Virtual Open Systems

AGL Virtio-loopback
Code review planned on (9/11/2022)

A work carried on by Virtual Open Systems, on behalf of Linux Foundation, to enhance Automotive Grade Linux (AGL)

contact@virtualopensystems.com

www.virtualopensystems.com

Discussion Index

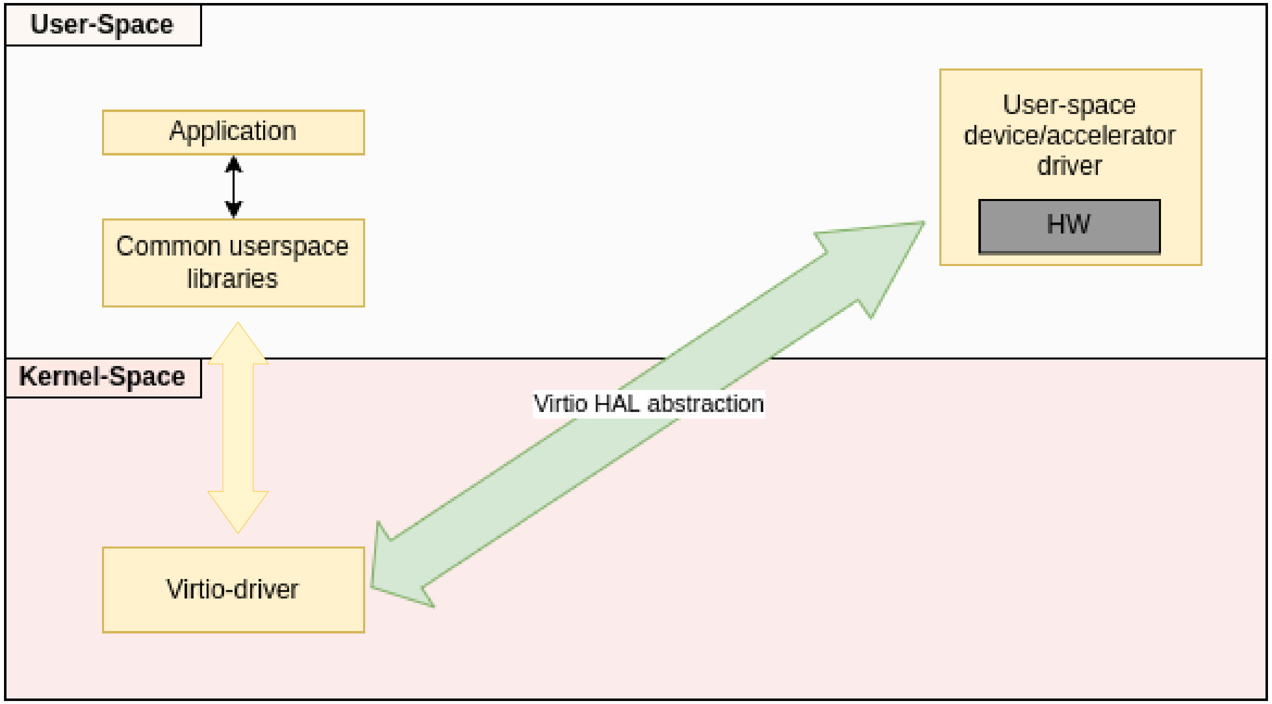
- Brief design description and current status
- Code review
  - Control plane
  - Communication mechanisms
  - Memory mapping (data plane)
- Live demo
- Upstream
- Next steps
- Questions

Discussion Index

- **Brief design description and current status**
- Code review
  - Control plane
  - Communication mechanisms
  - Memory mapping (data plane)
- Live demo
- Upstream
- Next steps
- Questions

Virtio-loopback approach

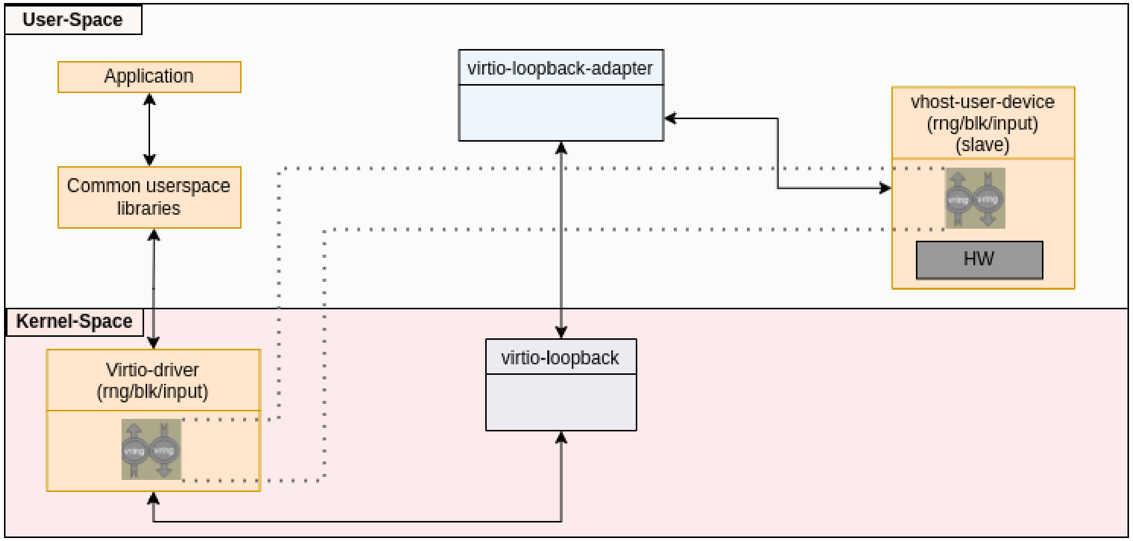
Virtio Loopback describes a new Hardware Abstraction Layer (HAL) for non-Hypervisor environments based on virtio.



Virtio-loopback gives the ability to host user-space applications to take advantage of user-space drivers

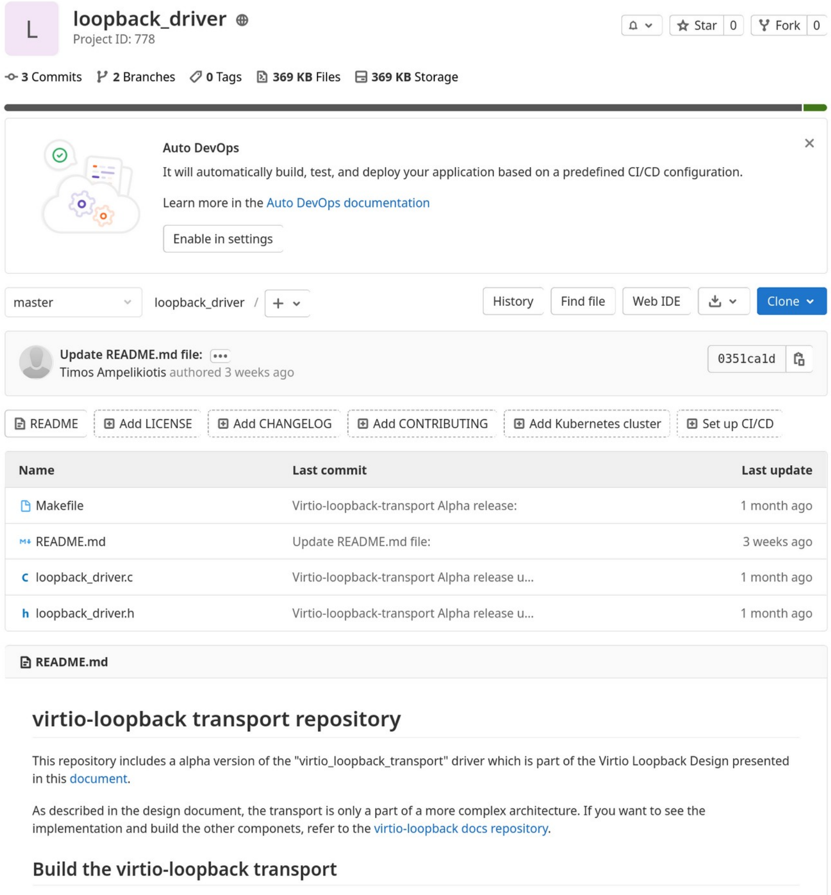
Virtio-loopback components

- The kernel space component is a **new virtio transport** that forwards driver calls in user space where the device is implemented.
- The second component is an **application in user space** (virtio-loopback-adapter) that is particularly important for the set-up of the system configuration, but that does not impact the data plane path to avoid overhead.


