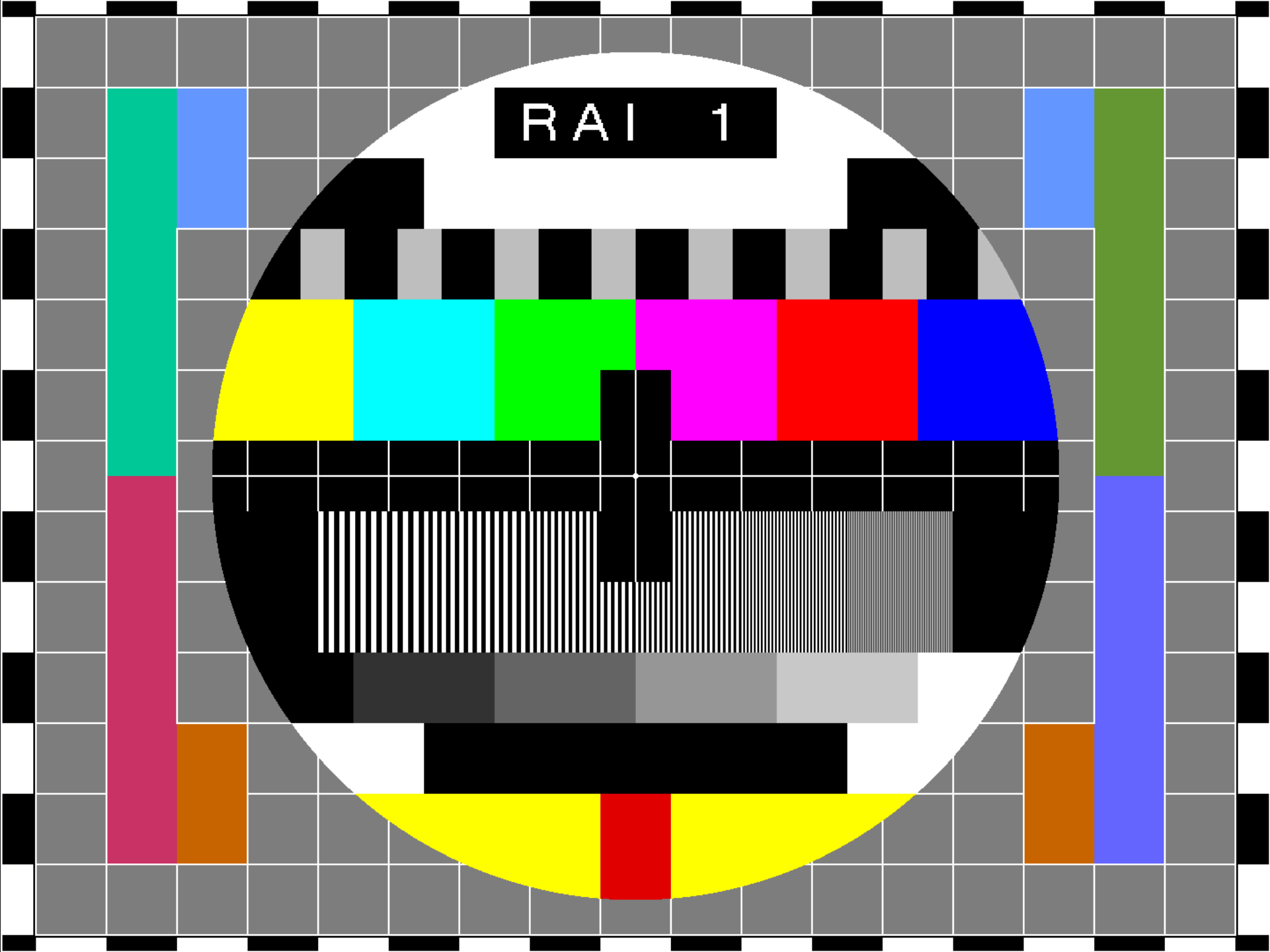


RAI 1






tiny VMs for kernel development

Rob Norris, Klara Inc.

Hello!

