The Modern Operating System in 2018

Justin Cormack

docker

Who am I?

Engineer at Docker in Cambridge, UK. Formerly Unikernel Systems.

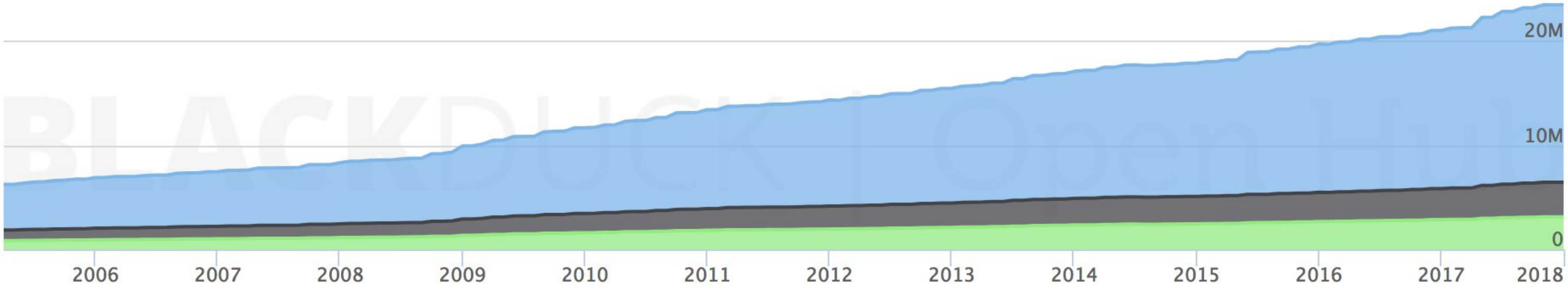
Work on security, systems software, LinuxKit, containers

@justincormack

The last monolith

Lines of code in the Linux kernel

Windows is around 50 million... a Linux distro is over 500 million lines



Windows git repo

- 3.5 million files
- 270GB
- 8,421 pushes per day (on average)
- 2,500 pull requests, with 6,600 reviewers per work day (on average)
- 4,352 active topic branches
- 1,760 official builds per day

The largest git repo on the planet

Declining number of operating systems

- Only three operating systems with significant market share
  - Linux, Android
  - Windows
  - iOS, MacOS
- For server applications only two have significant market share
  - Linux
  - Windows

Everything is wrong with this

- monoculture
- monolith
- everything is written in C or C++
- unrelated to how we do software now

Unikernels: the radical answer

Unikernels

- operating system as a library you link to your application
- boot your application directly on a VM or hardware
- just run your application, nothing else
- specialise everything for a single application
- not monolithic
- pick and choose different implementations from libraries
- pick the language you want to use

Unikernels: successes

- Microsoft shipped SQL Server for Linux as a unikernel
- Growing communities around key projects
  - Mirage (OCaml)
  - IncludeOS (C++)
  - Unik (tooling)
- Many other smaller projects
- Many closed source internal projects
- Come to Felipe Huici talk up next for practical tips
- AMA at 16.05 too!

