Hardening Emulated Devices in OpenBSD's vmd(8) Hypervisor

fork&
exec&
fork&
exec.

Dave Voutila <dv@openbsd.org>, AsiaBSDCon 2023

Dave Voutila (dv@)

Vermont 🍁, USA 🇺🇸

(40 mins from Québec 🇨🇦)

Hypervisors as High Value Targets

Why do you rob a bank? It’s where the money is. 💰

- Shift to multi-tenant public cloud means target rich environments
- If it’s networked, it’s vulnerable.
- Pop one ESXi instance, now you own many systems
- “it’s ok, i’m running it in a vm”

A Potpourri of CVE's

Some classic guest-to-host escapes

- QEMU
  - CVE-2015-3456 — "VENOM"
    - aka the QEMU floppy disk one
  - CVE-2015-7504 — network device escape
- VMWare
  - CVE-2020-3967 — EHCI heap overflow
- OpenBSD
  - DHCP packet handler stack overflow (6.8, 6.9)

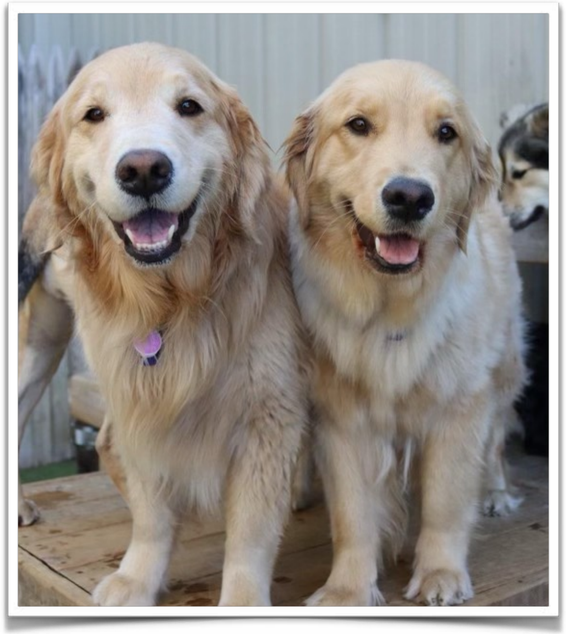
Emulated Devices are the Problem

When in doubt, cut it out. ✂️

- By definition, always handling untrusted (guest) input
- Need read/write to guest physical memory
- Need read/write to host interfaces (network, files, etc.)
- Emulating in kernel is asking for trouble, so commonly done in userland
  - In OpenBSD, only pvclock(4) is handled in the kernel (vmm(4))

Multi-process QEMU

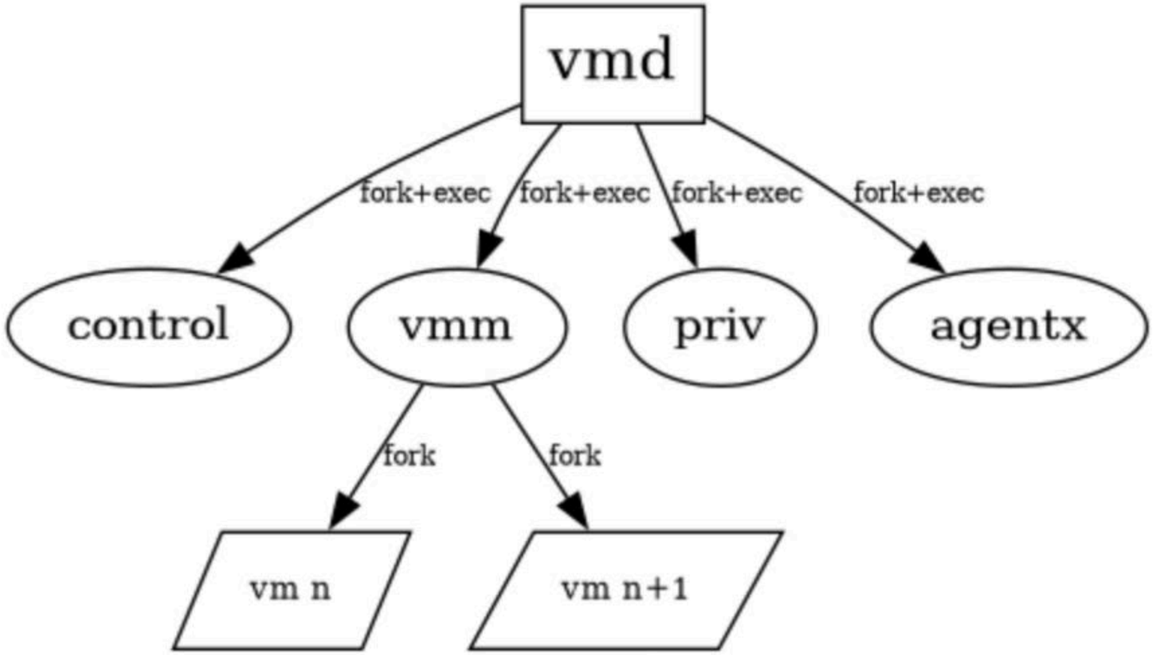
First Type-2 open source hypervisor doing this?



