ZACK'S KERNEL NEWS

Speedier Driver Merges

Recently, Linus Torvalds reduced the hurdles required to get code into the kernel. When Linux first entered the scene, one of Linus's main priorities was to encourage contributions. To that end, he made a point of being responsive to any patches that came across the group, accepting many and even publishing handcrafted statistics about the patches received. As contributions increased in the late 1990s, his responsiveness dwindled, and he became more likely to drop patches on the floor if he didn't like them. Linus also began insisting that the design and algorithms be beautiful and that different parts of the kernel communicated in the ways natural to them.

As Linus adopted the "stable" and "development" kernel trees, he maintained his insistence on good taste, but he became even more strict during the "stable" cycle. With the overt structuring of kernel code submission into a hierarchy of maintainers and "lieutenants," Linus began to create a culture of adherence to his coding preferences – wherein other people who understood his tastes could act as gatekeepers – in addition to doing the technical work of coding and reviewing code for bugs. With his use of BitKeeper and the git, Linus's culture of "coding taste" could be further distributed to particular projects working in isolation, in which the individual contributors could review each other's work without it first having to be part of the main kernel tree.

During the 2.6 tree, Linus abandoned the somewhat uncomfortable swings between stable and development trees. He then reinstituted a set of micro-forks for stable development, in which the main 2.6 tree never left the development phase and each release spawned a new stable fork, just for bug fixes. The decision to abandon the original stable/development cycles marked a time of rethinking a variety of problems. One of the main justifications for the change was that the Linux distros always layered their own specialized patches on top of the official kernel releases. This made at least some of the efforts at stability somewhat moot, because the distributions often would use cutting-edge patches that could not have been as thoroughly tested and reviewed as the official stable kernel.

Because the onus of providing true stability would always fall on the Linux distributions, Linus formally imposed the obligation on them and removed it from his primary development work. That decision indicated a new approach to kernel development – one that did not abandon the need to stabilize aspects of the kernel but that did put a higher focus on letting contributors cut loose a bit.

Now, Linus has started accepting driver patches in which the drivers contain obvious problems. Recently, Adrian Bunk complained about one driver submission that was accepted into the kernel despite having more than 250 "checkpatch" errors and more than 2,000 warnings. In the past, these problems needed to be fixed before a driver would be accepted into the tree. Part of Linus's justification for the change in policy is the acknowledgment that code is much more likely to be tested and fixed if in the official tree than out of it.

Also, the assumption is that average users rely on their distribution kernel, rather than the official release, whereas the kernel developers are the primary users of the official release and can better handle uglier, less polished drivers.

Linus points out that this new policy targets driver code, which is by definition peripheral to the main body of code. For kernel internals, presumably somewhat higher standards for what will be allowed to go in still exist. Driver code tends to stand alone and not interfere with any other parts of the kernel.

Linus says it's important that drivers are well tested before going into the kernel. He's not so concerned with how the code looks, but he does want to make sure that it works and doesn't break too badly or cost users their data.

This new direction does not simplify life for Adrian Bunk, who often sifts through large quantities of kernel code to remove bad and ugly wreckage. In fact, identifying code to remove from the kernel might become significantly more difficult as less clean code is merged and the barrier to entry becomes lower. On the other hand, Greg Kroah-Hartman, Jeff Garzik, and Arjan van de Ven felt the new direction would improve the kernel. With no major outcry, even Adrian didn't seem too concerned, and the discussion shifted to how to fix the "checkpatch" script independently of this new policy.

Undoubtedly, a lot more driver code will be going into the kernel, and the "stable team" and distributions will continue to ensure that regular users have a good experience. Also, there will be further changes in how the kernel is developed in the future, adjusting for whatever problems emerge out of this new approach.

The Linux kernel mailing list comprises the core of Linux development activities. Traffic volumes are immense, often reaching ten thousand messages in a given 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.
Our regular monthly column keeps you abreast of the latest discussions and decisions, selected and summarized by Zack. Zack has been publishing a weekly online digest, the Kernel Traffic newsletter for over five years now. Even reading Kernel Traffic alone can be a time consuming task.
Linux Magazine now provides you with the quintessence of Linux Kernel activities, straight from the horse's mouth.

Watching Subsystem Merges

Now that it's a lot easier to get code into the kernel, Andrew Morton had some misgivings and wanted to make sure that everything didn't break. His original idea was to create a linux-next git tree, wherein patches could pass through on their way to Linus, but this idea quickly changed to focus on subsystems only.

A big problem for subsystems was the number of merge conflicts. Every time a new merge window opened into Linus's tree, everyone had to scramble around trying to resolve all the subsystem conflicts. Andrew's idea involved a volunteer keeping a running tree of all the subsystem code, thus helping identify merge problems, build problems, and possibly even run-time problems if users could be solicited to test it. However, to create such a linux-next tree, the merge practices of the subsystem maintainers must be significantly altered.

Stephen Rothwell announced the creation of a linux-next tree for subsystems, as well as the *linux-next@vger.kernel.org* mailing list for discussion about the new tree. Stephen invited all subsystem maintainers to send him the addresses of their git trees or quilt series so that he automatically can pull from all these sources on a daily basis. Any tree with merge conflicts would be dropped from that day's pull, and the maintainer of that subsystem would be informed automatically via email. Stephen also would perform automated builds for as many architectures as possible. Any subsystems that failed to build likewise would be dropped from the day's tree.

Stephen was not the only volunteer: Frank Seidel, Ann Davis, and Harvey Harrison also volunteered to maintain the tree. Andrew ultimately selected Stephen, but Stephen hopes the other volunteers are willing to help as needed.

James Bottomley also has been maintaining a tree similar to linux-next, which he handed off to Stephen. Andrew knew about James's work already, but there were significant differences that made it insufficient to solve the problems Andrew wanted to solve. For example, James did not conduct automated build tests. James's tree also pulled in only about 46 of the estimated 80 subsystems that Andrew wanted to include in linux-next.

The linux-next idea might lead to many more changes in the way code is tested and submitted. Subsystem maintainers need to get control over how they make patches and what parts of the kernel they touch. Problems uncovered by linux-next might end up having default responses, such as patches being rejected if they haven't been in the tree long enough to be tested. Ultimately, it could be that Linus's decision to loosen the restrictions on what code gets into the main tree results in tighter restrictions by the people submitting the code, such as setting up things like linux-next.

Clearly, the kernel development process is continuing to undergo the major changes inaugurated by the 2.6 release.

1 2 3 4 5 6

Easy MPI Debugging

PGDBG is an easy-to-use graphical parallel MPI, OpenMP and hybrid MPI+OpenMP debugger for Linux and Windows clusters. PGI CDK compilers and tools are available directly from most cluster suppliers. Take a free test drive today at www.pgroup.com/reasons

PGI CDK® Cluster Development Kit®