Change the OS: the incremental answer

The five big changes

1. Performance
2. Operations
3. Portability
4. Scarcity
5. Security

1. Performance

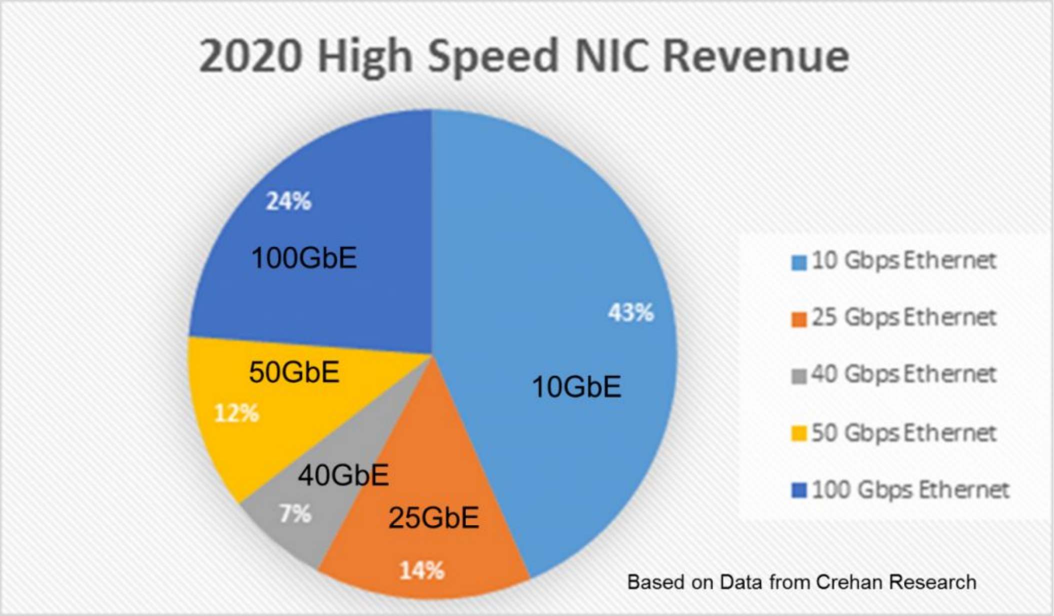
Performance

“A supercomputer is a device for turning compute-bound problems into I/O bound problems.”

Ken Batcher

Storage and network got *much* faster

- cheap 10 gigabit ethernet
- 100 gigabit ethernet
- millions of packets/sec
- SSD, NVMe, NVDIMM
- millions of IO/sec
- IO bandwidth way up
- clock speeds only doubled
- lots of CPU cores



Based on Data from Crehan Research

This is changing everything

- 1Gb ethernet to 100Gb, two orders of magnitude faster
- SSD seek time two orders of magnitude faster than disk

- Back in the early 2000s in memory databases were the big thing
- C10K, 10 thousand connections on a server, was hard
- epoll was invented to fix this, and events not threads

- SSD can now commit at network wire speed
- C10M is possible now
- every CPU cycle counts, 10GbE is up to 14m packets/s
- only 130 clock cycles per packet!

How to fix it 1
Userspace

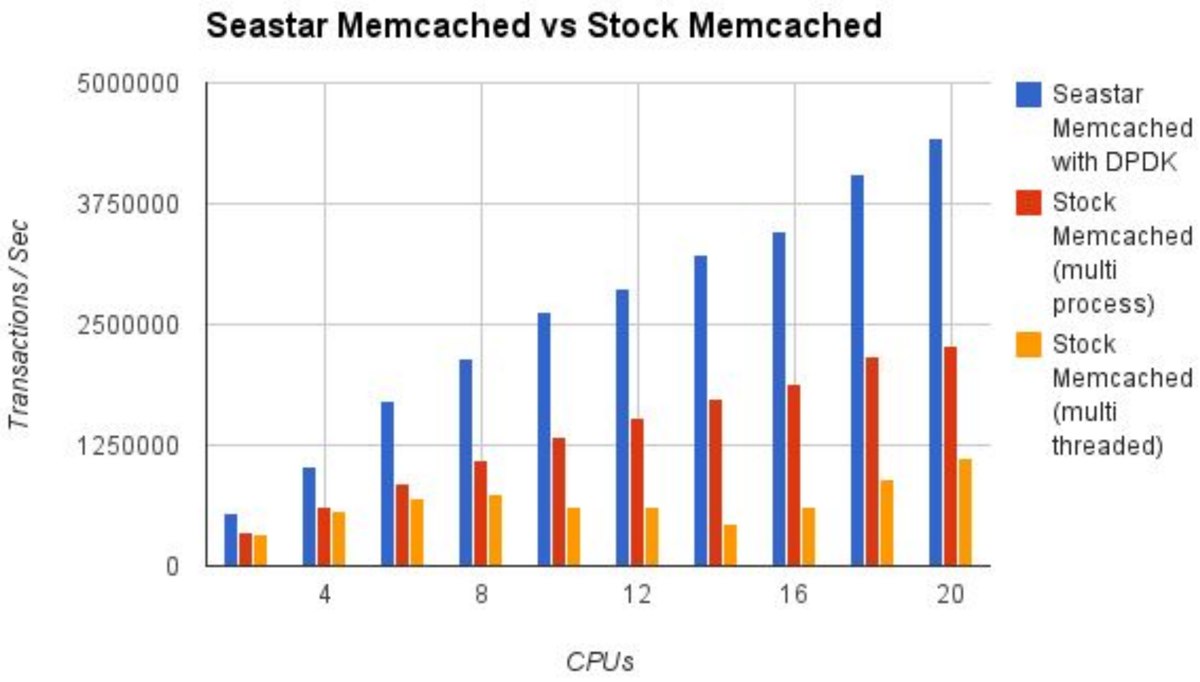
Avoid the kernel userspace switch latency

