DOCA DevEmu Virtio
# Table of contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOCA DevEmu Virtio-FS</td>
<td>9</td>
</tr>
</tbody>
</table>
Introduction

DOCA DevEmu Virtio, which is part of the DOCA Device Emulation subsystem, introduces low-level software APIs that provide building blocks for developing and manipulating virtio devices using the device emulation capability of NVIDIA® BlueField®. This subsystem incorporates a core library that handles a common logic for various types of virtio devices, such as virtio-FS. One of its key responsibilities is managing the standard "device reset" procedure outlined in the virtio specification. This core library serves as a foundation for implementing shared functionalities across different virtio device types, ensuring consistency and efficiency in device operations and behaviors.

DOCA provides support for emulating virtio devices over the PCIe bus. The PCIe transport is commonly used for virtio devices. Configuration, discovery, and features related to PCIe (such as MSI-X and PCIe device hot plug/unplug) are managed through the DOCA DevEmu PCI APIs. This modular design enables each layer within the DOCA Device Emulation subsystem to manage its own business logic and facilitates seamless integration with the other layers, ensuring independent functionality and operation throughout the system.

This subsystem also includes device-specific libraries for various virtio device types (e.g., a library for a virtio-FS device).

From the host's perspective, there is no difference between para-virtual, DOCA-emulated, and actual hardware devices. The host uses the same virtio device drivers to operate the device under all circumstances.

Prerequisites

Virtio device emulation is part of the DOCA Device Emulation subsystem. It is, therefore, recommended to read the following guides beforehand:
• DOCA Device Emulation

• DOCA DevEmu PCI

Environment

DOCA DevEmu Virtio is supported on the BlueField target only.

The BlueField must meet the following requirements

• DOCA version 2.7.0 or greater

• BlueField-3 firmware 32.41.1000 or higher

![Info]

Please refer to the [DOCA Backward Compatibility Policy](#).

Library must be run with root privileges.

Architecture

The DOCA DevEmu Virtio core library provides the following software abstractions:

• Virtio type – extends the PCIe type, represents common/default virtio configurations of emulated virtio devices

• Virtio device – extends the PCIe device, represents an instance of an emulated virtio device

• Virtio IO context – represents a progress context which is responsible for processing virtio descriptors and their associated virtio queues

DOCA DevEmu Virtio library does not provide APIs to configure the entire BAR layout of the virtio device as this configuration is done internally. However, the library offers APIs to configure some of the registers within the common configuration structure (see [Virtio Device](#)).
Virtio Common Configuration

According to the virtio specification, the common PCIe configuration structure layout is as follows:

```c
struct virtio_pci_common_cfg {
    /* About the whole device. */
    le32 device_feature_select; /* read-write */
    le32 device_feature; /* read-only for driver */
    le32 driver_feature_select; /* read-write */
    le32 driver_feature; /* read-write */
    le16 config_msix_vector; /* read-write */
    le16 num_queues; /* read-only for driver */
    u8 device_status; /* read-write */
    u8 config_generation; /* read-only for driver */

    /* About a specific virtqueue. */
    le16 queue_select; /* read-write */
    le16 queue_size; /* read-write */
    le16 queue_msix_vector; /* read-write */
    le16 queue_enable; /* read-write */
    le16 queue_notify_off; /* read-only for driver */
    le64 queue_desc; /* read-write */
    le64 queue_driver; /* read-write */
    le64 queue_device; /* read-write */
    le16 queue_notify_data; /* read-only for driver */
    le16 queue_reset; /* read-write */
};
```

The DOCA DevEmu Virtio core library provides the ability to configure some of the listed registers using the appropriate setters.

Virtio Type

The virtio type extends the PCIe type and describes the common/default configuration of emulated virtio devices, including the common virtio configuration space registers (such as num_queues, queue_size, and others).
Virtio type is currently read-only (i.e., only getter APIs are available to retrieve information). The following methods can be used for this purpose:

- `doca_devemu_virtio_type_get_num_queues` – for getting the default initial value of the `num_queues` register for the associated virtio devices
- `doca_devemu_virtio_type_get_queue_size` – for getting the default initial value of the `queue_size` register for the associated virtio devices
- `doca_devemu_virtio_type_get_device_features_63_0` – for getting the default initial values of the `device_feature` bits (0-63) for the associated virtio devices
- `doca_devemu_virtio_type_get_config_generation` – for getting the default initial value of the `config_generation` register for the associated virtio devices

The default virtio type is extended by a virtio device's specific type (e.g., virtio-FS type) and cannot be created on demand.

