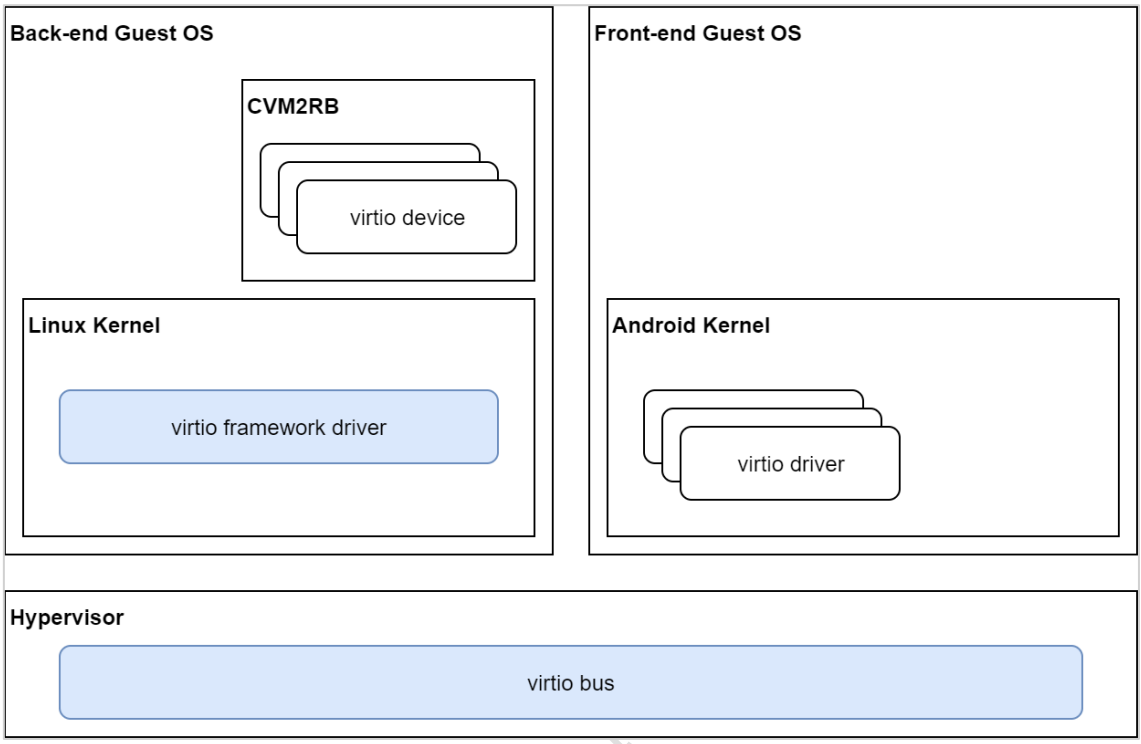
[virtio] Configuration Guide

2022.07.08

Table of Contents

1 The virtio framework

The virtio architecture is illustrated by the below figure:



It consists in the following software components:

- **CVM2RB**
  Provide the virtio devices
- **Linux virtio framework driver**
  Implements the virtio framework's control plane and data transaction
- **Android virtio driver**
  Implements the "guest" side of the virtio interface.
- **Hypervisor virtio bus**
  Trap any guest VM load/store instruction which accesses a virtio register and forward it to the host VM for emulation.

2 Virtio Framework Configuration

To use virtio devices, the Virtio Framework must be enabled in both the Linux SYS VM and Android VM. In addition, the Hypervisor configuration must describe how control path communication shall be established between the virtio devices and the virtio drivers. Moreover, the cvm2rb must automatically launch as a daemon in SYS.

2.1 Configuring the Linux SYS VM

The Linux SYS VM hosts cvm2rb, the virtio devices. It include the following new kernel configuration options:

- **VLX_VIRTIO**
  Enable the virtio framework that allows implementing virtio devices in the Linux kernel or in user space.
- **VLX_VIRTIO_VMSTOP_TIMEOUT_AUTOACK**
  Acknowledge VMSTOP events when the configured time-out expires.
- **VLX_VIRTIO_VMSTOP_TIMEOUT**
  Time-out on VMSTOP event acknowledgment in seconds.
- **VLX_VIRTIO_RETRY_TIMEOUT_MS**
  Retry time-out in milliseconds waiting for the device to be ready.