- system calls are relatively slow
- run all the code in userspace to avoid switches
- minimal use of the kernel!
- involves writing device drivers in userspace
- DPDK (networking) is the most widely used framework
- also SPDK (NVMe), Snabb (networking)
- userspace drivers getting easier, firmware provides higher level API
- eg Mellanox has a single driver API for 10-100Gb ethernet
- NVMe is widespread standard API for storage

Example: SeaStar

- SeaStar: high performance database application
- Originally the company shipped as a unikernel
- Now a framework hosted in Linux but not using much of Linux
- C++
- DPDK
- userspace TCP stack
- no locking, just message passing with ring buffers
- Cassandra, Memcached and Redis compatible backends
- https://github.com/scylladb/seastar

SeaStar performance



How to fix it 2
Kernel space

Never leave the kernel!

- the context switch is too expensive
- put everything in the kernel?
- the kernel was hard to code for though, C code, modules etc
- create a new in kernel programming interface

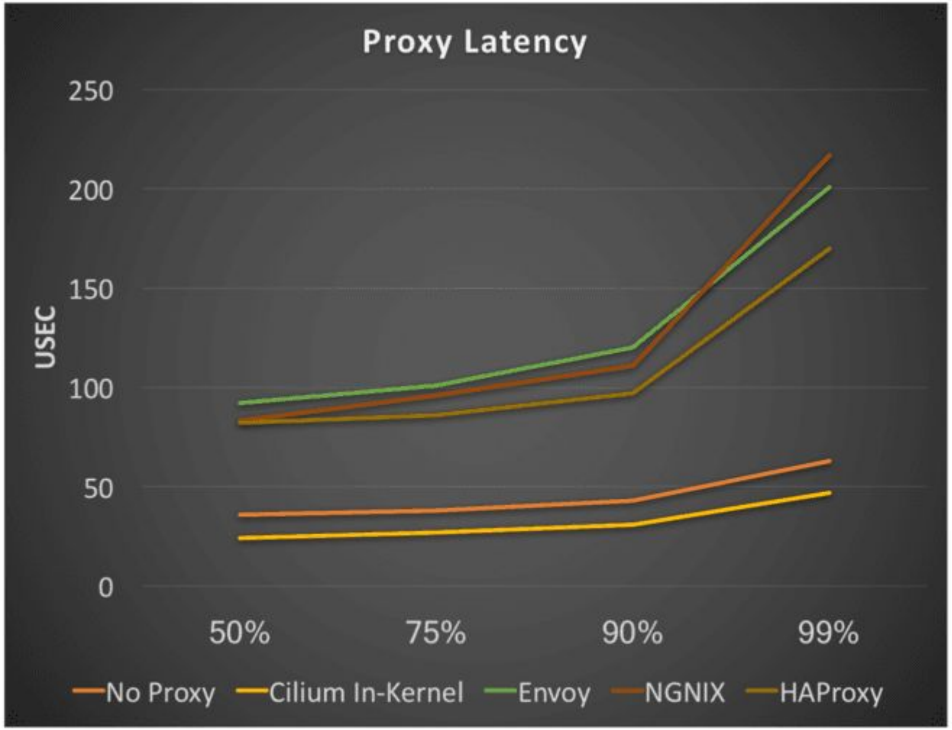
eBPF is AWS Lambda for the Linux kernel

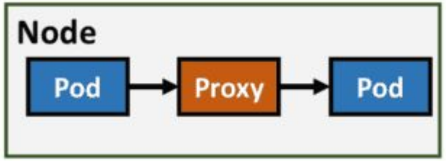
- attach functions to many kernel events
- eBPF is a limited safe language subset, LLVM toolchain
- being extended, eg supports function calls now
- XDP, the network framework is the most advanced part so far
- forwarding, filtering, routing, load balancing

