Bring Your Own Virtual Devices

Frameworks for Software and Hardware Device Virtualization

Stefan Hajnoczi

stefanha@redhat.com

Red Hat

About me

Virtualization team at Red Hat
QEMU and Linux VIRTIO drivers

VIRTIO Technical Committee
Virtio-vsock and virtio-fs devices

Areas
Storage, device emulation, tracing

Online
https://blog.vmsplice.net/, stefanha on #qemu IRC,
@stefanha:matrix.org

Linux VFIO

VFIO/mdev

vDPA

An overview of out-of-process device interfaces for QEMU/KVM

vhost (kernel)

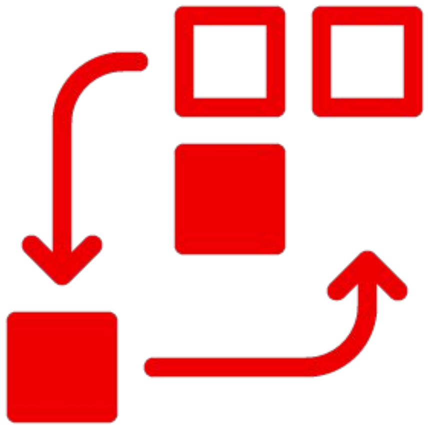
vfio-user

VDUSE

vhost-user

Red Hat

What are out-of-process devices?



Devices implemented outside the Virtual Machine Monitor (VMM) or Hypervisor

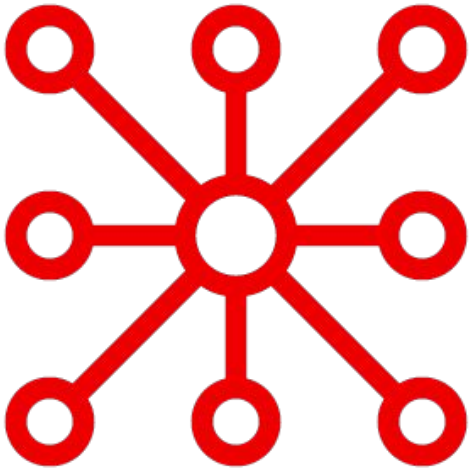
- Appear like any other device to the virtual machine
- Can be added to a VM without installing new VMM or Hypervisor software

Red Hat

Why this talk?

Out-of-process devices present a interesting combination of:

- Proven real-world applications
- Active development
- Rich area for systems research

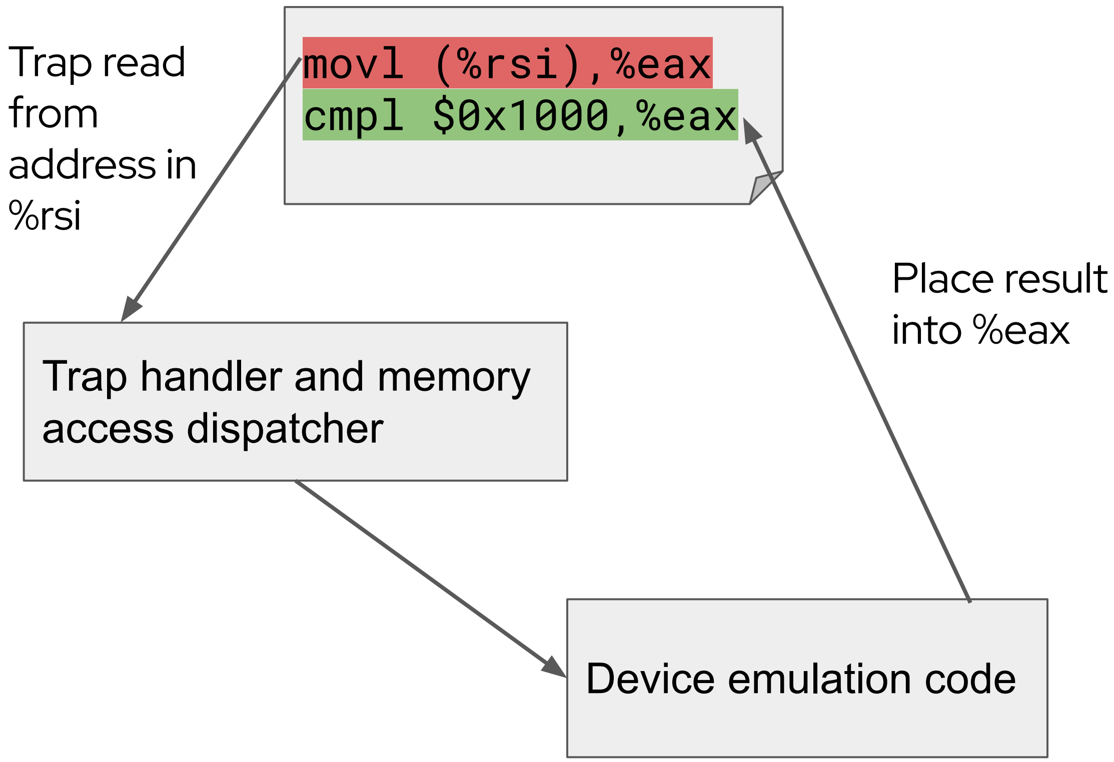


HPC has high I/O requirements and creating your own devices can yield significant improvements

Trap-and-emulate devices

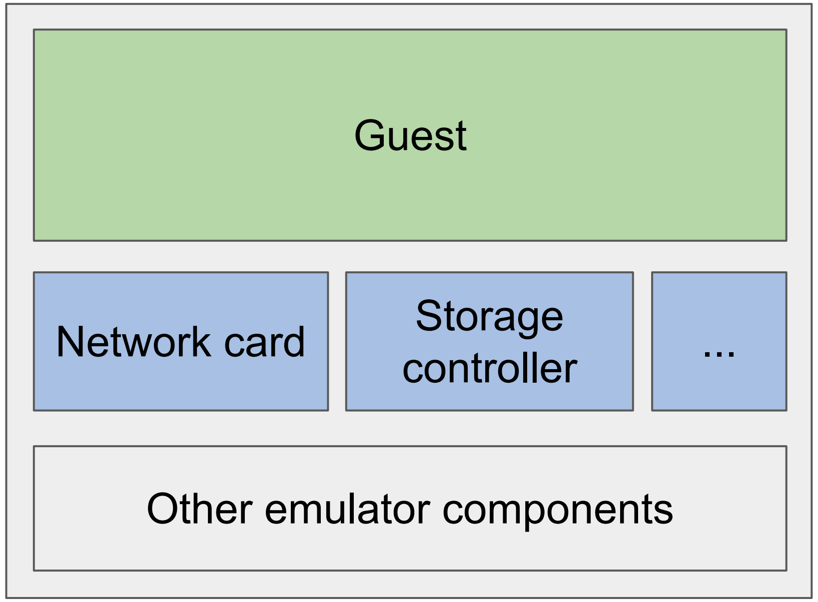
Traditional emulators implement device hardware register accesses through trap-and-emulate

