Solaris to Linux Migration 2017

05 Sep 2017

Many people have contacted me recently about switching from Solaris (or illumos) to Linux, especially since most of the Solaris kernel team were let go this year (including my former colleagues, I'm sorry to hear). This includes many great engineers who I'm sure will excel in whatever they choose to work on next. They have been asking me about Linux because I've worked for years on each platform: Solaris, illumos, and Linux, in all cases full time and as a subject matter expert. I've also done some work on BSD, which is another compelling choice, but I'll discuss that another time. The following is my opinion and not an official guide to any OS.

Switching from Solaris to Linux has become much easier in the last two years, with Linux developments in ZFS, Zones, and DTrace. I've been contributing (out of necessity), including porting my DTraceToolkit tools to Linux, which also work on BSD. What follows are topics that may be of interest to anyone looking to migrate their systems and skillset: scan these to find topics that interest you.

ZFS

ZFS is available for Linux via the zfsonlinux and OpenZFS projects, and more recently was included in Canonical's Ubuntu Linux distribution: Ubuntu Xenial 16.04 LTS (April 2016). It uses a Solaris Porting Layer (SPL) to provide a Solaris-kernel interface on Linux, so that unmodified ZFS code can execute.

My company uses ZFS on Linux in production, and I've been the go-to person for deep ZFS problems. It feels largely the same, except kstats are in /proc/spl/kstat/zfs/arcstats, and I debug it with Linux tracing tools instead of DTrace (more on that next). There have been some issues on Linux, but overall it's been ok, especially given how hard we push ZFS. We've used it for our container hosts (codename Titus) that do frequent snapshots, use send/recv, etc.

I think the ARC memory counters need more work, as people keep capping the ARC to avoid keeping memory from applications, and the ARC should already handle that (with the exception of massive allocations). There's also a ZFS send/recv code path that should try to use the TASK_INTERRUPTIBLE flag (as suggested by a coworker), to avoid a kernel hang (can't kill -9 the process). Both of those should be easy fixes. There are plenty of other bugs to fix, though, which you can see in the issue list on github.

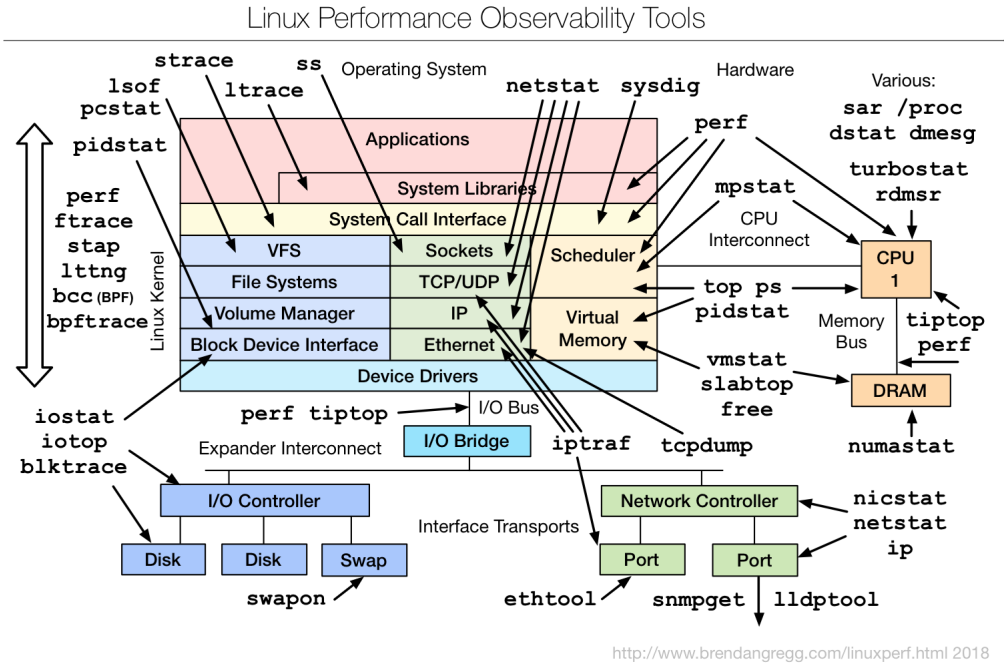
Linux has also been developing its own ZFS-like filesystem, btrfs. Since it's been developed in the open (unlike early ZFS), people tried earlier ("IS EXPERIMENTAL") versions that had serious issues, which gave it something of a bad reputation. It's much better nowadays, and has been integrated in the Linux kernel tree (fs/btrfs), where it is maintained and improved along with the kernel code. Since ZFS is an add-on developed out-of-tree, it will always be harder to get the same level of attention.

We're now testing container hosts in production on btrfs, instead of ZFS. Facebook have been using btrfs for a while in production, and key btrfs developers now work at Facebook and continue its development. There is a btrfs status page, but for the latest in development see btrfs posts to the linux kernel mailing list and btrfs

sections on kernelnewbies. It's a bit early for me to say which is better nowadays on Linux, ZFS or btrfs, but my company is certainly learning the answer by running the same production workload on both. I suspect we'll share findings in a later blog post.

Observability

Here's the big picture of performance observability tools on Linux, from my Linux performance page, where I also have diagrams for other tool types, as well as videos and slides of prior Linux performance talks:



I also have a USE Method: Linux Performance Checklist, as a different way to navigate and apply the tools.

Linux has many more text interfaces in /proc that Solaris does, which help for ad hoc debugging. It sounds inefficient, but I've never seen /proc readers show up in CPU flame graphs.

