

RIK FARROW

musings

rik@usenix.org



LISA '08 IN SAN DIEGO SEEMS TO ME like a place far away, in both time and space. But the memories are still fresh in my mind as I write this: sitting outside eating, drinking, and talking, listening to talks and papers, and meeting people I haven't seen in a while in the halls. In this column, I am going to pick up on a couple of subthemes from the conference: documentation and system configuration.

You can read about the invited talk on writing documentation in the reports on LISA in this issue. As I read them, they brought to mind a couple of things from my own deep, dark past. Yes, I once wrote documentation *and liked doing it*. Somewhere along the way I lost that feeling, and I am still wondering why.

Hardware

When I started consulting, I fell into writing hardware documentation. I had pestered George Morrow [1] and his wife about getting work, and one day I got a call asking me if I could revise a disk controller manual. They had taken an existing floppy-disk controller card and reworked it so that it was I/O-mapped instead of memory-mapped. Rewriting the manual meant translating all memory references to I/O references, starting with the old manual as a WordStar file.

I could read circuit diagrams and had actually patched CP/M device drivers, even relocated one so that it worked at different memory addresses (by using a FORTH program), so this sounded easy. And it was. I whipped through the job, turned it in (on an 8-inch floppy disk), and got paid on the spot. This project led to many other writing projects for Bay Area companies in the early 1980s.

I was a carefree sort of guy back then and spent three months sailing to Panama in 1984. When I went back to Morrow Designs, I discovered that the next hardware manual had been given to a professional documentation writer. I was disappointed, but the manager explained that it would cost them a lot less.

Right on cue, the documentation writer shows up, and she tells everyone that she has written the first 10 pages of documentation. Then she asks the engineers sitting nearby how to install this hard disk controller she needed to document in her single-board computer system. This was an impossibility,

as there were no slots for add-ons in her system. The manager asked me to leave the two of them alone for a few minutes. Shortly after, I learned I had gotten the job.

The disk controller was very cool: An early RISC processor handled both DMA and disk reads and writes. Timing was critical, as early disk controllers actually read and wrote analog data from the drives. The consultant who had designed the board had created his own language for programming the RISC chip, and the 30-some pages of printout were my main source for writing the manual.

The controller emulated IBM mainframe channel controllers, in that the device driver programmer could create a linked list of commands in memory, then tell the controller to carry out the list of commands, notifying the CPU with interrupts when desired. Documenting the short list of commands only took perhaps 30 pages. Later, a device driver programmer told me that writing the device driver from my documentation was easier than any other driver he had ever written. The slick design was certainly mostly responsible. But so was having accurate documentation.

I later learned that the professional writer had complained that she was rejected because a man had shown up. However, that wasn't the real issue: Her talent was in writing, not in reading circuit diagrams and homebrew source code for rare RISC processors. I could do that as well as write.

And that, I believe, speaks to the core issue in documentation. People who write most documentation are writers, and not generally technically apt. Real engineers, for the most part, hate writing documentation. Once a project, be it hardware or software, is working, it is no longer interesting. So the documentation gets written by someone with at best a weak technical background and grudging access to the engineers. And you get documentation written by someone who has little clue.

Today, buying software without any documentation is common, and hardware documentation rarely goes beyond badly translated explanations of how to connect cables and perhaps install firmware (but only when using Windows). A huge after-market in books that provide documentation (consider O'Reilly's missing manual series) has developed, where an outsider reverse-engineers a product and writes about it in a book that will be outdated within a year.

Perhaps there are examples of great documentation out there, for hardware or software. I fondly recall unpacking Sun 3/60s, finding a short length of cable and a BNC T connector, and wondering what the hell it was for (Ethernet, dummy!). Like all "real men," and probably many women as well, I had to figure it out for myself. Well, who bothers to read the manual?

Sysadmins and Documentation

When I began writing my system administration book, I had already written several UNIX system manuals. One thing I had learned to do was take good notes, and I preached that in my first chapter of my book.

Today, I take notes inconsistently. And I regret this whenever I am faced with solving the same problem I solved months earlier but didn't take good notes about the process. My usual excuse was just like the engineers': I am too busy to waste time taking notes. Later I pay for not "wasting" the time, by wasting more time. Puzzle-solving is fun, but not so fun when the time pressure is great.

Janice Gelb, in her “WTFM: Documentation and the System Administrator” invited talk, covers this angle and much more, so read the report—the slides are online and this talk is available in streaming video from LinuxPro[2]. And start writing good documentation. If you cannot write, work with someone else who can write by patiently answering their questions. Provide examples with comments. If your work is poorly documented, you might be the only one who ever uses it, no matter how brilliantly designed it is.

Configuration Management

The configuration management tools war appeared to have cooled down. Either there weren't loud arguments for a particular tool during workshops, or I missed them. Yet the architects of the four most popular tools—Cfengine, Bcfg2, LCFG, and Puppet—were all present, and I got to talk with all of them in the hallway or over meals.