-  Australian
- Klara, Inc.
- OpenZFS developer
- Recovering Linux sysadmin
- FreeBSD non-committer



exploratory programming

```
shell: ~/code/quiz
quiz on | main [!+]
$ ./quiz-prepare-kernel
[quiz-prepare-kernel] 20240531-15:01:00 FATAL no kernel version supplied; one of -k or -K required

quiz on | main [!+]
X1 $
```

```
quiz-prepare-kernel (~/.code/quiz) - VIM
~/usr/bin/env bash

# This Source Code Form is subject to the terms of the Mozilla Public
# License, v. 2.0. If a copy of the MPL was not distributed with this
# file, You can obtain one at http://mozilla.org/MPL/2.0/.

# Copyright (c) 2023, Rob Norris <robn@despairlabs.com>

set -uo pipefail

usage() {
    cat <<EOF
compile a kernel optimised for the quiz microvm
usage: quiz-prepare-kernel [opts]

options:
-k <kernel>      kernel to build
                  (can be used multiple times)
-e/-d/-m <key>  set kernel config key to enable/disable/module (Y/N/M)
                  (can be used multiple times)
-u              update standard config for this kernel minor version
-L            build with LLVM/Clang
-C            don't install kernel config before build
-B            don't build kernel
-K            build latest of all kernel series with available config
-h            this help
EOF
    exit 1
}

trace() {
    local STAMP=$(date +%Y%m%d-%H:%M:%S)
    echo "[quiz-prepare-kernel] $STAMP $@" >&2
}

fail() {
    trace "FATAL $@"
    exit 1
}
trap fail ERR

RUNDIR=$(realpath $(dirname $0))
source $RUNDIR/quiz-config
source $RUNDIR/quiz-lib

opt_kernel=""
opt_no_config=""
opt_no_build=""
opt_llvm=""
opt_update_config=""
opt_modify_config=""
opt_all_kernels=""

OPTIND=1
while getopts "k:CBe:d:m:uLKh" opt
NORMAL quiz-prepare-kernel bash 0% ln:1/200=1:1
"quiz-prepare-kernel" 200L, 5499B
```

exploratory kernel programming

- every crash is a reboot
- every deadlock is a reboot
- boot times are slow
- unclean shutdown damages filesystems
- traditional VMs are a pain to manage if you're blowing them up all the time
- I get bored and distracted very easily

Big thoughts

- We run programs in modified environments all the time:
 - alternate environment: `env VAR=val /some/program`
 - alternate filesystem: `chroot /some/path /some/program`
 - alternate language: `bash /some/program.sh`, `perl /some/program.pl`
- If you squint:
 - a hypervisor is just a program that runs a kernel
 - a kernel is just a program that runs a program called `init`
 - `init` is just a program that runs another program

Big thoughts

```
$ zfs-kernel-runner my-zfs-test-script.sh
```


Squad goals

- Feels just like another program
 - Output to stdout, so we can grep it
 - Ctrl-C will kill it
- Gets into the test program in a couple of seconds
- Completely gone without a trace when it completes
- Minimal extra typing
- Get new code and test programs direct from the host filesystem



quiz

<https://github.com/robn/quiz>



quiz

- QEMU `microvm` profile
- Custom build of Linux kernel
- Minimal Debian userspace
- Custom boot process
- 9pfs+overlayfs to build the root filesystem
- Run profiles to add devices or facilities to this run
- OpenZFS build support

QEMU `microvm` profile

- A minimalist /x86_64amd64 machine model
 - yes: ISA bus, LAPIC, IOAPIC, clock, virtio-mmio slots
 - no: PCI bus, ACPI, option ROMs, ISA serial, PIC, PIT, RTC
- Fast boot: nothing to discover, nothing to initialise
 - Known, fixed, minimal set of devices

Custom kernel build

- Bare minimum device support
 - no PCI bus? no PCI support needed!
 - no time lost enumerating bus
- All drivers built into kernel, no modules
 - No initrd required to boot!

Minimal Debian userspace

- `minbase` variant: "required" packages + package manager
 - (sort of like `base.tgz`)
- plus useful tools for this task:
 - performance and profiling: `perf`, `bpftrace`, `fio`, `gdb`, `strace`, `blktrace` ...
 - block device construction: `gdisk`, `dmsetup`, `cryptsetup`, ...
 - OpenZFS test suite support: `ksh`, ...
 - Boot support: `tini`, `udev`, `kmod`, ...

