EDITORIAL

Musings

RIK FARROW

Rik is the editor of *;login:*.
rik@usenix.org

I've often rambled on about the future of operating systems, imagining something completely different and new. Of course, there are loads of practical issues with that path, like the inability to run any existing software on the spanking new OS. And it turns out there are still things about existing operating systems that can surprise me.

I speculated that a future operating system might not look at all like Linux, today's favorite OS for servers and for OS research projects too. Linux is large, complex, and difficult when it comes to incorporating large changes into it because of its history and design. While the BSD kernels are more modularly designed, they are less popular, and thus not as interesting, or even as well-known. And both are enormous, with many millions of lines of code. While Minix3 is much smaller and actually takes a new approach, it too suffers from the "not as popular as Linux" issue, like the BSDs.

I've watched the OS space for a while, curious to see if some of the less popular directions taken will pick up a lot of interest. And that interest generally comes from providing features that users, whether they are running servers in some cluster or researchers looking to add the next neat feature or improvement, just can't live without.

Size Is Not Everything

Today's OSes are huge. When I worked with Morrow Designs in the '80s, I actually put together a set of two, double-density, floppy disks that contained a bootable kernel and utilities you needed to recover an unbootable system. That was a total of less than 800 kilobytes of code for the equivalent of a rescue CD, which should sound ridiculous in this day and age. But is it really?

The Internet of Things (IoT) already includes inexpensive devices, and that means slower CPUs, small memories, and sometimes relatively generous amounts of flash. With small memories, these devices won't be booting a generic kernel but one trimmed down to the bare essentials. In one sense, that's easy enough: You can build a kernel without support for file systems and devices you will never use in a diskless system on chip (SoC) device. Popular examples of this include the Raspberry Pi and the BeagleBone Black.

But even these devices are overkill for many IoT applications. Another popular example is the Arduino family, which does not run *nix but may still include networking. Even simpler (and slower with less memory) are the Peripheral Interface Controllers (PICs), favored not just by hobbyists but also by device designers. These devices have really tiny amounts of RAM (really just RAM as registers), yet are more than adequate for many household and industrial devices. They do not run *nix, or even what could ever be called an operating system.

Let's head to the opposite extreme and consider IBM's Sequoia (Blue Gene/Q) that was installed at Lawrence Livermore National Labs in 2011. Like others in this series, the Sequoia's compute nodes (some 98,000 of them) run the Compute Node Kernel (CNK). The CNK operating system is just 5000 lines of C++, just enough to communicate with I/O nodes and launch applications that have been compiled just for the CNK environment. The concept behind CNK is simple: the bare minimum of memory and processing required so that

most of the CPU and memory can be devoted to computation. And it works, as the Sequoia was the world's fastest computer for a while, as well as using 37% less energy than the computer (BG/K) it replaced.

So, the Sequoia runs a more sophisticated version of what runs on Arduinos on its compute nodes. There is no memory management or thread scheduler: Applications are single threaded and run in physical rather than virtual memory.

Stripped Down

The same stripped down to bare essentials approach can be found in rump kernels. The brainchild of Antti Kantee, rump kernels provide just those portions of an operating system needed to run a single application in physical memory, with no scheduler. Kantee refactored the NetBSD kernel into a base and three modules that allow the rump kernel to support applications that can run on bare metal or on top of a hypervisor. Not that rump kernels are the only game in town: OSv, MirageOS, and Erlang-on-Xen all are designed to remove the need for a full operating system and its environment when running on top of a hypervisor.

There's yet another way to stop layering operating systems over a hypervisor operating system, and it has been around for many years. You may have heard of LXC, a project that has been used for years as a way of providing the illusion of having your own hosted system. With LXC, and related technology like Solaris Zones, there is only one operating system. LXC, or other container software, provides the illusion of being the master, root, of your own system, when what you really are running is a group of processes in a jail. Just as the BSD jail has evolved over time, so has the Linux container. James Bottomley discusses Linux containers in an article in this issue, and he and his co-author have left me feeling like real progress has been made in making containers both secure and efficient.

Still in the theme of "stripped down," but not related to operating systems, I had hoped to get Ben Treynor (Google) to write about the concept of the *error budget*. Treynor introduced this idea during his keynote at the first SREcon, and I will try to cover it concisely here. Imagine that you are running software-as-a-service (SaaS) on an immense scale, that you must do so efficiently (no operators, just skilled SREs) but do not want to violate your service level agreement of five 9s, or 99.999% uptime. At the same time, you continually need to update your client-facing software. Your error budget includes some tiny fraction of your total capacity for providing SaaS for testing. And the better job you do of testing, the further your error budget, that .001%, can stretch. Read Dan Klein's article and perhaps you will see how Google's approach to updating software fits into this concept of the error budget.

I still hope that Ben Treynor will have the time to write for us someday.

The Lineup

We begin this issue with two operating systems-related articles. When I met Kirill Korotaev (Parallels) during Linux FAST '14, I was already interested in Linux container technology. I caught up with Kirill during a break, and asked him to write about Linux containers. Kirill suggested James Bottomley, and James agreed to write, working with Pavel Emelyanov. They've produced both a history and an excellent description of Linux containers for this issue.