DTrace

Linux 4.9 provides the raw capabilities to implement DTrace-like scripts, allowing me to port over many of my DTraceToolkit scripts (they also work on BSD). The hardest part on Linux is now done: kernel support. I wrote about it in a previous post, DTrace for Linux 2016. You might also like my Give me 15 minutes and I'll change your view of Linux tracing video as an introduction to the different built-in Linux tracers.

Nowadays, there are three built-in tracers that you should know about:

- **ftrace**: since 2008, this serves many tracing needs, and has been enhanced recently with hist triggers for custom histograms. It's fast, but limited in places, and usually only suited as a single-user tool (there are workarounds). I wrote an ftrace toolkit, perf-tools, and the article Ftrace: the hidden light switch.
- **perf**: since 2009, this started as a PMC profiler but can do tracing now as well, usually in a dump-and-post-process style. It's the official profiler. I wrote a page on it: perf.
- **eBPF**: tracing features completed in 2016, this provides efficient programmatic tracing to existing kernel frameworks. Many new tools can now be written, and the main toolkit we're working on is bcc.

Here's some output from my zfsdist tool, in bcc/BPF, which measures ZFS latency as a histogram on Linux:

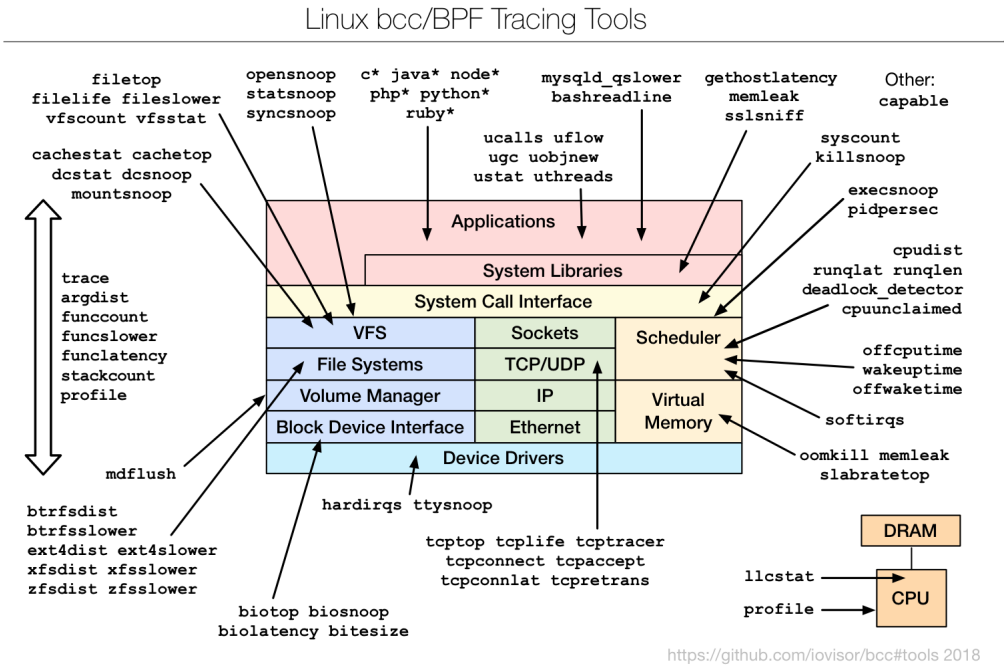
```
# zfsdist
Tracing ZFS operation latency... Hit Ctrl-C to end.
^C

operation = 'read'
     usecs               : count     distribution
         0 -> 1          : 0         |                                        |
         2 -> 3          : 0         |                                        |
         4 -> 7          : 4479      |****************************************|
         8 -> 15         : 1028      |*********                               |
        16 -> 31         : 14        |                                        |
        32 -> 63         : 1         |                                        |
[...]
```

Linux has been adding tracing technologies over the years: kprobes (kernel dynamic tracing), uprobes (user-level dynamic tracing), tracepoints (static tracing), and perf_events (profiling and hardware counters). The final piece was enhanced BPF (aka eBPF: enhanced Berkeley Packet Filter), which provided the custom in-kernel programmability needed for an advanced tracer, created by Alexei Starovoitov (now at Facebook).

There's a front-end for BPF, bcc (BPF Compiler Collection), which has many single- and multi-purpose tools written by me and others. Check it out. It's currently much more difficult to write a bcc/BPF script than a DTrace script, but at least it's now possible (using Linux built-ins), and one day there might be an easier front-end.

I have a page on eBPF tracing, and the current bcc/BPF tools are:



