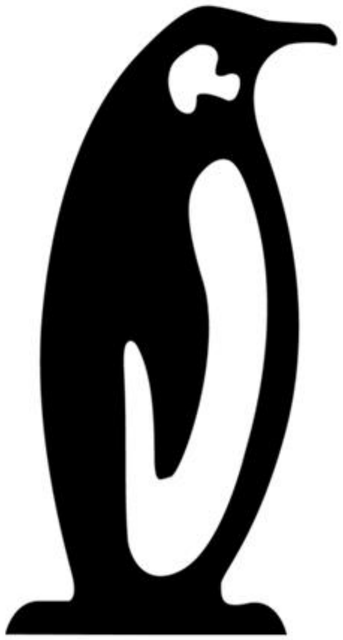


Linux Plumbers Conference

Vienna, Austria | September 18-20, 2024

OpenHCL: A Linux and Rust based paravisor

Chris Oo – Microsoft

LINUX
PLUMBERS
CONFERENCE Vienna, Austria / Sept. 18-20, 2024

Microsoft

What is a paravisor?

- Firmware component that runs inside the guest at a higher privilege level
- Provide emulation for unenlightened guests on CVM platforms
  - APIC emulation and interrupt virtualization
- Provide services for guests
  - vTPM
  - Legacy emulated devices like serial
  - Device translation

Why have a paravisor?

- Run guests that are not fully enlightened such as Windows and older Linux
- Provide security isolation for guests that are not fully hardened
- Provide emulated devices such as vTPM, serial
- Provide device translation
  - Translate NVMe to paravirtualized storage
- Host debuggers and diagnostic processes
  - Allow debugging guests where traditional debuggers are hard like CVMs

Why run vmm code in a paravisor?

- Move emulated devices inside into the guest
  - Require guest to host compromise outside of compromising an emulated device
- Support legacy OSes with accelerated devices assigned to a guest
  - Translate an assigned NVMe device to emulated IDE inside the paravisor
- Share confidential and non-confidential architecture
  - Run the same VMM in the same environment for both

OpenHCL overview

- Linux and usermode Rust based paravisor from the OpenVMM project
- Runs at a higher privilege level
  - VTL2 on Hyper-V
  - L1 VM on Intel TDX
  - VMPL0 on AMD SEV-SNP
- Provides various services to the guest
  - CVM enlightenments and support
  - Emulated device support
  - Device translation such as from NVMe to paravirt storage
  - Diagnostics

OpenHCL features

- Supports Hyper-V isolation (VTLs) on x86-64 and ARM64, AMD SEV-SNP, Intel TDX
- Supports device translation
- Supports vTPM
- Supports various legacy device emulators such as serial
- Supports Hyper-V legacy bios, Hyper-V UEFI and Linux direct boot guests

Usage in Azure

- Used in new Azure Boost SKUs
- Meets storage and networking performance requirements
- Used in over 10M cores and counting

Design philosophy

- Track upstream kernel
  - Aim to upstream all kernel patches or have a path to upstream
- Do as much in usermode as possible
  - Host the VMM itself in usermode
  - Device drivers in usermode
- Do as much in safe idiomatic rust as possible
- Rust async-focused usermode VMM
- Keep VMM code OS agnostic
  - Allows for running outside of OpenHCL

Why Rust?

- Prevent whole classes of memory safety issues
  - Borrow checker
  - Send & Sync traits
- Modern language with ecosystem of useful crates
  - Traits
  - Async
  - Modern tooling with rust-analyzer
- Still provides enough low level control
  - Able to use C APIs and talk directly with hardware