- VMM dispatches memory access to device emulation code
- Device emulation runs in vCPU thread while vCPU is paused
- Result is returned to guest and vCPU resumes



Red Hat

In-process devices



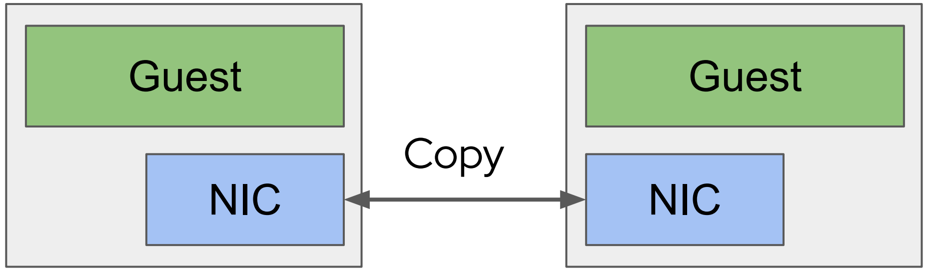
Emulator with in-process devices

Devices are part of the emulator and operate in close proximity to the guest

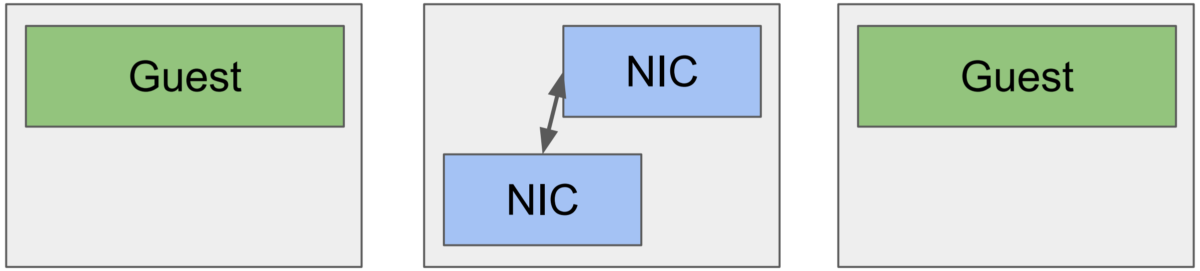
- QEMU, crosvm, etc employ multi-threading or forking models but devices are fundamentally part of the VMM
- No ability to add/remove device implementations

Use cases for out-of-process devices: Performance

- In some situations it's faster to centralize device emulation into a single process serving multiple VMs
- Example: avoiding IPC memory copies in a software-defined networking switch
- Example: dedicating host CPU cores to polling



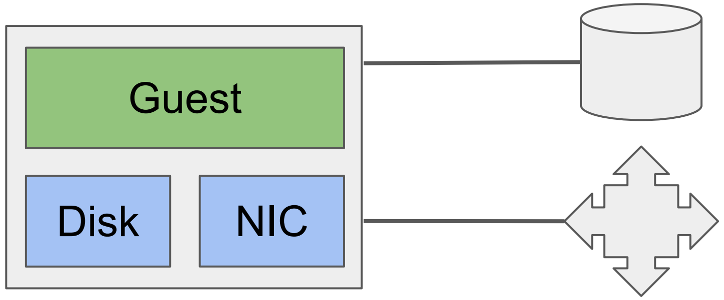
In-process devices



Out-of-process devices

Use cases for out-of-process devices: Security

- Fine-grained device processes helps with privilege separation
- Tighter sandboxing (seccomp, SELinux, namespaces)

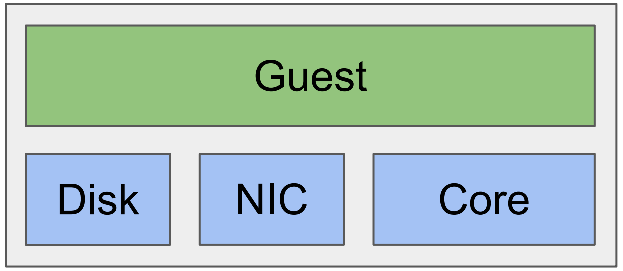


In-process devices

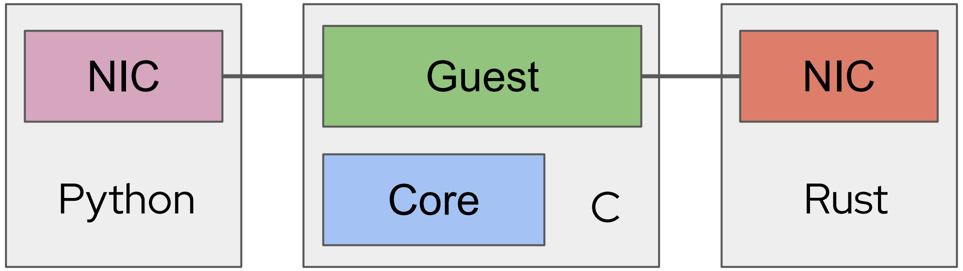
Out-of-process devices

Use cases for out-of-process devices: Polyglot emulators

- Mix and match programming languages
- Linking cross-language code is possible within a process but messy and complex
- Example: C core, Rust devices, Python fuzzer devices



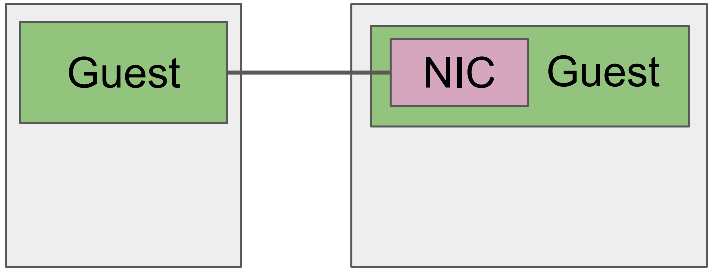
In-process devices



Out-of-process devices

Use cases for out-of-process devices: Inter-VM device emulation

- Placing emulated devices into VMs
- Stronger isolation of devices
- Easy to deploy in compute clouds where users cannot run processes on the host