User-Space
Application
Common userspace libraries
virtio-loopback-adapter
vhost-user-device (rng/blk/input) (slave)
vring
vring
HW
Kernel-Space
Virtio-driver (rng/blk/input)
vring
vring
virtio-loopback

Current status of the activity

- **Alpha release**: Publicly released with docs and demo
- **Beta release**: Done for this review
- Next steps:
  - Merge the beta into the master
  - Benchmarks
  - Prepare AGL patches



Auto DevOps

It will automatically build, test, and deploy your application based on a predefined CI/CD configuration.

Learn more in the Auto DevOps documentation

Update README.md file:
Timos Ampelikiotis authored 3 weeks ago

Name	Last commit	Last update
Makefile	Virtio-loopback-transport Alpha release:	1 month ago
README.md	Update README.md file:	3 weeks ago
loopback_driver.c	Virtio-loopback-transport Alpha release u...	1 month ago
loopback_driver.h	Virtio-loopback-transport Alpha release u...	1 month ago

virtio-loopback transport repository

This repository includes a alpha version of the "virtio_loopback_transport" driver which is part of the Virtio Loopback Design presented in this document.

As described in the design document, the transport is only a part of a more complex architecture. If you want to see the implementation and build the other componets, refer to the virtio-loopback docs repository.

Build the virtio-loopback transport

Discussion Index

- Brief design description and current status
- **Code review**
  - Control plane
  - Communication mechanisms
  - Memory mapping (data plane)
- Live demo
- Upstream
- Next steps
- Questions

Discussion Index

- Brief design description and current status
- Code review
  - **Control plane**
  - Communication mechanisms
  - Memory mapping (data plane)
- Live demo
- Upstream
- Next steps
- Questions

Control Plane (1)

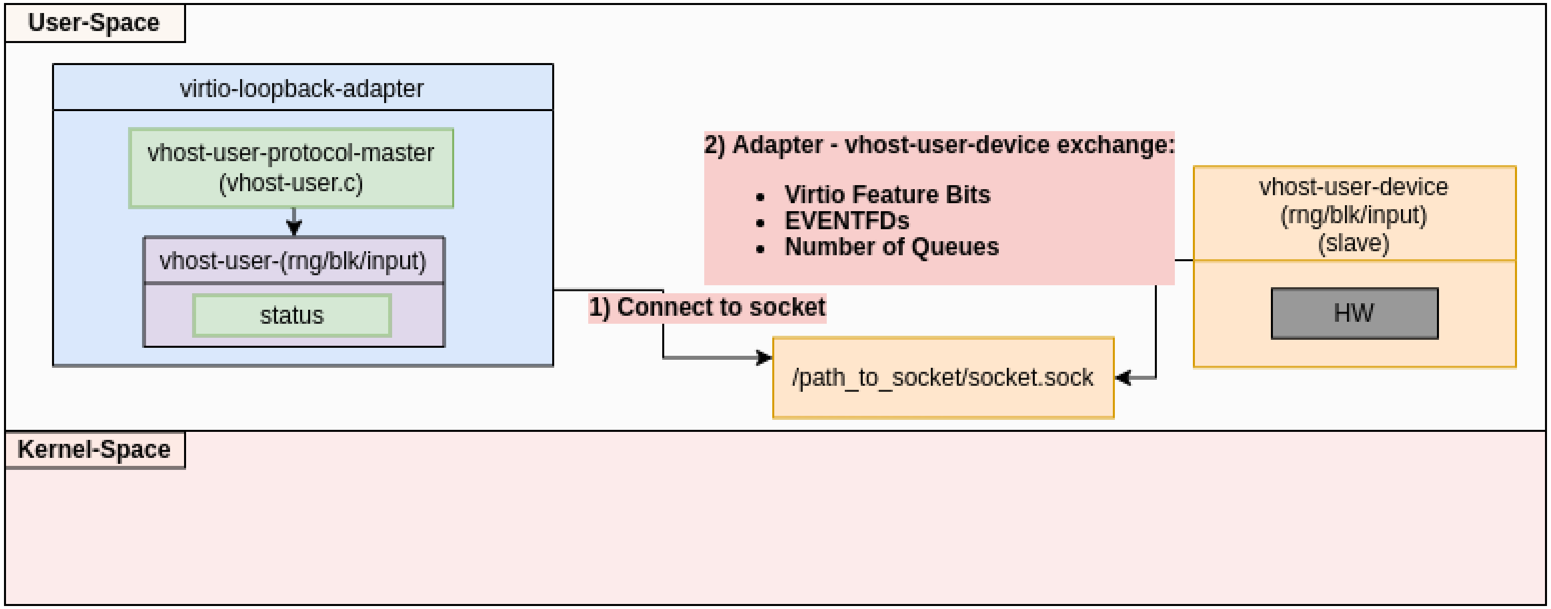
The following three steps are describing briefly the control plane which takes place before the whole system is ready to exchange any data.

The control plane consists of stages of communication:

- [Stage 1] Adapter ↔ Vhost-user-device
- [Stage 2] Adapter ↔ Transport driver
- [Stage 3] Adapter ↔ Vhost-user-device

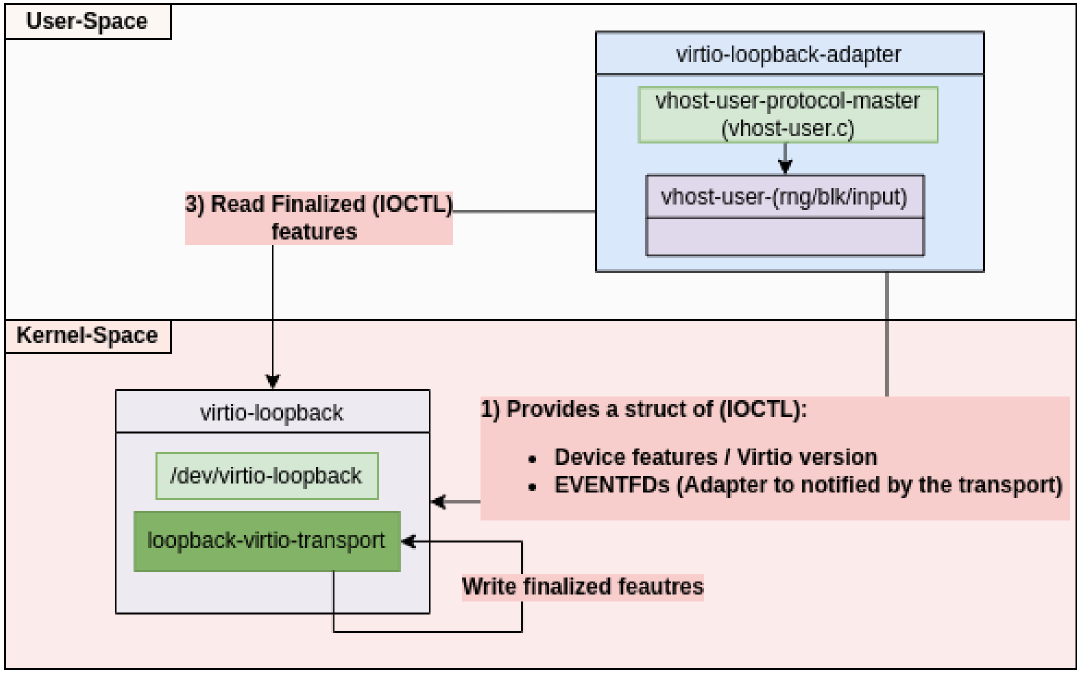
Control Plane (2)