Example: Cilium

- working on a full in kernel datapath for networking
- Linux has in kernel TCP so can terminate in kernel
- Can transparently bypass TCP for local sockets
- Much faster than mixed kernel/userspace dataplane eg Nginx, Envoy
- https://github.com/cilium/cilium

Cilium performance





Notes:

- This excludes parsing logic and policy rule execution
- Istio may route requests through mixer which will add latency

More on eBPF

- See Gilberto Bertin's talk at 2.55 on XDP at CloudFlare
- XDP eXpress Data Path provides a high performance, programmable network data path in the kernel using eBPF
- Networking is the most mature part of the eBPF in kernel stack
- Ready for production on a modern kernel

Choosing one or the other

Kernel space or userspace?

- userspace
  - use any programming language, tooling
  - debugging and programming is more like what you are used to
  - shortage of comprehensive libraries
- kernel space
  - you can reuse much of the Linux kernel infrastructure
  - very limited tooling

Both are getting better fast! Most people doing high performance work are doing one or the other now for new projects.

Unlike the rest of the world

- operating system code not designed for reuse
- not enough system libraries for fast development
- most OS code is in C, developers want C++, Rust, Go, OCaml, ...
- you can borrow code from the BSDs, eg TCP stacks
- unikernels are building those libraries too
- as more people work with these tools they get better!

2. Operations

Cattle not pets

- operations has changed a huge amount too in the decade
- the vast majority of operating systems never have a person log in
- most are created via APIs and automation
- immutable infrastructure: build once, then deploy
- tooling for automated installs not manual tweaking
- move away from the Sun workstation of the 1990s

Immutable delivery at Netflix

“In the cloud, we know exactly what we want a server to be, and if we want to change that we simply terminate it and launch a new server with a new AMI.”

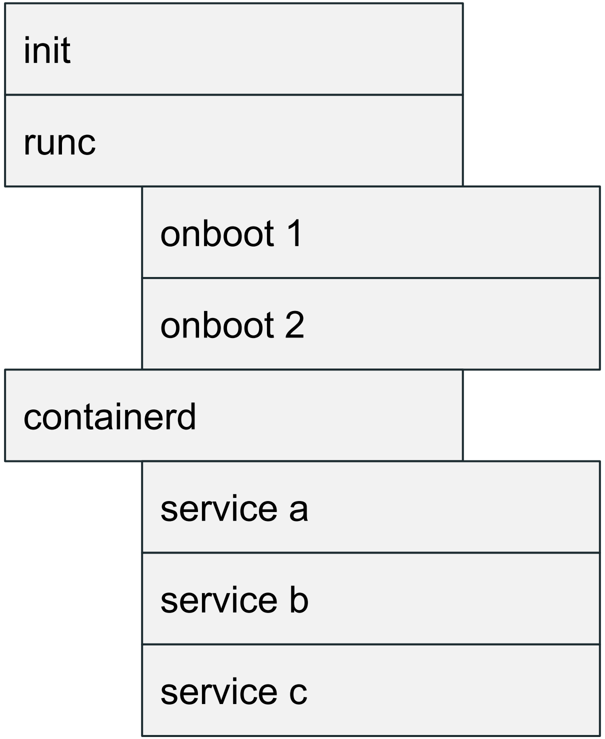
Netflix Building with Legos, 2011

LinuxKit

LinuxKit

- it is a kit, with enough pieces to get you started
- everything can easily be replaced if required
- designed to be built and tested in a CI pipeline
- build times just a minute or so
- test locally then ship to production
- minimal so boots fast
- small so secure and does not need updating so much