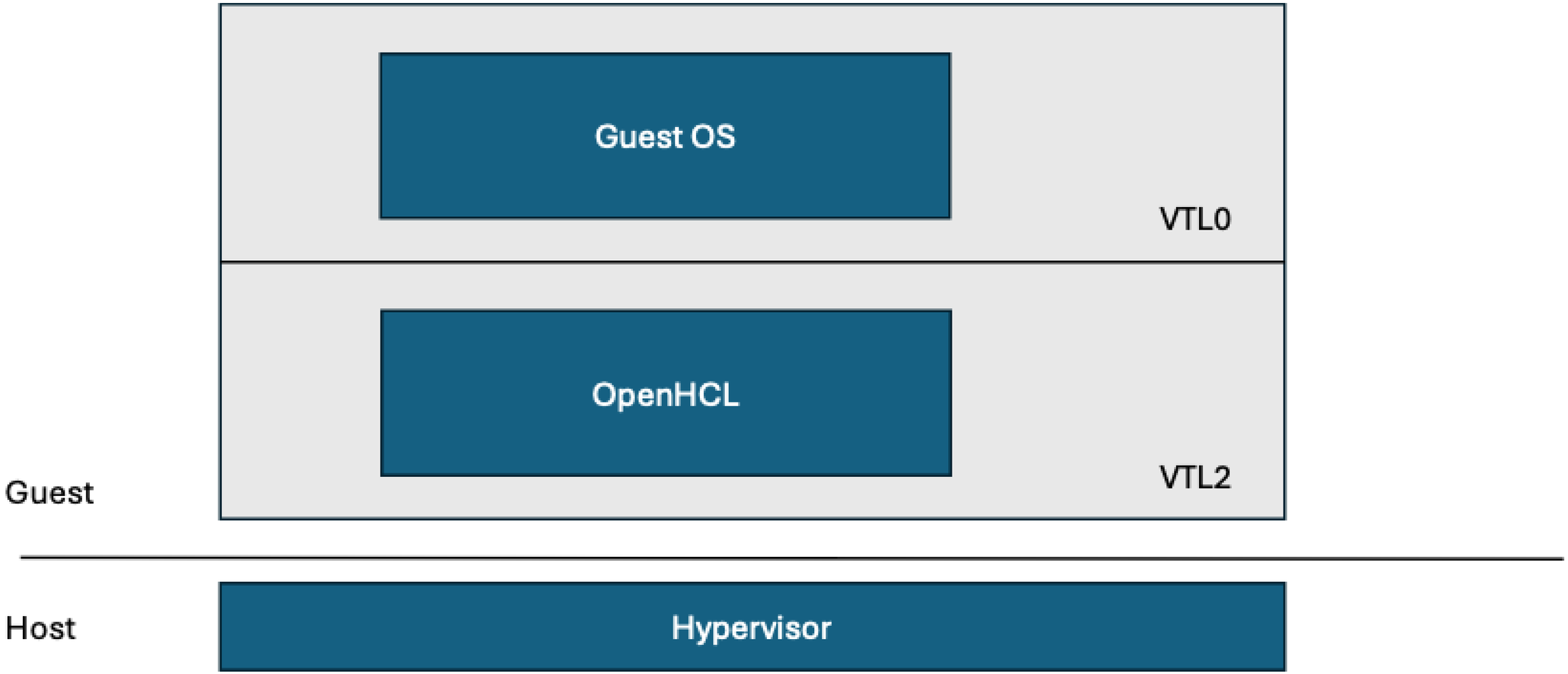
Why async Rust?

- Support a variety of different devices, especially high performance devices
- Control execution inside different environments
  - The VMM run in different environments, so being able to tailor executors without needing to rewrite device code
- Control OS scheduling overhead
  - Executor could be multithreaded, or singlethreaded or whatever works best for that environment

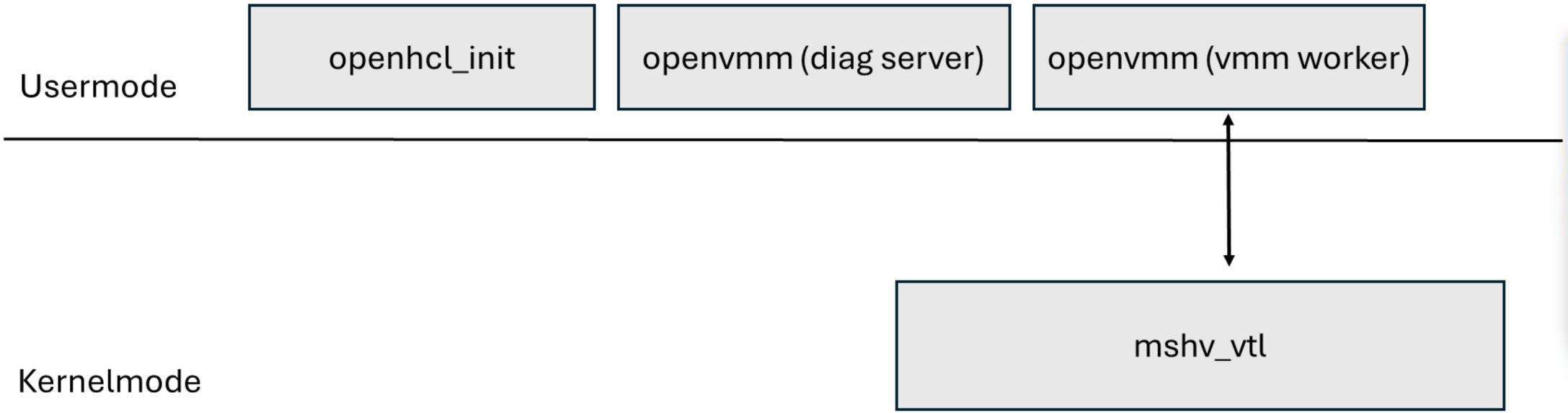
Async Rust in OpenHCL

- Minimize paravisor runtime by minimizing OS threads
- Utilize per vcpu executors for most devices
  - Each thread affinitized to a single vcpu
  - Minimize context switch overhead between VMM exit handler and devices
  - Very important for device translation, where OpenHCL is in the hotpath
- Some low performance devices and diagnostic services handled on separate executor thread