- [Stage 1] Adapter ↔ Vhost-user-device
  - The vhost-user-device sends via the Unix socket to the adapter:
    - Virtio features, vhost-user protocol features, virtio device configuration



Control Plane (3)

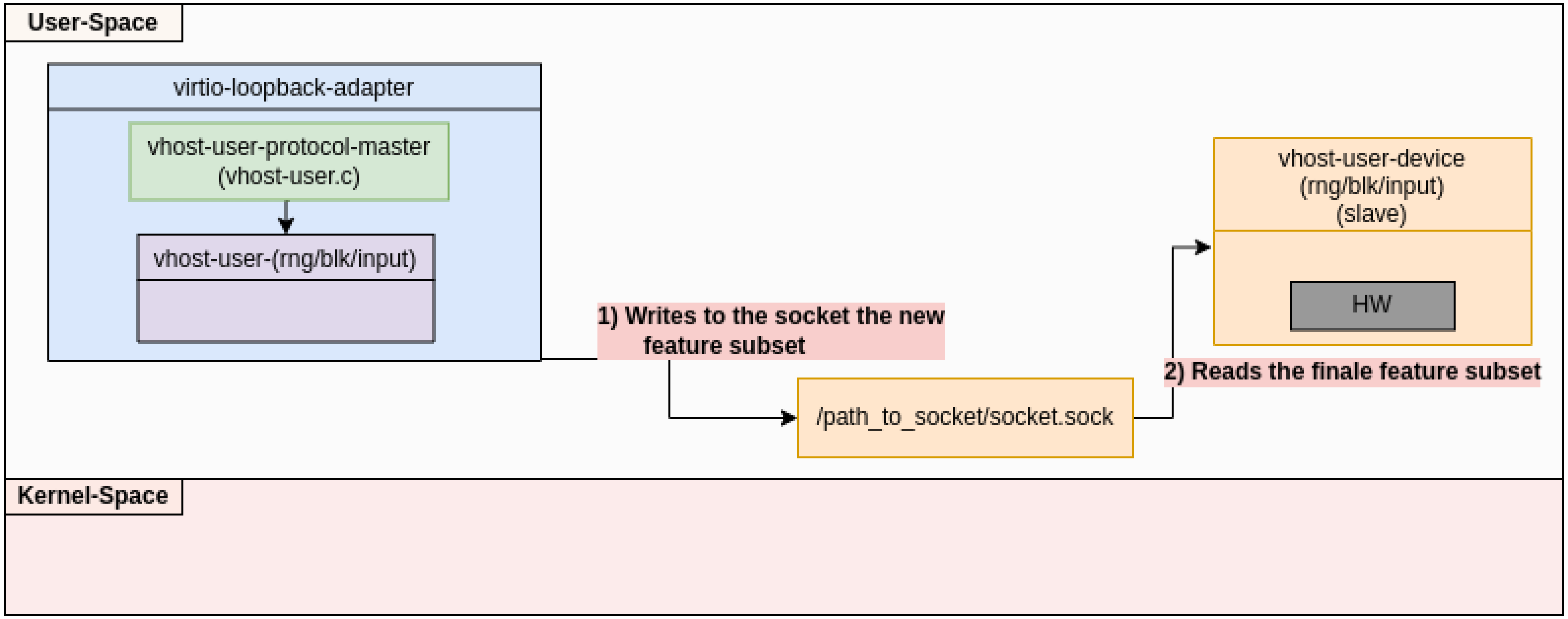
- [Stage 2] Adapter ↔ Transport driver
  - The adapter sends to the loopback driver:
    - Virtio specific information: Device id, Vendor, magic number
    - Virtio device features
  - The virtio-loopback-transport starts and register the corresponding virtio device (blk, input, rng)
  - Acknowledges the features and writes back to the adapter



User-Space
virtio-loopback-adapter
vhost-user-protocol-master (vhost-user.c)
vhost-user-(rng/blk/input)
3) Read Finalized (IOCTL) features
Kernel-Space
virtio-loopback
/dev/virtio-loopback
loopback-virtio-transport
1) Provides a struct of (IOCTL):
• Device features / Virtio version
• EVENTFDs (Adapter to notified by the transport)
Write finalized feautres

Control Plane (4)

- [Stage 3] Adapter ↔ Vhost-user-device
  - The adapter sends to the vhost-user-device:
    - The acknowledged virtio features

