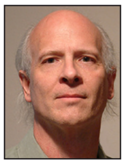
EDITORIAL

Musings

RIK FARROW



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

I was deeply disappointed by the operating systems class I took in 1978. An advanced CS class, the focus was on IBM mainframe architecture, and that appeared grossly inappropriate to me. By that time, the Apple II and the Altair 8080 had been out for a year, and it was obvious to me that the computers most people used would be changing.

The lab for the class used two Digital Equipment Corporation PDP 11/45s, and students were supposed to build an operating system, starting with the keyboard driver, proceeding to a file system, then the ability to load and execute code, all on a mini-computer with a very different architecture than the mainframe. Oh, and the mainframe didn't have a file system, and used Job Control Language for running programs.

In despair, I asked the teaching assistant if there wasn't something more appropriate to use as a way of understanding operating systems, and he said there wasn't. Keep in mind that AT&T had been licensing UNIX to universities for several years by then, a textbook had been written about 6th Edition UNIX, and that UNIX ran on DEC mini-computers.

I never finished that class. Competing for time on the lab systems with 200 other students, when all you could get was one-hour time slots, was too frustrating. I aced the other CS course I took that year and got a job working for a small embedded systems company, where I began learning about operating systems.

Computers have gotten a lot more complicated than they were in the seventies. I built my own computer, from a kit, in 1979. A couple of years later, I wrote my first C program, one that provided all of the file system features of CP/M [1], and the device driver, in two pages of code. With only 56 Kb of memory, having an intelligent floppy disk controller, one quite similar to the one in the DEC mini-computer, made the task simpler.

Today, Linux has more than 120 file systems, designed for different use cases. Device drivers have gotten more complex, and programmers now have gigabytes of memory to work with. Those choices are there because certain file systems perform much better for particular workloads.

Even if you are not a programmer, you still need to understand some operating systems basics. Caskey Dickson, co-chair of LISA17, has been teaching such a class at LISA, and there are several good books out about operating systems.

In this issue, we are featuring two freely available sources for learning about operating systems. The first is a three-section online book, with exercises and material for helping instructors, by Remzi Arpaci-Dusseau and Andrea Arpaci-Dusseau. Their material most closely follows what I've seen in OS textbooks, but their tone is conversational and much less daunting.

The second example comes from classes taught using FreeBSD. George V. Neville-Neil and Robert N. M. Watson developed this material for three styles of classes: those taught as college courses, one as a tutorial for BSD conference attendees, and a third as online videos. Their classes have more pragmatic focuses, with two versions actually using BeagleBone Black computers running FreeBSD and used for probing the internals of running systems. Their materials are open source under a BSD-style license.

Either of these would have been much better than using a textbook and having class lectures about a mainframe system that was over 10 years old in 1978 and soon to be usurped.

While there were certainly good jobs to be had as systems programmers, as almost anything you did on these computers required writing assembly language patches, there really weren't that many mainframes. Especially when you compare that to the revolution that was on the horizon.

The Lineup

I've already introduced the first two articles, resources for learning about operating systems, so I'll move on to the rest.

Kees Cook has written about security improvements to the Linux kernel. Kees works on the Kernel Self-Protection Project [2], and he describes a lot of the work that has already been done to make Linux kernels more difficult to exploit.

I have two interviews for this issue, both systems-related. Jeff Mogul has done many things in his career, and in this longer than usual interview, I begin by focusing on what Jeff worked on when the Internet, and later the Web, was young. We also discuss research labs and why we have CS proceedings instead of journals.

I talked with Amit Levy about TockOS. TockOS will replace TinyOS, both operating systems for very resource-limited embedded devices. Amit's interest in TockOS includes building a secure system, something the world of IoT desperately needs.

I discovered MarFS from a talk given at SNIA's SDC conference. Jeff Inman et al. explain how they built a nearly POSIX front end for a massively parallel, object file system back end. While their focus is on HPC, MarFS and the ideas illustrated by their system can certainly be applied to other large-scale and high performance storage systems. And the software they developed is available online.

Sergey Bratus and crew have written about how to parse input securely. If you've ever written any code, including shell scripts, you likely have noticed how much time you spend on parsing input. Yet mistakes in parsing input, that often mean accepting invalid input, lead to the majority of exploits we see in both programs and operating systems. The authors use their published work, where they replace the buggy code for an industrial control system, as examples as they explain how to do this correctly, as well as how coders usually do this poorly.

In the area of system administration and SRE, Lunney et al. cover the proper handling of postmortem action items. While postmortems are now recognized as important methods for improving the quality and stability of systems, Lunney et al. explain how they take advantage of the output of postmortems to drive corrective work.

The final article comes from research published at OSDI. Lion et al. were examining the performance of popular distributed systems, like Hadoop and Spark, looking at overhead. They discovered that a large proportion of the time spent running these applications was wasted on loading and interpreting Java classes every time another request was made. They produced HotTub, a version of the JVM, that caches warmed-up JVMs for reuse, improving the performance of HDFS by 21% and Spark by 33%.

David Beazley has written about what's new in Python 3.6. Hint: it's cool and not at all backwards-compatible with Python 2.

David N. Blank-Edelman pulls off a tour de force by creating a database from the output of `ls -lR`, then creates a Web page and Perl scripts that work with Google Charts, finally creating a spiffy chart showing the number of files created each month over many years.

Kelsey Hightower and Dave Josephsen decided not to write for this issue.

Dan Geer and Eric Jardine examine cybersecurity workload trends, using the NIST vulnerability workload and data providing estimates of the number of people working on remediating vulnerabilities to produce some trend lines.

Robert G. Ferrell has written about being totally truthful, and creates an example where withholding information has both good and bad effects.

Mark Lamourine has written three book reviews. The first two cover books about Angular 2, a framework for writing Web client applications. The third review is on the third edition of *The Practice of System and Network Administration*. I've reviewed bunnie Huang's book called *Hacking Hardware*.

During LISA16, someone asked me how I'd become so interested in security. I replied that security required that I understand programming, networking, system administration, file systems, and operating systems, and that I loved having one field cut across so many other areas of interest. Exploiting computers is definitely a form of hacking because the successful exploit requires seeing the system in a manner that the designers of that system didn't anticipate. Defending systems also means going "outside the box," although I have to concede that the attackers had, and still have, the upper hand.

While your work focus may not be security, understanding as much as you can about the operating systems that provide resources, and hopefully, security to your applications should only make your work easier.

References

[1] CP/M: https://en.wikipedia.org/wiki/CP/M.

[2] Kernel Self-Protection Project: https://kernsec.org/wiki/index.php/Kernel_Self_Protection_Project.