

MARC E. FIUCZYNSKI

better tools for kernel evolution, please!



Dr. Marc E. Fiuczynski is a research scientist in the Computer Science department at Princeton University and a member of the PlanetLab R&D team.

■ mef@cs.princeton.edu

THIS ARTICLE SUMMARIZES A FORAY into the land of Linux, revealing the soft underbelly of the animal called kernel. This voracious animal is eagerly eating up others, but how much longer can it do so before its belly bursts?

Analogies aside, to many users Linux is great—it is cheap (free in many cases) and often works quite well for the intended purposes. Generally, such users treat Linux as a black box by using an unmodified distribution such as Fedora, SuSE, etc. For users for whom a conventional distribution is insufficient, Linux, as one of the quintessential open source projects, offers unlimited flexibility for customization. Moreover, various kernel extensions released as patch sets, or simply patches, offer unique and useful features not available in the mainline version of the kernel.

Unfortunately, maintaining a customized kernel can be challenging, particularly when it is necessary to keep track of security updates, bug fixes, and general enhancements to the mainline kernel. The reason is that externally developed kernel extensions are often available as patches that typically apply only to the vanilla mainline kernel. While these patches sometimes apply to the latest mainline kernel, often they do not and so require integration work.

While such integration (merging) may be trivial at times, it can quickly get out of hand. Why? Even when these patches apply cleanly to the latest release from kernel.org, they often do not to the latest releases from distributions such as Fedora or SuSE. These distributions, to set themselves apart from others, introduce their own value-added modifications to the kernel and, in some cases, tend to be ahead of the stable 2.6 release by integrating features from the unstable kernel.org releases. Consequently, for those using a customized kernel in a production setting, the job of keeping on top with the latest and greatest kernel can become a tedious one that is pure overhead.

This has been my experience while maintaining the Linux kernel used for PlanetLab (<http://www.planet-lab.org>). PlanetLab is a geographically distributed overlay platform designed to support the deployment and evaluation of planetary-scale network services. As of August 2005, it consists of over 580 machines at 275 sites in 30 countries, and it has supported over 450 research projects. PlanetLab continues to grow at a rate of approximately five sites and 10 machines per month. Each machine runs a customized version of Linux to support a “virtual private server” (VPS) model, which is used to isolate separate research projects running on a single machine from

each other. At one point the kernel for PlanetLab was modified by 28 patches—both externally developed and homegrown. The kernel was so tedious to maintain that at one point it lagged eight minor releases behind the 2.4 mainline kernel release. Upon switching to the 2.6 kernel release, we focused on reducing the patch count to a minimum with the goal of keeping close to the latest mainline kernel release. Nonetheless, our kernel still uses several large patches to support performance isolation and namespace isolation for said VPS support.

At this point, you, likely a Linux user, may be thinking, “Hey man, quit the whining. This problem only affects a small group that have a vested interest in maintaining a highly customized kernel.” I humbly disagree. With the rampant success of Linux, this “small” group is growing by leaps and bounds. Besides well-known players like RedHat, SuSE, and IBM, there is a growing number of corporations (e.g., PalmSource, Wind River Systems, Panasonic, NEC, NTT DoCoMo, to name just a few) and government agencies worldwide that are putting a tremendous amount of effort into the Linux kernel. The happy days of the Linux phenomenon probably are numbered. Why? Rather than working toward the general good, corporate kernel programmers will try as hard as possible to push their “modifications” into the Linux kernel in pursuit of their own agendas. It is just a matter of time before the soft underbelly bursts.

When this happens, significant infighting will ensue, leading to fragmentation or who knows what—nothing that’s good for any community. What can be done about this? Will Linus Torvalds keep everything under control so that the rest of us can continue to obliviously toil along with Linux? No! To address this problem it is not prudent to rely on a bazaar, cabala, cathedral, benevolent pope, or any sociopolitical model. No one truly understands or can predict the long-term outcome of such models. As engineers we have two choices: (1) blissfully ignore the problem and hope the day of reckoning will never arrive, or (2) turn it into a technological problem that we can attempt to solve—i.e., find better ways to evolve the kernel proper with kernel extensions.