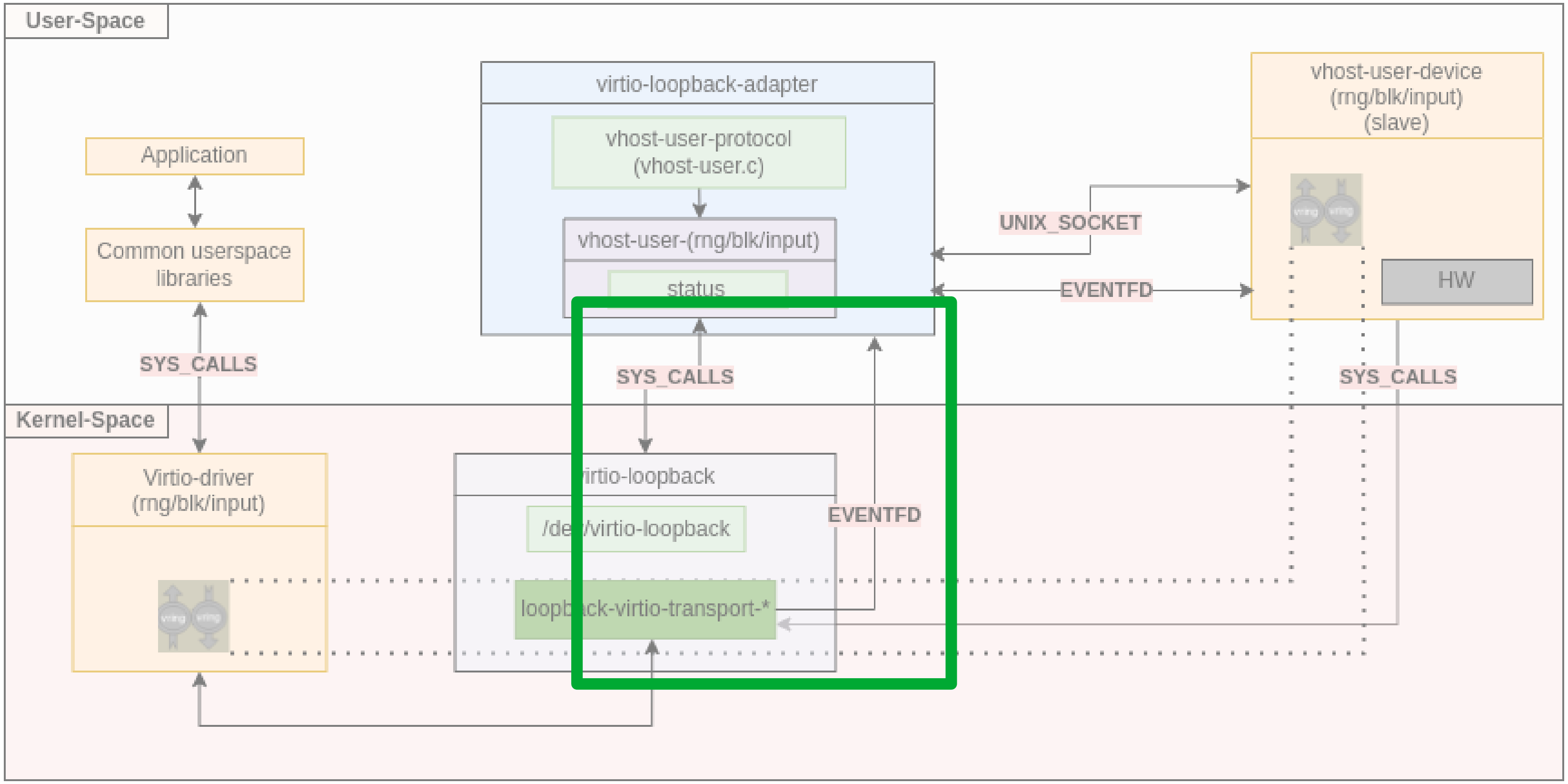

User-Space
virtio-loopback-adapter
vhost-user-protocol-master (vhost-user.c)
vhost-user-(rng/blk/input)
1) Writes to the socket the new feature subset
/path_to_socket/socket.sock
vhost-user-device (rng/blk/input) (slave)
HW
2) Reads the finale feature subset
Kernel-Space

Discussion Index

- Brief design description and current status
- Code review
  - Control plane
  - **Communication mechanisms**
  - Memory mapping (data plane)
- Live demo
- Upstream
- Next steps
- Questions

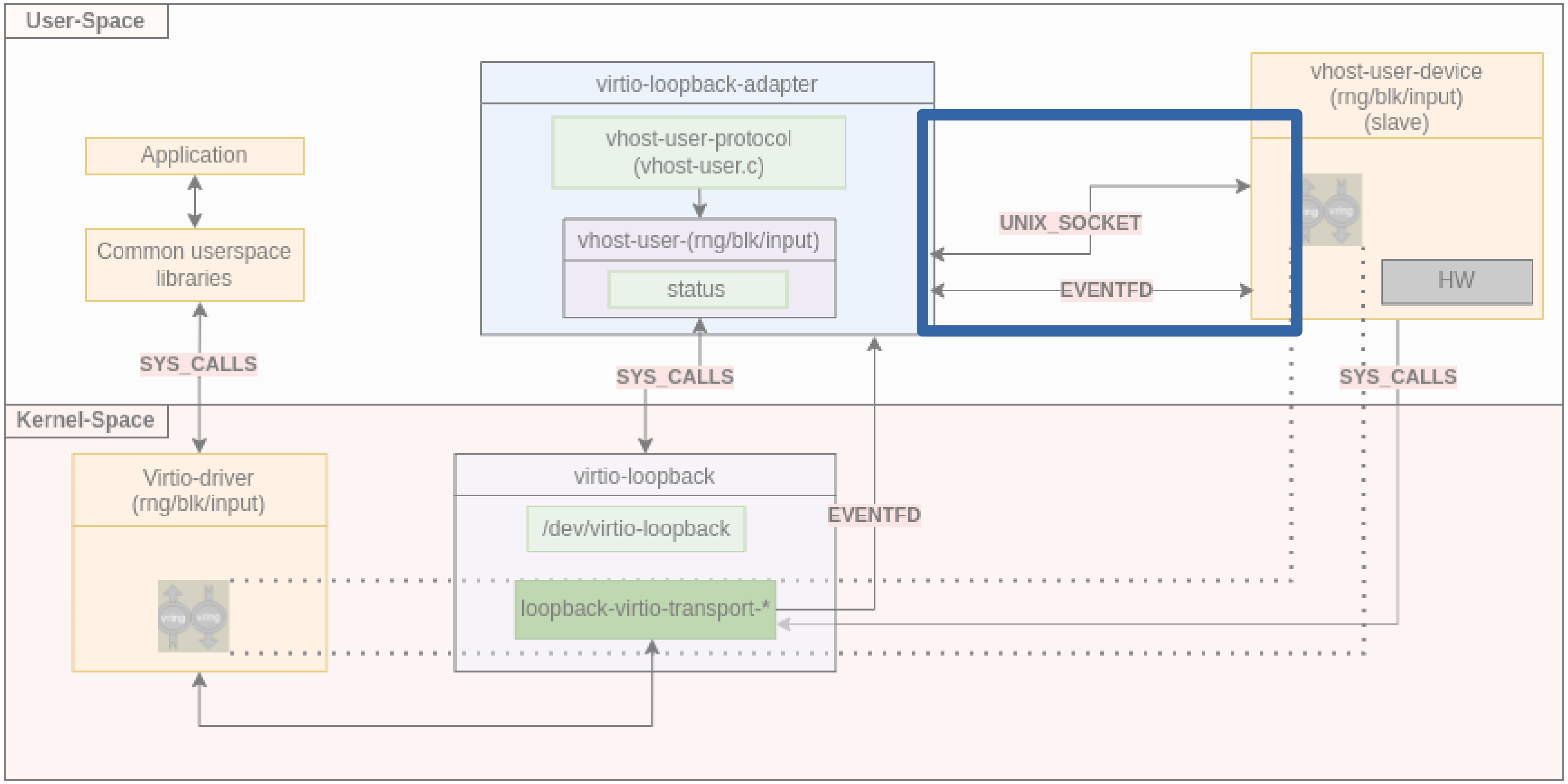
Communication mechanisms used between the components

- **Adapter ↔ driver : Eventfds, SYS_CALLS**


User-Space
virtio-loopback-adapter
vhost-user-protocol (vhost-user.c)
vhost-user-(rng/blk/input)
status
vhost-user-device (rng/blk/input) (slave)
Application
Common userspace libraries
UNIX_SOCKET
EVENTFD
HW
SYS_CALLS
SYS_CALLS
SYS_CALLS
Kernel-Space
Virtio-driver (rng/blk/input)
virtio-loopback
/dev/virtio-loopback
EVENTFD
loopback-virtio-transport-*
vring
vring

