VDUSE – vDPA Device in Userspace

Yongji Xie
ByteDance STE Team

KVM Forum 2021
Agenda

- Background
- Design & Implementation
- Status & Future Work
vDPA Overview

vDPA
- Virtio Data Path Acceleration

vDPA Device
- Virtio compatible datapath
- Vendor specific control path

vDPA Kernel Subsystem
- vDPA Bus
- vDPA Device (Abstraction)
- vDPA Bus Driver, including virtio-vDPA and vhost-vDPA
What is VDUSE

Based on vDPA subsystem, a framework to implement userspace vDPA Device

Provide an unified userspace approach for both VM and container workloads
Why Userspace

DEVELOPMENT LIFECYCLE  MAINTAINABILITY  FLEXIBILITY
Design & Implementation
Architecture Overview

- **Kernel Subsystem**
  - Virtio Driver
  - Vhost Subsystem
    - Vhost Device
      - Vhost-vDPA Bus Driver
  - Virtio Device
    - Virtio-vDPA Bus Driver

- **Device Emulation**
  - Device Emulation
  - Virtio Dataplane

- **VDUSE Driver**
  - vDPA Device
  - MMU-Based Software IOTLB

- **VDUSE Daemon**
  - /dev/vduse/$NAME

- **Control Path**
  - virtqueue
  - virtio bus

- **Data Path**
  - virtqueue
  - share memory
  - virtio bus
  - vDPA bus

- **Container**
  - Device Interface

- **VM**
  - Virtio Driver
  - virtqueue

- **Kernel**

---

**Container**

**VM**

**VDUSE Daemon**

**Kernel**
Control Path

Mostly handled in kernel

Some ioctls is introduced to initialize, query and update configuration of device, e.g.
- Initialize virtio features
- Update configuration space
- Query virtqueue information

Message mechanism is used to forward some control messages from vDPA Bus Driver to userspace, e.g.
- Set device status
- Get virtqueue state
The core is how to access the data of DMA buffer in userspace:
- In virtio-vdpa cases (Hosts), bounce buffer mechanism is introduced.
- In vhost-vdpa cases (VMs), memory is shared.

Eventfd is used to receive kick.

Ioctl is used to inject irq.
How to implement a VDUSE Daemon

Create VDUSE Device
- ioctl(/dev/vduse/control, VDUSE_CREATE_DEV, struct vduse_dev_config)

```c
struct vduse_dev_config {
    #define VDUSE_NAME_MAX 256
    char name[VDUSE_NAME_MAX]; /* vduse device name, uniquely identify a VDUSE device */
    __u32 vendor_id; /* virtio vendor id */
    __u32 device_id; /* virtio device id */
    __u64 features; /* virtio features */
    __u32 vq_num; /* the number of virtqueues */
    __u32 vq_align; /* the allocation alignment of virtqueue's metadata */
    __u32 reserved[13]; /* for future use, needs to be initialized to zero */
    __u32 config_size; /* the size of the configuration space */
    __u8 config[]; /* the buffer of the configuration space */
};
```
How to implement a VDUSE Daemon

Create VDUSE Device
- ioctl(/dev/vduse/control, VDUSE_CREATE_DEV, struct vduse_dev_config)
- A char device interface (/dev/vduse/$NAME) will be exported to userspace
How to implement a VDUSE Daemon

Create VDUSE Device
- ioctl(/dev/vduse/control, VDUSE_CREATE_DEV, struct vduse_dev_config)
- A char device interface (/dev/vduse/$NAME) will be exported to userspace

Setup Virtqueues
- ioctl(/dev/vduse/$NAME, VDUSE_VQ_SETUP, struct vduse_vq_config)

```
struct vduse_vq_config {
    __u32 index;  /* virtqueue index */
    __u16 max_size;  /* the max size of virtqueue */
    __u16 reserved[13];  /* for future use, needs to be initialized to zero */
};
```
How to implement a VDUSE Daemon

Begin processing VDUSE messages from /dev/vduse/$NAME
- The first messages will arrive while attaching the VDUSE device to vDPA bus via VDPA_CMD_DEV_NEW netlink message
How to implement a VDUSE Daemon

Begin processing VDUSE messages from /dev/vduse/$NAME
- The first messages will arrive while attaching the VDUSE device to vDPA bus via VDPA_CMD_DEV_NEW netlink message