Custom boot process

- `init1`: first stage; build the root filesystem, pivot
- `init2`: second stage; prepare environment, run `tini`
- `tini`: bare-minimum PID 1, run the test program

demo #1

basic operation



`init1`: filesystem construction

- base system image: ext2
 - Debian `minbase` + extras
 - `init1` is built into this image
- quiz init dir: host dir via 9pfs
 - script fragments, config, etc created by `quiz` script for this run
- quiz system dir: host dir via 9pfs
 - install target for OpenZFS, built outside
- quiz user dir: host dir via 9pfs
 - test scripts and other random stuff I want inside
- top: tmpfs
 - so writes inside the VM can work, and disappear later

`init2`: prepare environment

- set the hostname
- get `/dev` nodes up (`udev`)
- mount debug filesystems
- exec `tini` as PID 1, which runs either a shell or the requested program

tini: the littlest PID 1 that could

<https://github.com/krallin/tini>

- runs a program
- reaps zombies
- provides default signal handlers
- the "standard" PID 1 for containers

quiz: profiles

- add extra stuff to this run
- profiles can:
 - run stuff on the host, before the VM starts
 - run stuff in the guest, before the user program starts
 - provide extra files that will be included in the guest

quiz: zfs profile

- host: run depmod to ensure module linkage is correct
- guest: install `zfs` module

quiz: memdev profile

- guest: create small memory-backed block devices

quiz: blockdev profile

- host:
 - create 1G sparse files as block device backing
 - extend `qemu` command line to attach them as virtio-blk devices

quiz: ztest profile

- files: provide "no-op" variants of `sudo` and `id` to work around ZTS assumptions

 demo #2

profiles



OpenZFS build support

OpenZFS build support

The dream:

```
$ ./autogen.sh  
$ ./configure --prefix=/path/to/quiz/system  
$ make -j6  
$ make install
```

OpenZFS build support

The reality:

```
$ ./autogen.sh
$ ./configure \
  --with-linux=/path/to/quiz/build/kernel/linux-x.y.z \
  --with-linux-obj=/path/to/quiz/build/kernel/linux-x.y.z \
  --prefix=/usr/local \
  --disable-sysvinit \
  --disable-systemd \
  --disable-pam \
  'lt_cv_sys_lib_dlsearch_path_spec=/lib /usr/lib /lib/i686-linux-gnu /usr/lib/x86_64-linux-gnu' \
$ make -j6
$ make install DESTDIR=/path/to/quiz/system
```

OpenZFS build support

For now:

```
$ ./autogen.sh  
$ quiz-build-zfs configure  
$ make -j6  
$ quiz-build-zfs make install
```

Kernel features

- multiple kernels, selectable with `-k`
- oneshot kernel builds with changed options
- GCC and LLVM/Clang variants

Plans and dreams

- multiple architectures (proper ZFS big-endian testing)
- writable host mount (save logs and build artifacts)
- profiles for building block devices out of dm stacks
- remote tmux (mess with program run live)
- multiple instances, for each OpenZFS checkout
- and...



FreeBSD support

FreeBSD guest support

- `FIRECRACKER` kernel config
- Cross-build on Linux host
- Missing: 9pfs (coming soon!)
- Missing: overlays (coming soon?)

FreeBSD host support

- Needs hypervisor support, either:
 - `qemu` needs hardware acceleration
 - `bhyve` needs support for one-shot, diskless Linux VMs

Direct Linux boot



Direct Linux boot

```
$ qemu-system-x86_64 \  
-nodefaults -no-user-config -nographic \  
-enable-kvm -cpu host -smp 2 -m 1G \  
-serial stdio \  
-kernel /boot/vmlinuz-6.1.0-21-amd64 \  
-append 'console=ttyS0'
```



