WeGet letters

by Michael W Lucas

Dear FreeBSD Journal Letters Column Answerer,

There's a bunch of excitement over how ARM64 is now Tier 1. How should I use it? Should I immediately switch my desktop, my servers, and my media server over to ARM64? Just from reading this very issue, I've gotten really excited about it. How fast should I move?

—Searching Out Amazing Machines

"My day is my own," I thought when I woke up this morning. "I can lounge around and think about the glorious success that will inevitably descend upon me if only I can keep the bit about the wombats, the school bus, and the algae bollard under wraps, which should be simplicity incarnate because nobody knows what a bollard is except for a handful of literati who know nothing of wombats. I should probably write down a sentence or two, something about how the information reported by hard drives is not merely deceitful but actively treacherous, just so I can claim that I'm doing real work instead of wandering about the house listening to wire recording mix tapes from the 1930s and wondering how I can keep the squirrels from nesting in my emergency pants. It's not that I need pants all that often. They make bad days wholly terrible, like when I need to leave the house to find a gelato service that understands the difference between promising and boasting. The current one isn't it. Maybe the next."

And then your letter arrives, SOAM, ruining an otherwise perfect day.

On the plus side, I get to crush your hope. That's always nice.

All operating systems have an idea of tier 1 architectures. This means that the operating system can be installed on that architecture, that it runs, and that updates will be available to fix the inevitable bugs. Crash dumps will receive the same mixed attention as those of any other major platform. That sounds fine, right?

The problem is that sysadmins don't run hardware. We don't even run operating systems. We run applications. FreeBSD might be tier 1 on ARM64, but that doesn't mean your application is. Sure, there's lots of packages available. Many ports build. Perhaps even most. But just because the code compiles doesn't mean that it works let alone interoperates with whatever malware you're passing off as an application stack. People are using ARM64 for real work out in the real world, but that doesn't mean that you can. You thought Linux-isms were bad? Wait until you get a look at Intel-isms. Sure, people are working on their applications to make them work on ARM64, but the change in architecture has opened vast new realms of bugs. The obvious bugs have been found. What remains are the highly specific ones. Your environment is highly specific. Logically, these bugs all belong to you.

Many technologists claim that ARM64 is inevitable. The only inevitabilities are core dumps and that orange—and—green rash on my neck. People who should know better tout the advantages of ARM64 as if anything in computing could ever be improved when we all know that the pain never goes away, only changes. Install an ARM64 web server, and you'll discover tiny changes in behavior will put your application at risk. The people pushing ARM64 keep babbling about "reduced power consumption" and "open platforms," and they're extremely stubborn, so I suspect that they'll eventually get their way. A change of pains is as good as a rest.

So, what do you do?

You could start by not writing letters to this journal. That would have been an improvement.

Failing that, you should prepare for failure.

Your vital application runs on ARM64? Great… for some value of "great."

You can't start using it yet. Even if you set up an ARM64 system purely for testing and turn on all the debugging you can find so you can catch application errors and submit bug reports, you almost certainly have no idea what normal looks like. Your idea of normal is a quiet helpdesk phone. When your brand-new ARM64 system starts spewing cryptic messages about locks and updates and whatever sort of flimflam the developers yammered that made your organization decide that this particular group of lies would solve their problems, you'll have no idea if these are normal or not.

You're starting in the wrong place.

Application developers rarely design useful logs. A few intend to.

Application developers rarely design useful logs. A few intend to. Many design logs that they find useful, which is not the same as useful to you. You need to know what normal logs look like, so you can recognize abnormal ones.

Start playing with ARM64 by going to your legacy environment, full of AMD64 or MIPS or even (ugh) i386 hardware. Make a list of your vital applications. For each one, figure out how to gather debugging data. Wholesome systems send everything to syslog, where you could distribute it into individual log files as needed, but many modern developers have abandoned this healthy practice in favor of randomly selected logging systems that happen to conform to their prejudices so you'll have to (ugh, ugh) read the documentation. Some sysadmins have centralized logging servers where they can perform analysis of messages from every system they manage, but they are overachievers, and we will discuss them no further. Worst case, find a convenient log4j instance on the public Internet and dump all your debugging there. They won't mind.

While you are digging up logging configuration information on your every critical application, make a list of how to file bug reports on each and every one. It's much easier to do this before a notably vexing bug raises your blood pressure and triggers your brain's wired-in "kill one developer or massacre them all?" decision-making circuits.

Now that you have a baseline for comparison, you can install your ARM64 system and see what happens. Don't get me wrong, it's going to fail. As always, the question is how it will fail. Your ARM64 systems will have logs stuffed with cryptic meaningless messages. Fortunately, you already have functioning servers that have their own cryptic, meaningless log messages. You can compare the two and, with luck and perhaps a simple conjuration at an abandoned crossroads during the new moon, sort out messages that indicate your actual error.

Prepare a bug report.

Send it to the application developers.

If they answer, they will almost certainly wonder why you're using their application in a way that was never intended, but that's exactly what UNIX is for, so ignore the sniveling. Work the problems, one after the other, until your application truly runs on ARM64. That's how open source software works. It's people like you, doing the grunt work of polishing and problem-solving, so that future decades of lazy bastards like myself can reap the rewards.

If that's not enough of an answer for you, too bad. I hear a squirrel gagging in the garage, so I know where my emergency pants are. I should probably wash them one year.

Have a question for Michael?
Send it to letters@freebsdjournal.org

MICHAEL W LUCAS has spent too many decades debugging hardware platform migrations. His latest books include *DNSSEC Mastery* and *$git sync murder*. He's also released *Letters to Ed (1)*, a collection of the first three years of this very column. Why he thinks you'll pay good money for something you get for free right here, we have no idea.