Zack's Kernel News

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

By Zack Brown

Android in the Main Line

Anyone who has an Android device is running Linux, whether they know it or not. But it's a version of Linux that has been heavily modified by Google and others and is undergoing continuing development. Tim Bird recently announced the Android Mainlining Project [1]. Its purpose is to incorporate the features of Android versions of Linux into the main kernel sources so that Linux may one day run unmodified on Android devices.

With the recent discovery of Carrier IQ software secretly tracking the behavior of many millions of users, the need to retain control of one's own hardware could not be more pronounced. Although Carrier IQ is not restricted to Android systems, it has affected many Android users. If device providers will only address the issue of user privacy when they get caught violating it on a massive scale, then the onus must return to us, to secure it for ourselves.

But the main motivation for Tim's project is not to respond to the threat of things like Carrier IQ – it's to port Linux to another cool hardware platform. And this benefits everyone for the same reason that other ports are better than outright forks. The more elements of the Android OS that are incorporated back into the mainline kernel, the less effort Google must expend maintaining their own sets of patches and the greater the ability of regular users to play on new toys. Google knew and approved of the fact that this would be the most likely path of development before they started the project, or they wouldn't have selected a GPL'd project like Linux to be their codebase.

Cleaning Up the VFS

Al Viro, patron saint of the VFS (virtual filesystem), is cleaning up some strictly internal VFS code, so the rest of the kernel doesn't have to see a bunch of data that isn't relevant to it.

Specifically, the vfsmount structure has accumulated a big pile of data that isn't needed anywhere in the main kernel. In fact, most of it isn't even needed in the bulk of the VFS itself. Al's plan was to gut the entire structure, set up a new structure entirely private to the VFS, embed the old structure within the new, and have a beer.

In fact, he'd already made the changes and sent them to the linux-next tree, where he hoped fresh eyes would confirm they were working properly. Part of the reasoning behind how he was organizing the structures was to avoid requiring massive changes to filesystem code.

For example, Matthew Wilcox suggested just making the vfsmount structure entirely private to the VFS and providing external functions so the rest of the kernel could access just the data that was relevant to it. But as Al pointed out, that would require changing all the code that used vfsmount. Another problem was that non-inlined function calls at such a speed-sensitive area of the kernel might significantly slow down normal operations.

But, he added, if Matthew's seemingly cleaner solution ever did become feasible, Al would prefer it as well. His current design was largely influenced by the desire not to have to inflict too much change on the kernel in one fell swoop.

Meanwhile, James Morris tried out Al's patches and found them to work fine. Nothing is directly user-facing in any of this stuff; it's all under the hood. In fact it's underneath hoods that sit beneath other hoods that are under the hood. But Al does amazing stuff, like making it possible for all of us to use our hard drives, so it's nice to say something about that once in a while.

New Xen Port; Compatibility Discussion

Stefano Stabellini announced that he and a number of other folks had ported Xen to the Cortez-A15 processor – a powerful ARM chip for use mainly in embedded devices. The port, as Stefano said, is "nascent," but in his announcement, he said it had already successfully booted a Linux 3.0 virtual machine and gotten a shell prompt.

Arnd Bergmann liked the work and suggested that some effort be made to align the support of Xen and KVM, so the same kernels would be able to run successfully in each hypervisor implementation. Stefano agreed this would be best and also explained that their code made only minimal changes to the core kernel sources. The discussion descended into a lot of technical details, with a good bit of focus on making Xen and KVM play nicely – or at least play similarly.

Virtualization is pretty neat in general. It provides a layer of security for the underlying

INFO

[1] Android Mainlining Project: *http://elinux.org/Android_Mainlining_Project*

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**.

OS that can recover more easily from security problems encountered on the virtualized systems it hosts. Also, under some circumstances, running multiple virtualized systems on a single piece of hardware can result in using all the resources of that hardware more efficiently: Sometimes a networked solution makes more sense than trying to cram everything onto a single system.

In the old days, running software written for a different OS under Linux required things like dosemu and Wine, which only partially worked and had a lot of headaches associated with them. Nowadays, running Microsoft OS and others under Linux has become almost trivial.

Display Driver Madness

Tomi Valkeinen, the OMAP display driver author, suggested creating a common display framework that all the display drivers could make use of. Something that would centralize all the heavy lifting, and make it trivial – or at least much easier – to write a display driver for each new device that cropped up.

His own proposal for how to do this was understandably OMAP-centric. Keith Packard pointed out that Jesse Barnes had put together a similar proposal that was more DRM-centric.

Tomi felt that the DRM approach left out things like touch screens on embedded devices, and it didn't seem to take into account framebuffer requirements either. Keith thought that most problems could be overcome with DRM, although he did affirm, "backlight remains a mess in the desktop world; with many custom backlight drivers along with generic ACPI and then per-video-device drivers as well."

Alan Cox seemed to favor the DRM approach as well, suggesting that framebuffer would eventually just be a legacy feature that could sit on top of DRM.

Florian Tobias Schandinat didn't like any of the proposals; in particular, he said, "I'll never accept any change to the fb infrastructure that requires DRM." Later he added that he was "not against taking DRM's current implementation as a base but the steps required to make it generally acceptable would be to split it off, probably as a standalone module and strip all DRM specific things off."

He also added, "I think it's too late to really fix this thing. We now have 3 APIs in the kernel that have to be kept. Probably the best we can do now is figure out how we can reduce code duplication and do extensions to those APIs in a way that they are compatible with each other or completely independent and can be used across the APIs."

But Alan suggested that the framebuffer code was on the way out. He said, "I think it comes down to 'when nobody is using the old fb drivers they can drop into staging and oblivion'. Right now the fb layer is essentially compatibility glue on most modern x86 platforms." In spite of the acrimony between the various groups, the discussion continued. It gradually seemed to become clear to the participants that creating a single common display infrastructure was not yet a feasible option, given the history and the existing competing systems.

A better interim step, everyone seemed to agree, would be to try to make the existing display systems more compatible with each other. This offered several benefits, including allowing the various competing systems to continue to develop somewhat independently, while ultimately leading in a direction that, in the future, might naturally suggest how the truly correct solution to the dilemma might develop. For the moment, it seems, Tomi's and others' dreams of a unified display driver will have to wait for the political differences to resolve and for the existing technical differences to begin to converge. Ultimately, though, it does seem as though the developers have a strong motivation to work out their differences because it would potentially result in a great simplification of code and a much cleaner support process for future hardware.