Direct Linux boot

```
$ qemu-system-x86_64 -nodefaults -no-user-config -nographic \  
-enable-kvm -cpu host -smp 2 -m 1G -serial stdio \  
-kernel /boot/vmlinuz-6.1.0-21-amd64 -append 'console=ttyS0'  
[ 0.000000] Linux version 6.1.0-21-amd64 (debian-kernel@lists.debian.org)  
          (gcc-12 (Debian 12.2.0-14) 12.2.0, GNU ld (GNU Binutils for Debian) 2.40)  
          #1 SMP PREEMPT_DYNAMIC Debian 6.1.90-1 (2024-05-03)  
[ 0.000000] Command line: console=ttyS0  
[ 0.000000] BIOS-provided physical RAM map:  
[ 0.000000] BIOS-e820: [mem 0x0000000000000000-0x0000000000009fbff] usable  
[ 0.000000] BIOS-e820: [mem 0x0000000000009fc00-0x0000000000009ffff] reserved  
...  
[ 0.511354] List of all partitions:  
[ 0.511853] No filesystem could mount root, tried:  
[ 0.511854]  
[ 0.512705] Kernel panic - not syncing: VFS: Unable to mount root fs on unknown-block(0,0)  
[ 0.513862] CPU: 0 PID: 1 Comm: swapper/0 Not tainted 6.1.0-21-amd64 #1 Debian 6.1.90-1  
[ 0.514940] Hardware name: QEMU Standard PC (i440FX + PIIX, 1996),  
          BIOS 1.16.2-debian-1.16.2-1 04/01/2014  
[ 0.516269] Call Trace:  
[ 0.516619]  <TASK>  
[ 0.516932]  dump_stack_lvl+0x44/0x5c  
[ 0.517450]  panic+0x118/0x2f4  
[ 0.517932]  mount_block_root+0x1d3/0x1e6  
[ 0.518516]  prepare_namespace+0x136/0x165  
[ 0.519078]  kernel_init_freeable+0x25c/0x286  
[ 0.519633]  ? rest_init+0xd0/0xd0  
[ 0.520088]  kernel_init+0x16/0x130  
[ 0.520571]  ret_from_fork+0x1f/0x30  
[ 0.521059]  </TASK>  
[ 0.521486] Kernel Offset: 0x2fe00000 from 0xffffffff81000000  
          (relocation range: 0xffffffff80000000-0xffffffffbfffffff)  
[ 0.522815] ---[ end Kernel panic - not syncing: VFS: Unable to mount root fs on unknown-block(0,0) ]---
```

bhyve: direct Linux boot

```
$ bhyve -DPHA -c 2 -m 1G \  
-s 0,hostbridge -s 1,lpc -l com1,stdio \  
...
```

bhyve: boot process

- allocates a big chunk of memory
- maps a "BIOS" boot ROM device into that space
- points the first CPU at it
- standard PC boot: find boot device, load the bootloader, ...


bhyve: boot process

- allocates a big chunk of memory
- *puts stuff in memory*
- points the first CPU at it
- standard PC boot: find boot device, load the bootloader, ...

bhyve: boot process

- allocates a big chunk of memory
- *puts stuff in memory*
- *sets CPU state to match*
- standard PC boot: find boot device, load the bootloader, ...

bhyve: boot process

- allocates a big chunk of memory
- *puts stuff in memory*
- *sets CPU state to match*
- *go!* 

bhyve: loader infrastructure

```
$ bhyve -DPHA -c 2 -m 1G \  
-s 0,hostbridge -s 1,lpc -l com1,stdio \  
-o loader.name=linux \  
-o loader.kernel=vmlinuz-6.1.0-21-amd64 \  
-o loader.cmdline='console=ttyS0'
```