Red Hat

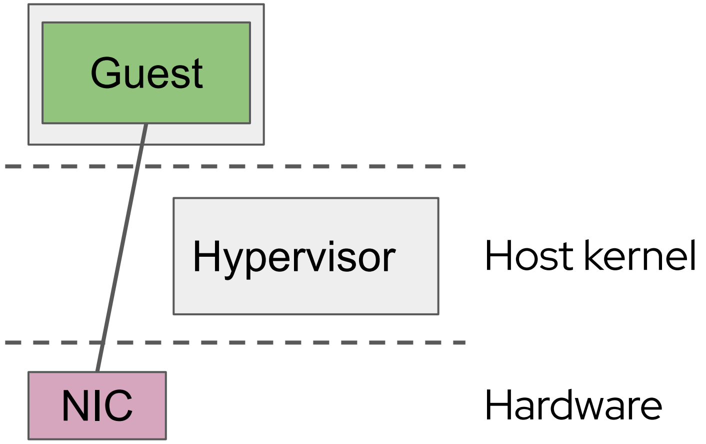
Use cases for out-of-process devices: More…

- Sharing device emulation code between VMMs
- Special-purpose device implementations to achieve niche requirements
- …

- The pros/cons depend on the details of your VMM and Hypervisor but there are many use cases.

Types of OoP device interfaces: Hardware passthrough

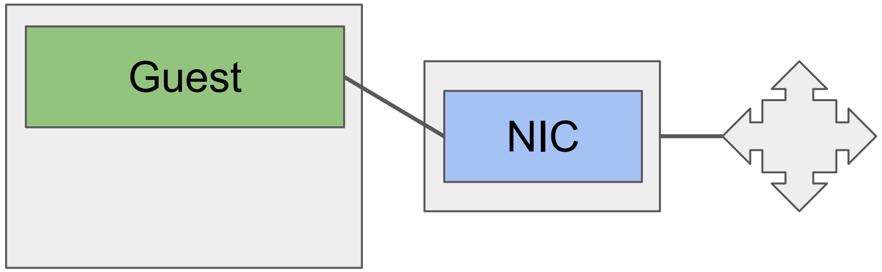
- Give guest access to a physical device
- PCI, PCI SR-IOV, accelerators, SmartNICs
- Relatively high barrier to creating your own device



We will cover **Linux VFIO**

Red Hat

Types of OoP device interfaces: Software

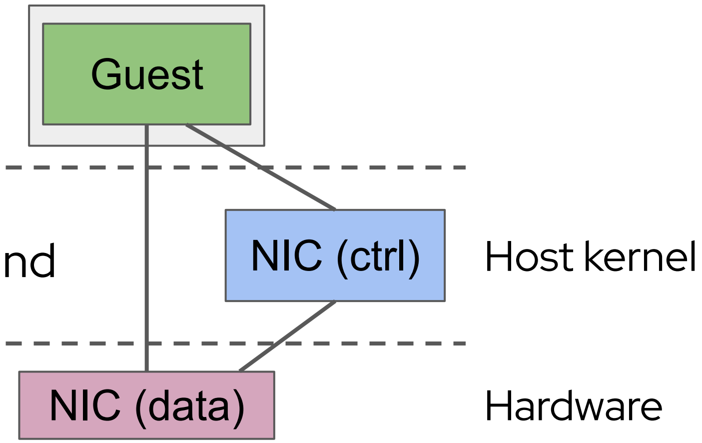


- Run device emulation in a separate software component
- vhost (VIRTIO-based) devices
- Relatively easy to create your own device

We will cover **vhost (kernel)**, **vhost-user**, **vfio-user**, **VDUSE**

Types of OoP device interfaces: Software/hardware hybrid

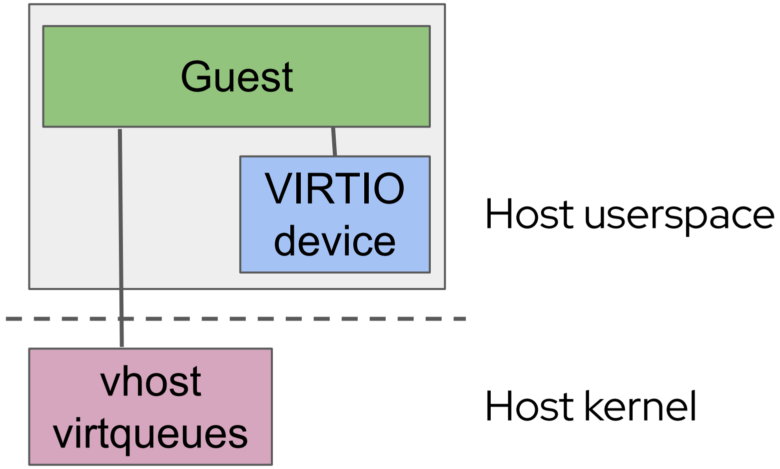
- Give guest data path access to physical device with control path managed in software
- Mediated devices, Intel® Scalable I/O Virtualization, etc
- Combines qualities of hardware passthrough with more flexible and lightweight software control



We will cover **Linux VFIO/mdev** and **vDPA**

vhost (kernel)