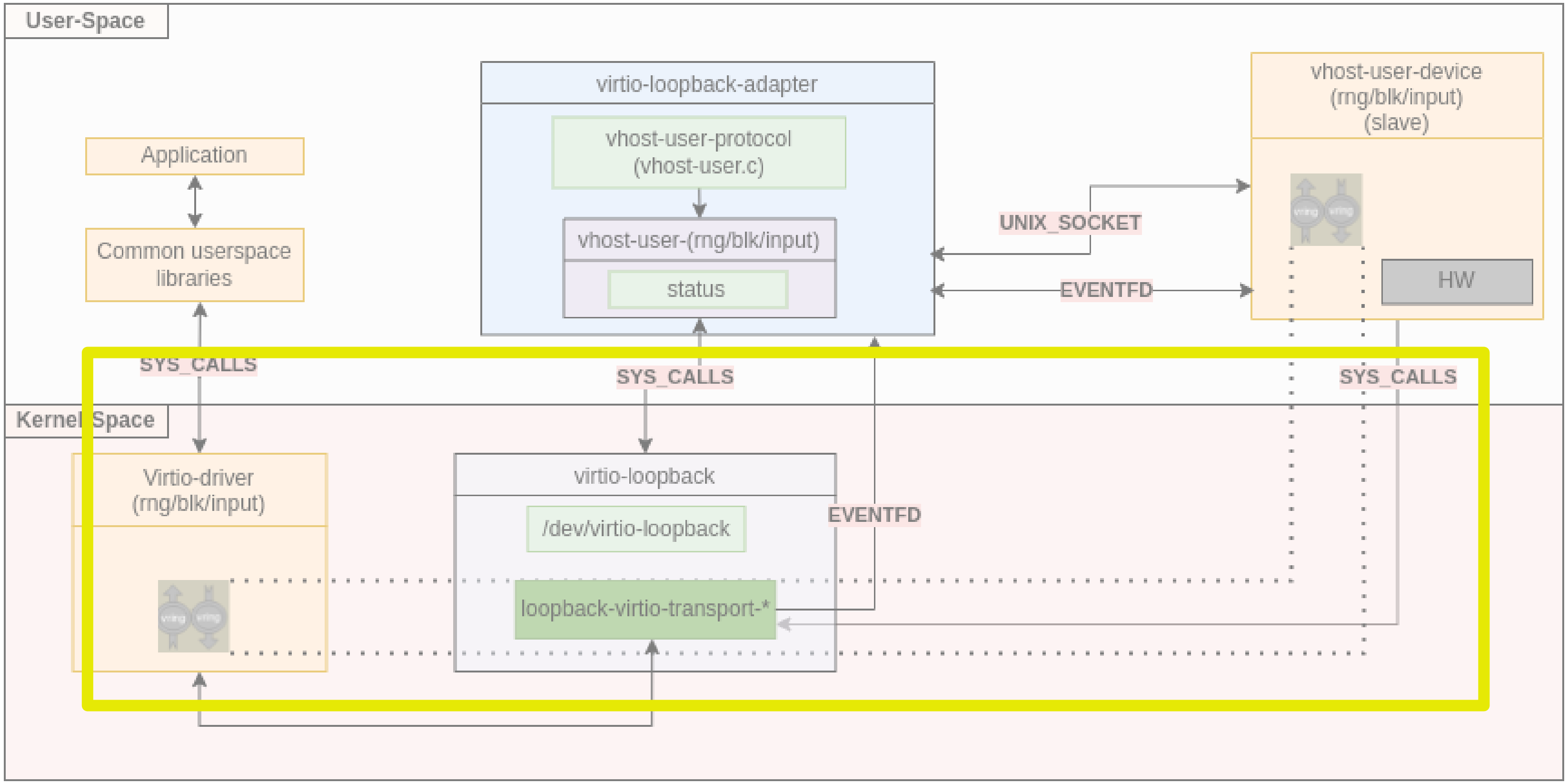
Communication mechanisms used between the components

- Adapter ↔ driver : Eventfds, SYS_CALLS
- **Vhost-user-device** ↔ **adapter** : Eventfds, Unix Socket


User-Space
Application
Common userspace libraries
SYS_CALLS
virtio-loopback-adapter
vhost-user-protocol (vhost-user.c)
vhost-user-(rng/blk/input)
status
SYS_CALLS
UNIX_SOCKET
EVENTFD
vhost-user-device (rng/blk/input) (slave)
vring
vring
HW
SYS_CALLS
Kernel-Space
Virtio-driver (rng/blk/input)
vring
vring
virtio-loopback
/dev/virtio-loopback
loopback-virtio-transport-*
EVENTFD

Communication mechanisms used between the components

- Adapter ↔ driver : Eventfds, SYS_CALLS
- Vhost-user-device ↔ adapter : Eventfds, Unix Socket
- **Vhost-user-device ↔ driver**: SYS_CALLS


User-Space
virtio-loopback-adapter
vhost-user-protocol (vhost-user.c)
vhost-user-(rng/blk/input)
status
vhost-user-device (rng/blk/input) (slave)
Application
Common userspace libraries
UNIX_SOCKET
EVENTFD
HW
vring
vring
SYS_CALLS
SYS_CALLS
SYS_CALLS
Kernel Space
Virtio-driver (rng/blk/input)
virtio-loopback
/dev/virtio-loopback
EVENTFD
loopback-virtio-transport-*
vring
vring

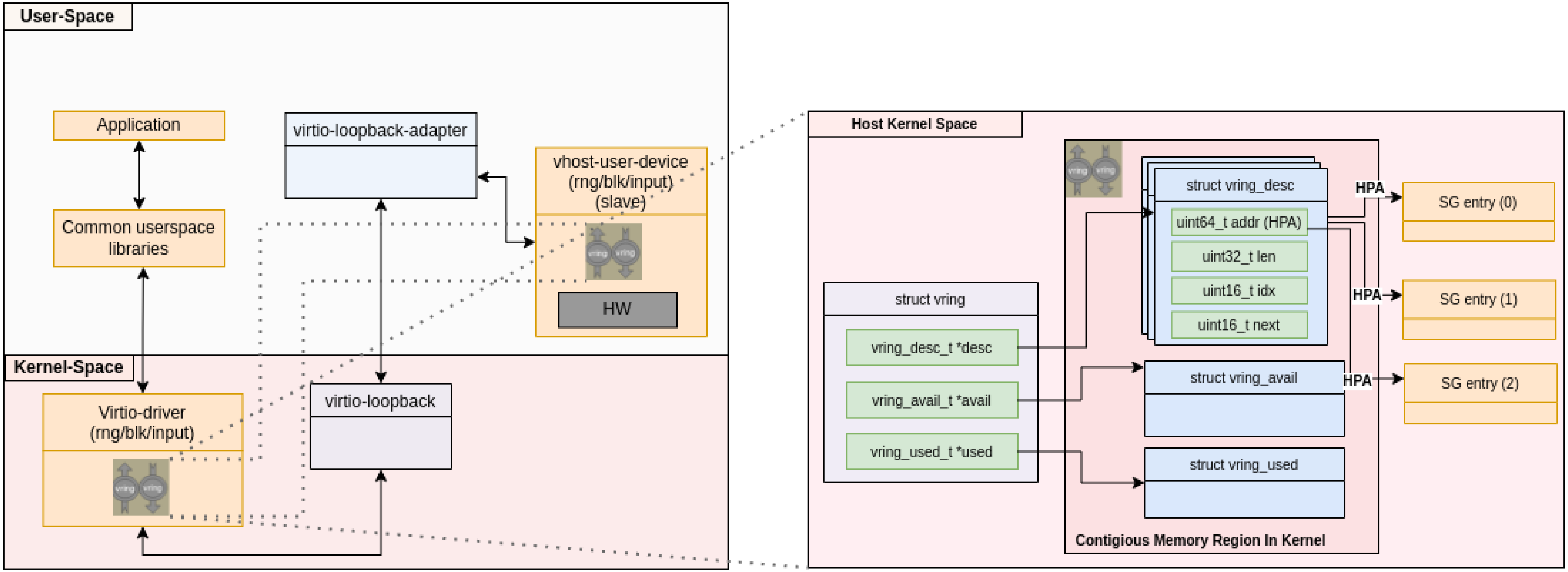
Discussion Index

- Brief design description and current status
- Code review
  - Control plane
  - Communication mechanisms
  - **Memory mapping (data plane)**
- Live demo
- Upstream
- Next steps
- Questions

Memory mapping mechanism (data plane)
Vhost-user mmap the Vrings (part 1)

The vhost-user-device access the data exchanged with the virtio device in two parts. First, access the vring data structure (in kernel space) and then access the SG list entries pointed by the vrings.

