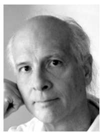
OPINION

Musings

RIK FARROW



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

Walking near where I live on a warm winter afternoon, it hit me. Layers. Everywhere I looked there were layers of rocks, leftovers from millions of years ago, now exposed by weathering. How apropos to many of the topics covered in this issue. There are layers in virtualization, layers in file systems, layers in networking, as well as pluses and minuses with having so many layers.

Layers are not necessarily a bad thing. I was once asked to create drop-down, cascading menus using a primitive drawing library. Popping open a menu of choices was fairly easy, as was drawing a second-level menu. I just created a bunch of text rectangles of the correct size, making certain they stacked perfectly. But I also needed to deal with saving the pixels that were present before I drew the menu and with replacing those pixels once a selection was made. Today, you would just use a library, a list of menu items, a corresponding list of functions to call, and everything would get taken care of. In other words, you would take advantage of higher layers of software that made a task that was once difficult easy to do.

Layers

Layers occur everywhere in computers. As I type this sentence, each keypress gets converted into a set of bits, sent over a serial link, received by a character device driver, passed through the line handling code, placed in a queue associated with a particular virtual TTY and copied to the program I am using, which then interprets the character and writes it to an X Window library routine, which eventually calls a kernel routine so that it can display some bits on my screen. And, of course, I am simplifying things quite a lot. But it is the many layers that make what appears to be, and really needs to be, a simple action appear to work in a trivial fashion.

Simple devices have fewer layers. Your microwave may have a menu of items it “knows” how to cook, displayed in funky letters on its display. On such simple systems there are few layers, and a lot of programming effort is required just to spell out BEEF. Move up to a smartphone, and even though the device is smaller, the computing power is immensely bigger, and the layers have grown as well. Then move onto a modern PC running Linux or Windows 7, and the number of layers grows even faster. You might think that a smartphone running Linux or Windows would have just as many layers as a desktop running the same OS, but you’d be wrong. The smartphone is more limited, with a simplified API that programmers are required to use.

Semantic File Systems

Instead of thinking about characters on a screen, let's consider file systems. At the hardware interface, the OS presents sector-sized blocks to devices along with directions about where to write the sector. Modern disk drives may ignore the location directions and just write the sector in the first free block available, keeping track of where the OS thinks the block resides. So even disks have their own layers—internal, hidden levels of indirection.

Now let's get really squirrelly and pop way up the stack to an operating system running within a VM. This virtualized OS "thinks" it is writing to a disk, but really it is sitting atop a hypervisor which may take the block write and convert it into a network write to some remote storage device. From the perspective of the virtualized OS, it is convenient not to have to consider what really happens when the OS writes a sector. But from other perspectives, blindly treating blocks as blocks wastes lots of information.

Including semantics in file systems is not a new idea. File metadata has provided some level of semantics in just about every file system, with the exception of mainframe OSes. But virtualization rips away the assumption that a block written on a disk includes some semantic information, because VMMs today are, for the most part, blind to this information.

Storage companies just love this. I really wondered why EMC bought VMware, until I realized just how much virtualization features, like migration, rely on SANs. And with the semantic information about what is being written lost, the opportunity to do clever things is greatly reduced. Sure, a smart filer can handle deduplication, as data is just data. But from a system administrator's perspective, the blocks on those filers are just blocks. They no longer represent anything meaningful. Instead, the amount of storage required increases.

The Lineup

During the enormous poster session at OSDI (75 posters!), Dutch Meyer managed to catch my eye. Perhaps it was because I knew Meyer from his work as a summarizer, but I think it was really because he and his co-authors are looking at the issue of layers in virtualization in their research. In their article they point out just how much is lost, and how much there is to gain, by preserving file semantics below the level of a VM.

I also met Rob Sherwood during OSDI. Sherwood presented a paper on FlowVisor, a prototype implementation of network slicing that relies on OpenFlow. FlowVisor allows new services to be tested on live networks by partitioning the network based on how traffic is switched. OpenFlow by itself stands to be a game-breaking technology for the operators of large clusters of systems.