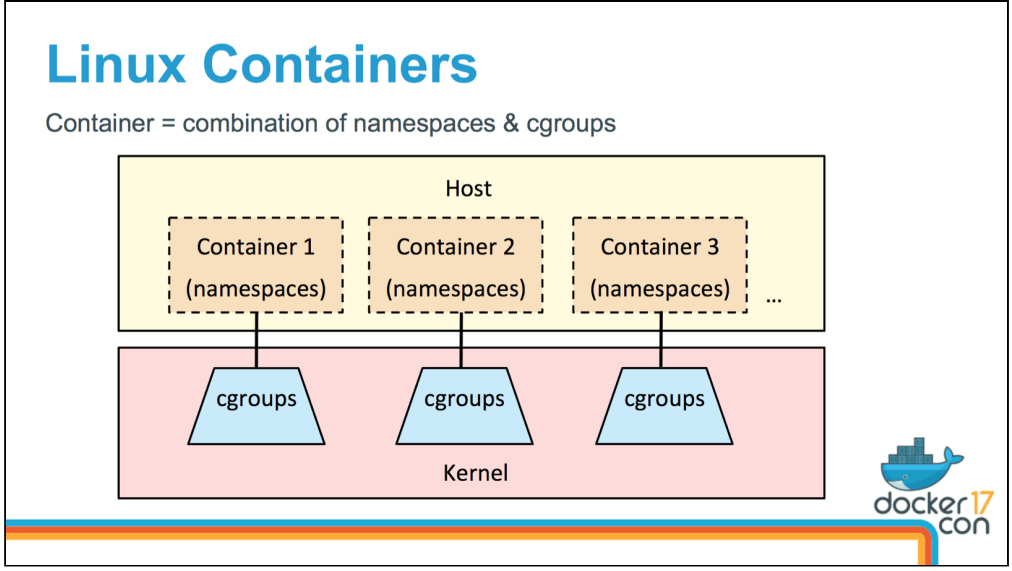
There have been other tracing projects for Linux, and some companies found them useful for their needs, but the big problem was that they weren't merged in mainline Linux. Now that eBPF has been, many of these tracing projects may switch to using it as a backend since it is stable, or, they could further specialize in what they do (non-BPF related), eg, offline analysis of a capture file (LTTng, sysdig).

If you're on an older Linux kernel (3.x), you can use ftrace for some tracing needs. My perf-tools includes single purpose tools like opensnoop, execsnoop, iosnoop, and more, and multi-purpose tools like funccount, kprobe, and uprobe. I intended perf-tools as a hacky workaround until eBPF was available, but ftrace has since been developed further (hist triggers) so perf-tools may have a reason to continue.

Zones

I'd recommend this post about Zones vs Containers by Jessie Frazelle.

On Linux, containers are a combination of namespaces (restricting what a process sees) and cgroups (similar to Solaris resource controls). People rarely create them manually. They use third-party software like Docker or Kubernetes to simplify their administration. I gave a talk about container performance recently at DockerCon, and included a quick summary of how they work: YouTube, SlideShare:



If you search of slideshare and youtube, you'll find many other good talks on containers as well. Apart from Jessie, I also like talks by Jérôme Petazzoni, and Tejun Heo.

Linux containers have been in rapid development in the last few years. It's the container team at my employer that runs the newest Linux kernels, since they need the latest features and fixes, and you should try to run the newest as well. Currently that means at least Linux 4.9.

There's a lot about Linux containers that isn't well documented yet, especially since it's a moving target. (Zones lacked docs when they came out too, which is why I wrote the first Zones resource control docs.) Search for recent blog posts on Linux containers, and try them out, and you'll piece together their capabilities and workings bit by bit. Here are some documents for understanding internals:

- Linux Namespaces from Wikipedia
- Linux Cgroups from Wikipedia
- Documentation/cgroup-v1 from the Linux source
- Documentation/cgroup-v2.txt from the Linux source

One feature Linux containers lack is a container ID in the kernel. It's been proposed on lkml, but the patches have not been integrated yet (it was last discussed two weeks ago). Some people argue that the kernel shouldn't have one, since a container is a collection of namespaces and cgroups defined in user-space (by Docker, etc), and it's therefore up to user-space to track it. As a performance engineer who does kernel tracing, I find the lack of an ID I can trace in the kernel to be pretty annoying. There are workarounds: I can use the perf_events cgroup ID, provided the container software is configuring it (they do).

Some specific differences that got my attention: you can access a container's mount namespace from the host (global zone) via /proc/PID/root, given a PID in a container. But understanding if a PID belongs to a container is surprisingly difficult: there's no -Z option to tools like ps, since there's no container ID in the kernel. From the host (global zone), given PID 18300:

```
host# grep NSpid /proc/18300/status
NSpid:  18300   1
host# grep 18300 /sys/fs/cgroup/perf_event/*/*/tasks
/sys/fs/cgroup/perf_event/docker/439e9f99850a9875e890130a2469114536f8aa55d7a1b37f86201e115b27fc0f
```

The first command shows that PID 18300 is really PID 1 in another process namespace: a telltale sign it's in a container. I also checked a task list from /sys/fs/cgroup, and saw it's in a docker cgroup. I've suggested adding a command to docker to make listing at least the top-level PIDs in containers easier.

Virtual Machines

The two main technologies on Linux are Xen and KVM (and there's Bhyve for BSD). Xen is a type 1 hypervisor that runs on bare metal, and KVM is a ~~type 2~~ hypervisor that runs as processes in a host OS, and can make use of kernel modules for faster device access (so it's both type 2- and type 1-ish). Oracle VM Server is based on

Xen. Xen's biggest user is the Amazon EC2 cloud, which has over one million customers, and appears to be a custom version (it self identifies as version "3.4.3.amazon"). Outside of EC2, many other providers are deploying on KVM.

Both Xen and KVM have had many performance and security improvements, and workloads can now be tuned to run at almost bare metal speeds (say, a 3% loss or less). At my employer we sometimes use SR-IOV for direct network interface access, and NVMe for direct disk access. Some years ago, it was easy to make the case to switch from VMs to containers due to the performance improvements alone, as VMs had to emulate everything. Not so today, although this comes at the cost of complexity and required tunables. In general, I find Xen more complicated to work with than KVM. (FWIW, I contributed some patches to Xen to allow a subset of PMCs to be accessed.)