LinuxKit startup



init

runc

onboot 1

onboot 2

containerd

service a

service b

service c

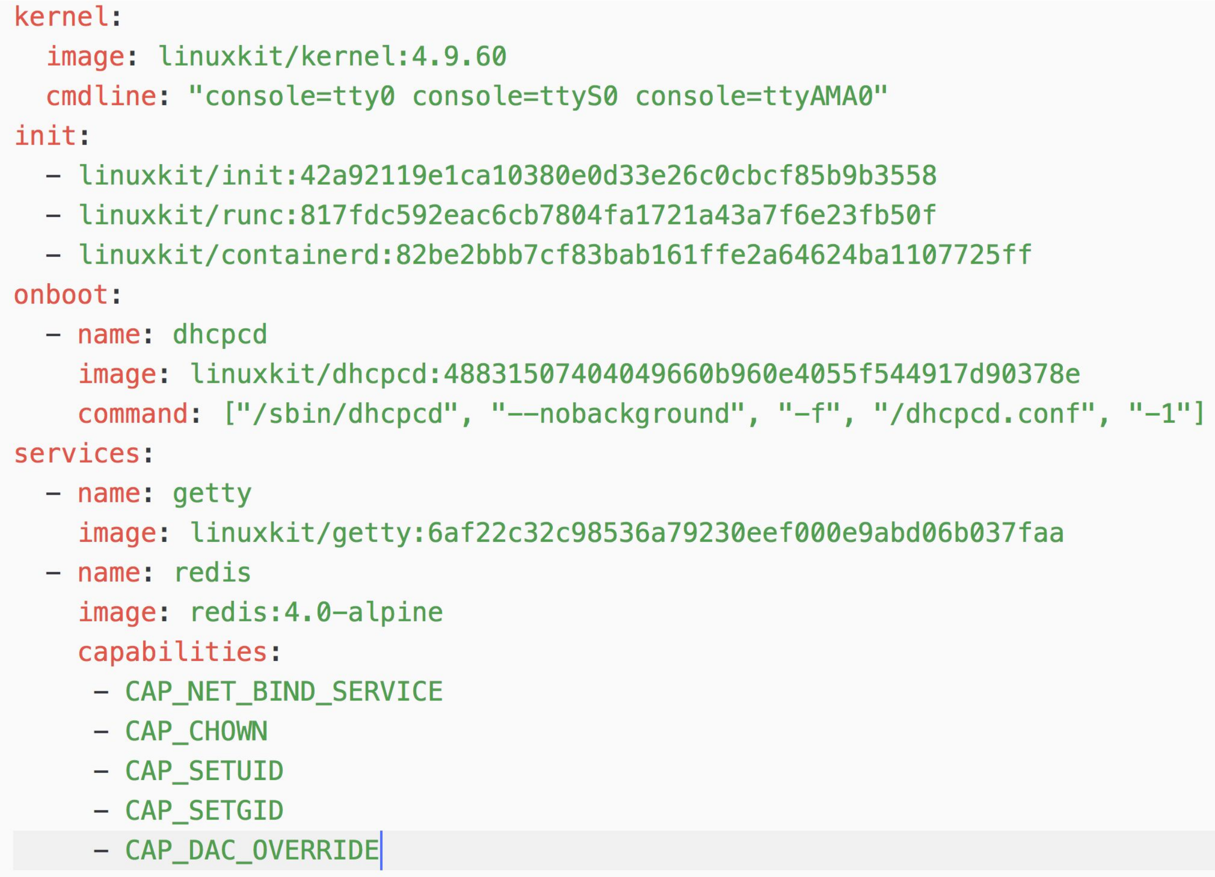
sequential startup

eg network configuration, disks

services start up in parallel after initialization

same design as pods in Kubernetes

Configure this from a yaml file



```yaml
kernel:
  image: linuxkit/kernel:4.9.60
  cmdline: "console=tty0 console=ttyS0 console=ttyAMA0"
init:
  - linuxkit/init:42a92119e1ca10380e0d33e26c0cbcf85b9b3558
  - linuxkit/runc:817fdc592eac6cb7804fa1721a43a7f6e23fb50f
  - linuxkit/containerd:82be2bbb7cf83bab161ffe2a64624ba1107725ff
onboot:
  - name: dhcpcd
    image: linuxkit/dhcpcd:48831507404049660b960e4055f544917d90378e
    command: ["/sbin/dhcpcd", "--nobackground", "-f", "/dhcpcd.conf", "-1"]
services:
  - name: getty
    image: linuxkit/getty:6af22c32c98536a79230eef000e9abd06b037faa
  - name: redis
    image: redis:4.0-alpine
    capabilities:
     - CAP_NET_BIND_SERVICE
     - CAP_CHOWN
     - CAP_SETUID
     - CAP_SETGID
     - CAP_DAC_OVERRIDE
```

important differences

- root filesystem is immutable
- can run from ISO, initramfs, squashfs, ...
- no package manager
- no possibility to update at runtime
- replace with a new image to update software
- if you want dynamic services you use Docker or Kubernetes on top
- removes all complexity of install, update, reboot