**Virtio Device**

The virtio device extends the PCIe device. Before using the DOCA DevEmu Virtio device, it is recommended to read the guidelines of DOCA DevEmu PCI device and DOCA Core context configuration phase.

The virtio device is extended by a virtio-specific device (e.g., virtio FS device) and cannot be created on demand.

**Virtio Device Configurations**

The virtio device context can be configured to match the application use case and optimize the utilization of system resources.

**Mandatory Configurations**

The mandatory configurations are as follows:

- `doca_devemu_virtio_dev_set_num_required_running_virtio_io_ctxs` – to set the number of required running virtio IO contexts to be bound to the virtio device context. The virtio device context does not move to running state (according to the DOCA Core
context state machine) before having this amount of running virtio IO contexts bound to it.

- doca_devemu_virtio_dev_event_reset_register – to register to the virtio device reset event. This configuration is mandatory

Optional Configurations

The optional configurations are as follows:

- doca_devemu_virtio_dev_set_device_features_63_0 – to set the values of the device_feature bits (0-63). If not set, the default value is taken from the virtio type configuration.

- doca_devemu_virtio_dev_set_num_queues – to set the value of the num_queues register. If not set, the default value is taken from the virtio type configuration.

- doca_devemu_virtio_dev_set_queue_size – to set the value of the queue_size register for all virtio queues. If not set, the default value is taken from the virtio type configuration.

Events

DOCA DevEmu Virtio device exposes asynchronous events to notify about sudden changes, according to DOCA Core architecture.

Info

Common events are described in DOCA DevEmu PCI Device events and in DOCA Core Event.

Reset Event

The reset event allows users to receive notifications whenever the device reset flow is initialized by the device driver. Upon receiving this event, it is guaranteed that no further
requests are routed to the user via any associated virtio IO context until the reset flow is completed.

To complete the reset flow the user must:

1. Flush all outstanding requests back to the virtio IO context associated with the request.

2. Perform one of the following:
   - Call `doca_devemu_virtio_dev_reset_complete`.
   - Follow FLR flow:
     1. `doca_ctx_stop` – stop the virtio device with its associated virtio IO contexts and wait until the device and its associated virtio IO contexts transition to idle state
     2. `doca_ctx_start` – start the virtio device with its associated virtio IO contexts and wait until the device and its associated virtio IO contexts transition to running state

Now, the device and its associated virtio IO contexts should be fully operational again, the device is allowed to route new requests via any associated virtio IO context.

**Virtio IO**

The virtio IO context extends the DOCA Core context. Before using the DOCA DevEmu Virtio IO, it is recommended to read the guidelines of DOCA Core context configuration phase.

This context is associated with a single DOCA virtio device and is bound to the virtio device context upon start. The virtio IO context is a thread-unsafe object and is progressed by a single DOCA Core progress engine. Usually, users configure a single virtio IO context per BlueField core used by the application service.
The virtio IO context is responsible to route new incoming virtio requests towards the application and to complete handled requests back to the device driver. It can only route requests while in running state and when its associated virtio device is also in running state.
DOCA DevEmu Virtio-FS

Note

This library is supported at alpha level; backward compatibility is not guaranteed.

Introduction

The DOCA DevEmu Virtio-FS library is part of the DOCA DevEmu Virtio subsystem. It provides low-level software APIs that provide building blocks for developing and manipulating virtio filesystem devices using the device emulation capability of NVIDIA® BlueField® DPUs.

DOCA supports emulating virtio-FS devices over the PCIe bus. The PCIe transport is the common transport used for virtio devices. Configuration, discovery, and features related to PCIe (e.g., MSI-X and PCIe device hot plug/unplug) are managed through the DOCA DevEmu PCI APIs. Configuring common virtio registers and handling generic virtio logic (e.g., virtio device reset flow) is handled by the DOCA Virtio common library. This modular design enables each layer within the DOCA Device Emulation subsystem to manage its own business logic. It facilitates seamless integration with the other layers, ensuring independent functionality and operation throughout the system.

The DOCA Devemu Virtio-FS library efficiently handles virtio descriptors, carrying FUSE requests, sent by the device driver, and translating them into abstract virtio-FS requests which are then routed to the user. This translation process ensures that the underlying device-specific acceleration details are abstracted away, allowing applications to interact with abstracted virtio-FS requests.

Users of this library are responsible for developing a virtio-FS controller, which manages the underlying DOCA Devemu Virtio-FS device alongside an external backend file system which is outside DOCA's scope. The controller application is designed to receive DOCA
Virtio-FS requests and process them according to virtio-FS and FUSE specifications, translating FUSE-based commands into the appropriate backend filesystem protocol.