During USENIX Security '10, David Barrera proposed sharing work he had done with Glenn Wurster and Paul Van Oorschot on improving a part of IPv6 that has security implications. In IPv6, the lower 48 bits of an address are, by default, the MAC address of the network interface. But that address is supposed to be unique. And that implies that an adversary could track the mobile devices as they move from one IPv6 network to another. Barrera shares their approach to fixing this issue, along with a very nice explanation of IPv6 host addresses.

Mona Attariyan and Jason Flynn (also met during OSDI) share their work on providing an automated way of solving configuration error problems. Their project involves statically tracing execution flow, then monitoring execution until an error occurs. They can revisit the execution, trying out different paths, until they determine which variables, identified with taint, were most likely to have caused the error. Very cool and useful work.

Josh Fiske shares his experience using Linux virtualization from his work at Clarkson University. Fiske takes advantage of layers, by automating the process of spinning up new VMs and configuring them, as well as installing and configuring a set of application packages.

Ole Tange shows off his own software project, GNU Parallel. GNU Parallel is a replacement for xargs with a focus on forking as many processes in parallel as desired, allowing you to take better advantage of multicore or multi-threaded systems. Tange has also designed GNU Parallel to avoid some weaknesses in how xargs processes its arguments, making it an excellent replacement.

David Blank-Edelman gets right into the theme of file systems by exploring some of the included Perl libraries for copying and renaming files. He also takes a look at CPAN modules that go well beyond the basics, such as using FTP, SFTP, SCP, and the wrappers for rsync.

Peter Galvin compares virtualization options in Solaris, AIX, and RHEL. Expanding on the comparison in his December 2010 column, Galvin explores the pluses and minuses of these three enterprise-ready operating systems. Not surprisingly, hardware support does make a difference here.

Dave Josephsen continues his exploration of Ganglia. In this column, Josephsen demonstrates how to write plug-ins in C for the data-collecting daemon, gmond. While writing C programs may not be something everyone feels comfortable with, for often repeated tasks on critical servers their performance cannot be beat. And, as Josephsen points out, using C means that other packages do not need to be installed for this trick to work.

Robert Ferrell explains how the threat of worms like Stuxnet requires us to think outside the box, or at least the comic book, to find new solutions.

Elizabeth Zwicky explains how she can review so many books each issue, tells us about her experience reading eBooks, then presents us with her views of three new books. I take a quick look at a book about building your own PCs, and I like what I see enough to order the recommended list of parts for my new desktop. Sam Stover waxes enthusiastic over a book about lock picking, a great hobby for any geek, as well as a useful skill for physical penetration testers.

This issue includes summaries from OSDI '10. LISA summaries were not complete when I turned this issue in for printing (really!), so they will be out in April 2011. We also have summaries from four workshops, including some excellent advice from the Diversity Workshop to anyone either in grad school or planning to work toward an advanced degree.

I can look out my office window and see the layers in the rock, similar to the Coconino Sandstone and Hermit Shale layers seen in the Grand Canyon [1]. If I move my chair a little, I can see basalt that capped the Mogollon Rim with hard rock from volcanic eruptions millions of years ago. These layers make for spectacular views, as well as supporting the local economy by attracting hordes of tourists.

Our computer systems are composed of many more layers, rapidly deposited over a period of just decades. If our computers performed as slowly as they did in 1969, where a dual-CPU MULTICS system peaked out at six million instructions per second, we wouldn't have so many layers—they would be too performance-intensive.

Layers make programming and, to some extent, using computers much simpler. They also have other implications. More lines of code means more bugs. And deep software stacks also allow operating system vendors to lock in customers, as the only way to bypass these layers is to port software to other operating systems' layers. I've been thinking and writing about the implications of these layers for many years, and you can take a look at where I hope we are going both in security and in dealing with these layers [2].

References

[1] Grand Canyon Layers: http://www.bobspixels.com/kaibab.org/geology/gc_layer.htm and http://education.usgs.gov/schoolyard/IMAGES/GrandCanyon.jpg.

[2] Rik Farrow, "Software Insecurity," *IQT Quarterly,* vol. 2, no. 2: http://www.rikfarrow.com/IQT_Quarterly_2010.pdf.