libvfio-user status update

john.levon@nutanix.com

thanos.makatos@nutanix.com

swapnil.ingle@nutanix.com

2021-09-16
"MUSER: mediated userspace device" by Thanos Makatos & Swapnil Ingle

- Working proof of concept, but had drawbacks
  - Kernel module, patch
- Much has changed in two years
  - Worked with community on better approach
vfio-user

- **vfio-user** is a protocol for managing external device servers
  - motivations: performance, security, resilience
- control plane focus
- message protocol over a communication channel
- analogous to, and based on, Linux's **vfio ioctl()** interface
- similar to **vhost-user**, but not **virtio** specific
- VMM agnostic
# vfio-user message types

<table>
<thead>
<tr>
<th>Name</th>
<th>Purpose</th>
<th>Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>VFIO_USER_VERSION</td>
<td>Lifecycle</td>
<td>client -&gt; server</td>
</tr>
<tr>
<td>VFIO_USER_DEVICE_RESET</td>
<td>Lifecycle</td>
<td>client -&gt; server</td>
</tr>
<tr>
<td>VFIO_USER_DEVICE_GET_INFO</td>
<td>Lifecycle</td>
<td>client -&gt; server</td>
</tr>
<tr>
<td>VFIO_USERDEVICE_GET_IRQ_INFO</td>
<td>IRQs</td>
<td>client -&gt; server</td>
</tr>
<tr>
<td>VFIO_USERDEVICE_SET_IRQS</td>
<td>IRQs</td>
<td>client -&gt; server</td>
</tr>
<tr>
<td>VFIO_USERDEVICE_GET_REGION_INFO</td>
<td>Device mem</td>
<td>client -&gt; server</td>
</tr>
<tr>
<td>VFIO_USERDEVICE_GET_REGION_IO_FDS</td>
<td>Device mem</td>
<td>client -&gt; server</td>
</tr>
<tr>
<td>VFIO_USER_REGION_READ/WRITE</td>
<td>Device mem</td>
<td>client -&gt; server</td>
</tr>
<tr>
<td>VFIO_USER_DMA_MAP/UNMAP</td>
<td>Guest mem</td>
<td>client -&gt; server</td>
</tr>
<tr>
<td>VFIO_USER_DMA_READ/WRITE</td>
<td>Guest mem</td>
<td>server -&gt; client</td>
</tr>
<tr>
<td>VFIO_USER_DIRTY_PAGES</td>
<td>Guest mem</td>
<td>client -&gt; server</td>
</tr>
</tbody>
</table>
**libvfio-user**

- C library with two roles
  - `vfio-user` socket server
  - PCI device emulation wrapper
- API for sync and async/non-blocking mode
int main()
{
    vfu_ctx_t *ctx = vfu_create_ctx(VFU_TRANS_SOCK, "/tmp.sock", 0, NULL, VFU_DEV_TYPE_PCI);

    vfu_pci_init(ctx, VFU_PCI_TYPE_EXPRESS, PCI_HEADER_TYPE_NORMAL, 0);
    vfu_pci_set_id(ctx, 0x494f, 0x0dc8, 0x0, 0x0);

    vfu_setup_region(ctx, VFU_PCI_DEV_BAR2_REGION_IDX, 0x100, bar2_access, VFU_REGION_FLAG_RW, NULL, 0, -1, 0);
    vfu_setup_device_nr_irqs(ctx, VFU_DEV_INTX_IRQ, 1);
    vfu_setup_device_dma(ctx, dma_register, dma_unregister);
    vfu_realize_ctx(ctx);

    /* accept() on the socket */
    vfu_attach_ctx(ctx);

    do {
        err = vfu_run_ctx(ctx);
    } while (err != -1);
}

static ssize_t bar2_access(vfu_ctx_t *ctx, char *buf, size_t count, loff_t off, bool is_write)
{
    if (off == DEV_REG_OFF_CTRL) {
        ...
    }
    ...
}

typedef struct vfu_dma_info {
    struct iovec iova;
    void *vaddr;
    struct iovec mapping;
    size_t page_size;
    uint32_t prot;
} vfu_dma_info_t;

/* handle new guest memory mapping */
static void dma_register(vfu_ctx_t *ctx, vfu_dma_info_t *info)
{
    ...
}
PCI device support

- each context corresponds to a PCI endpoint
- library handles (most) standard config space accesses
  - or, fully delegated (for **MPQEMU**)
- other accesses are handled via application callbacks
  - e.g. access to BAR regions
- user can register (extended) capabilities
  - vendor caps via callbacks
IRQ handling
Device region access: MMIO and DMA

- **VFIO_USERDEVICE_GET_REGION_INFO**
  - info on device regions (e.g. BARs)
  - can be (partially) mapped directly into VM

- **VFIO_USER_REGION_READ/WRITE**
  - read/write via `vfio-user` message
Guest memory: Device DMA

- `vfu_addr_to_sg(gpa, count, ...)`
  - convert guest PA to scatter-gather list array
- `vfu_dma_read/write(sg, ...)`
  - read/write via `vfio-user` message (so a misnomer!)
- `vfu_map_sg(sg, &iov, &iovcnt, ...)`
  - provide direct-mapped access
  - dirty page tracking
libvfio-user: Migration states

- Uses VFIO migration sub-protocol
  - Migration region with special regs.
  - libvfio-user migration API
- Transition: device transitions between states
  - running/stopped
  - pre-copy/stop-and-copy
  - resuming
libvfio-user: copying migration data

- Migrating from
  - get_pending_bytes: QEMU asks how much data need migrating
  - prepare/read_data: QEMU tells device to prep. migr. data, reads migr. data

- Migrating to
  - prepare_data: device tells QEMU where to write migr. data
  - write_data/data_written: guest writes migr. data to device
Demo

- GPIO sample from original MUSER presentation
  - Simple device with external pin
  - Pin can be either zero or one and can be read by host driver
  - Now using latest libvfio-user
- Live migrate from C implementation to Rust implementation
Future Work

• Stability (1.0 API/ABI)
• ioeventfd & ioregionfd support: cut QEMU out of the loop for port/MMIO accesses
• vIOMMU support
• DMA controller / API improvements
• Better PCI support (caps, non-endpoint)
• Multi-threading
• Restartable device emulation
  • Fault tolerance
  • Seamless upgrade
• Other transports, device types
• H/W device mediation / SR-IOV
libvfio-user

- GitHub: https://github.com/nutanix/libvfio-user
  - BSD licensed
  - Contributions are welcome
- Mailing list: libvfio-user-devel@nongnu.org
- Slack: https://libvfio-user.slack.com
Thank you