Practicalities

Simple tooling for lots of use cases

- Tooling can build most kinds of image needed to boot VMs or bare metal
  - ISO for EFI or BIOS
  - raw disk images
  - AWS AMIs
  - GCP disk format
  - QCOW2 for qemu and KVM
  - VHD
  - VMDK
  - raw kernel and initramfs
  - Raspberry Pi3 image

Simple tooling for lots of use cases

- Simple build, push, run workflow for many common use cases
  - AWS
  - GCP
  - Azure
  - OpenStack
  - Vcenter
  - Packet.net iPXE
  - Hyperkit for MacOS
  - Hyper-V for Windows
  - KVM for Linux
  - VMware Fusion
  - Virtualbox

Simple tooling for lots of use cases

Generally (example Google Cloud)

```
linuxkit build file.yml
linuxkit push gcp filename
linuxkit run gcp filename
```

Some platforms have additional options. You can always use the native tooling to expose all the options.

Demo

3. Portability

docker

Linus decided Linux would have a stable ABI

“If a change results in user programs breaking, it's a bug in the kernel. We never EVER blame the user programs. How hard can this be to understand?”

Linus Torvalds

This gave a stable emulation target

- Linux emulation originally implemented on NetBSD in 1995
- Solaris implementation in 2004
- Ported to FreeBSD in 2006
- Reimplemented and updated on SmartOS in 2015
- Windows Subsystem for Linux introduced in 2016

These have been getting much better, especially WSL

Linux ABI is still large but possible to do this with hard work.

What does this mean?

- non performance critical software can be emulated elsewhere
- this will increasingly be used for security isolation
- for high security applications this will become increasingly important
- also just useful, eg for running existing code on Windows
  - WSL becoming increasingly good
  - integrating better with Windows programs

WSL

- Rich Turner and Tara Raj will be presenting about the Windows Subsystem for Linux at 13.40

4. Scarcity

99¢ ONLY
99 Thanks...!
99¢ ONLY
NOTHING OVER
99¢ EVER
DAYS
99¢

Operating systems used to be multi user

- the design of Unix was around sharing scarce resources
- computers were expensive so shared between people
- now we all have lots of computers which we do not share
- there are only a few "scarce" resources left
  - memory
  - I/O bandwidth
- for most applications these are not the limiting factors
- operating systems do not guarantee what applications actually want
  - tail latency
  - SLAs

Remove scarcity

- virtualize or namespace resources
  - containers have their own filesystem, network
  - an IP address for every container, so port 80 not a scarce resource
  - SR-IOV, everyone gets a PCI device
  - virtual machines, everyone gets a whole computer
- a computer for every application
  - computers are cheap
  - available in many sizes
  - automation makes this more reasonable
- total control of the computer or part of computer in order to
  - control tail latency
  - provide SLA

5. Security

Security was the other drive for unikernels

- security as a driver for operating system change is slow
- Meltdown and Spectre maybe changed that a bit
- prevalent encryption will too, as key management becomes common
- security has yet to change the design space a lot
- secure IoT is one space that will change
- demand for security and privacy slowly rising
  - still denial this will change how applications are built

We still don't have the "Ruby on Rails" of secure application frameworks.
Will it be a unikernel?

Summary

Summary

- Operating systems did change after all!
- Performance meant we created two new ways to run code
  - userspace, self contained using little of OS
  - in kernel eBPF, Lambda for Linux
- Unikernels are being used, if mostly on Linux so far
- Emulation is making code more portable
- Security will lead to the next changes
- The diversity of languages and tooling for systems software is growing

THANK YOU