There were two invited talks, one by Paul Anderson, architect of LCFG, who wanted to look at configuration management from a different perspective, and has written about it in this issue (page 20). Then there was an invited talk from Ticketmaster about its own tool, Spine. Like many before them, the Ticketmaster sysadmins decided that nothing in existence really worked for the company's model, so they built their own toolset.

I got another data point about the state of these tools when architects from two of the big four approached Jordan Hubbard, Director of UNIX Development at Apple (and well-known FreeBSD developer before that) after Jordan's talk. Each claimed that his tool was a popular way of managing Apple products, and each time Jordan suggested that they create a document describing the configuration knobs they wanted from Apple to expose via APIs.

Jordan's comment got me thinking. His goal was to publish an API that would isolate the internal details from configuration management tools. I imagined that this would mean that Apple engineers could document this API and still be free to mangle the file formats used for configuration however they liked. But I also wondered what this API would need to cover to handle the most common configuration management tasks.

Currently, all tools edit configuration files, along with other tasks such as restarting or reloading services. If there was a common API for most operating systems, this would make system management easier. However, current tools present a one-tool-does-everything approach to configuration management. In my mind, I can break down the tool into a set of components: a secure communication protocol, a client-side, and a server-side. The server-side seems like the real place to innovate, with components for storing current and past configurations, a GUI to make using the tool simple, more GUI tools for monitoring the effectiveness of the tool, and, of course, CLI versions.

So even though there are many configuration management tools in existence today, there still seems room for research on any of these aspects. I can imagine sysadmins being able to mix-and-match their favorite tools, based on desired features, something that I can only dream about today because of the monolithic architectures.

The Lineup

I started the lineup with an article about insecurities in Linux package management. Cappos and Samuels have written for *login*: before (February

2008), and when they got slash-dotted this summer, I asked if they would write for us again.

The issue this time has to do with how different Linux distros secure their package management systems. The right way to do it is to have a signed manifest from a trusted source and to use that manifest to verify packages that can be downloaded using mirrors. But this is not how most distros do this, and the consequences are scary indeed.

Dehus and Grunwald cover a different angle of package management. Based on their LISA '08 paper, they describe the system they developed for instantiating virtual servers. A key feature of their project, STORM, is that packages that are common to many services get shared, so that these packages can be updated in just one location.

Paul Anderson expounds on the topic of his invited talk at LISA in the next article. Paul ponders about the parallels between the adoption of programming languages and the adoption of configuration management tools and languages. He wonders whether that aren't lessons to be learned there.

Corey Brune decided to evangelize Python for system administration. He wrote a script for extracting password records, converting them to LDAP data, and adding them to an LDAP database. Brune uses this to explain features of Python, a scripting language that is showing up in more places all the time.

David Blank-Edelman shows how to screen-scrape Web pages using WWW::Mechanize. As usual, David provides us with clear examples for using this module, including methods for automating form-filling and submission.

Peter Galvin waxes enthusiastic over a DTrace-based feature found in new Sun storage appliances. Although these are truly appliances, with no visible traces of either DTrace or Solaris, Peter both explains and demonstrates the cool ways of viewing current and past performance of a storage appliance that uses DTrace under the hood.

Dave Josephsen bemoans the death of email, then quickly moves on to explain the use of SMS or Asterisk as a replacement for remote notification systems. I'd heard about texting used in this way before, and it's cool to have a concrete set of examples using Nagios and Nokia phones.

Robert Ferrell goes where no sane sysadmin has gone, and that is into the thicket that is the vi versus emacs debate. Well, sort of.

In the book reviews, Elizabeth Zwicky leads off with three graphic introductions to statistics (no kidding). I add a nongraphic suggestion later, for those of you who don't care whether your statistics textbook is funny. Zwicky then writes about Peter Salus's new book on open source software, then on a book concerning the use of diagrams for explanations and as a way of winning arguments. Finally, she compares two books on the use of Photoshop CS4 for photographers.

Jason Dusek is up next, beginning with a review of *Maven: The Definitive Guide*. Maven is a tool for code build management, and Jason clearly describes how this book handles the uneven documentation surrounding the open-source project. Then Jason takes a deeper dive into *Design Concepts in Programming Languages*, a book about programming language theory that appears most useful to those designing languages, whether big or small.

Sam Stover reviews *The Craft of System Security* and explains why he likes it. I have two short reviews, one on *Statistics in a Nutshell* and the other on *Getting Started with Arduino*.

Although I never set out to be a writer, and even hated writing right through college, I eventually became not just a writer but a successful one. Along the way, I've noticed just how important it is, in the businesses of system administration, consulting, and publishing, to be able to write skillfully. If I had really understood the importance of English 101, I would have looked at it very differently, as I do now.

REFERENCES

- [1] George Morrow: [http://en.wikipedia.org/wiki/George_Morrow_\(computers\)](http://en.wikipedia.org/wiki/George_Morrow_(computers)).
- [2] LISA08, Thursday Tech lineup: <http://www.usenix.org/events/lisa08/tech/#thursday>.