- **Vring data structure** The device uses **mmap** in order to obtain access



Memory mapping mechanism (data plane)

Vhost-user-device mmap the SG list (part 2)

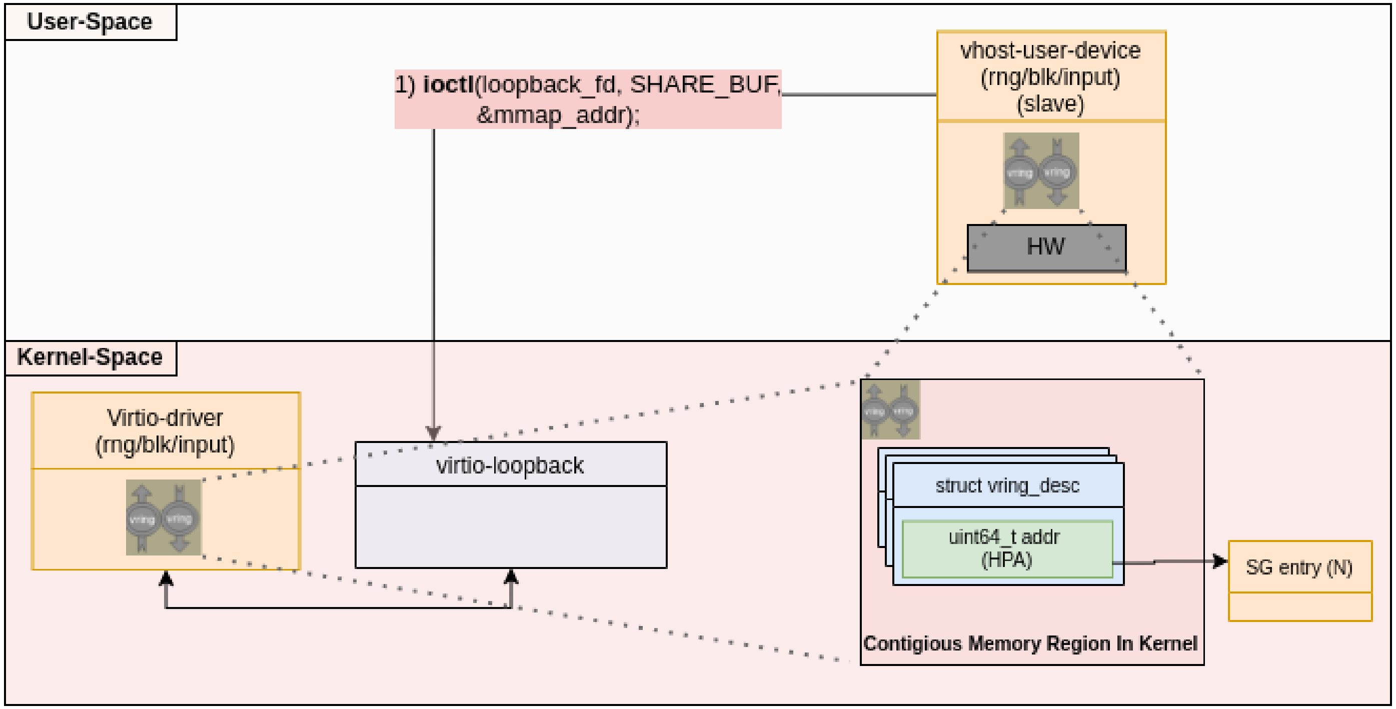
SG list entries The vhost-user-device uses the ioctl in order to ask the transport driver for access to the buffer (Host Physical Address) pointed the descriptors' table

1) If the driver has already shared this memory page with the vhost-user-device, returns an VMA (Virtual Memory address) in the 'mmap_addr' argument.
2) Otherwise the driver returns 0 and vhost-user-device goes on with calling 'mmap' function
3) The vhost-user-device obtains a pointer to this SG list element

Memory mapping mechanism (data plane)

Vhost-user-device mmap the SG list (part 2)

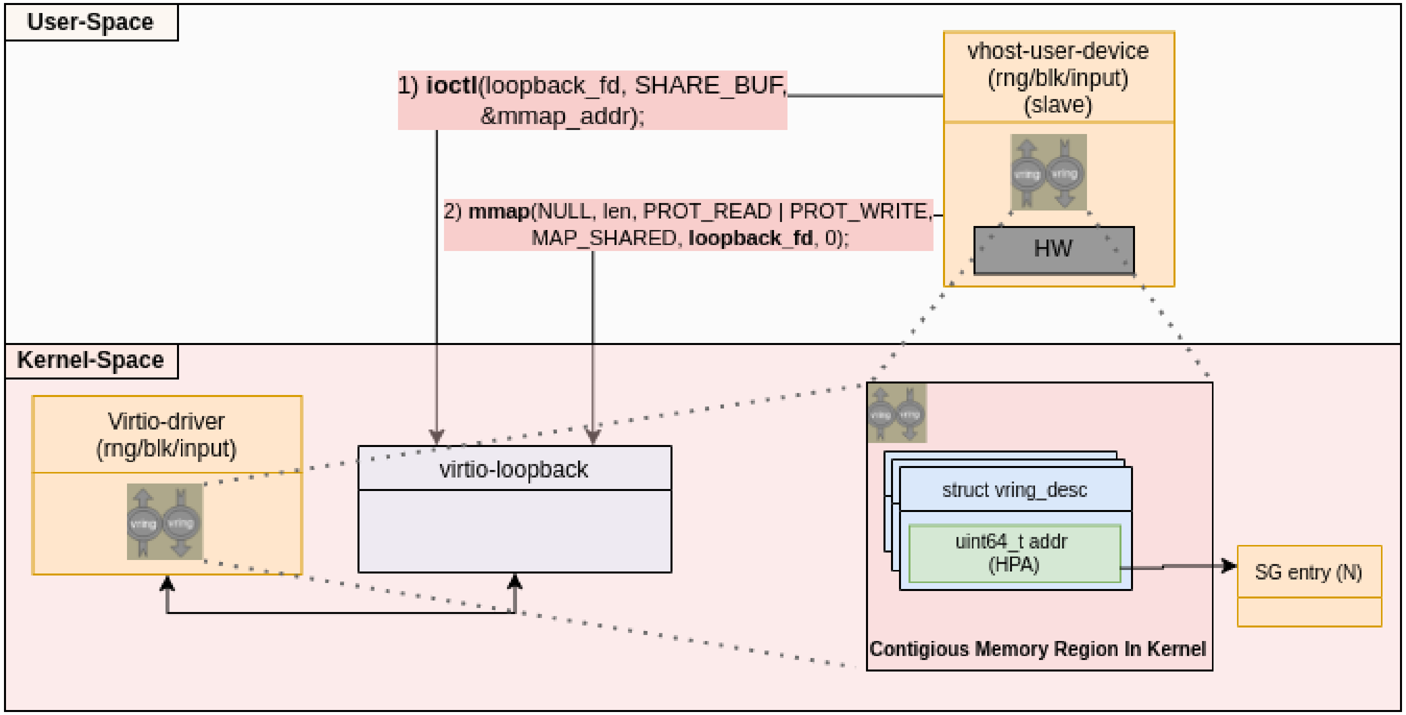
1) **If the driver has already shared this memory page with the vhost-user-device, returns an VMA (Virtual Memory address) in the ‘mmap_addr’ argument.**
2) Otherwise the driver returns 0 and vhost-user-device goes on with calling ‘mmap’ function
3) The vhost-user-device obtains a pointer to this SG list element


User-Space
1) ioctl(loopback_fd, SHARE_BUF, &mmap_addr);
vhost-user-device (rng/blk/input) (slave)
vring
vring
HW
Kernel-Space
Virtio-driver (rng/blk/input)
vring
vring
virtio-loopback
vring
vring
struct vring_desc
uint64_t addr (HPA)
SG entry (N)
Contigious Memory Region In Kernel

Memory mapping mechanism (data plane)

Vhost-user-device mmap the SG list (part 2)