- There are now three types of messages introduced:
  - VDUSE_GET_VQ_STATE: Get the state for virtqueue
  - VDUSE_UPDATE_IOTLB: Notify userspace to update the memory mapping for specified IOVA range
  - VDUSE_SET_STATUS: Set the device status
How to implement a VDUSE Daemon

Start the dataplane processing
- Start after DRIVER_OK status bit is set via the VDUSE_SET_STATUS message
How to implement a VDUSE Daemon

Start the dataplane processing
- Start after DRIVER_OK status bit is set via the VDUSE_SET_STATUS message
- Get information of virtqueues
  - ioctl(/dev/vduse/$NAME, VDUSE_VQ_GET_INFO, struct vduse_vq_info)

```c
struct vduse_vq_info {
    __u32 index;    /* virtqueue index */
    __u32 num;      /* the size of virtqueue */
    __u64 desc_addr; /* address of desc area */
    __u64 driver_addr; /* address of driver area */
    __u64 device_addr; /* address of device area */
    union {
        struct vduse_vq_state_split split; /* split virtqueue state */
        struct vduse_vq_state_packed packed; /* packed virtqueue state */
    };
    __u8 ready;     /* ready status of virtqueue */
};
```
How to implement a VDUSE Daemon

Start the dataplane processing
- Start after DRIVER_OK status bit is set via the VDUSE_SET_STATUS message

- Get information of virtqueues
  - ioctl(/dev/vduse/$NAME, VDUSE_VQ_GET_INFO, struct vduse_vq_info)

- Map IOVA regions related to virtqueues into userspace
  - ioctl(/dev/vduse/$NAME, VDUSE_IOTLB_GET_FD, struct vduse_iotlb_entry)

```
struct vduse_iotlb_entry {
    __u64 offset; /* the mmap offset on returned file descriptor */
    __u64 start; /* start of the IOVA range: [start, last] */
    __u64 last; /* last of the IOVA range: [start, last] */
    #define VDUSE_ACCESS_RO 0x1
    #define VDUSE_ACCESS_WO 0x2
    #define VDUSE_ACCESS_RW 0x3
    __u8 perm; /* access permission of this region */
};
```
How to implement a VDUSE Daemon

Start the dataplane processing
- Setup the kick eventfd for virtqueues (optional)
  - ioctl(/dev/vduse/$NAME, VDUSE_VQ_SETUP_KICKFD, struct vduse_vq_eventfd)

```c
struct vduse_vq_eventfd {
    __u32 index; /* virtqueue index */
    #define VDUSE_EVENTFD_DEASSIGN -1
    int fd; /* eventfd, -1 means de-assigning the eventfd */
};
```
How to implement a VDUSE Daemon

Start the dataplane processing
- Setup the kick eventfd for virtqueues (optional)
  - ioctl(/dev/vduse/$NAME, VDUSE_VQ_SETUP_KICKFD, struct vduse_vq_eventfd)

- Listen to the kick eventfd (optional) and consume the available ring
  - The buffer described by the descriptors in the descriptor table should be also mapped into userspace via the VDUSE_IOTLB_GET_FD ioctl before accessing
How to implement a VDUSE Daemon

Start the dataplane processing
- Setup the kick eventfd for virtqueues (optional)
  - ioctl(/dev/vduse/$NAME, VDUSE_VQ_SETUP_KICKFD, struct vduse_vq_eventfd)

- Listen to the kick eventfd (optional) and consume the available ring
  - The buffer described by the descriptors in the descriptor table should be also mapped into userspace via the VDUSE_IOTLB_GET_FD ioctl before accessing

- Inject an interrupt for specific virtqueue after the used ring is filled
  - ioctl(/dev/vduse/$NAME, VDUSE_VQ_INJECT_IRQ, __u32)
Status & Future Work
Status & Future Work

Status
- Kernel patchset v11 posted
  - https://lore.kernel.org/kvm/20210818120642.165-1-xieyongji@bytedance.com/
  - https://github.com/bytedance/linux/tree/vduse
- A userspace daemon example
  - https://github.com/bytedance/qemu/tree/vduse

Future Work
- Userspace library
- More device types support
- Improve performance
THANKS.