- Oracle started work in 2017, landed in QEMU December 2020
  - Elena Ufimtseva, Jag Raman, John G. Johnson
  - https://lists.gnu.org/archive/html/qemu-devel/2020-12/msg00268.html
- …but, who uses it?
  - I'd presume Oracle Cloud
  - ¯_(ツ)_/¯ 🤷
- Documentation is primarily about design, future ideas, & not present day usage.
  - Additional burden placed upon the poor administrators 😩

vmm(4) / vmd(8)
OpenBSD’s native hypervisor

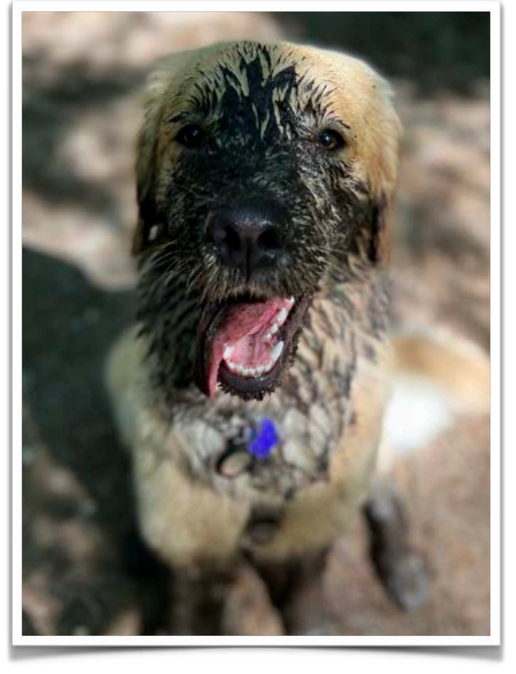
- Originally released with OpenBSD 5.9 by mlarkin@ & reyk@
- Currently amd64 only with support for both amd64 and i386 guests
- Adopted privilege separation design
  - fork+exec —> chroot(2) & pledge(2)
  - drop from root to _vmd
- Components
  - vmm(4) — in kernel monitor
  - vmd(8) — userland daemon
  - vmctl(8) — userland control utility



vmd(8) gaps & weaknesses

Room for improvement

- vm process is fork(2)'d from the vmm process, without execv*(2)
  - Every vm gets same address space layout
  - Every vm has junk left over from vmm process (global state)
- vm process isn't chroot'ed & vmm process isn't running as root
- vm process has multiple pledge(2) promises beyond "stdio"
  - "recvfd" — vm send/recv
  - "vmm" — vmm(4) ioctl for vcpu, interrupts, registers r/w



