Zack's Kernel News

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

By Zack Brown

Logging Early Crashes

Ahmed S. Darwish was getting system crashes so early in the boot process that no logging information had time to be saved. He didn't like this, so he wrote a patch that would rely on low-level BIOS routines to store logging information to disk, even if the system crashed near the very start of the boot process.

Ingo Molnár replied, "I have to admit that while I'm a rabid BIOS-hater, I find this debug feature very very interesting, for the plain reason that if it's implemented in a robust and clever way then this has a chance to improve debuggability of pretty much any Linux laptop quite enormously!"

Ingo's primary concern with Ahmed's code, was to make sure there was absolutely no chance that it might corrupt any real data. As a secondary concern, he hoped it would be possible to find an unused part of the disk to store these logs, so it wouldn't be necessary for users to do a complete reinstall to take advantage of the feature.

Tejun Heo was highly skeptical that any such features could work reliably – although he acknowledged that stranger things had happened in kernel land. And H. Peter Anvin felt that trying to prevent the BIOS from trashing user data would be an extremely hard problem.

At one point, Linus Torvalds intervened in the discussion to say, "Over the years, many people have tried to write things to disk on oops. I refuse to take it. No way in hell do I want the situation of 'the system is screwed, so let's overwrite the disk' to be something the kernel I release might do. It's crazy. That disk is a lot more important than the kernel, and overwriting it when we might have serious memory corruption issues or something is not a thing I feel is appropriate."

Folks continued discussing the technical obstacles, and how to overcome them, but Linus seemed pretty definite on the point. This kind of code will not go into the kernel.

Too Many Clocks?

Jeremy Kerr bemoaned the existence of more than 20 distinct implementations of the clock structure in the ARM architecture. He proposed some code that would unify all of these into a single implementation and presumably let folks write architecture-independent code without worrying about making the wrong assumptions about `struct clk`. His goal was to share the clock code among the various hardware platforms, while allowing folks to create clock devices dynamically in a platform-independent way.

Several folks offered some technical suggestions. Apparently, various tests and error conditions were a bit tricky to avoid, such as the proper way to detect when two devices were using the same clock. It was also important to make sure that generic clock features didn't try to do too much and get in the way of architecture-specific special handling requirements.

The code looks as if it is destined to be included in the official kernel. Although regular users will probably not notice this feature at all, it will simply make it easier for kernel developers to do more and better stuff with the kernel.

Generic IRQ Revamp

Thomas Gleixner submitted a big overhaul of the generic interrupt code. He fixed up the namespace to have less confusing accessor functions. He also did a pile of work encapsulating the code, so that not as much of the guts of the generic interrupt handler code is exposed.

Specifically, by further encapsulating the code, he made it much easier to detect when other folks' code was reaching too far into the IRQ code. This would at least force people to discuss their special requirements with him, instead of doing what they had been doing, which was just to reach into the code in odd and hard-to-detect ways. Using his new infrastructure, it should be possible to enhance the IRQ code gradually in a regular way to meet the needs of all comers.

Sam Ravnborg suggested documenting exactly what Thomas expected from anyone using the code, so they wouldn't end up trying to do the wrong thing, seeing compiler warnings, and perhaps having to have the situation explained to them.

There was also some concern over the size of Thomas' patches. He had billed the changes as just fixes, but Linus Torvalds noticed that a lot of new code was going in as well. Thomas explained that much of this was just transitional code to prevent breakage and would ultimately go away.

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**.

Limiting Global Capabilities

Eric Paris bemoaned the decision to remove the global capability bounding set from the kernel. The decision to replace it with a per-task bounding set that would be inherited by the children, was fine, he said, as long as you want to trust the parent process to make the proper choices. But, with the absence of a global capability bounding set, there was no way to enforce the removal of a particular capability from the system as a whole.

He had tried various clever solutions to this problem, such as dropping the unwanted capability by the init process before any other process had run. With this approach, all new processes would be children of init, and there would be no problem. But, a way around this technique was for a user to cause the kernel to try to autoload a module, at which point the kernel would produce a process that could have all capabilities enabled.

After much thought, Eric decided there was no way to do what he wanted within the existing infrastructure, so he posted a patch to reintroduce the global capability bounding set. In fact, his new version was more extreme, because not even init could reintroduce a capability that had been excluded by this new bounding set. Once a capability is dropped by the system, it never comes back. Only certain kernel threads might still have that capability, but they would not be able to pass that capability along to their children.

Various folks suggested existing security solutions, such as LSM and SELinux, as alternatives, but that can of worms was not fully opened in this discussion.

Serge E. Hallyn had a technical objection, pointing out that Eric's code actually changed the manner in which capabilities were inherited by processes. Eric agreed that this was a problem and said he would work to match the previous behavior but retain the global bounding feature.

Meanwhile, Andrew G. Morgan also objected to Eric's patch, saying that if a running system could simply drop capabilities out from under processes that might be using them, it could cause odd problems. Eric replied that he wasn't going to drop capabilities from processes that had them already, only from newly created processes.

In particular, Eric said, he wanted to boot the system, drop `CAP_SYS_MODULE` and `CAP_SYS_RAWIO`, and then hand root-level privileges to an untrusted user. That was the primary goal of his submission.

Steve Grub pointed out that Eric's idea was actually somewhat different from the normal threat model that Linux developers consider. The assumption, Steve said, is that once a user gains root, he can do anything he wants to the system. So, the focus of any security system is rather to prevent a user from gaining root in the first place. The idea of handing root privileges over to a user and then expecting to be able to control what happens after that point is not an approach taken by most folks working on Linux security issues.

The discussion then veered off into a consideration of virtualization, in which a user has root on a virtual machine and might want to attack another virtual machine elsewhere on the same physical system. So, it seems as though whatever Eric originally had in mind, the ultimate behavior might end up quite different from his implementation. But, at least he has piqued the interest of a number of kernel folks in doing something related to his original suggestion. ■■■