1) If the driver has already shared this memory page with the vhost-user-device, returns an VMA (Virtual Memory address) in the ‘mmap_addr’ argument.
2) **Otherwise the driver returns 0 and vhost-user-device goes on with calling ‘mmap’ function**
3) The vhost-user-device obtains a pointer to this SG list element



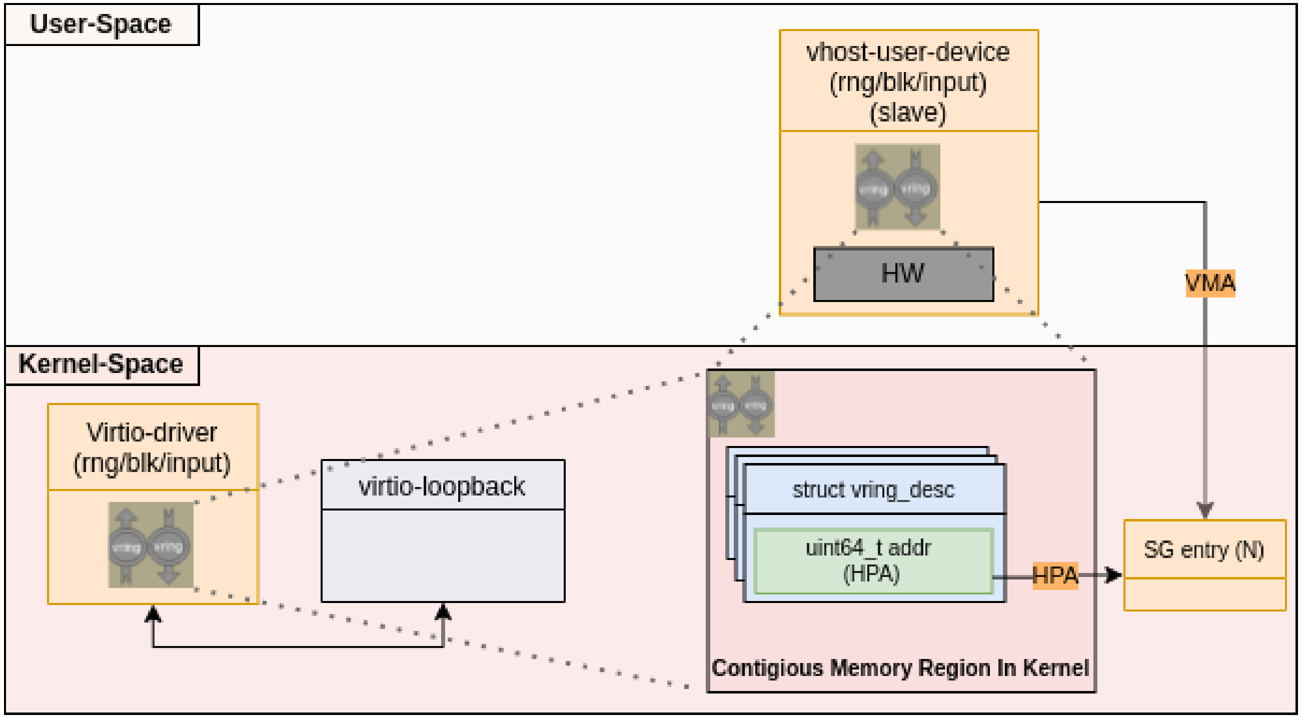
1) **ioctl**(loopback_fd, SHARE_BUF, &mmap_addr);

2) **mmap**(NULL, len, PROT_READ | PROT_WRITE, MAP_SHARED, **loopback_fd**, 0);

Memory mapping mechanism (data plane)

Vhost-user-device mmap the SG list (part 2)

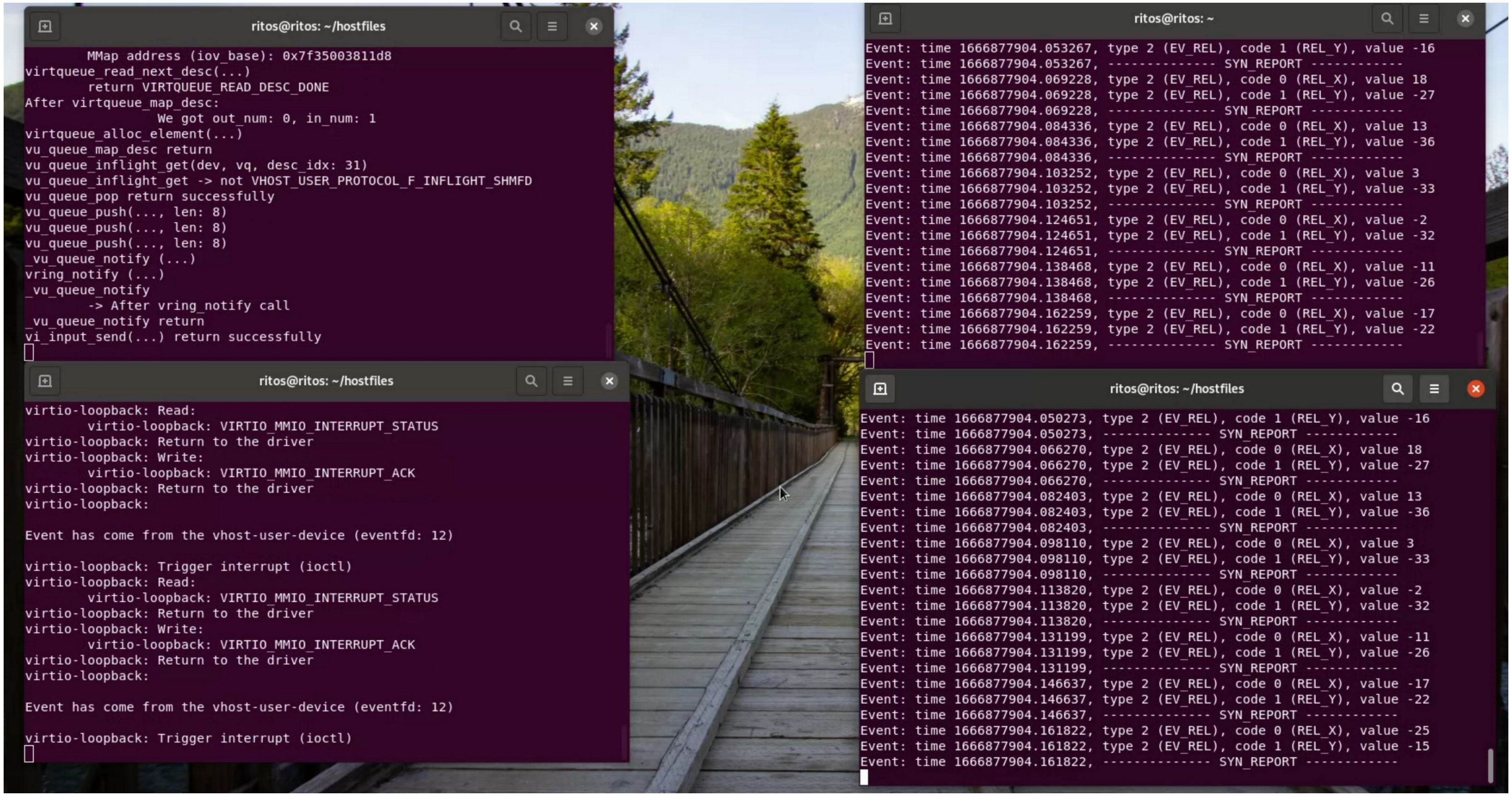
1) If the driver has already shared this memory page with the vhost-user-device, returns an VMA (Virtual Memory address) in the ‘mmap_addr’ argument.
2) Otherwise the driver returns 0 and vhost-user-device goes on with calling ‘mmap’ function
3) **The vhost-user-device obtains a pointer to this SG list element**