If you switch from managing Oracle VM to Xen, it will hopefully feel very similar. If you switch to KVM, it will be quite different, but hopefully easier.

SMF

I personally introduced hundreds of customers to SMF while teaching Solaris 10 classes. I came up with a great demo where I could break Solaris 9 and 10 servers in the same way, then demonstrate how it would take an hour and a reboot cycle to fix Solaris 9, but minutes and no reboot to fix Solaris 10. I also wrote and published an entertaining SMF manifest that played music. A lot of people got it and learned to love SMF. But some still hated it and the new universe of stuff one had to learn. Some vowed to remain on Solaris 9 forever in protest, or to switch to Linux.

Linux is now going through this with systemd, and has its own share of systemd critics, encouraging distros to remove systemd. I suspect it will prevail, just as SMF did. There are many implementation differences, but the same general idea: a coordinated system to manage parallel application startup and dependency state.

If you absolutely can't stand systemd or SMF, there is BSD, which doesn't use them. You should probably talk to someone who knows systemd very well first, because they can explain in detail why you should like it.

Performance

Linux should be faster out of the box for many production workloads, due to improvements in scheduling (including a tickless kernel), driver support, newer syscalls features, newer TCP feature support, processor optimizations (often provided by Intel engineers directly), a lazy TLB, and more. There's also better compiler and application support: in some cases applications run faster on Linux, not because the kernel is faster, but because that compilation target has had more attention. I've even seen cases where the Makefile compiles on Linux with -O3, and Solaris with -O0, thus crippling Solaris performance, for no legitimate reason.

How much Linux is faster depends on the workload: I'd expect between zero and a few percent typically. There are some extreme cases, where a lack of proper driver support on Solaris can have Linux run 10x faster. I'd also expect you could still find a workload where Linux is slower. For example, although it's very minor, the /dev/*random devices were faster on Solaris last time I checked, as Linux was spending more effort on entropy for improving security. (Or, from a different point of view, Solaris was less secure.)

I spoke about the performance differences in my 2014 SCALE keynote "What Linux can learn from Solaris performance and vice-versa" (slides) where the conclusion was that Linux may run faster out of the box, but I could typically make Solaris run much faster thanks to optimizations found using DTrace. DTrace didn't exist for Linux at the time, but now we have BPF (see previous section). There have been many other improvements to Linux since then, as well.

Security

Key Linux security technologies to learn:

- LSM: Linux Security Modules

- AppArmor: application access control (LSM)
- seccomp: secure computing mode, restricts system call usage
- SELinux: Security-Enhanced Linux (LSM), for access control and security policies (alternate to apparmor)
- Linux audit: event logging
- eBPF (which is used to enhance seccomp)
- iptables: network firewalling

There are many more: browse the release notes on kernelnewbies. Live kernel patching is another capability, that is currently being integrated in the 4.x series. And namespaces, used for Linux containers, are also a relevant technology.

There have been security vulnerabilities, just like there are with any software. This is especially true for Linux, which is used everywhere and has a lot of attention. The way the cloud is used helps with security: most instances at my employer have only been up for one or two days. We're constantly creating and destroying instances from a base image, which means that when we update that base image with security patches, they get rolled out very quickly.

Reliability

Our production servers, running Ubuntu, have been rock solid. In over three years, I've only seen three kernel panics, for an enormous deployed fleet (tens of thousands of Linux instances). Working on Solaris, I'd usually see several different panics per year. I would not attribute this to, say, a more limited range of workloads at my company: we have a wide range of different things running internally. I would, however, attribute some of it to our virtualized environment, running virtual machines: the hypervisor will handle some hardware problems before the guest kernel sees them, which I suspect helps us avoid some hardware-related panics.

In a test environment, I've seen several more Linux panics in the past three years. Five of those were my own kernel bugs, when I was doing kernel development. Two others were on the latest "release candidate" (-rc) kernel from kernel.org – the bleeding edge of kernel development. If you do run the latest -rc kernel and hit a bug, please share on the Linux kernel developers mailing list (lkml) where it should be quickly fixed.

In production, people often stick to the Long Term Support (LTS) kernel releases. Vendors (see later section) are usually quick to make sure these LTS kernel releases have all the right patches and are reliable. My rock solid experience with Ubuntu is on an LTS release.

Crash Dump Analysis

It can be done. One technique is kdump, which uses a capture kernel configured in grub. Execution switches to the capture kernel during a panic, so that a working kernel can capture the state of the system. I've set this up a few times and successfully debugged kernel panics.

It's worth describing what commonly happens with Linux kernel panics. In an environment like ours (patched LTS kernels running in VMs), panics are rare. The odd time we hit them, we'll take the "oops message" – a dump of the kernel stack trace and other details from the system log – and search the Internet. We almost always find that someone else has hit it and had it fixed, and so then we track the patch to the kernel update and deploy that. There's so many people running Linux, and given that we're usually on LTS and not the release candidates, it's rare that we're the first to hit a panic.

For that rare case where we are first to hit a panic: by posting the entire oops message to the right mailing list, the responsible engineer will usually fix it quick (by figuring out how to reproduce from the oops message alone), and then we track their patch into a kernel update. That mailing list would be lkml if we're running the latest rc (only in test), or the mailing list identified in the MAINTAINERS file (more on that later). In Solaris, we'd only really do panic analysis given a crash dump, but Linux gets lots of mileage from the oops message alone. Just from a quick search, see this presentation PDF, which digs into oops message components.