The first option configures the virtio framework kernel driver itself.
The following two options specify the actions taken by the kernel driver if an Android VM that hosts virtio drivers is abruptly stopped. The Android VM's restart is delayed until each virtio device acknowledges that its peer virtio driver is gone. Specifically, the devices must ensure that no DMA transaction targeting the Android VM's memory is still pending before they send their acknowledgement. If a virtual device fails to respond, the virtual framework device driver can issue the acknowledgement on its behalf after the configured timeout expires.
The last option specifies the action taken by the kernel driver when a virtio driver is running but the corresponding virtio device is not ready yet. A negative value means an infinite time-out; the framework retries the instruction until device becomes ready. A zero value means no time-out; if the device is not ready then -1 is immediately returned on a load instruction and a store instruction is ignored. A positive value means that the framework retries the instruction that accesses the virtio device until either the device becomes ready, or the time-out expires.
These are global parameters that apply to all the configured virtio devices.

Make sure that the Android's kernel features the virtual drivers, and then add the following options to the kernel configuration:

```
CONFIG_VLX_VIRTIO=y
CONFIG_VLX_VIRTIO_VMSTOP_TIMEOUT=5
CONFIG_VLX_VIRTIO_VMSTOP_TIMEOUT_AUTOACK=y
```


After rebooting the target platform, the virtio framework kernel driver creates new /dev/virtio/<device-vmid>/cvm2rb-pci character device files that match the virtio configuration present in the virtual platform device tree. These device files are used by the cvm2rb to access the virtio framework kernel driver.

2.2 Configuring the Android VM

The Android VM hosts the virtio drivers. The virtio drivers are built as part of the kernel or as external, dynamically-loadable kernel modules. The virtio drivers must be added to the Android VM configuration as described below.

The virtio PCI must be added to the Linux kernel's configuration device tree (DT).

```
pci {
    compatible = "pci-host-cam-generic";
    device_type = "pci";
    reg = <0x0 0x10000 0x1000000>;
    ranges =
       <0x3000000 0x00 0x01010000 0x00 0x01010000 0x0 0x00100000>;
    bus-range = <0x0 0x0>;
    #address-cells = <3>;
    #size-cells = <2>;
    #interrupt-cells = <0x1>;
    interrupt-map-mask = <0xf800 0x0 0x0 0x7>;
    interrupt-map = <
       0x0800 0x0 0x0 0x1 &gic 0x0 0x03 0x4
       0x1000 0x0 0x0 0x1 &gic 0x0 0x04 0x4
       0x1800 0x0 0x0 0x1 &gic 0x0 0x05 0x4
       ...
       0x9800 0x0 0x0 0x1 &gic 0x0 0x15 0x4
       0xa000 0x0 0x0 0x1 &gic 0x0 0x16 0x4
       >;
    dma-coherent;
};
```

The properties in this node are defined as:

- **compatible**
  "pci-host-cam-generic"
- **reg**
  Control registers base address and size of the PCI configuration space.
- **interrupts**
  The interrupts defined for each virtio devices

In the above configuration, the virtio PCI defined virtual address 0x10000 and 20 interrupts from SPI interrupt 3.
The virtual address and interrupt granted to the device are platform-specific. The Linux kernel's /proc/iomem file shows the virtio PCI address as following:

```
00010000-0100ffff : PCI ECAM
01010000-0110ffff : pci
  01010000-01017fff : 0000:00:01.0
    01010000-01017fff : virtio-pci-modern
  01018000-0101ffff : 0000:00:02.0
    01018000-0101ffff : virtio-pci-modern
  01020000-01027fff : 0000:00:03.0
    01020000-01027fff : virtio-pci-modern
  01028000-0102ffff : 0000:00:04.0
    01028000-0102ffff : virtio-pci-modern
  01030000-01037fff : 0000:00:05.0
    01030000-01037fff : virtio-pci-modern
```

The virtio PCI base address must be aligned on a page boundary. The entire 4KB page (i.e. 0x1000 bytes) must be available to the device. Similarly, the /proc/interrupt file shows the interrupts that are already claimed by each devices.

```
       CPU0   CPU1   CPU2   CPU3   CPU4
144:      0      0      0      0      0 GIC-0 35 Level virtio0
145:      1      0      0      0      0 GIC-0 36 Level virtio1
146:      7      0      0      0      0 GIC-0 37 Level virtio2
147:      0      0      0      0      0 GIC-0 38 Level virtio3
148:      0      0      0      0      0 GIC-0 39 Level virtio4
```