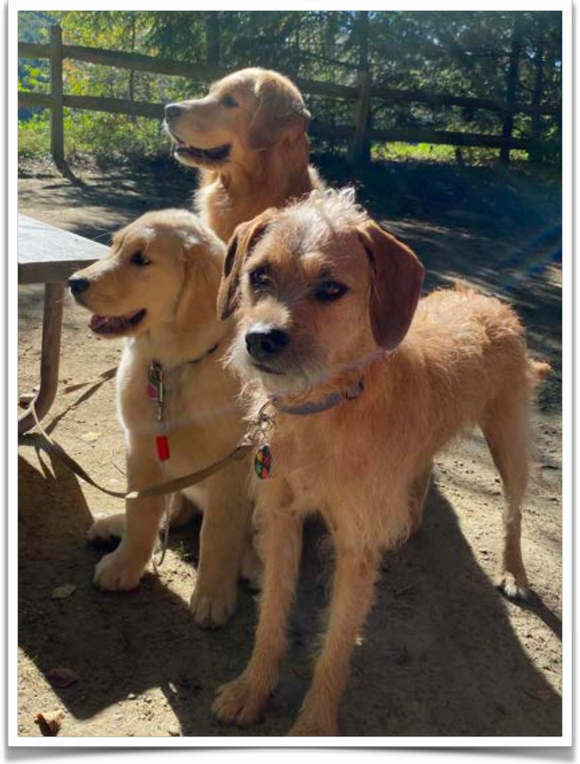
Existing Synthetic Mitigations

A house is only as good as its foundation.

- ASLR — force attackers to need info leaks (3.4)
- RETGUARD — control flow integrity (6.4 for amd64?)
- pledge(2) — minimize syscalls available to attackers (5.9)
- unveil(2) — hide parts of the file system without root (6.4)
- In `-current` and landing as part OpenBSD 7.3:
  - mimmutable(2) — prevent changing page permissions or mappings
  - pinsyscall(2) — minimize allowed call point for syscalls like execve
  - execute-only — enforce execute-only on .text to prevent ROP (amd64 via kernel PK

Step 1: fork+exec for each vm

Minimizing info leaks across vm's



- Switch vmm process from just fork(2) to: fork(2) + execvp(2)
- Easy wins 🙌
  - Reuse socketpair for IPC between vmm proc and vm
  - Simply pass the fd number for the channel in argv
- Headaches 😖
  - vmm process needs absolute path to vmd executable
  - vm process can't rely on existing global state for configuration

Step 2: Isolate the VirtIO Device

“Breaking up is hard to do.” — the Oracle Blog post

- vmd uses multiple `vm_mem_ranges` (bios/reserved, mmio, regular)
- The approach:
  - shm_mkstemp(3) — create temporary file for mapping shared memory
  - ftruncate(2) & shm_unlink(3) — size and remove the temp file
  - fcntl(2) — set the fd to not close on exec
  - mmap(2) guest memory ranges MAP_SHARED | MAP_CONCEAL
  - Pass the fd number after exec & re-mmap `vm_mem_ranges`