My position paper titled “patch(1) Considered Harmful,” presented at this year’s Workshop on Hot Topics in Operating Systems, outlines why patch is harmful for the evolution of the kernel (see the HotOS summaries in this issue of *login*:). In a nutshell, patch is good for localized fixes but bad for kernel extensions that introduce changes which crosscut many files and/or functions. Analysis of patches revealed that kernel extensions can easily cover a hundred existing kernel files, even though it represents a logical unit, expressing a single crosscutting concern. The crux of the problem is that with today’s tools (patch, cvs, bk,

etc.) these changes need to be integrated into the kernel proper, hence driving kernel programmers to achieve the ultimate integration of their modification into the mainline kernel managed by Torvalds. To avoid this problem altogether, I am working with a team spread across New York University, University of Victoria, and Princeton University on a toolkit called C4, for CrossCutting C Code.

As part of our work on C4, our analysis of various patch sets reveals that kernel extensions primarily make intramodule changes: a coherent collection of modifications encapsulated within an existing module. These changes may modify many of the functions within a particular module, but they do not change the externally visible interface. Clients of the module do not need to change their code and usage patterns. Please note the loose use of the word “module” to mean any collection of components with a well-defined external interface, including kernel subsystems. In particular, a module is not limited to a single kernel source file.

Some kernel extensions also make intermodule changes: modifications that change intermodule interfaces, or the visible semantics of an existing interface, in fundamental ways. For instance, modifications of a function’s type or the field makeup of a data structure (e.g., adding, deleting, or changing the type of a field) are intermodule changes. Making such changes can have far-reaching consequences: it can require updating all modules that directly use the changed code. This is prohibitive when the interface changes are in the kernel proper or in the generic device driver framework and trigger corresponding changes in specific device drivers—there might be hundreds. Doing such updates manually is error-prone and time-consuming.

Recognizing that intramodule and intermodule changes are common to new kernel extensions for Linux, our approach with C4 is to make them part of the kernel’s architecture by leveraging aspect-oriented programming (AOP) techniques. Whereas object-oriented programming provides linguistic mechanisms for structuring self-contained units of code, AOP provides linguistic mechanisms for structuring concerns that naturally cut across primary modules of a system. More specifically, our approach is to express intramodule changes as semantic patches using aspects, which provide a language-supported methodology for integrating crosscutting concerns with a program.

The benefits of aspects are twofold. First, they provide a well-defined specification of domain-specific features that is separate from baseline functionality, yet can be automatically integrated with the kernel. Second, we believe that aspects will enable tools to perform automatic analysis of the implications of

composing several crosscutting concerns and therefore help identify true semantic conflicts, as opposed to the line-by-line conflicts identified by patch.

Thus, a toolkit like C4 provides an opportunity to the growing number of well-funded (and under-funded) kernel developers to have their code included with the latest mainline kernel. The automatic integration of kernel extensions at build time will leave the kernel proper largely unencumbered. In contrast, the existing model litters the kernel proper with unnecessary #include statements and code fragments that do nothing until the corresponding CONFIG option is enabled. More important, Torvalds et al. no longer will need to decide what kernel extensions make it onto the mainline kernel.org release. Rather, since the

extensions will be part of the mainline release, the kernel.org folks can declare a preferred composition of these extensions, but others will be truly free to choose what they prefer—i.e., no longer will there be a need to patch in code and resolve merge conflicts between separate extensions.

Building a toolkit to solve this problem is a tall order, and the C4 toolkit is just one approach. There are undoubtedly others. For more information and an initial release of the C4 toolkit, please see <http://c4.cs.princeton.edu>.

Thanks to USENIX Supporting Members

ADDISON-WESLEY/PRENTICE HALL PTR

AMD

ASIAN DEVELOPMENT BANK

CAMBRIDGE COMPUTER SERVICES, INC.

DELMAR LEARNING

ELECTRONIC FRONTIER FOUNDATION

ELI RESEARCH

HEWLETT-PACKARD

IBM

INTEL

INTERHACK

THE MEASUREMENT FACTORY

MICROSOFT RESEARCH

NETAPP

ORACLE

OSDL

PERFECT ORDER

RAYTHEON

RIPE NCC

SPLUNK

SUN MICROSYSTEMS, INC.

TAOS

TELLME NETWORKS

UUNET TECHNOLOGIES, INC.

It is with the generous financial support of our supporting members that USENIX is able to fulfill its mission to:

- Foster technical excellence and innovation
- Support and disseminate research with a practical bias
- Provide a neutral forum for discussion of technical issues
- Encourage computing outreach into the community at large

We encourage your organization to become a supporting member. Send email to Catherine Allman, Sales Director, sales@usenix.org, or phone her at 510-528-8649 extension 32. For more information about memberships, see <http://www.usenix.org/membership/classes.html>.