EDITORIAL

Musings

RIK FARROW

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

While I was attending my very first FAST conference, in 2006, a journalist told me that he had a question for a file system expert. I walked him toward registration and quickly spotted someone who I thought could answer his question. After all, I didn't think it would be that hard.

"Why are there so many file systems?" he asked. I was floored. At the time, I thought this was a naive question and felt embarrassed for the journalist. But I later realized I shouldn't have been, because while I had accepted the fact that there were many file systems, multiple ones for every popular operating system, there actually are many reasons why there are so many.

On the surface, you might think that each group of operating systems designers wants to write their very own file system. And you'd be right, up to a point. If you look at the Wikipedia page that compares file systems [1], all of the most commonly used file systems share features. But this misses two important points.

First, the underlying operating systems cannot support dropping in existing file systems. The interfaces to key support functions—for example, allocation of memory—will be different. These differences go even deeper than interfaces. Windows NT, now known as Windows, has a very different design philosophy from any UNIX-related operating system. One manifestation of this is that the NTFS keeps file and directory names, attributes, and block lists in the Master File Table whenever possible. This allows for much faster filename searches and file openings but at the price of having a single point-of-failure—yes, the MFT is a file. I'm exaggerating slightly, as there is a copy of the MFT, and the locations of each are stored in the boot block. UNIX-style systems have superblocks for file-system metadata, inodes for the attributes and blocks in files, and directories themselves are files. I'm skipping over how large files and extended attributes are handled for simplicity, but UNIX and NT file systems handle key features very differently.

The second reason for the variety found in file systems is that the science behind building fast and reliable file systems advances. Fragmentation was a huge problem for the Version 7 UNIX file system, which continued to be used in System V. The Fast File System solved this through the use of cylinder groups, something used in Linux ext and the NTFS as well. FFS also introduced the notion of having multiple copies of the superblock. But ext2 experimented with a dangerous technique that has turned out to work: it dispensed with synchronous writes of metadata, which block any further writing to the file system. Unpacking a tar archive on the same hardware was 10 times faster using ext2 than FFS. I know, because I experimented with both BSDI and Linux in 1992.

Finally, file systems exist to support applications, and not all applications require the same style of operations. Some applications create large sequential reads or writes, some create lots of small files (mail systems), high performance computing systems create thousands of checkpoint files nearly simultaneously, and so on. So while ext4 works fine for most Linux installs, some applications demand the use of XFS for handling large files.

The NSF Visioning Workshop [2], with a report published in 2019, focuses on the potential future design requirements for storage and file systems. Some things in the report just seem like common sense: for example, creating support for commonly used lower level file system tasks that can be reused. Other facets of this report cover newer applications, such as edge computing and machine learning, that have different needs than older applications.

The five lead authors of the workshop report (you can find a list of the workshop participants at [2]), wrote about some of the report's findings for this issue of *;login:*.

The Lineup

Given that build-up, you might expect that this issue will be all about file systems and storage, but you'd be wrong. We actually start out with several articles related to security.

Jean Camp writes about her experience monitoring BGP mistakes and attacks. BGP was designed when there was little concern for security, and Internet operators were happy enough to have a routing advertisement protocol that just worked. There have been many attempts to secure routing updates that have not succeeded in the decades since BGP was adopted. Camp explains how important it is to consider the geopolitical aspects of running BGP safely, and describes a tool that permits blocking access to networks affected by mistaken or malicious route updates.

Reardon et al. share their work on finding side and covert channels in Android. They built upon their prior work in improving the usability of the Android permission system and creating a monitoring tool to check how well the permissions systems worked. While testing the accuracy of their tools, they uncovered lots of malfeasance, including in libraries specifically for apps for children, a violation of the law in many countries.

Kulandaivel et al. talk about the CAN bus scanner they developed, called CANvas. While you can purchase information about replacing your car's Electronic Control Units, that tells you only about the ECUs the manufacturer installed in your car, not anything added later. The authors explain why a scanner is useful but also difficult to implement for the CAN bus.

Peter Peterson and Rob Jansen have produced a summary of the CSET '19 workshop. CSET had a broader focus this year, and the chairs describe how this new focus and a larger program committee worked out. They also include summaries of all the presentations from CSET '19.

Kirill Levchenko is interested in aviation, but his interest goes deeper than just being a passenger. Levchenko has been working with a multi-institution group to create testbeds for the communication buses used in large passenger aircraft, like the Boeing 737. Some of what they have discovered turns out to be comforting rather than frightening, as I discovered while interviewing him.

Amvrosiadis et al. share some of the results of the NSF-sponsored future of file systems and storage 2025 workshop mentioned in the opening. The idea behind the workshop was to provide directions for future research and development in these areas. In particular, the workshop participants determined important new areas that have, or are expected to have, unusual storage requirements, such as machine learning and the Internet of Things.

Terence Kelly has been designing programming paradigms for persistent memory for many years. In this article, Kelly demonstrates two different techniques, one a programming style for traditional storage-backed memory, and the second, a mechanism for making changes to the backing-store atomic.

Effie Mouzeli shares her perspective on "Ask-Me-Anything" engineering. Not all system engineers (SEs) work at huge companies. Mouzeli provides useful advice for people working alone or in small teams of SREs in the often chaotic environments of startups and at smaller organizations, where the SE must be able to solve almost any problem—thus the AMA designation. Mouzeli writes from her own life experiences, including about the five stages of technical debt, with humor and honesty.