OpenHCL VM



OpenHCL architecture



Why use Linux?

- Write std Rust for VMM code
  - no_std Rust is especially difficult to code in
  - We want to use stable Rust toolchains, not nightly for some no_std features
- Supports standard tooling like gdbserver, perf, etc
- Write usermode drivers via VFIO
- Broadly supported Rust toolchain and crate ecosystem
- Familiar OS platform for contributors

Linux Kernel details

- Boot via device tree, no ACPI
- Minimal Kconfig
  - Minimize binary size and runtime RAM usage
  - Device drivers in usermode via VFIO
- mshv_vtl driver
  - Provides virtualization APIs for usermode
  - Provides access to physical addresses via mmap
- Otherwise standard APIs used by usermode VMM

Usermode processes

- Packaged as a single binary with multiple aliases and launch names
- openhcl_init
- openhcl_dump
- openhcl_crash
- VMM processes
  - control & diagnostics server
  - main vmm worker

openhcl_init

- Lightweight and simple init process to set various settings and launch other processes
  - Minimize binary size and complexity vs other options
- Setup kmsg logging
- Launches main VMM process next

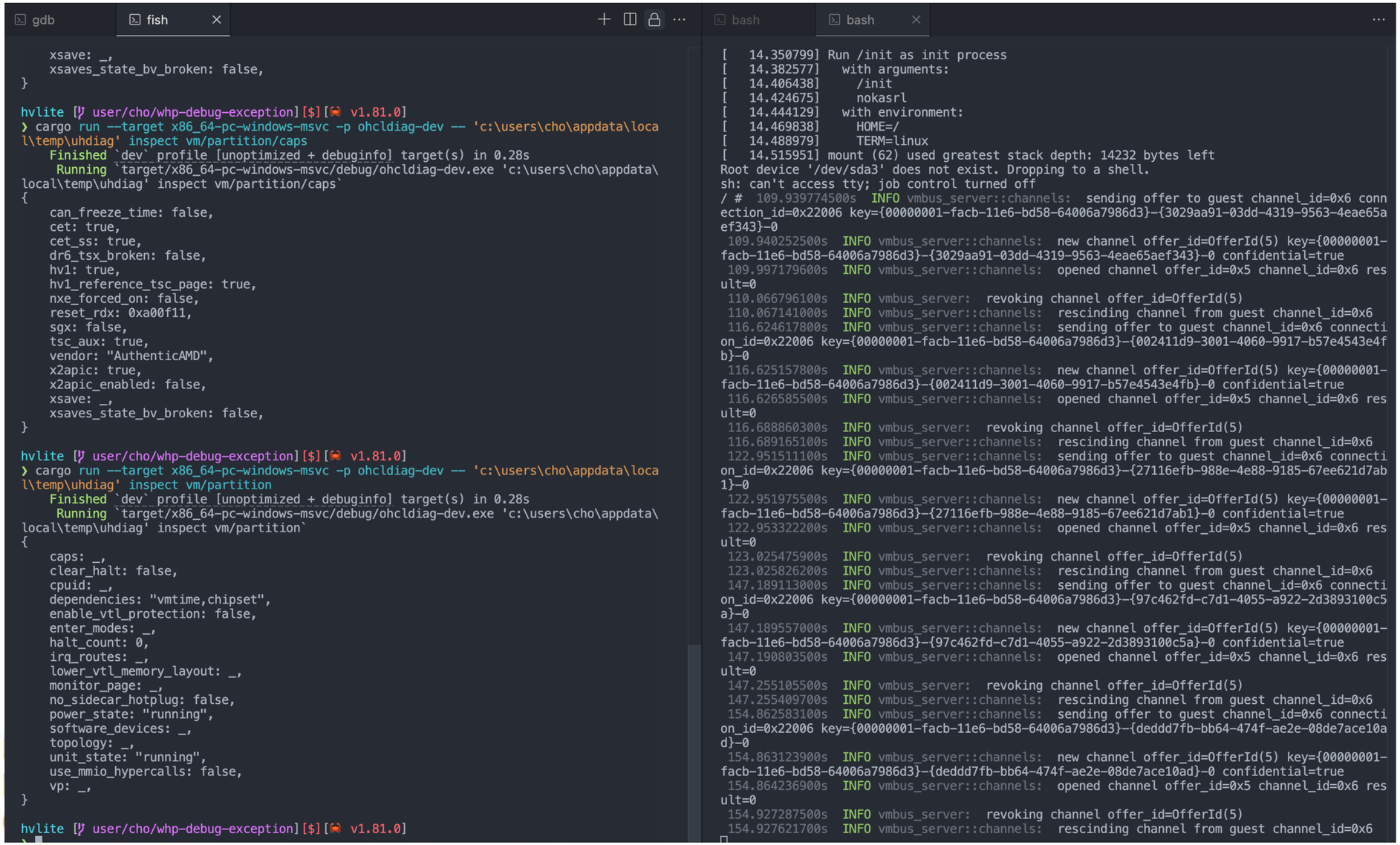
openvmm (diag server)