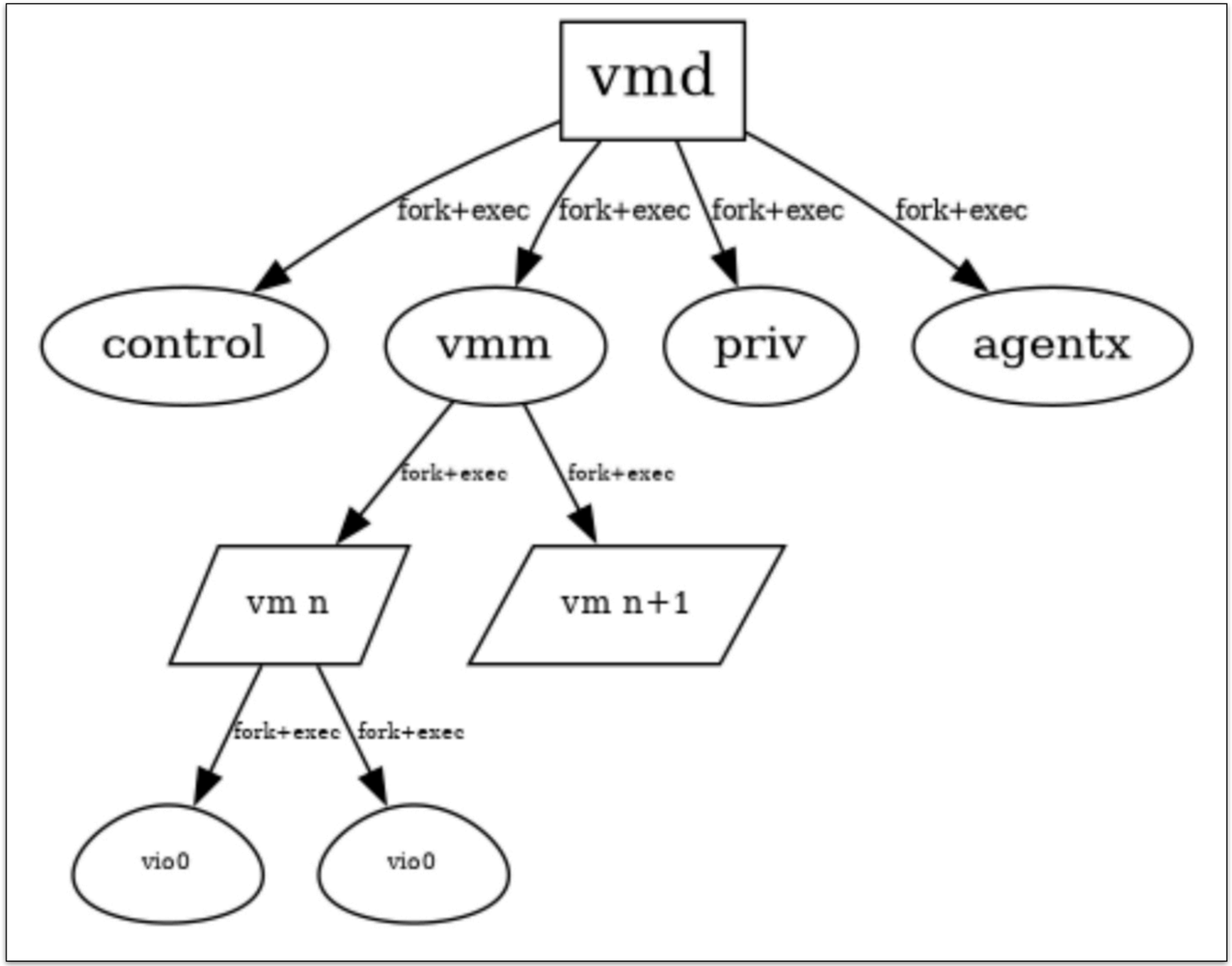
Step 3: Wiring up RPC

Here's a dime…call someone who cares.

- Sync Channel
  - Bootstrapping device config post-exec
  - Virtio PCI register reads/writes
- Async Channel
  - Lifecycle events (vm pause/resume, shutdown)
  - Assert/Deassert IRQ
  - Set host MAC (vionet)