Greg Burd (Amazon) had suggested that I publish an article about rump kernels in 2013, but I didn't think the technology was ready. When Antti Kantee volunteered to write about rump kernels this summer, I took another look. Kantee actually wrote his PhD dissertation about refactoring the NetBSD kernel to support the concept of rump kernels: a method of supplying the parts of an OS you need for a particular application, and no more. He and Justin Cormack continue to work at making rump kernels easier to use, and their article in this issue explains the concept in detail.

I met Steve Muir during ATC '14. While Steve presented the paper, four other people from Comcast were involved in the research. Their goal was to create an in-memory database for a read-only service that could be transparently updated. Their use of Paxos as a means of managing updates between a hierarchy of servers got me interested, plus their software is open source.

A student of John Ousterhout, Diego Ongaro, presented a Best Paper at ATC '14, "In Search of an Understandable Consensus Algorithm," which the researchers offer as a replacement for Paxos. Although the subject is not covered in this issue, Raft is focused on applications like the one the Comcast people wrote, and on RAMCloud (of course). You can find the Ongaro paper on the USENIX Web site, as well as videos explaining how Raft works, by searching online.

Dan Klein, Dina Betser, and Mathew Monroe have written about the process they use within Google to push software updates. While the process is quite involved, I had heard about parts of it before Dan volunteered to write for *;login:* from various sources. And it both makes sense and realizes a cautious yet realistic approach to upgrading software without causing catastrophic failures—perhaps just small-scale ones within the error budget. Klein's article covers not only updating but also a further optimization that will make the process more efficient, involving less human interaction.

Andy Seely continues his series of columns about managing system administrators. In this contribution, Andy relates a set of three parables he uses as guides and stories he can share to motivate co-workers.

EDITORIAL

Musings

Charles Border and Kyrre Begnum introduce a workshop and a new journal. The Summit for Educators in System Administration had its first official meet at LISA '13, and will occur again at LISA14. The *Journal of Education in System Administration* (JESA) provides a mechanism for publishing research about educating system administrators year round.

Dilma Da Silva has written her second article about CRA-W, the organization devoted to helping woman PhD candidates in the fields of computer science and engineering. Dilma discusses the Grad Cohort, a yearly gathering of grad students and mentors focused on providing useful information about both completing grad school successfully and planning beyond grad school. And right now (late 2014) is the time to be making plans, and applying for support, to attend Grad Cohort 2015.

David Blank-Edelman claims that this time he is going to be totally practical about his chosen topic. I would claim David is always practical and pragmatic. David has been working on health checks for a small cluster of LDAP servers, and he takes us through both aspects of what a health check requires and Perl support for querying LDAP servers.

Dave Beazley considers Python's problems with paths. It's not so much that Python can't manipulate pathnames. It's just that the ways of doing so have been disjointed, involving multiple OS modules. Well, things have gotten more elegant with a new module, pathlib, available as part of Python 3.4.

Dave Josephsen continues on his mission of evangelizing for the proper design and use of monitoring systems. In this column, Dave rails against the arithmetic mean, showing just how badly the mean works when used to summarize/compress time series data. And, of course, Dave offers alternatives.

Dan Geer has written a concise article clearing up the confusion surrounding terms like false positive and true negative. Dan not only does this, but also provides an example for determining the most efficient ordering of tests for sensitivity and specificity.

Robert Ferrell, having recently retired from being a badge-carrying Fed (bet you didn't suspect that), has decided to poke fun at the Internet of Things. Even as people rush to connect their cars and thermostats up to the Internet, Robert points out that the security of these devices is about on par with that of the Internet—in 1994.

I've written a review of the new edition of the *Design and Implementation of the FreeBSD Operating System*. It's not the first time I've taken a look at similar volumes, as past USENIX president Kirk McKusick has been part of writing about BSD operating systems for over 20 years. This edition, the first in 10 years, contains several new chapters as well as much updated material.

Mark Lamourine, while technically a system administrator, continues to write excellent reviews of books on programming topics. This time, he covers books on when and how to use Bayesian statistics, understanding when refactoring an imperative program to use functional programming features can help, and an experimental work called the *Go Developer's Notebook*.

We have lots of summaries: ATC '14, HotCloud '14, HotStorage '14, WiAC '14, and ICAC '14. Most are incomplete, as there were too many sessions to cover and not enough volunteers—with the exception of the WiAC summary, which was thoroughly covered by Amy Yin. If you are planning on attending LISA14, and want to be certain a favorite session gets covered, contact me to volunteer.

The first time I attended the OSDI conference, I asked someone I knew why there weren't any papers about new OS designs. His answer was simple: It's hard. Designing a new OS takes many years and is also a risky endeavor. That's why I now look more closely at important but incremental changes, like unified container support for Linux, and at work like Kantee's, where he has converted a complete kernel into a more modular form. And, I continue to watch seL4, which just went open source (July 2014), Arrakis, and Minix3.