- Main control process
- Launches child vmm worker that handles exits from the guest
- Handles diagnostics requests via ohcldiag-dev
  - ohcldiag-dev runs on the host, and communicates with diag_server via VSOCK

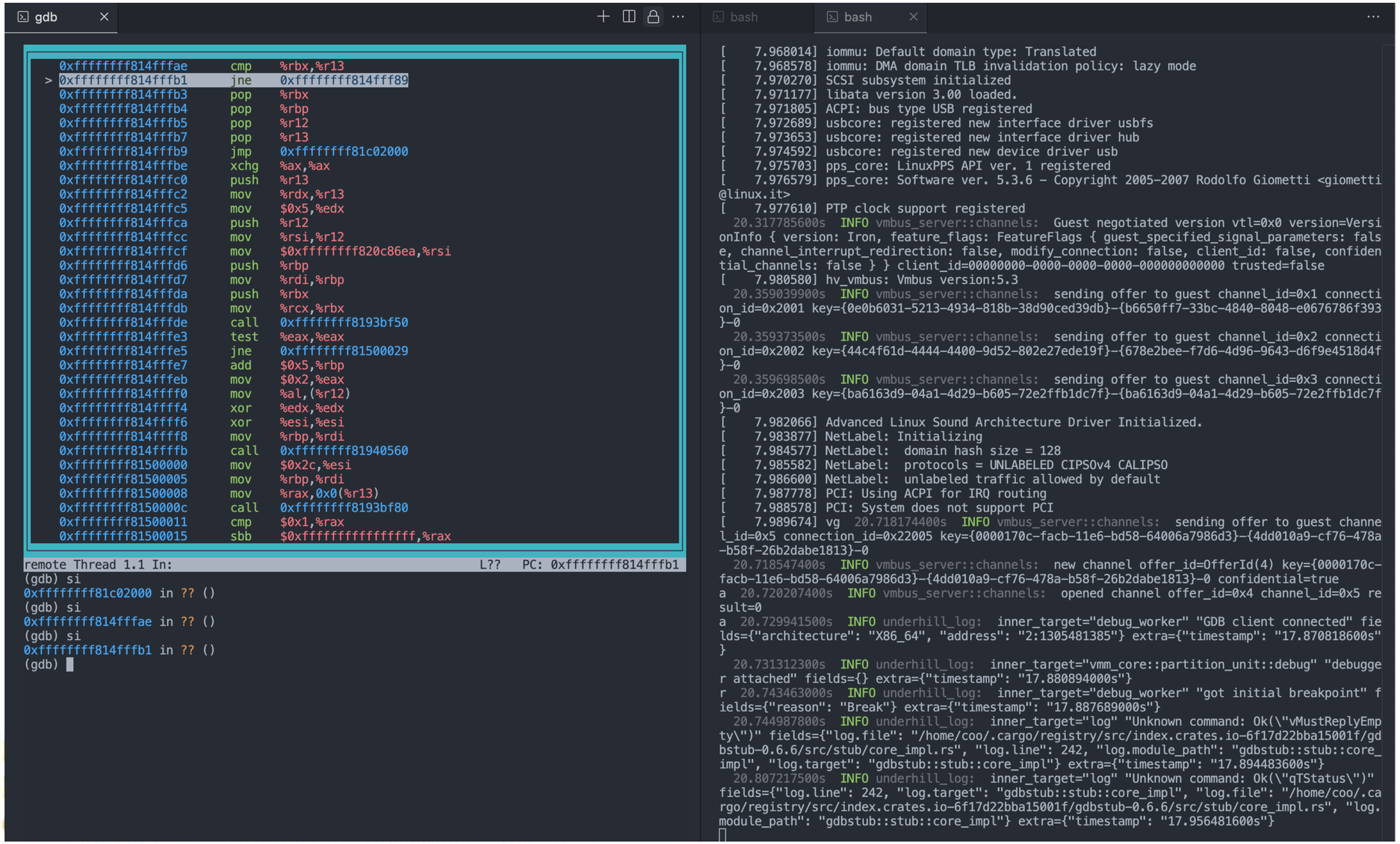
Example diagnostic commands

- kmsg
  - Dump the current kmsg log in OpenHCL
- inspect
  - Objects implement the Inspect trait throughout the codebase
  - Allows inspection of object & system state via human readable text
  - Some values modifiable at runtime, such as tracing log filter
- shell
  - Remote shell into OpenHCL, useful for interactive development
  - Provided via BusyBox

ohcldiag-dev demo



gdbstub demo



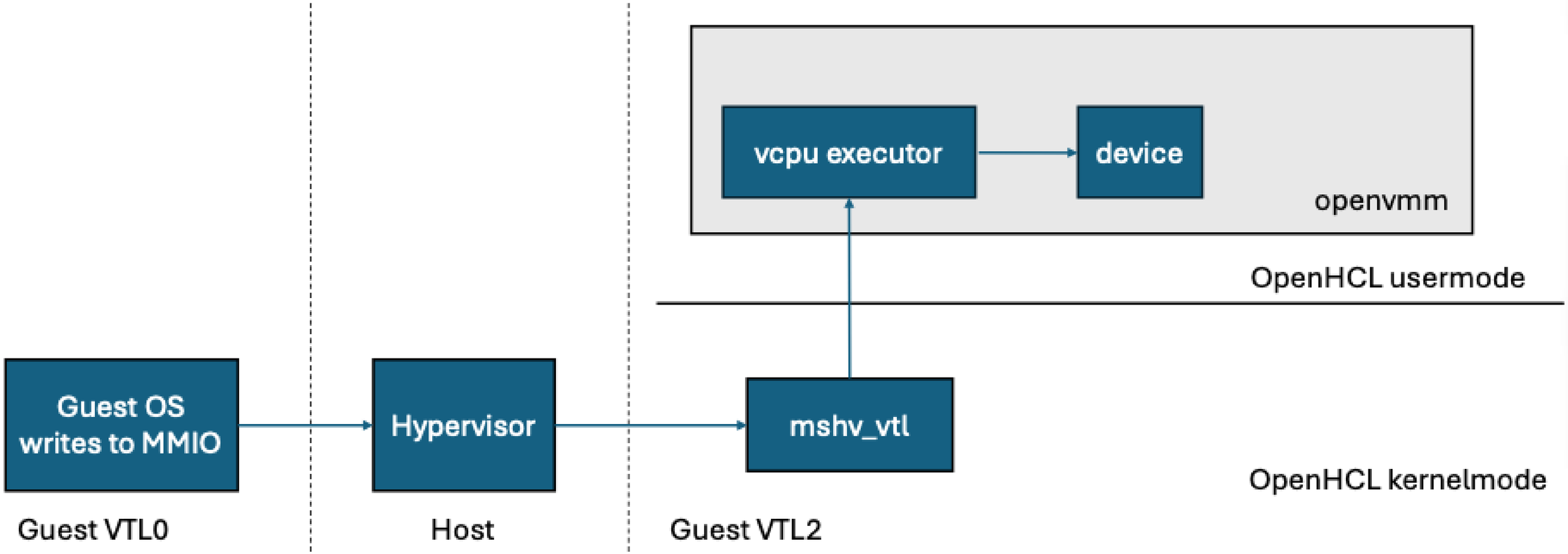
openvmm (VMM worker)

- Main process that acts as a VMM for the guest
- Handle exits from the platform
- Per vCPU executor hosting async tasks
- Interacts with mshv_vtl driver to perform VMM functions
  - Modifying register state
  - Accessing ram via mmap

Other utility processes

- openhcl_dump
  - Collect core dumps of processes and write them to openhcl_crash
  - Separate process to allow dumping any other process in the system
- openhcl_crash
  - Write core dumps of usermode crashes to the host via hvsock

MMIO dispatch flow



Future roadmap

- Open source later this year
- Support ARM CCA
- Support KVM as host
- Support hosting devices in separate processes
  - IE sandbox vTPM from other devices

Q/A