Another difference: kernel panics don't always reboot the system. Linux can oops and kill a process, but not reboot if it doesn't think it needs to, instead leaving it up so you can login and debug. It's also why you should

always run "dmesg" at the start of any investigation, to check if the system (that's still up!) has in fact oops'd.

As for hitting a panic for the first time, posting an oops message, but finding no one wants to fix it: I haven't seen that yet in 3 years. The day it does happen, I'll set up the capture kernel, get a crash dump, and do the analysis myself. You can also ask your Linux OS vendor, if you are paying one for support.

Debugging Tools

Instead of mdb you'll be using gdb, which has been improving, and even has a TUI mode nowadays. I wrote gdb Debugging Full Example (Tutorial), and I also recommend you watch Greg Law's talk Give me 15 minutes and I'll change your view of GDB.

I've noticed that newer projects are using lldb. Here's an lldb to gdb command map.

Other Tools

If you never found this before, it's been a great resource over the years: the Rosetta stone of Unix, from which you can draw a table of just Linux and Solaris.

	redhat-config-xfree86 /etc/X11/?dm	
TASK \ OS	**Linux**	**Solaris**
read a disk label	fdisk -l	prtvtoc
whole disk in partition	/dev/hda (e.g. if */dev/hda1* is a partition)	2
label a disk	cfdisk fdisk gdisk e2label	format prtvtoc *(x86)* fdisk
partition a disk	parted *(if you have it)* cfdisk fdisk gdisk pdisk *(on a MAC)* *(deb)* mac-fdisk *(on a MAC)* *(md)* diskdrake	format fmthard
TASK \ OS	**Linux**	**Solaris**
kernel	/boot/vmlinuz* /boot/bootlx	/kernel/genunix /platform/^ uname -m`/ kernel/unix

There's also a new effort to recreate it, which is online at the Command Line Rosetta Stone. Oracle have a similar useful page as well: the Linux to Oracle Solaris 11 comparison, as well as a procedure for migrating from Solaris to Linux.

A few other tool differences that stood out to me:

- syscall tracing: truss → strace
- packet sniffing: snoop → tcpdump
- process tree: ptree → pstree -ps
- kernel tuning: ndd → sysctl
- binary dumping: elfdump → objdump
- kernel module list: modinfo → lsmod
- swap status (swap often isn't used): swap → swapon

Other Kernel Differences

Linux supports overcommit: instead of guaranteeing that all virtual memory can be stored when needed, including on swap devices, Linux bets that it won't need to, so allows more virtual memory allocations than it could possibly store. This means that malloc() almost always returns successfully, so much so that some programmers on Linux don't bother checking its return value. What happens if processes really do try to populate all that virtual memory? The system runs out, and the kernel's out-of-memory killer (OOM killer) will pick a sacrificial process and kill it. If that seems wildly unacceptable, note that you can tune overcommit on Linux to not do this, and behave more like Solaris (see sysctl vm.overcommit_memory).

I covered various kernel differences in my SCALE 2014 talk What Linux can learn from Solaris performance and vice-versa, and of course my book Systems Performance: Enterprise and the Cloud where I cover both Linux and Solaris.

OS Vendors and Paying for Linux

If you're already an Oracle customer and switch to Linux, then there is Oracle Linux. Other vendors who offer support include Red Hat, Canonical, and SUSE.

However, most companies don't pay for Linux. How does it get developed? Often companies want features and will develop and upstream them to meet their own needs. But once it's part of mainline Linux, their contribution may end there. There may be no real documentation written, no marketing of the feature, and no education of the community. Just code that appears in the Linux source because IBM/Cisco/Hitachi/whoever needed it there for their own internal project. This lack of supporting efforts can make learning Linux capabilities more challenging.

Linux Kernel Engineering

If you want to get into Linux kernel development, you'll need to get familiar with Coding Style, Submitting Patches, and the Submit Checklist. You could also read On submitting kernel patches (see section 14.1 and imagine how different Solaris would be if Linux accepted that patch!).

There are also many blog posts on how to compile the Linux kernel and submit your first patch, just search for "compiling the Linux kernel". It can be menu driven or automated. Just as an example, here's my build script for automating Linux kernel builds for my custom EC2 environment (it's custom, you don't want to use it, just giving you an idea).

You'll especially want to understand the MAINTAINERS file. It's very unlikely you'll be submitting patches to Linus Torvalds (nor the github repo, read why). You'll almost always be sending your patches to "maintainers", who will do code review and then pass your patch on to Linus. There are over one thousand subsystems in Linux (many for device drivers), each has one or more maintainers. Maintainers make the day-to-day decisions in Linux development. Apart from reading the MAINTAINERS file (which includes a legend at the top), you can query it. Eg, to see who maintains tcp_output.c:

```
linux$ ./scripts/get_maintainer.pl -f net/ipv4/tcp_output.c
"David S. Miller"  (maintainer:NETWORKING [IPv4/IPv6])
Alexey Kuznetsov  (maintainer:NETWORKING [IPv4/IPv6])
James Morris  (maintainer:NETWORKING [IPv4/IPv6])
Hideaki YOSHIFUJI  (maintainer:NETWORKING [IPv4/IPv6])
Patrick McHardy  (maintainer:NETWORKING [IPv4/IPv6])
netdev@vger.kernel.org (open list:NETWORKING [IPv4/IPv6])
linux-kernel@vger.kernel.org (open list)
```

The MAINTAINERS file also shows the mailing lists for each subsystem. Patches often get hashed out there and polished long before they are sent by the maintainer to Linus on lkml.