Amit Gud explains the benefits of multi-tenancy in microservice environments. Multi-tenancy means that data in flight and when stored include labels that are used to control the flow and usage of data in these environments. Uses include testing code or configuration changes and designing more modularity into systems.

Laura Nolan writes about the pitfalls of dynamic control systems. Ever wonder why the servers you once had in your racks were more reliable than the complex systems run by gigantic cloud services? Wonder no more.

Peter Norton wants to teach you how to add useful profiling to your Python scripts with the goal of looking at different visualizations. The default Python profiler doesn't produce as useful results as newer tools, so Norton demonstrates other tooling.

Dave Josephsen tells us about distributed tracing. There have been two popular projects, OpenTracing and OpenCensus, which are being merged into one. And the IETF has been working on a way to use HTTP headers to do this called OpenTelemetry. Dave explains the differences between these approaches.

Dan Geer and Jason Crabtree challenge us to get clear about our security metrics. If everyone creates and uses their own set of in-house metrics, we cannot share measurable information about attacks, risk, and the success or failure of defenses.

Robert Ferrell muses about possible solutions to the wave of ransomware affecting systems throughout the world. I pointed out several obvious weaknesses in most of his approaches, and he reproached me. It *is* a humor column, Robert reminded me.

Mark Lamourine has reviewed cookbooks about Kubernetes, Ansible, and OpenStack in this issue, and I cover Randall Munroe's cookbook for how to solve common problems using absurd scientific advice.

While considering Robert Ferrell's absurd solutions to ransomware, I wondered why victims almost never have good backups prepared. We all know about the importance of backing up systems and testing these systems before we need them. Could it be that most IT departments cannot handle this most basic of tasks, one that comes right after user management?

The failure of so many organizations tells us volumes about the world we live in. The real world is prone to failure and is not full of people eager to do the repetitive work of having to create routine backups—even though the occasional consequences are far worse than the boredom entailed while spot-checking backups for correct functionality—or the more difficult task of setting up and enforcing a site-wise backup system or policy.

Sometimes I think that I live in a world populated by teenagers, all in revolt against everything that has stood the test of time and willing to risk everything just because they can. I was glad when my teenagers outgrew that period of their lives, and I can't wait for the rest of the world to get there, too.

References
[1] Comparison of File Systems: https://en.wikipedia.org/wiki/Comparison_of_file_systems.

[2] G. Amvrosiadis, A. R. Butt, V. Tarasov, E. Zadok, and M. Zhao, Data Storage Research Vision 2025 Report on NSF Visioning Workshop, 2018: https://dl.acm.org/citation.cfm?id=3316807.

Letter to the Editor

Thanks for the enthusiastic intro to our "Not So Fast" work in the Musings editorial in the Fall 2019 issue.

We did want to highlight a few unfortunate misconceptions about Browsix-Wasm—the biggest being that Browsix is most emphatically not a browser plugin, and that it does not provide access to the host's file system!

Browsix works as a JavaScript/Wasm library that runs entirely within the browser, without the need to install plugins. The file system that Browsix exposes to programs (like SPEC) is entirely independent of the host OS's file system. For SPEC, all the files come from an HTTP server, and a writeable "overlay" file system provides ephemeral storage for the duration that a tab is alive (a very similar approach to how Docker provides layered file systems).

Like the file system, all of the operating system services that Browsix-Wasm provides (like processes and pipes) are built on top of standard browser APIs (like WebWorkers) within the confines of the browser sandbox. This approach is what enables us to work across all major browsers.

Given that safety, you and everyone else should feel just fine about running Browsix-Wasm on your daily driver browser, and indeed you might visit websites that use Browsix without you even knowing it! (Components of it help power the emulators on archive.org, like the Oregon Trail: https://archive.org/details/msdos_Oregon_Trail_The_1990.)

Many thanks in advance,

—Abhinav, Bobby, Emery, and Arjun

Correction

In the book review of *Concurrency in Go* (Summer 2019 issue, page 59), C. A. R. Hoare was incorrectly listed as C. Anthony and R. Hoare in the reference at the end of the review. We apologize for the error.

Save the Dates!

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

FAST '20

18th USENIX Conference on File and Storage Technologies

February 24–27, 2020 | Santa Clara, CA, USA

Sponsored by USENIX in cooperation with ACM SIGOPS

Co-located with NSDI '20

www.usenix.org/fast20

The 18th USENIX Conference on File and Storage Technologies (FAST '20) brings together storage-system researchers and practitioners to explore new directions in the design, implementation, evaluation, and deployment of storage systems.

The program committee will interpret "storage systems" broadly; papers on low-level storage devices, distributed storage systems, and information management are all of interest. The conference will consist of technical presentations including refereed papers, Work-in-Progress (WiP) reports, poster sessions, and tutorials.

---

nsdi '20

17th USENIX Symposium on Networked Systems Design and Implementation

February 25–27, 2020 | Santa Clara, CA, USA

Sponsored by USENIX in cooperation with ACM SIGCOMM and ACM SIGOPS

Co-located with FAST '20

www.usenix.org/nsdi20

NSDI focuses on the design principles, implementation, and practical evaluation of networked and distributed systems. Our goal is to bring together researchers from across the networking and systems community to foster a broad approach to addressing overlapping research challenges.

NSDI provides a high-quality, single-track forum for presenting results and discussing ideas that further the knowledge and understanding of the networked systems community as a whole, continue a significant research dialog, or push the architectural boundaries of network services.

The full programs and registration will be available in December.