Putting it all Together

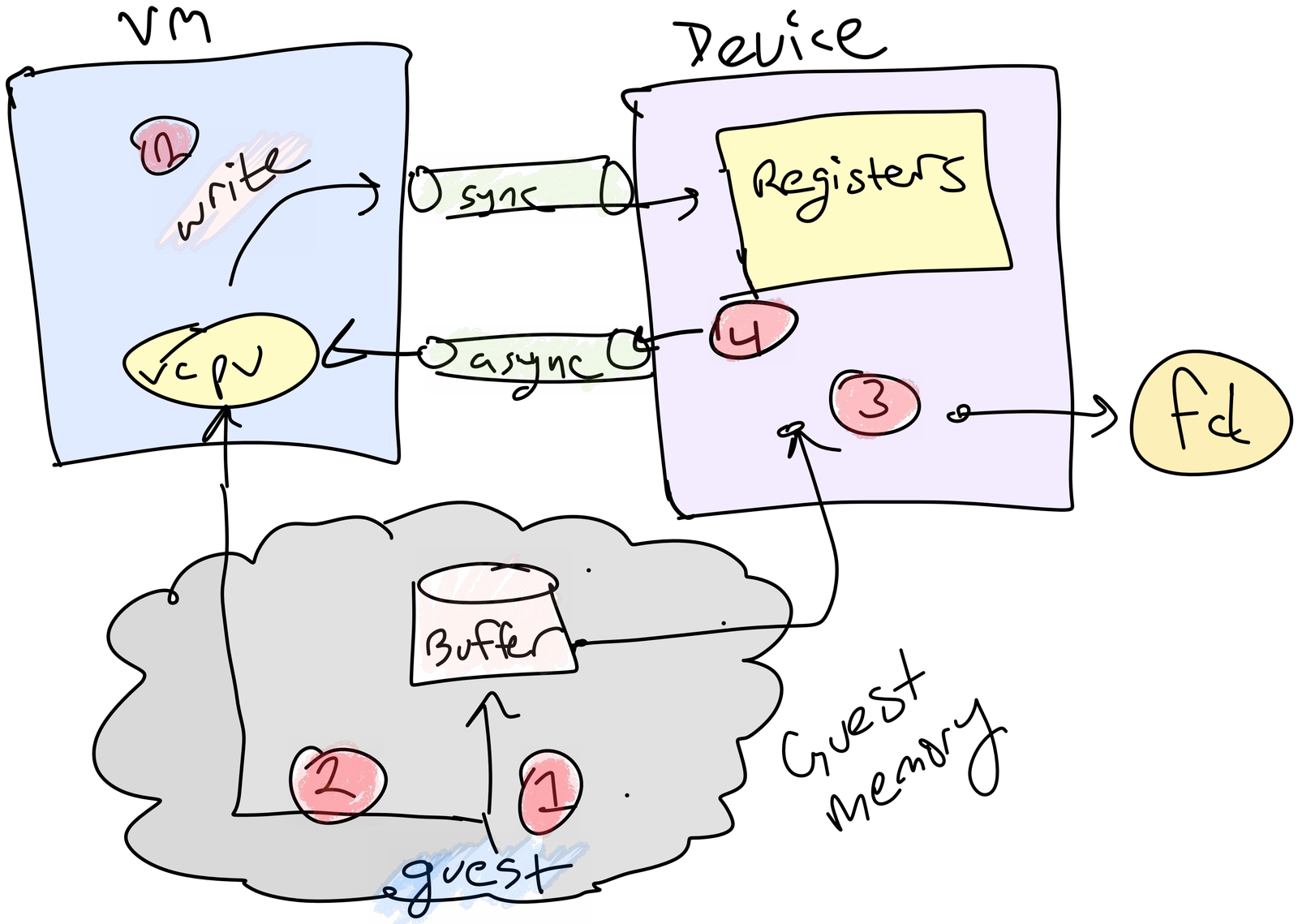
Sorry for my artwork 👨‍🎨



High-level Message Flow

Sorry for my artwork 👨‍🎨

1. Guest fills buffers, updates virtqueues, etc.
2. Guest writes to Device register via IO instructions
3. Device is notified it can process data. Writes it to fd.
4. Device kicks guest via vcpu interrupt to notify buffers are processed



Security! But at what cost?

What about the user/admin experience? Does it change?

- OpenBSD 7.2
  - # vmd -d
  - # vmctl start -Lc -d disk.raw -m 8g guest
- Prototype
  - # vmd -d
  - # vmctl start -Lc -d disk.raw -m 8g guest

Security! But at what cost?

What about the user/admin experience? Does it change?

Security! But at what Cost?

