Zack's Kernel News

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

By Zack Brown

3.0 Fallout

Linus Torvalds finally pulled the trigger on 3.0, setting off a vast array of minor issues in a kind of "Y2K event" that's still going on.

For one thing, the Git tree itself, where the kernel sources are housed, had "2.6" in the title. So, shortly after sending out the 3.0 release, Linus changed the Git URL to `git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git`. The old URL might still work for a while, but if you're keeping up with the source tree, you should probably update the URL in your `.git/config` file.

Then, as a ripple effect, the kernel.org admins didn't update the homepage to show the new URL, so it was broken, and Vitaliy Ivanov had to remind them to fix it.

Meanwhile, Jesper Juhl pointed out that the build scripts still expect a three-number version number, with a fourth available for the stable series guys, whereas the kernel would now have only a two-number version number, with a third available for the stable series guys. Traditionally, the third number had been called EXTRAVERSION, and the stable series number had been called SUBLEVEL. Jesper asked which of these two (if any) were going away, so he could start updating the scripts. Randy Dunlap also wanted to know, because he'd have to fix some scripts himself, depending on the answer.

Linus didn't have a clear answer, partly because he thought that actually getting the kernel to recognize itself as a two-number version might be very messy to implement, and that just leaving it as a three-number version internally might be the easiest way. But this doesn't really help the folks who are trying to maintain scripts understand how the version numbers will actually work in practice.

The problem also goes deeper than these inconveniences, as Andi Kleen pointed out. He had programs – some of them existing in binary form only – that broke under Linux 3.0, just because of the version number.

His ugly, horrifying solution was to patch the kernel so it would still report a "2.6.40" version number, with a special string appended to indicate the additional version numbers in the 3.x series. Andi said that in spite of its disgustingness, this seemed to be the best workaround he could find, and it seemed important because maintaining backwards compatibility was a high priority always. It's unclear whether his patch will be incorporated into the kernel, but it certainly does highlight a real problem posed by the transition to the 3.x versioning scheme.

New OpenRISC Architecture

Jonas Bonn submitted code for the OpenRISC architecture to be included in one of the early 3.x kernels. The project is part of opencores.org and is housed at openrisc.net. The idea is to create an open source hardware design for a CPU. After a period of languishing without much work going on, the OpenRISC project has been revived, and during the past year, the ranks of its developers have swelled from five to 25. After tracking the kernel with their own port since 2.6.35, they feel ready to have their code exist in the upstream source tree as a legitimate architecture.

As it happens, their code was accepted by Linus Torvalds, after a bit of a catch-22. Apparently, the OpenRISC code wouldn't build successfully without a modules patch from Rusty Russell, and Linus didn't want to accept the OpenRISC code if it didn't build;. However, Rusty had been waiting until the OpenRISC code had been accepted before submitting his modules patch because his patch would be much more trivial that way.

So, they untangled that loop, and Linus accepted first the modules patch and then the OpenRISC patch. Now, it looks as if OpenRISC will be a part of the 3.1 kernel release.

New Architecture for Texas Instruments Chips

Mark Salter posted a patch to port Linux to a new architecture – the C64x digital signal processors from Texas Instruments. This single-core and multicore family of processors lacks some hardware elements common to other CPUs. For example, the multicore C64x processors don't support cache coherency (i.e., providing participating cores with predictably similar data when they query the same location in a shared data store). Without cache coherency, supporting an SMP operating system becomes more challenging.

But, this wouldn't prevent the code from being included in the kernel. Linux runs on plenty of relatively simple processors. But, as Mark pointed out in his announcement, al-

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**.

though the C64x port has been around in one form or another since Linux 2.6.13, there has never been a port of GCC that would run on it, until now. Without GCC, getting a full system up and running on these chips would be problematic even with a functioning kernel.

The port of GCC to this architecture is still very much in the "early access" stage of development, but it has apparently become functional enough that Mark felt the time was right to submit the C64x port for inclusion in the main kernel tree and set up a developer wiki at linux-c6x.org. Developers from Texas Instruments, he said, were also ready to address any issues the Linux folks might have with the port as it currently stood.

At this point, a number of developers, including Randy Dunlap, Milton Miller, and Arnd Bergmann, descended on the project and pointed out some issues standing between Mark's code and inclusion into the kernel tree. These issues ranged from cosmetic things, like which directory to house various files, to deeper and more structural issues, such as rewriting all the board files to be devicetree source (DTS) files, as well as a lot of other issues.

Mark seemed to keep up with all these suggestions fairly well, but it's unclear how long it will take to implement most of them. It does seem clear that the C64x port will have to go through several more iterations before it'll be ready for inclusion in the main tree.

Framebuffer Advancements

Florian Tobias Schandinat has posted code to allow multiple simultaneous visible consoles in the framebuffer. This wouldn't work with graphical applications – only console shells. Still, it's a step in a fun direction.

The Linux framebuffer is an attempt to put control over the graphical displays of a given system back into the hands of the kernel. The standard approach is to use the X Window System, which takes control of a lot of the graphical hardware that would more naturally be the purview of the kernel itself. Typically, the kernel is the only entity on the system that controls hardware resources, divvying them up and dishing them out to the applications that want access. This allows all of the many applications each to use a bit of RAM without having to control all available RAM themselves. Pretty much all hardware resources are the same. Various applications want control of a given resource, but the kernel decides which application gets control over which resource and for how long.

With graphics, it's a different story. In the case of graphics hardware, the kernel hands basically all control over to the X Window System, which in turn handles the resource allocation for all the various applications.

The Linux framebuffer is intended to give control over graphics hardware back to the kernel, where it belongs. But, it has never developed the same power as the X Window System – partly because it's just hard to code all the controls for all the different pieces of graphics hardware in the world, but also because the X Window System does a lot more than simply control the graphics hardware: It acts as a networked communication protocol between the applications, providing tremendous flexibility, as well as a set of APIs that graphics software has relied on for decades.

To truly replace the X Window System, Linux would not only have to provide solid controls for the graphics hardware, but it also would have to emulate all the APIs provided by the X Window System. Effectively, it would have to incorporate all of X into the kernel source tree itself. This would pose a lot of technical problems, as well as the political problems of attempting to change the way the X developers do their work. It would also probably result in converting an actively developed and maintained system (the X Window System) into a pile of code that no one would be actively maintaining or developing at all.

Still, the Linux framebuffer represents the idealistic dream that one day, if the proper features ever were incorporated into it, the graphical software packages out in the world might eventually be ported over, and there would be no more need for the large and unwieldy X Window System.