bhyve: loader infrastructure

```
$ bhyve -DPHA -c 2 -m 1G -s 0,hostbridge -s 1,lpc -l com1,stdio \  
-o loader.name=linux -o loader.kernel=vmlinuz-6.1.0-21-amd64 -o loader.cmdline='console=ttyS0'  
[ 0.000000] Linux version 6.1.0-21-amd64 (debian-kernel@lists.debian.org)  
          (gcc-12 (Debian 12.2.0-14) 12.2.0, GNU ld (GNU Binutils for Debian) 2.40)  
          #1 SMP PREEMPT_DYNAMIC Debian 6.1.90-1 (2024-05-03)  
[ 0.000000] Command line: console=ttyS0  
[ 0.000000] BIOS-provided physical RAM map:  
[ 0.000000] BIOS-e820: [mem 0x0000000000000000-0x0000000000009ffff] usable  
[ 0.000000] BIOS-e820: [mem 0x0000000000100000-0x0000000003fffffff] usable  
...  
[ 0.543927] List of all partitions:  
[ 0.544070] No filesystem could mount root, tried:  
[ 0.544071]  
[ 0.544329] Kernel panic - not syncing: VFS: Unable to mount root fs on unknown-block(0,0)  
[ 0.544617] CPU: 0 PID: 1 Comm: swapper/0 Not tainted 6.1.0-21-amd64 #1 Debian 6.1.90-1  
[ 0.544862] Hardware name: FreeBSD BHYVE/BHYVE, BIOS 14.0 10/17/2021  
[ 0.545056] Call Trace:  
[ 0.545139] <TASK>  
[ 0.545212] dump_stack_lvl+0x44/0x5c  
[ 0.545332] panic+0x118/0x2f4  
[ 0.545436] mount_block_root+0x1d3/0x1e6  
[ 0.545565] prepare_namespace+0x136/0x165  
[ 0.545695] kernel_init_freeable+0x25c/0x286  
[ 0.545837] ? rest_init+0xd0/0xd0  
[ 0.545949] kernel_init+0x16/0x130  
[ 0.546062] ret_from_fork+0x22/0x30  
[ 0.546180] </TASK>  
[ 0.547128] Kernel Offset: 0x32800000 from 0xffffffff81000000  
          (relocation range: 0xffffffff80000000-0xffffffffbfffffff)  
[ 0.547455] ---[ end Kernel panic - not syncing: VFS: Unable to mount root fs on unknown-block(0,0) ]---
```



Boot protocol

- A kernel is just a program
- The CPU jumps to it and starts executing

- What is the memory layout?
- Where are the basic devices (clocks, buses, ...)
- Where are the commandline args?
- Where is the support code? (initrd, loader, drivers, ...)



Linux x86 64-bit boot protocol

- Copy header template from image
- Sanity checks
 - Magic number
 - Protocol version (2.02+)
- Compute compressed kernel offset within image
- Select memory location, load compressed kernel
- Select memory location, copy command line in
- Select memory location, load initrd image (if required)



Linux x86 64-bit boot protocol

- Fill out header
 - Set loader type ("undefined" 0xff)
 - Set commandline start/length
 - Set initrd start/length (if required)
 - Install e820 memory map



Linux x86 64-bit boot protocol

- Setup registers
 - GDT : 4G each CODE , DATA , TSS
 - IDT : zero
 - CS : GDT[CODE]
 - DS , ES , FS , GS , SS : GDT[DATA]
 - TR : GDT[TSS]
 - PDE , PDPTE , PML4 : identity page table
 - CR0 , CR3 , CR4 , EFER : 64-bit long mode, paging enabled
 - EFLAGS : interrupts disabled
 - RIP : 64-bit entry point: kernel load address + 0x200
 - RSI : header start
 - RSP , RBP : initial stack



bhyve: multiboot2 loader

```
$ bhyve -DPHA -c 2 -m 1G -s 0,hostbridge -s 1,lpc -l com1,stdio \  
-o loader.name=multiboot2 -o loader.image=hobby-os  
...
```

bhyve quiz: TODO

- loader infrastructure + Linux loader: aiming for FreeBSD 15
- unprivileged bhyve: 15
- anonymous VMs: ...?
- 9pfs: 15
- overlayfs: ...?



QEMU: bhyve/vmm acceleration

- QEMU currently uses software CPU acceleration on FreeBSD
- `vmm.ko` (bhyve kernel component) fundamentally incompatible
 - QEMU wants to allocate and initialise memory and map devices itself, then hand that to the accelerator
 - `vmm.ko` expects to allocate memory and map devices, and give them to userspace to use
- So...?
 - Remake `vmm.ko` the "right" way?
 - Port `nvmm(4)` from NetBSD?
 - Just add more devices and things to `bhyve` so we don't need `qemu`?

kernels are
just programs
do not listen
to their bulls••t