Discussion Index

- Brief design description and current status
- Code review
  - Control plane
  - Communication mechanisms
  - Memory mapping (data plane)
- **Live demo**
- Upstream
- Next steps
- Questions

Live demo (input + blk)



```
        MMap address (iov_base): 0x7f35003811d8
virtqueue_read_next_desc(...)
        return VIRTQUEUE_READ_DESC_DONE
After virtqueue_map_desc:
                 We got out_num: 0, in_num: 1
virtqueue_alloc_element(...)
vu_queue_map_desc return
vu_queue_inflight_get(dev, vq, desc_idx: 31)
vu_queue_inflight_get -> not VHOST_USER_PROTOCOL_F_INFLIGHT_SHMFD
vu_queue_pop return successfully
vu_queue_push(..., len: 8)
vu_queue_push(..., len: 8)
vu_queue_push(..., len: 8)
_vu_queue_notify (...)
vring_notify (...)
_vu_queue_notify
        -> After vring_notify call
_vu_queue_notify return
vi_input_send(...) return successfully
```

```
virtio-loopback: Read:
        virtio-loopback: VIRTIO_MMIO_INTERRUPT_STATUS
virtio-loopback: Return to the driver
virtio-loopback: Write:
        virtio-loopback: VIRTIO_MMIO_INTERRUPT_ACK
virtio-loopback: Return to the driver
virtio-loopback:

Event has come from the vhost-user-device (eventfd: 12)

virtio-loopback: Trigger interrupt (ioctl)
virtio-loopback: Read:
        virtio-loopback: VIRTIO_MMIO_INTERRUPT_STATUS
virtio-loopback: Return to the driver
virtio-loopback: Write:
        virtio-loopback: VIRTIO_MMIO_INTERRUPT_ACK
virtio-loopback: Return to the driver
virtio-loopback:

Event has come from the vhost-user-device (eventfd: 12)

virtio-loopback: Trigger interrupt (ioctl)
```

```
Event: time 1666877904.053267, type 2 (EV_REL), code 1 (REL_Y), value -16
Event: time 1666877904.053267, -------------- SYN_REPORT ------------
Event: time 1666877904.069228, type 2 (EV_REL), code 0 (REL_X), value 18
Event: time 1666877904.069228, type 2 (EV_REL), code 1 (REL_Y), value -27
Event: time 1666877904.069228, -------------- SYN_REPORT ------------
Event: time 1666877904.084336, type 2 (EV_REL), code 0 (REL_X), value 13
Event: time 1666877904.084336, type 2 (EV_REL), code 1 (REL_Y), value -36
Event: time 1666877904.084336, -------------- SYN_REPORT ------------
Event: time 1666877904.103252, type 2 (EV_REL), code 0 (REL_X), value 3
Event: time 1666877904.103252, type 2 (EV_REL), code 1 (REL_Y), value -33
Event: time 1666877904.103252, -------------- SYN_REPORT ------------
Event: time 1666877904.124651, type 2 (EV_REL), code 0 (REL_X), value -2
Event: time 1666877904.124651, type 2 (EV_REL), code 1 (REL_Y), value -32
Event: time 1666877904.124651, -------------- SYN_REPORT ------------
Event: time 1666877904.138468, type 2 (EV_REL), code 0 (REL_X), value -11
Event: time 1666877904.138468, type 2 (EV_REL), code 1 (REL_Y), value -26
Event: time 1666877904.138468, -------------- SYN_REPORT ------------
Event: time 1666877904.162259, type 2 (EV_REL), code 0 (REL_X), value -17
Event: time 1666877904.162259, type 2 (EV_REL), code 1 (REL_Y), value -22
Event: time 1666877904.162259, -------------- SYN_REPORT ------------
```

```
Event: time 1666877904.050273, type 2 (EV_REL), code 1 (REL_Y), value -16
Event: time 1666877904.050273, -------------- SYN_REPORT ------------
Event: time 1666877904.066270, type 2 (EV_REL), code 0 (REL_X), value 18
Event: time 1666877904.066270, type 2 (EV_REL), code 1 (REL_Y), value -27
Event: time 1666877904.066270, -------------- SYN_REPORT ------------
Event: time 1666877904.082403, type 2 (EV_REL), code 0 (REL_X), value 13
Event: time 1666877904.082403, type 2 (EV_REL), code 1 (REL_Y), value -36
Event: time 1666877904.082403, -------------- SYN_REPORT ------------
Event: time 1666877904.098110, type 2 (EV_REL), code 0 (REL_X), value 3
Event: time 1666877904.098110, type 2 (EV_REL), code 1 (REL_Y), value -33
Event: time 1666877904.098110, -------------- SYN_REPORT ------------
Event: time 1666877904.113820, type 2 (EV_REL), code 0 (REL_X), value -2
Event: time 1666877904.113820, type 2 (EV_REL), code 1 (REL_Y), value -32
Event: time 1666877904.113820, -------------- SYN_REPORT ------------
Event: time 1666877904.131199, type 2 (EV_REL), code 0 (REL_X), value -11
Event: time 1666877904.131199, type 2 (EV_REL), code 1 (REL_Y), value -26
Event: time 1666877904.131199, -------------- SYN_REPORT ------------
Event: time 1666877904.146637, type 2 (EV_REL), code 0 (REL_X), value -17
Event: time 1666877904.146637, type 2 (EV_REL), code 1 (REL_Y), value -22
Event: time 1666877904.146637, -------------- SYN_REPORT ------------
Event: time 1666877904.161822, type 2 (EV_REL), code 0 (REL_X), value -25
Event: time 1666877904.161822, type 2 (EV_REL), code 1 (REL_Y), value -15
Event: time 1666877904.161822, -------------- SYN_REPORT ------------
```

Discussion Index

- Brief design description and current status
- Code review
  - Control plane
  - Communication mechanisms
  - Memory mapping (data plane)
- Live demo
- **Upstream**
- Next steps
- Questions

Upstream

The key upstream target for this work is AGL. Yocto layers and recipes will be created and integrated in meta-egvirt. However, each and every component could be integrated in the reference community

- Virtio-loopback-transport driver → Linux Kernel
  - (Mailing list: dev-mailing-lists-virtio)
- Virtio-loopback-adapter → Qemu project
- Vhost-user devices:
  - Vhost-user-library → Qemu project
  - Vhost-user-rng (RUST)
    - https://github.com/rust-vmm/vhost-device
    - https://github.com/rust-vmm/vm-virtio
    - https://github.com/rust-vmm/vhost-user-backend
    - https://github.com/rust-vmm/vhost

What's the best strategy to propose these components to the community(ies)?

Discussion Index

- Brief design description and current status
- Code review
  - Control plane
  - Communication mechanisms
  - Memory mapping (data plane)
- Live demo
- Upstream
- **Next steps**
- Questions

Next steps

- Merge beta version into master branch
  - More testing
  - Polish the code / address comments
  - Update / add documentation

- Run benchmarks for vhost-user-blk & input
  - Using ‘dd’ for measuring blk throughput
  - Measure event latency with the help of ‘vinput’

- Create patches for the AGL
  - Meta-egvirt additions

Discussion Index

- Brief design description and current status
- Code review
  - Control plane
  - Communication mechanisms
  - Memory mapping (data plane)
- Live demo
- Upstream
- Next steps
- **Questions**

Questions

?

Virtual Open Systems

contact@virtualopensystems.com

Web: virtualopensystems.com

Products: http://www.virtualopensystems.com/en/products/

Demos: virtualopensystems.com/en/solutions/demos/

Guides: virtualopensystems.com/en/solutions/guides/

Research projects: virtualopensystems.com/en/research/innovation-projects/