Since 2010



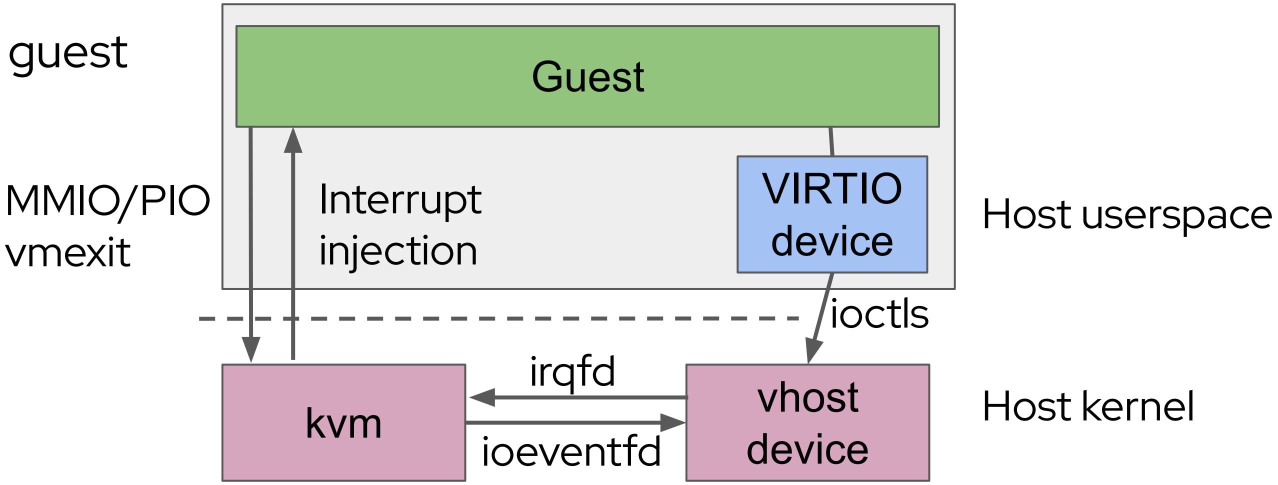
- Offload VIRTIO data path to host kernel
- Kernel code has access to special functionality:
  - Network stack, LIO SCSI target, etc
- VIRTIO control path handled by VMM
- Devices: vhost_net, vhost_vsock, vhost_scsi

Links:
https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/drivers/vhost/

Red Hat

How vhost (kernel) works

- ioeventfd signals vhost worker thread when guest hardware access causes vmexit
- irqfd injects interrupt into guest
- Memory regions are configured by VMM to provide access to guest RAM
- vhost lifecycle managed by VMM via ioctls



Links:
https://blog.vmsplice.net/2011/09/qemu-internals-vhost-architecture.html

vhost (kernel) ioctls

Command	Purpose
VHOST_GET/SET_FEATURES	VIRTIO feature negotiation
VHOST_SET/RESET_OWNER	Associating a device with a userspace process
VHOST_SET_MEM_TABLE	Configuring guest RAM
VHOST_SET_VRING_NUM/ADDR/BASE	Configuring vring memory structure
VHOST_SET_VRING_KICK	Assigning ioeventfd for driver->device notifications
VHOST_SET_VRING_CALL	Assigning irqfd for device->driver notifications

Links:
https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/include/uapi/linux/vhost.h

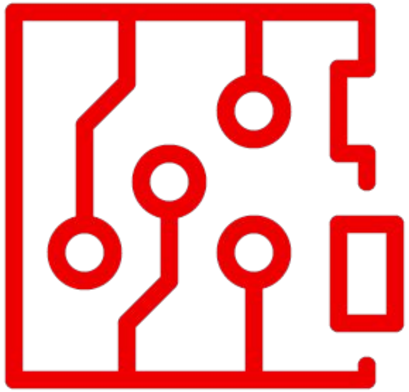
vhost (kernel) device implementation

- Add a new driver to Linux drivers/vhost/
- Modify VMM's VIRTIO device emulation to use your vhost driver
- Consider syzkaller for fuzzing your driver

Linux VFIO

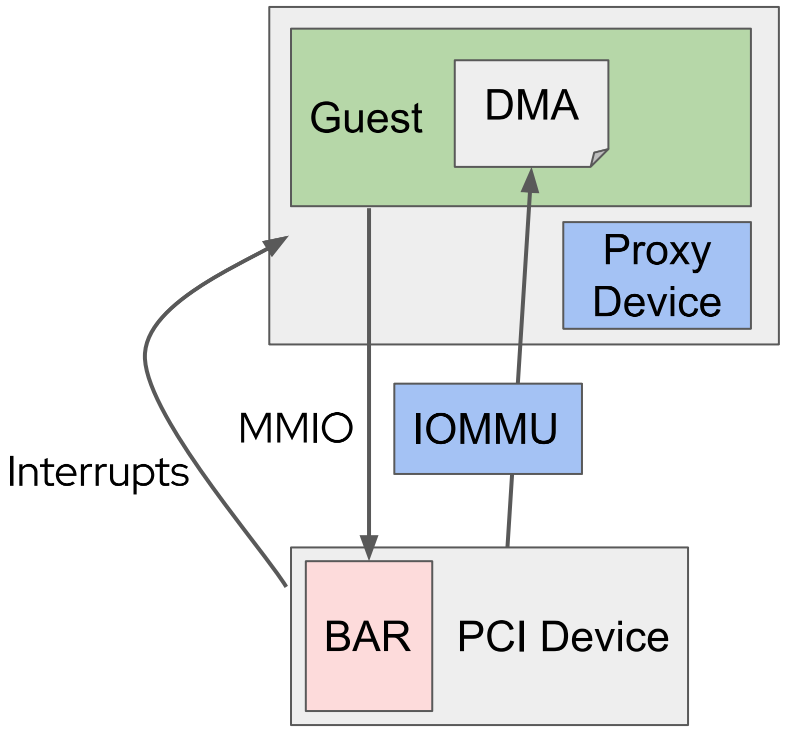
Since 2012

- Linux API for userspace device drivers (PCI and other busses)
- QEMU uses VFIO for hardware passthrough
- Devices are isolated by IOMMU
  - Device can only touch guest RAM
- Guest requires driver for the specific device
- Suitable for your own PCI and PCI SR-IOV devices like accelerators or SmartNICs



Links:
https://www.kernel.org/doc/html/latest/driver-api/vfio.html

How Linux VFIO works



- Memory-mapped I/O (MMIO) directly accesses device
- Interrupts are directly injected into guest using interrupt controller virtualization features
- DMA isolation enforced by host IOMMU
- PCI proxy device resides on an emulated PCI bus
  - PCI Configuration Space is still emulated
  - Other resources are typically passed through

Links:
Intel VT-d Posted Interrupts - https://software.intel.com/content/dam/develop/external/us/en/documents/vt-directed-io-spec.pdf

