger that users might get burned if they mount a filesystem on which another OS has created an empty symlink, Linus would rather eat sand than let that go unfixed. The real question may boil down to whether the status quo would burn anyone. At the moment, it still seems unclear.

Difficult Bug Hunt

Michael Hocko used `git bisect` to track down a problem resuming a suspended system. Instead of resuming, the system would just reboot. He posted a patch to revert the commit that seemed to cause the problem.

H. Peter Anvin asked for more details about Michael's system; H. Peter said, "This is one of a series of extremely bizarre suspend to RAM failures we are trying to make sense of." The particular cause of the problem, he said, was "not just bizarre, this is extremely disturbing."

The reason H. Peter found this so disturbing is that the piece of code Michael had reverted did nothing more than flip the NX bit. The NX bit is used in some CPUs to mark areas of memory as being "never executable" and flipping that bit should never affect anything just on its own. The only way it could be involved in a problem with suspend-to-RAM is if there were some deeper malignancy.

Linus joined the discussion and traced the problem to `__initdata`, a special part of the kernel that marks certain things as being solely related to initialization, so that once initial-

ization is completed, the kernel can free all associated memory.

Apparently, the NX bit had been preventing a particular region of memory from executing as code, and that region of memory had been getting corrupted by `__initdata`. As long as the NX bit had been set as it was, the corruption problem had been covered up. That's why Michael had been able to bisect the issue and trace it to a patch that flipped the NX bit – and why reverting that patch had seemed to fix the problem. Michael posted a fresh patch to fix the underlying issue with the `__initdata`, and Linus incorporated it immediately into his tree.

License Discussion

Eric Appleman opened a wriggling can of worms when he suggested gathering Linux kernel copyright holders (i.e., anyone who's contributed code to the kernel) into a group to enforce the GPL against license violators. Luke Leighton said he'd love to join this sort of thing, although he admitted that his copyrighted contributions were few. Almost as an aside, he remarked, "can it be specifically noted, from this moment onwards, that all contributions that I have made to the linux kernel are dual-licensed under both the GPLv2 and also the GPLv3+ license?"

Cole Johnson replied that as far as he knew, "Linus said he will NOT use the GPLv3 for the kernel." This apparently threw Luke into a rage, and he said neither Linus Torvalds nor anybody else could prevent him from releasing his copyrighted code under any license he wanted. Among the variety of relatively heated posts in this thread was Theodore Ts'o reply to the whole idea of dual-licensing with the GPLv3:

It's not just Linus; many senior Linux kernel developers have spoken very clearly that the anti-Tivoization clause in GPLv3 is totally unacceptable …. This means that GPLv3-only code is always going to be incompatible with code released as part of the Linux kernel, because substantial parts of the kernel have and will be available only under a GPLv2 only license.

If anyone wants to release their code under a dual-license, it's easiest if that's how you submitted the code originally. For example, … to encourage its use in other operating systems.

If you have only contributed a few lines … specifying that "these 15 lines of the function I_worship_at_the_altar_of_rms() are under the GPLv2/v3, even through the rest of the file is GPLv2-only" is not something that we generally do. Speaking as a subsystem maintainer, … if someone insisted on line-level copyright statements, I'd just simply reject the patch rather than dealing with the accounting nightmare. If you want to add a GPLv2/GPLv3 dual license to a file, … you'll need to get the consent of everyone who has contributed changes to that file.

Finally, as Jonas has stated, if you are trying to impose the anti-Tivoization clause through the back door, it's not going to have that effect, since people can always choose either license for dual-licensed code, and for the kernel GPLv2 always has to be one of the choices.

A bit later in the discussion, he posted this equally fascinating follow-up [3]:

*The more subtle thing to consider is that with dual-licensed code, ***anyone*** has the ability to strip one of the licenses from the code in the course of making [a] modification. … That's a completely legal thing to do …. The reason why I dislike someone taking GPLv2/v3 code and stripping out the GPLv2 license is because it makes new versions of code which I had originally written becoming available only under a GPLv3 license.*

*But there's a flip side to this, which is, the same legal argument ***also*** allows a kernel maintainer to take a contribution which is under a GPLv2/v3 or GPLv2+ license, and incorporate it into a GPLv2-only file, and not bother to mark that it originally came from a GPLv2+ or GPLv2/v3 contribution. … you could find that contribution and extract that code and use it in some other GPLv3 project. But we are under no obligation to mark that a particular set of lines in a file originally came from a GPLv2/v3 or GPLv2+ contribution. … That's not to say that certain drivers won't be dual licensed, for specific reasons, but you shouldn't expect that core kernel files will be GPLv3 compatible in the near future.*

Rob Landley gave a very bleak yet highly interesting assessment of the modern history of "copyleft" [4]. To Luke, he said: *You're aware that copyleft in general is declining, right?*

"The GPL" was synonymous with copyleft … and the only thing programmers had to know is whether or not some other license was GPL-compatible. If it was: treat it as GPL. If it wasn't: ignore it.

But there's no "the GPL" anymore. Linux and Samba can't share code, even though they implement two ends of the same protocol. And making your project "GPLv2 or later" means you can't take code from _either_ source. These days the GPL largely serves to _prevent_ code reuse, and people have responded to the perceived problems with "GPL-next" initiatives where they fragment copyleft further with Affero variants, by using creative commons on code, and so on. But copyleft only ever worked as one big universal license, and now it doesn't.

… the most common license on Github is "no license specified," and that's not just ignorance, that's napster-style civil disobedience from a generation of coders who lump copyright in with software patents and consider it all "too dumb to live." (You think GPL enforcement suits are viewed any differently than DMCA takedown notices on youtube, both coming from clueless old people?)

Now add in Android's official "no GPL in userspace" policy, which means that if you preinstall GPL (or LGPL) software in your install image, you can't use the Android trademark to describe your product. (Did I mention that smartphones are replacing the PC? …) I'm sorry, but Richard Stallman _screwed_ up_. GPLv3 succeeded where Sun's CDDL failed: it split copyleft into incompatible warring factions which are collectively shrinking in market share because none of them are as useful as "The GPL" was. … Advocating for GPLv2 to go away is sad, but understandable. Expecting GPLv2 to be replaced by GPLv3 is just delusional. ■■■

INFO

[1] Al Viro on George Spelvin's "hackish" solution: *http://lkml.indiana.edu/hypermail/linux/kernel/1303.2/02764.html*

[2] Tux3 fsync debate: *http://lkml.indiana.edu/hypermail/linux/kernel/1305.1/02246.html*

[3] Theodore Ts'o on dual-licensed code in the kernel: *https://lkml.org/lkml/2013/5/20/218*

[4] Rob Landley on copyleft: *https://lkml.org/lkml/2013/5/21/139*