The kernel development cycle: It begins with a new release (eg, 4.13), and then every Sunday (or whenever Linus decides) a release candidate in posted for the next release (eg, 4.14). So there'll be 4.14-rc1 (which may take two weeks, to allow for large changes, the "merge window"), then 4.14-rc2, etc, usually up to -rc7 or -rc8, which will be the final release candidates, and then Linus will cut that release (eg, version 4.14: with no "-rc"). All major changes are supposed to go in the first or second release candidate, and then minor bug fixes by rc7. For example, Linus just released 4.13, saying:

```
So last week was actually somewhat eventful, but not enough to push me
to delay 4.13.

Most of the changes since rc7 are actually networking fixes, the bulk
of them to various drivers. With apologies to the authors of said
patches, they don't look all that interesting (which is definitely
exactly what you want just before a release).  Details in the appended
shortlog.
[...]
```

If you make some major changes or feature additions, and Linux is currently on rc3 or later, your patches are unlikely to be integrated in that release. The maintainers will hold on to them: they often have their own forks of Linux for this purpose.

As for brilliant jerks: Linux has them. So did Solaris. You know what I mean: the difference between saying "this code is idiotic" (probably ok) and "you are an idiot" (probably not ok). I don't believe in such behavior, and I think it's even more problematic for Linux given so many volunteers who could choose to do something else if pushed the wrong way. Fortunately, my own experience with Linux has been largely positive.

To get started on Linux kernel development, I'd subscribe to lkml and other lists, then offer to code review and test patches that you see posted. A lot of people are writing code, but fewer offering to help code review and test (and write docs). This should be an easy way to get started, build some credibility, and make valuable contributions. Sometimes good patches are posted and slip through the cracks, so replying with "I tested it, it works, thanks!" can help get things integrated, and the engineers will be grateful for your help.

Community & Experts

The Linux community is massive. Here are areas you should know about:

- kernelnewbies: Posts changelogs for each Linux release (eventually), highlighting major and minor additions.
- lkml: The Linux Kernel Mailing List. This is the final staging ground for patches to be integrated into Linux, so following this will let you see what's happening right now. Be warned that it's high volume, and there are only a few reasons you should ever post there: 1. you are submitting a patch set and the MAINTAINERS file told you to CC lkml; 2. you are providing constructive expert comments on someone else's patch set, ideally after you tested it; or 3. you're running the latest -rc from kernel.org (or github/torvalds/linux) and hit a bug/panic.
- lwn.net: The best news feed of what's happening in Linux. It requires a subscription to read the latest articles, but if Linux is going to be a big part of your job, it's worth it.

Many of the experts in the Linux community are maintainers, as listed in the MAINTAINERS file. It's rare to bump into a maintainer: those I know keep their heads down working on lkml and the sublists, and usually avoid blog posts, meetups, and conferences (with some exceptions, like Linux Plumbers, Kernel Summit, NetDev, and Kernel Recipes. Which reminds me: I'm helping run the tracing micro conference at Plumbers this year, and I'm also speaking at Kernel Recipes, so if you manage to make it to either, I'll see you there.) I wish maintainers were more visible, as they can best respond to Linux suggestions and criticism (and bashing).

There was a phenomenon in Solaris where we, the engineers, began doing our own marketing and evangelism, out of desperation to save Solaris. I've never found that happening in Linux, where there's the belief that Linux is too big to fail. I think that is a weakness of Linux. When I first joined Sun in 2001, it was believed that Sun was too big to fail, as well. Nowadays, Sun is a cobweb-covered sign at the Facebook Menlo Park campus, kept as a warning to the next generation.

Documentation

The best documentation is under /Documentation in the kernel source (online at kernel.org/doc or github/torvalds). That documentation is correct but terse, written by the engineers as they commit code.

Full documentation for features, such as would be published by Sun, is often non-existent. Even the major releases can go undocumented for weeks until someone writes a summary on kernelnewbies, which is about as close to official release notes as you can get. I think the problem is a lack of paid tech writers working on Linux. Who would pay them?

This lack of documentation makes learning and discovering new Linux features difficult.

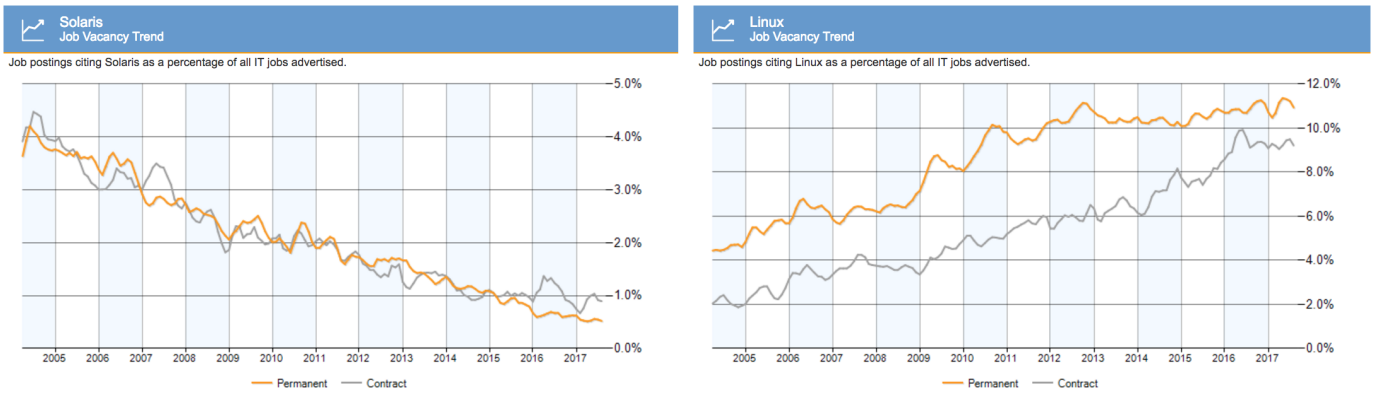
Update: there is a new effort to improve the kernel documentation (using RST and Sphinx), which you can see at www.kernel.org/doc/html/latest.