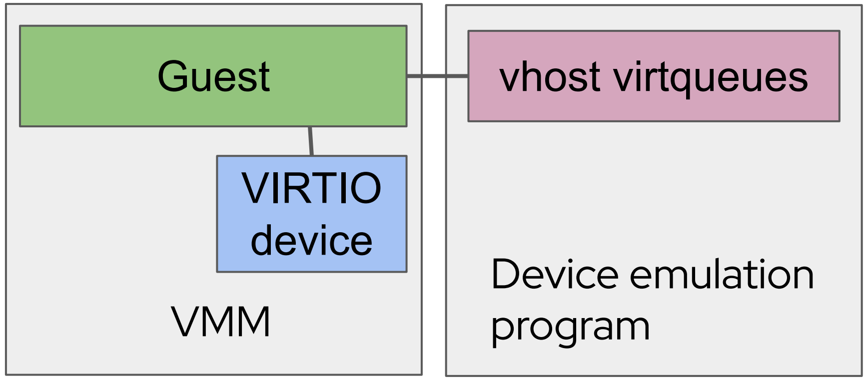
VFIO device implementation

- Develop and test PCI or PCI SR-IOV device
- Most existing PCI devices do not require changes to VFIO or QEMU, new devices should not require any changes

vhost-user

Since 2014

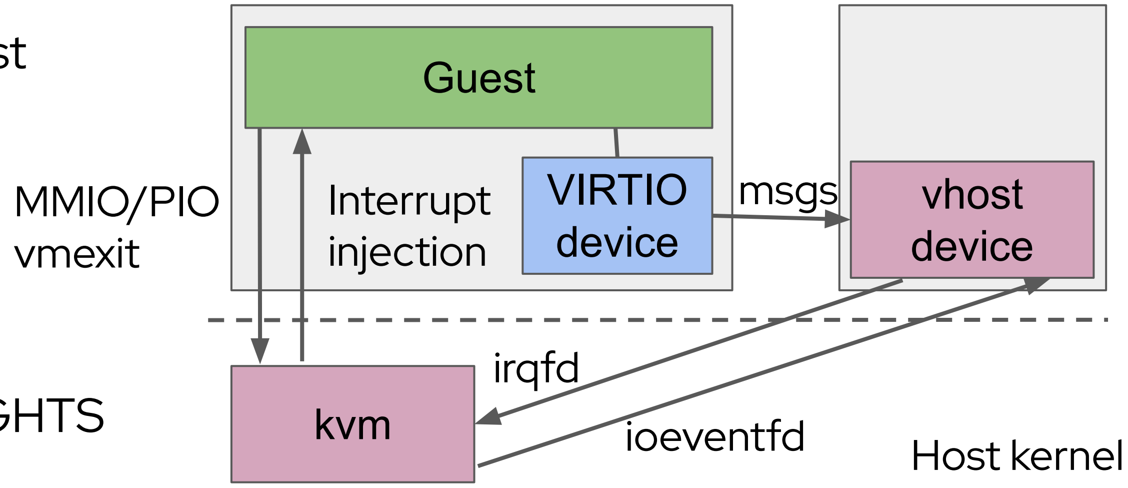
- Offload VIRTIO data path to a userspace process
- vhost-style ioctl commands over a UNIX domain socket
- VIRTIO control path handled by VMM
- Devices: vhost-user-net, vhost-user-blk, vhost-user-scsi, …
- Used for software-defined networking and storage, complex software devices like GPU or file system servers



Links:
https://gitlab.com/qemu-project/qemu/-/blob/master/docs/interop/vhost-user.rst

How vhost-user works

- Same ioeventfd and irqfd approach as vhost (kernel)
- Similar ioctls as vhost (kernel) but sent as message over UNIX domain socket
- Memory regions are shared using SCM_RIGHTS file descriptor passing and mmap(MAP_SHARED)



Red Hat

vhost-user device implementation

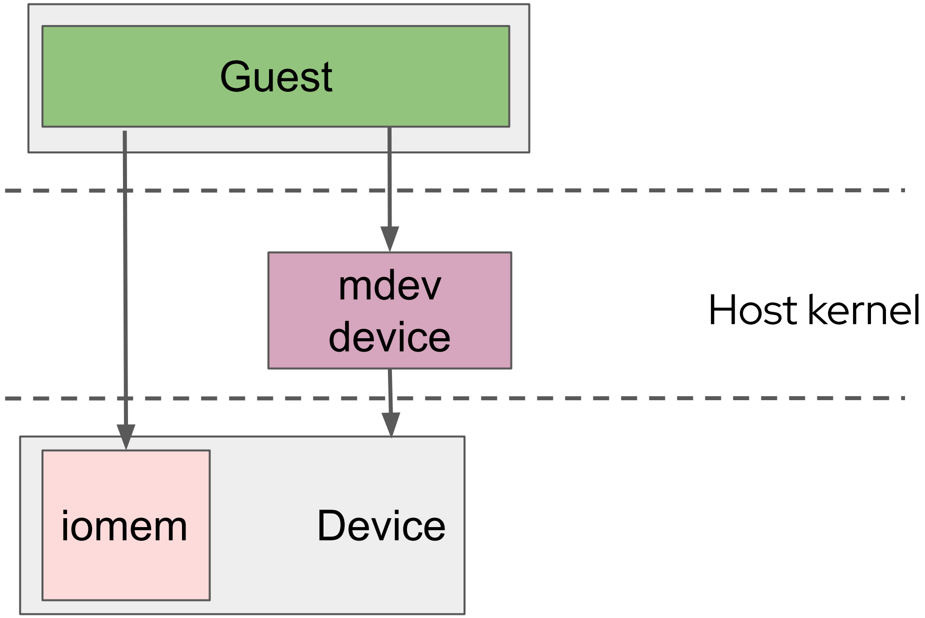
- libvhost-user
  - C library with optional glib integration
- rust-vmm's vhost-user-backend
  - Rust crate
- Add new protocol messages to vhost-user specification, if necessary

Links:
https://gitlab.com/qemu-project/qemu/-/tree/master/subprojects/libvhost-user
https://dpdk.org/https://spdk.io/https://github.com/rust-vmm/vhost-user-backend