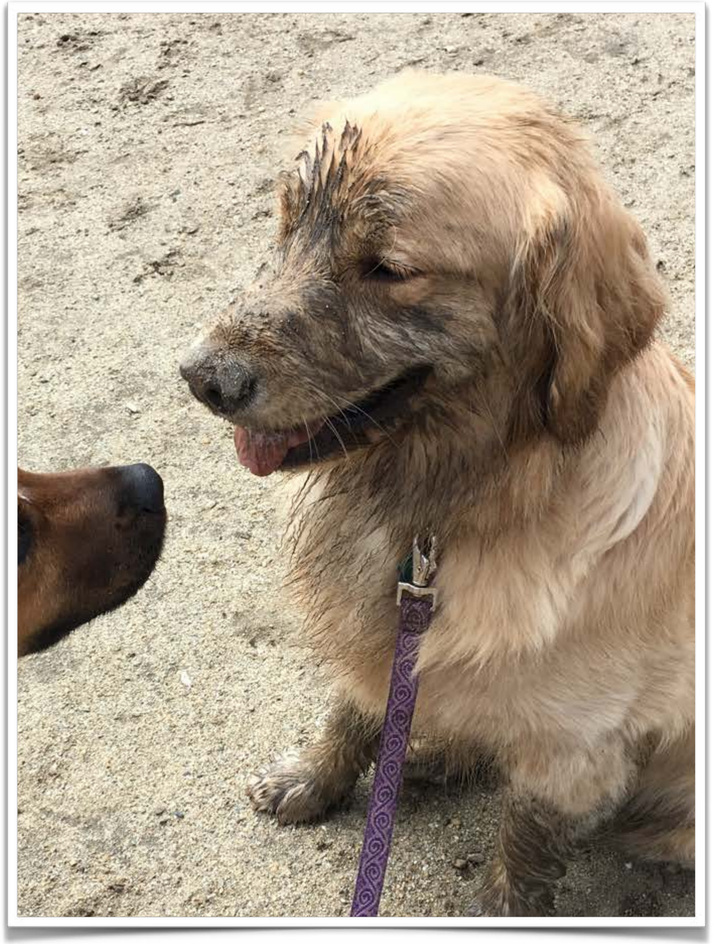
vmd(8) has room for performance improvement

- zero-copy virtio added only recently
- single emulated device thread handles all async io
  - virtio PCI devices — network, disk, cdrom
  - ns8250 serial device
  - i8253 programmable interrupt timer
- Mutexes guard device state from competing vcpu & event threads

Quick & Dirty Benchmarking

This is not a benchmarking talk 🤦

- Lenovo X1 Carbon (10th gen)
  - 12th gen Intel i7-1270P @ 2.2 GHz
  - 32 GiB RAM
  - 1 TB NVME disk
- Guest Operating Systems
  - OpenBSD 7.2
  - Alpine Linux 3.7 (kernel vTKTKT)



vioblk(4) benchmark

fio(1) [ports] performing 8 KiB random writes to 2GiB file for 5 minutes

- Very little difference in throughput
- Very little difference in latency
  - Long-tail slightly worse on Alpine??

Host Version	Guest Version	Throughput MiB/s	clat avg (usec)	clat stdev (usec)	99.90th % (usec)	99.95% (usec)
OpenBSD-current	OpenBSD-current	90.3	89.9	8730	338	379
Prototype	OpenBSD-current	100	76.4	7220	388	429
OpenBSD-current	Alpine Linux 3.17	131	11.2	534	32	38
Prototype	Alpine Linux 3.17	132	11.7	682	594	685

vio(4) benchmark

iperf3(1) [ports] with 1 thread run for 30 seconds, alternating TX/RX

- iperf(3) used with 1 thread in alternating client / server modes
- Observations
  - More consistent throughput in OpenBSD guests
  - Negligible performance decrease for Linux guests (not sure why)

Host Version	Guest Version	Receiving Bitrate (Mbit/s)	%	Sending Bitrate (MBit/s)	%
-current	OpenBSD 7.2	0.86	—	1.26	—
prototype	OpenBSD 7.2	1.22	**63%**	**1.35**	**7%**
-current	Alpine Linux 3.17	1.26	—	4.26	—
prototype	Alpine Linux 3.17	1.30	**5%**	**3.19**	**-25%**

Headaches!

Some things weren't so easy. 😰

- Dual-channel connectivity
  - synchronous register io from vcpu thread needs to avoid deadlocks
- vcpu vs. event thread isolation
  - libevent(3) is not thread-safe (OpenBSD bundles v1.x)
- Debugging multi-process, async code is often challenging
  - printf & ktrace can quickly generate lots of noise
  - nanosleep(2) + gdb attaching directly to device process helps

Future Work & Research

Plans for the next hackathon (m2k23) 📋

- Finish lifecycle cleanup (vm send/receive)
- QCOW2 disk support — only supports raw images at the moment
- Begin merging changes into tree after 7.3 release
  - vm fork+exec dance
  - vioblk
  - vionet
- Expand to other devices
  - ns8250?
- Tighten exposure of guest physical memory
  - guest aware drivers? (is there anything to gain by limiting how much of guest memory we remap?)

Thanks!

See you in Ottawa?