Other Differences

- **Packaging**: on Ubuntu (and similar) use "apt", on Red Hat (and similar) use "yum". They make it very easy to install packages, and automatically handle dependencies.
- **Driver & Platform Support**: Linux runs on practically everything.
- **Server Application Support**: Linux is usually the development environment for server applications, where things are most likely to work.
- **Desktop Support**: I miss CDE and dtksh. There's a lot of options on Linux, but I'm out of touch with current desktop environments, so can't recommend any.

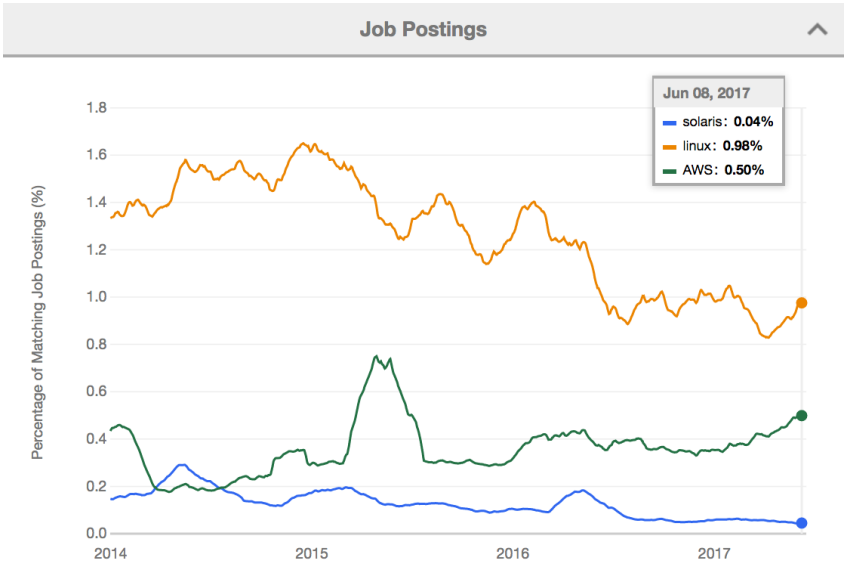
Job Market

The Linux job market Linux has been much healthier for a while and growing. Solaris vs Linux jobs in the UK:



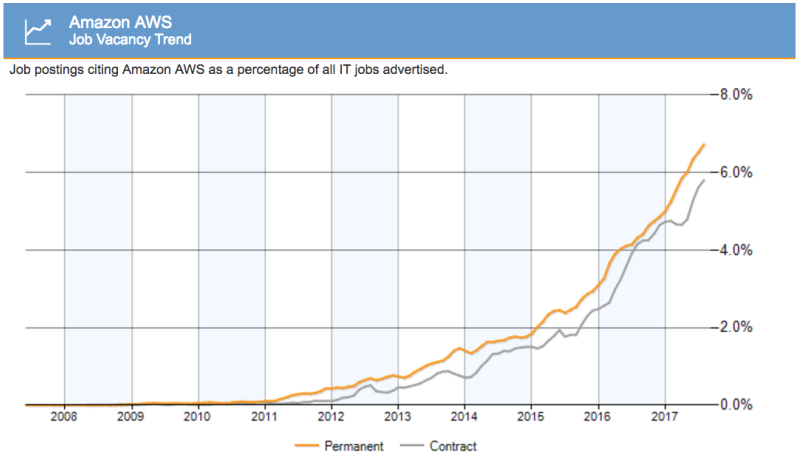
Source: www.itjobswatch.co.uk

But there's another factor at play: jobs are also migrating from both Solaris and Linux to cloud jobs instead, specifically AWS. From another tracker, for the US:



Source: www.indeed.com

The market for OS and kernel development roles is actually shrinking a little. The OS is becoming a forgotten cog in a much larger cloud-based system. The UK tracker plots the growth in AWS jobs clearly:



The job growth is in distributed systems, cloud SRE, data science, cloud network engineering, traffic and chaos engineering, container scheduling, and other new roles. While you might be considering switching to an equivalent Linux or BSD job, you should also consider a new role if that interests you. Leapfrogging to the next big thing was one of Deirdré Straughan's suggestions in Letting Go of a Beloved Technology.

I suspect at some point there'll be more jobs supporting the AWS cloud than there will be supporting Linux. If you choose this route, AWS makes it very easy to create servers (you just need a credit card) and learn how to use them. Which is also why it's been so successful: developers can create servers when they want them, without having to ask and wait for the system administration team. Adrian Cockcroft also wrote about moving from Solaris to AWS in his post Open letter to my Sun friends at Oracle.

As for companies: I can recommend Netflix, which has a culture that works really well.

If you stay working on the OS and kernel, there are still many jobs in support and development, and always will be. Large companies (like the one I work for) have OS teams to look after patching, releases, and performance. Appliance manufacturers hire kernel engineers to develop custom features, including storage appliances. There are several ZFS-based startups, who would appreciate your experience on ZFS.