VFIO/mdev

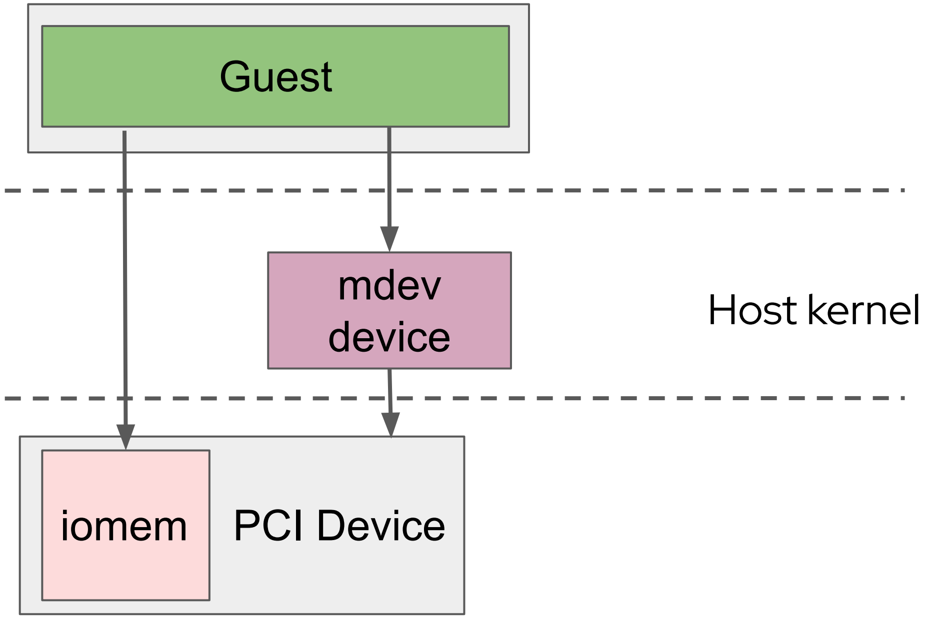
Since 2016

- Software devices in host kernel that implement VFIO ioctls
- Appear to host userspace as VFIO devices
- Can pass through hardware resources or can emulation device functionality in software
- Can use IOMMU for DMA isolation of hardware
- Suitable for complex devices, software alternative to SR-IOV



Links:
https://www.kernel.org/doc/html/latest/driver-api/vfio-mediated-device.html

How VFIO/mdev works



- mdev driver's ->ioctl() callback synthesizes VFIO ioctl responses
- VFIO_REGION_INFO_FLAG_MMAP and "sparse mmap" enable full or partial hardware passthrough
- Interrupts can be injected by the hardware or simulated by the software mdev driver
- mdev driver can enforce DMA isolation of hardware using IOMMU

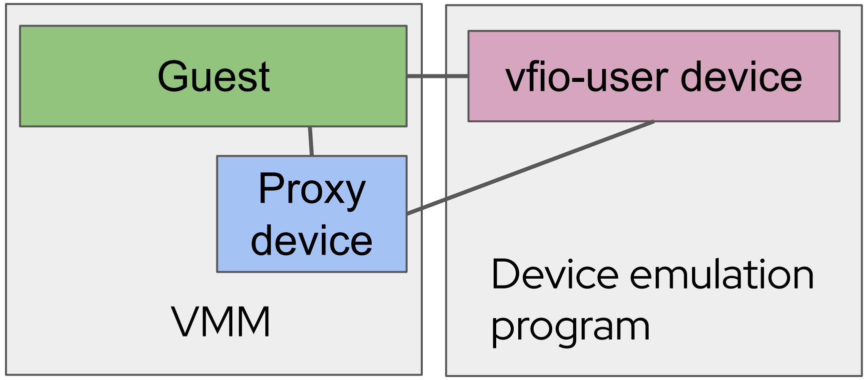
VFIO/mdev device implementation

- Develop a PCI device if hardware offload is desired
- Develop a Linux VFIO/mdev driver
- VMM accesses the device like a regular VFIO device, no code changes necessary

vfio-user

Currently in development

- PCI device emulation in userspace, maybe other busses in the future too
- VFIO-style ioctl commands over UNIX domain socket
- vhost-user-style design with shared memory and eventfds
- Software-defined networking and storage, complex software devices

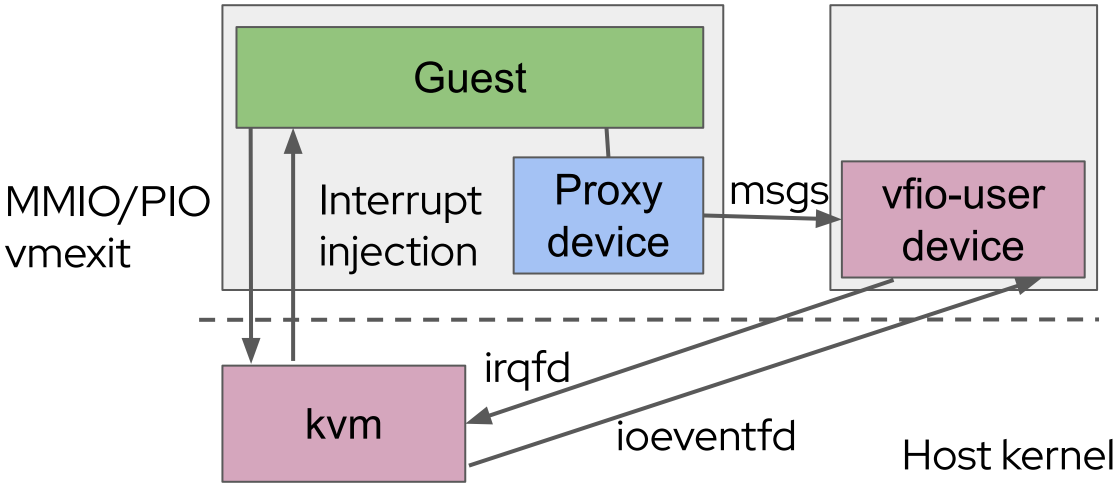


Links:
https://patchew.org/QEMU/20210614104608.212276-1-thanos.makatos@nutanix.com/

Red Hat

How vfio-user works

- PCI BARs can:
  a. Be mapped into guest
  b. Use ioeventfd
  c. Trap-and-emulate via a message
- Interrupts use irqfd
- Memory regions are shared using SCM_RIGHTS file descriptor passing and mmap(MAP_SHARED)