The VIRTIO_PCI and corresponding virtio driver of each device must be enabled in the Guest OS kernel configuration.

```
+CONFIG_HW_RANDOM_VIRTIO=m
+CONFIG_VIRTIO_CONSOLE=m
+CONFIG_VIRTIO_INPUT=m
+CONFIG_VIRTIO_NET=m
+CONFIG_VIRTIO_PCI=m
+CONFIG_VIRTIO_PCI_LEGACY=y
+CONFIG_VIRTIO_PCI_LIB=m
+CONFIG_VIRTIO_VSOCKETS=m
```

2.3 Configuring the Virtual Platform

The system configuration shall instruct the Hypervisor to map the virtio PCI base address to the Android VM's address space. This requires adding specific nodes to the Guest OS' virtual platform device tree.
The Hypervisor uses this information to intercept accesses to the area assigned to the virtio PCI and forward them to the cvm2rb hosted by the Linux SYS VM.

```
    cvm2rb-pci {
        compatible = "vl,virtio-device,mmio";
        reg = <0x00 0x00010000 0x01000000>,
              <0x00 0x01010000 0x00100000>;
        interrupts = <0x0 0x03 0x0>,
                     <0x0 0x04 0x0>,
                     <0x0 0x05 0x0>,
                     <0x0 0x06 0x0>,
                     ...
                     <0x0 0x1e 0x0>,
                     <0x0 0x1f 0x0>;
        vl,device-vmid = <2>;
        vl,device-name = "cvm2rb-pci";
    };
```


The properties are defined as follows:

- **compatible**
  "vl,virtio-device,mmio"
- **reg**
  The base address and size of the virtio device in the Guest OS address space.
- **interrupts**
  The interrupt sent to the Android VM when the virtio device notifies the driver.
- **vl,device-vmid**
  Identifies the VM where the virtio device implementation runs.

- **vl,device-name**
  Specifies the logical device name which will be available to the Backend VM (Linux SYS VM).

In addition, the Linux SYS VM must be granted access to the Android VM's memory partition, so that it can read and write the buffer descriptors, buffer rings and virtqueues allocated by the Android virtio drivers.
This is achieved by adding the following nodes to the SYS's virtual platform device tree:

```
    vm-memory@3 {
        compatible = "vl,vm-memory";
        import;
        reg = <0 0x0 0x0>;
        vl,vm-id = <3>;
        vl,mapped;
    };
```

In this example, the Android (vm3)'s entire memory partition will appear as reserved memory in the Linux SYS VM's address-space.
Finally, special virtual links must be configured to enable communication between the Linux SYS VM and the Hypervisor.
This is done by adding the following nodes to the Hypervisor configuration device tree:

```
&vm1_vdevs {
    virtio_bus: virtio@bus {
        compatible = "vl,virtio-bus";
        server;
    };
};
&vm2_vdevs {
    virtio@bus {
        peer-phandle = <&virtio_bus>;
        client;
    };
};
```

In the above example, the peer vdev nodes set up a "virtio-bus" vlink between the Linux SYS VM (vm2) and the Hypervisor, identified here as special vm1. This vlink is used by the virtio framework device driver to communicate with the Hypervisor's virtio bridge. The virtio bridge is the hypervisor component that enables control-path commands and data-path notifications to be exchanged between the Linux SYS VM and the Android VM.

2.4 Configuring the cvm2rb

In current release the cvm2rb is launched by systemd as a service:

```
root@euto-v9-discovery:~# cat /lib/systemd/system/virtio-devices.service
[Unit]
Description=Virtio Devices Service

# Make sure that the test partition device node is available before we
attempt
# to start cvm2rb.
After=dev-disk-by\x2dpartlabel-test_part.device
Requires=dev-disk-by\x2dpartlabel-test_part.device

[Service]
Type=simple
Restart=always
ExecStart=/usr/bin/cvm2rb run \
    --pci-bus-devfd=/dev/virtio/vm3/cvm2rb-pci \
    --cid=3 \
    --serial=hardware=virtio-console,num=1,type=file,path=/tmp/virtio-
console-output \
    --rwdisk=/dev/disk/by-partlabel/test_part \
    --host_ip 192.168.0.2 --netmask 255.255.255.0 --mac 42:42:42:42:42:42

# Add --evdev=/dev/input/eventXX to cvm2rb command line to expose the
specified
# input device through virtio-input.

[Install]
WantedBy=multi-user.target
```

In this service (showed above), cvm2rb enabled virtio-console, virtio-blk and virtio-net. If you want to add other virtio device, or you want to add you own virtio device, you can implement it in CVM2B and add the parameters of your own virtio device