Good Luck

This is the post I wish someone had written for me when I made the switch. The first few months were the hardest. It gets easier. It will also become easier if you contribute to Linux, or BSD, and fix the annoying things you discover. Solaris may not survive, but certain technologies and expertise will.

Here's the Sun I'd like to remember: lots of smart people, having fun, and doing great work (music might not play outside the US):

RIP, Sun.

Good luck to all, and let the spirit of great engineering live on in the next projects you choose.

References

ZFS

- http://zfsonlinux.org/
- http://www.open-zfs.org/wiki/Main_Page
- https://github.com/zfsonlinux/zfs/issues
- https://btrfs.wiki.kernel.org/index.php/Status
- http://vger.kernel.org/vger-lists.html#linux-kernel
- https://kernelnewbies.org/LinuxVersions

Observability

- http://www.brendangregg.com/linuxperf.html
- http://www.brendangregg.com/USEmethod/use-linux.html
- http://www.brendangregg.com/FlameGraphs/cpuflamegraphs.html

DTrace

- http://www.brendangregg.com/blog/2016-10-27/dtrace-for-linux-2016.html
- https://www.youtube.com/watch?v=GsMs3n8CB6g
- https://github.com/brendangregg/perf-tools
- https://lwn.net/Articles/608497/
- http://www.brendangregg.com/perf.html
- https://github.com/iovisor/bcc/blob/master/tools/zfsdist_example.txt
- https://github.com/iovisor/bcc
- http://www.brendangregg.com/ebpf.html

Zones

- https://blog.jessfraz.com/post/containers-zones-jails-vms/
- https://www.youtube.com/watch?v=bK9A5ODIgac
- https://www.slideshare.net/brendangregg/container-performance-analysis
- https://en.wikipedia.org/wiki/Linux_namespaces
- https://en.wikipedia.org/wiki/Cgroups
- https://www.kernel.org/doc/Documentation/cgroup-v1
- https://www.kernel.org/doc/Documentation/cgroup-v2.txt
- https://github.com/moby/moby/issues/32501

Performance

- http://www.slideshare.net/brendangregg/what-linux-can-learn-from-solaris-performance-and-viceversa

Security

- https://en.wikipedia.org/wiki/AppArmor
- https://en.wikipedia.org/wiki/Seccomp
- https://en.wikipedia.org/wiki/Security-Enhanced_Linux
- https://linux.die.net/man/5/auditd.conf
- https://www.kernel.org/doc/Documentation/networking/filter.txt
- https://en.wikipedia.org/wiki/Iptables
- https://en.wikipedia.org/wiki/Linux_Security_Modules
- https://kernelnewbies.org/LinuxVersions

Crash Dump & Debugging

- http://d3s.mff.cuni.cz/teaching/crash_dump_analysis/slides/08-linux.pdf
- http://www.brendangregg.com/blog/2016-08-09/gdb-example-ncurses.html

- http://undo.io/resources/presentations/cppcon-2015-greg-law-give-me-15-minutes-ill-change/
- http://lldb.llvm.org/lldb-gdb.html

Other Tools & Kernel Differences

- http://bhami.com/rosetta.html
- https://certsimple.com/blog/recreating-unix-rosetta-stone
- https://certsimple.com/rosetta-stone
- http://www.oracle.com/technetwork/server-storage/solaris11/overview/redhat-mapping-guide-1555721.html
- https://docs.oracle.com/cd/E12530_01/oam.1014/b32416/sol_linx.htm
- https://www.slideshare.net/brendangregg/what-linux-can-learn-from-solaris-performance-and-viceversa
- http://www.brendangregg.com/sysperfbook.html

Linux Kernel Engineering

- https://github.com/torvalds/linux/blob/master/Documentation/process/4.Coding.rst
- https://github.com/torvalds/linux/blob/master/Documentation/process/submitting-patches.rst
- https://github.com/torvalds/linux/blob/master/Documentation/process/submit-checklist.rst
- http://halobates.de/on-submitting-patches.pdf
- https://github.com/brendangregg/Misc/blob/master/linux/buildkernel_ec2xenial.sh
- https://github.com/torvalds/linux/blob/master/MAINTAINERS
- http://blog.ffwll.ch/2017/08/github-why-cant-host-the-kernel.html
- https://lkml.org/lkml/2017/9/3/155

Community & Experts

- https://kernelnewbies.org/LinuxVersions
- http://vger.kernel.org/vger-lists.html#linux-kernel
- https://lwn.net/

Documentation

- https://www.kernel.org/doc/Documentation/
- https://www.kernel.org/doc/html/latest/
- https://github.com/torvalds/linux/tree/master/Documentation

Job Market

- https://www.itjobswatch.co.uk/jobs/uk/solaris.do
- https://www.itjobswatch.co.uk/jobs/uk/linux.do
- https://www.itjobswatch.co.uk/jobs/uk/amazon%20aws.do
- https://www.indeed.com/jobtrends/q-solaris-q-linux-q-AWS.html
- http://www.beginningwithi.com/2016/10/08/letting-go-of-a-beloved-technology/
- https://medium.com/@adrianco/open-letter-to-my-sun-friends-at-oracle-updated-from-2010-post-1f8b2bcba693
- http://www.brendangregg.com/blog/2017-05-16/working-at-netflix-2017.html

Thanks to Deirdré Straughan for edits. Now to write a BSD version of this post...

You can comment here, but I can't guarantee your comment will remain here forever: I might switch comment systems at some point (eg, if disqus add advertisements).