Zack's Kernel News

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

By Zack Brown

Hand-Picking PIDs

Recently, Pavel Emelyanov, Oleg Nesterov, Tejun Heo, and a number of other folks have been working in loose collaboration, to implement a way for the kernel to create processes with a specified process ID (PID), instead of just generating one on an as-needed basis.

Typically, the process ID doesn't matter to userspace, it's just something the kernel uses essentially for resource allocation and accounting. Sometimes a user process might need the PID of another process to send it a signal or do some other kind of accounting with it, but in that case, the specific PID itself can usually be just whatever number the kernel originally assigned. The user process doesn't care, as long as the PID doesn't change in the middle.

But in some circumstances it's inconvenient to let the kernel pick PIDs on its own. The GNU Debugger (GDB), for example, can create a "checkpoint" in the middle of running a process and then return to that checkpoint later, having the process restored to exactly the state it was in when that checkpoint was created – or, almost exactly. To be truly indistinguishable from its previous state, the checkpoint process, created by cloning the process being debugged, would need to be given the original process's PID.

Another use, as Pedro Alves pointed out, would be to allow seamless migration of processes between nodes on a large-scale computing cluster, pausing and resuming processes at will, as the Berkeley Lab Checkpoint/Restart project is doing.

So the opportunity for total insanity of the fun, useful variety does exist for this feature, and Pavel and the rest want to implement it. But, the best way to do so has been giving them problems. Within the kernel, such a feature could be created in a number of different places and have a number of different characteristics; for example, as Oleg suggested, there could simply be a `set_last_pid` file in the `/proc` directory, and changing the contents of that file would change the PID of the most recently created process. Or, as Pavel suggested, the API for the `clone()` system call could be modified to accept a desired PID as input.

Other possibilities exist, but part of the problem is that any implementation will have an effect on the surrounding kernel features, so Pavel and the rest want to make sure they pick the right spot. Over the course of discussion, Linus Torvalds chastised them for not trying hard enough to rein in the effect of the feature.

Overall, though, and partly because of Linus's objection, it seems clear that he considers this a worthwhile feature, although certainly a "best" approach does need to be found, with a clean interface and a minimal effect on the rest of the kernel internals.

Massive Boot-Time Speedup

One problem with the hardware trend of piling more and more CPUs into a single machine is that those CPUs all need to be started up at boot time. Currently, the Linux kernel starts them all in sequential order, so the more CPUs you have, the longer it takes to boot up the system. As Arjan van de Ven recently pointed out, this could add up to multiple minutes of additional boot time on some large SGI systems in use today.

To solve this, Arjan posted a patch so that only the first CPU – the one responsible for bootup itself – would start first, and the remaining CPUs would start simultaneously. On his personal laptop, the patch cut boot time by 40%.

Ingo Molnár was highly impressed, and the discussion immediately turned toward additional speed improvements that could be possible in the future. For example, Arjan's patch included a slight delay at the start of bringing up each CPU, which he said was an important part of achieving the speedup. Without that delay, the CPU hotplug lock would be completely nailed down as the CPUs booted, and none of the other initialization code would be able to use it.

By adding the delay, Arjan was able to share the lock sufficiently to allow everything to steam forward without throttling. Ingo didn't like this use of "magic delays" that might work fine on one system, but could amount to just a waste of time on another. He felt that if a performance problem was associated with hogging the hotplug lock, they should first figure out the real nature of that problem and solve it appropriately, so the fix would work on all systems.

Linus Torvalds suggested adding a new, finer grained per-CPU lock that would allow them to drop the global hotplug lock earlier in the process and let the rest of the code use the lock as usual, while the CPUs only held

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 resources they actually needed for bootup. So, that would be one additional speedup that could be added to Arjan's patch. Another possibility, as Arjan pointed out in his initial announcement, would be to start userspace as soon as the first CPU was ready and allow the remaining CPUs to join in the fun as they each finished their own initialization. This could result in a truly massive reduction in boot time, but Arjan didn't want to put it into his initial patch because it had the potential to wreak havoc on kernel code throughout the source tree.

Ingo agreed that caution was necessary and that a lot of unpredictable breakage would probably occur as the change was introduced, requiring careful testing and fixing each revealed bug as it emerged. This does seem like the kind of project that will make it into the kernel sooner rather than later, especially given the 40% speedup just from Arjan's initial patch and the promise of even greater speedups to come. But as Arjan and Ingo have said, some features may take some time to test thoroughly before making their way into the official kernel tree.

Columns, Shmollums

For years, it's been part of kernel development policy to restrict the source code to 80 characters per line. This dates back to the use of the old text-mode terminals that many Linux users of today have probably never seen. However, that's why xterms start up with 80 columns by default – to be compatible with those TTYs of yesteryear – and that's why the kernel source code has been limited to 80 characters per line, because it assumed the lowest common denominator of kernel developer would be restricted to coding on those old TTYs.

Recently, we got to see what would happen when someone tried to change this. Ingo Molnár recently pointed out that 80-column terminals have not been used by kernel developers for years and years, and that, as far as he knew, he himself was the last one to use an old-style VGA text console. He pointed out that kernel developers had been going through some extravagant hoops to keep their code crammed within 80 columns, including making some very ugly line-breaking decisions that messed up the source tree and made the code harder to read. These shenanigans took extra time to produce and required extra time to interpret on the part of the kernel developers reviewing those patches. It was a mess! Ingo said:

Let's increase the limit to 100 cols – this is a nice round limit, and it also happens to match with most developer xterm sizes. Code that goes over 100 cols for no good reasons will be arguably something worth fixing. (100 cols is also arguably closer to various brain limits such as vision of field and resolution restrictions, so we'll likely not have to increase this limit for a couple of million years, for all retro human genome users.)

There were some general murmurs of assent, and some proposals of 96 columns instead of 100, because 96 is a multiple of eight and therefore, you know, better. But then Linus Torvalds torpedoed the entire concept, saying:

*The problem is not the 80 columns, it's that damn patch-check script that warns about people *occasionally* going over 80 columns.*

*But usually it's better to have the *occasional* 80+ column line, than try to split it up. So we do have lines that are longer than 80 columns, but that's not because 100 columns is ok – it's because 80+ columns is better than the alternative.*

*So it's a trade-off. Thinking that there is a hard limit is the problem. And extending that hard limit (and thinking that it's "ok" to be over 80 columns) is *also* a problem.*

So no, 100-char columns are not ok.

And that was that. ■■■