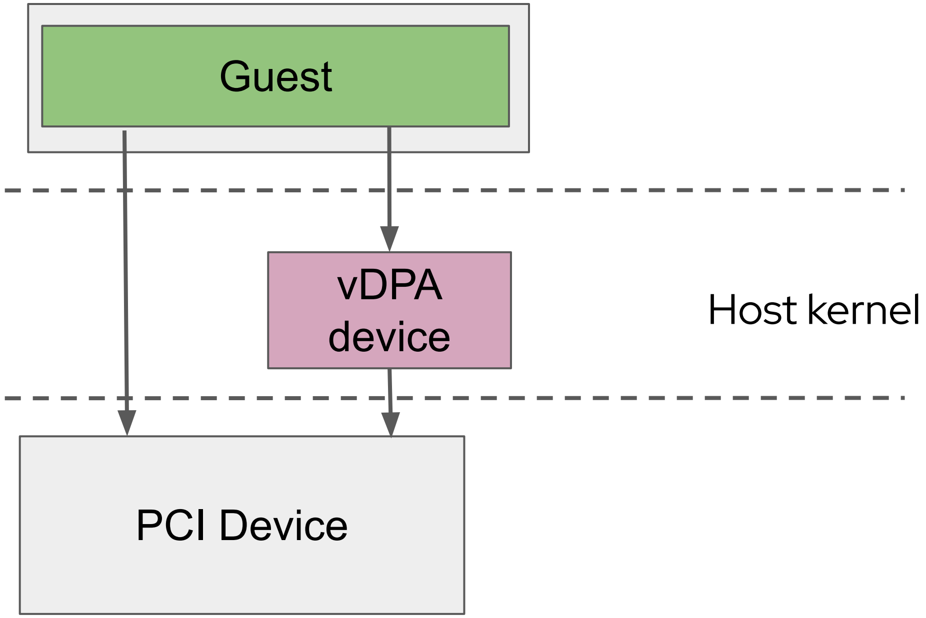
vfio-user device implementation

- libvfio-user
  - C library
  - Currently being used to develop QEMU and SPDK support

Links:
https://github.com/nutanix/libvfio-user

vDPA

Since 2020

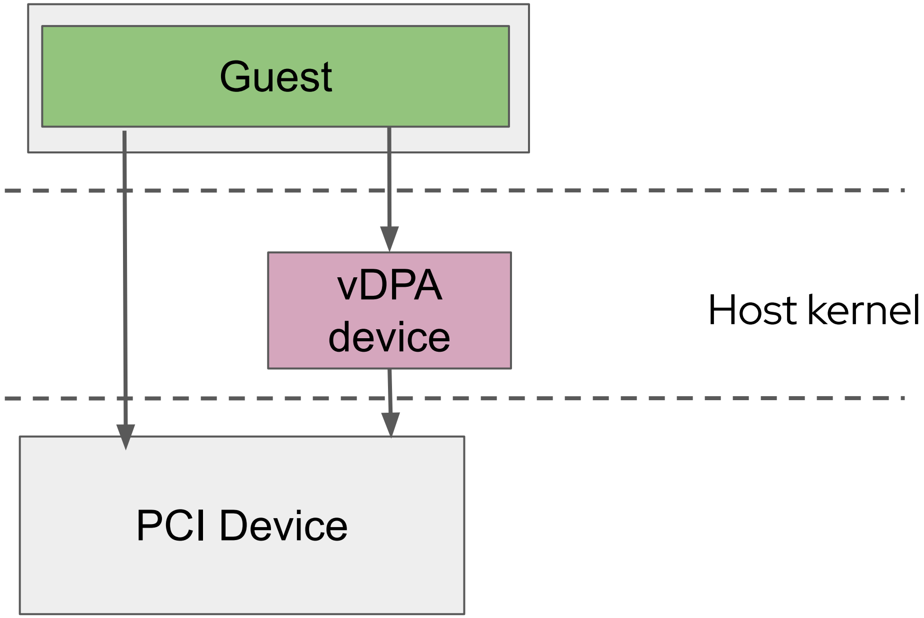


- Hybrid hardware/software VIRTIO devices
- Or pure software VIRTIO devices in host kernel
- Exposed to VMMs via vhost_vdpa Linux driver
  - Extended vhost (kernel) ioctl API
- Host applications/containers can access devices via virtio_vpda Linux driver
- Suitable for SmartNICs, accelerators, etc

Link:
https://www.redhat.com/en/blog/introduction-vdpa-kernel-framework

Red Hat

How vDPA works



- Virtqueue doorbell register writes can be passed through directly to hardware or handled in software
- Interrupts can be injected by the hardware or simulated by the software vDPA driver
- Guest RAM mappings provided by VMM via vhost IOTLB API, including host IOMMU support for hardware passthrough

Red Hat

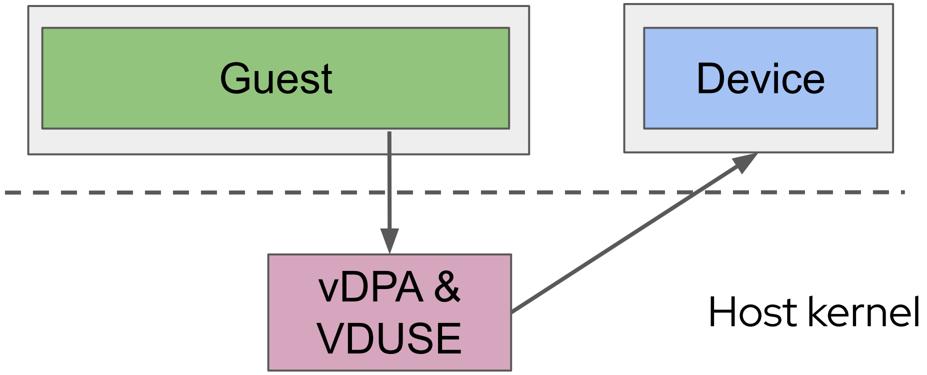
vDPA device implementation

- Develop a PCI device if hardware offload is desired
- Develop a Linux vDPA driver
- VMM needs vhost_vdpa support, which is similar to vhost (kernel)

VDUSE

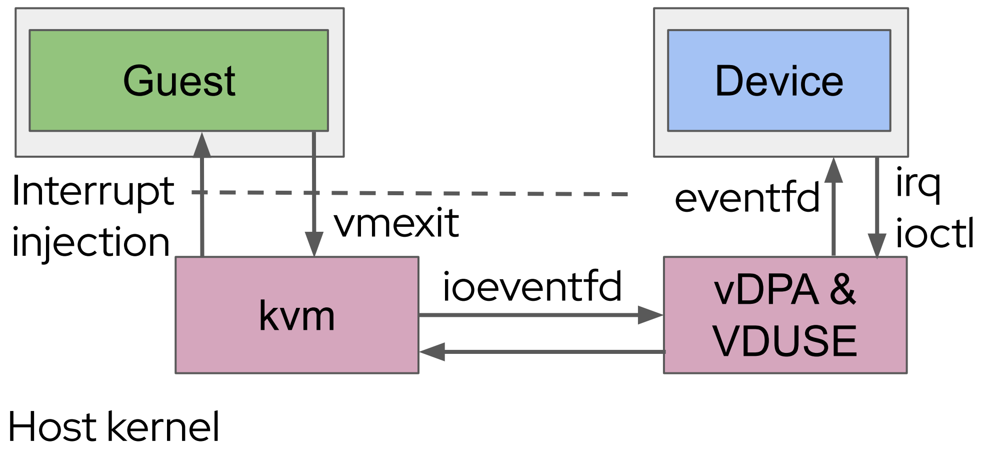
Currently in development

- Connects userspace devices to host kernel vDPA subsystem
- Devices can be attached to the host or exposed to guests via vhost_vdpa
- Similar to vhost-user except host can also access devices
- Devices: virtio-blk



Links:
https://lore.kernel.org/lkml/20210615141331.407-1-xieyongji@bytedance.com/

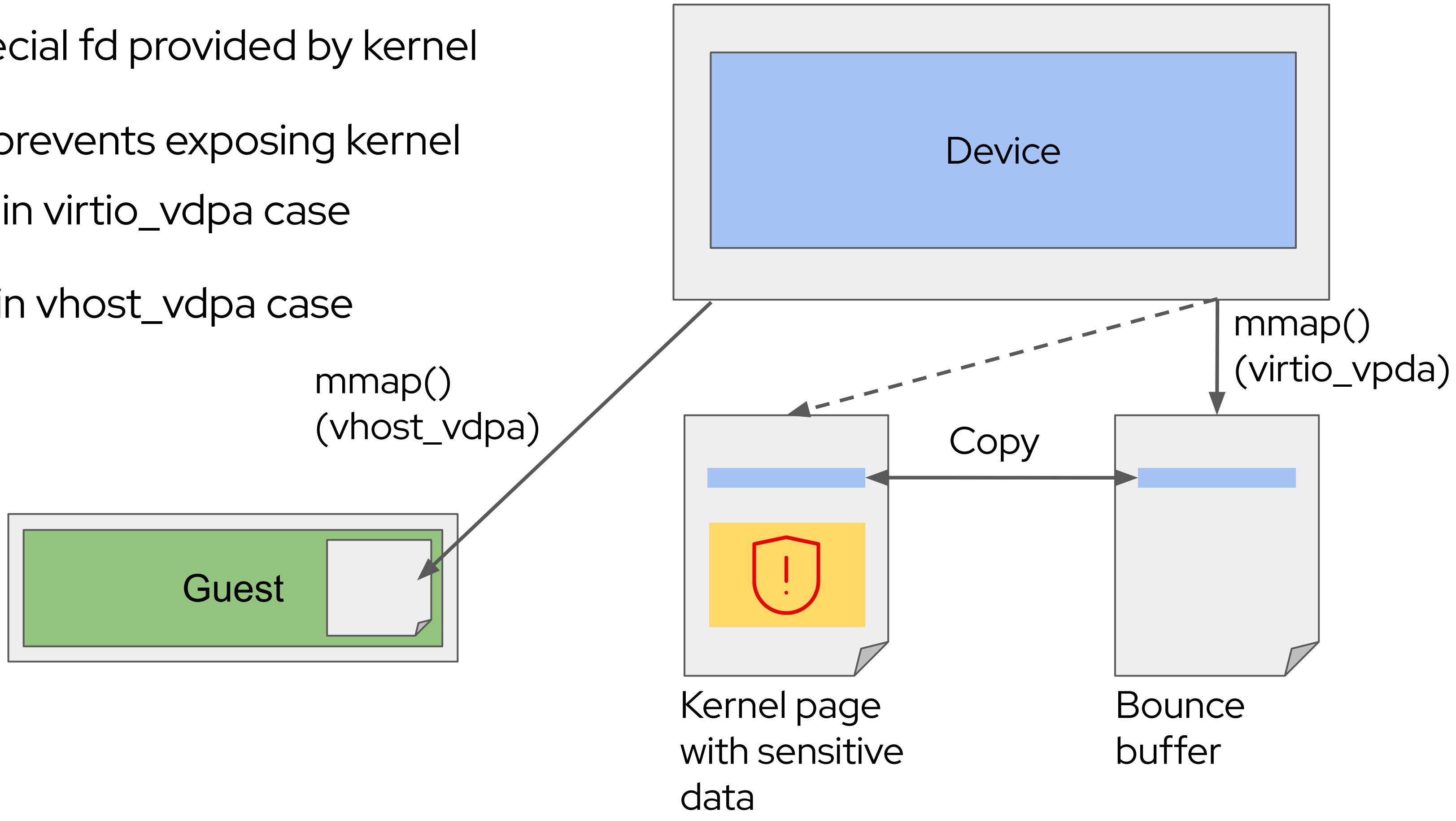
How VDUSE works



- Userspace device registers with host kernel VDUSE and vDPA subsystem
- ioeventfd signals vDPA and forwards to userspace eventfd when guest hardware access causes vmexit
- VDUSE ioctl injects interrupt

VDUSE IOTLB API

- Userspace mmaps special fd provided by kernel
- Kernel bounce buffer prevents exposing kernel memory to userspace in virtio_vdpa case
- Shared memory used in vhost_vdpa case



Red Hat

VDUSE device implementation

- No libraries available yet, code against kernel ioctl API
- No VMM changes necessary if vDPA device is already supported
- Currently limited to virtio-blk devices but expected to support more vDPA device types in the future

Choosing the appropriate solution

Out-of-process device interface	Recommended application
vhost (kernel)	Accessing host kernel functionality
VFIO	Accelerators and SmartNIC PCI devices
vhost-user	Software VIRTIO-based devices
VFIO/mdev	Complex PCI devices needing SR-IOV style functionality
vfio-user	Software PCI devices
vDPA	VIRTIO accelerators and SmartNICs, host applications/containers support
VDUSE	Software VIRTIO devices accessible from host applications/containers

Choosing the appropriate solution (2)

Security

Performance

Ease of development

Deployment cost

Hardware vs software implementation

Live migration

VIRTIO or PCI

Where to find out more

- All of these interfaces are open source
- Join the mailing lists and chat
- kvm@vger.kernel.org & qemu-devel@nongnu.org
- More about technical requirements of out-of-process device interfaces: https://blog.vmsplice.net/2020/10/requirements-for-out-of-process-device.html

Thank you

Red Hat is the world's leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.

linkedin.com/company/red-hat

youtube.com/user/RedHatVideos

facebook.com/redhatinc

twitter.com/RedHat

Red Hat