**Prerequisites**

Virtio-FS device emulation is part of DOCA DevEmu Virtio subsystem. It is, therefore, recommended to read the following guides before proceeding:

- [DOCA Device Emulation](#)
- [DOCA DevEmu PCI](#)
- [DOCA DevEmu Virtio](#)

**Environment**

DOCA DevEmu Virtio-FS is supported on the BlueField target only. The BlueField must meet the following requirements:

- DOCA version 2.7.0 or greater
- BlueField-3 firmware 32.41.1000 or higher

**Info**

Please refer to the [DOCA Backward Compatibility Policy](#).

**Note**

Library must be run with root privileges.

Perform the following:
1. Configure BlueField to work in DPU mode as described in NVIDIA BlueField Modes of Operation.

2. Enable emulation by running the following on the host or DPU:

   ```
   host/dpu> sudo mlxconfig -d /dev/mst/m41692_pciconf0 s VIRTIO_FS_EMULATION_ENABLE=1
   ```

3. Configure the number of static virtio-FS physical functions and the number of MSIX for each physical function to expose. This can be done by running the following command on the DPU:

   ```
   host/dpu> sudo mlxconfig -d /dev/mst/m41692_pciconf0 s VIRTIO_FS_EMULATION_NUM_PF=2
   VIRTIO_FS_EMULATION_NUM_MSIX=18
   ```

4. Perform a BlueField system reboot for the mlxconfig settings to take effect.

   **Note**
   DOCA does not support hot plugging virtio-FS PF devices into the host PCIe subsystem or SR-IOV for virtio-FS devices.

**Architecture**

The DOCA DevEmu Virtio-FS library provides the following main software abstractions:

- The virtio-FS type – extends the virtio type; represents common/default virtio-FS configurations of emulated virtio-FS devices

- The virtio-FS device – extends the virtio device; represents an instance of an emulated virtio-FS device

- The virtio-FS IO context – extends the virtio IO context; represents a progress context responsible for processing virtio descriptors, carrying FUSE requests, and
their associated virtio queues (e.g., hiprio, request, admin, and notification queues).

- The virtio-FS request

## Virtio-FS Feature Bits

According to the virtio specification, a virtio-FS device may report support for VIRTIO_FS_F_NOTIFICATION which indicates the ability to handle FUSE notify messages sent via the notification queue.

### Note

Currently, DOCA does not support reporting the VIRTIO_FS_F_NOTIFICATION feature to the driver.

## Virtio-FS Configuration Layout

According to the virtio specification, the virtio-FS configuration structure layout is as follows:

```c
struct virtio_fs_config {
    char tag[36];
    le32 num_request_queues;
    le32 notify_buf_size;
};
```

The tag and num_request_queues fields are always available. The notify_buf_size field is only available when VIRTIO_FS_F_NOTIFICATION is set.

### Note
**Virtio-FS Type**

The virtio-FS type extends the virtio type and describes the common/default configuration of emulated virtio-FS devices, including some of the virtio-FS configuration space registers (e.g., `num_request_queues`).

Currently, the virtio-FS type is read-only (i.e., only getter APIs are available to retrieve information). The following method can be used for this purpose:

- `doca_devemu_vfs_type_get_num_request_queues` – to get the default initial value of the `num_request_queues` register for the associated virtio-FS devices

DOCA supports the default virtio-FS type. To retrieve the default virtio-FS type, users use the following method:

- `doca_devemu_vfs_is_default_vfs_type_supported` – check if the default DOCA Virtio-FS type is supported by the device. If supported:
  - `doca_dev_open` – open supported DOCA device
  - `doca_devemu_vfs_find_default_vfs_type_by_dev` – get the default DOCA Virtio-FS type associated with the device

**Virtio-FS Device**

The virtio-FS device extends the virtio device. Before using the DOCA DevEmu Virtio-FS device, it is recommended to read the guidelines of [DOCA DevEmu Virtio device](#), [DOCA DevEmu PCI device](#), and [DOCA Core context configuration phase](#).

This section describes how to create, configure, and operate the virtio-FS device.
**Virtio-FS Device Configurations**

The virtio-FS emulated device might be in several different visibility levels from the host point of view:

- Visible/non-visible to the PCIe subsystem – If the device is visible to the PCIe subsystem, the user is not able to configure PCIe-related parameters (e.g., number of MSI-X vector, subsystem_id).

- Visible/non-visible to the virtio subsystem – If the device is visible to the virtio subsystem, the user is not be able to configure virtio-related parameters (e.g., number of queues, queue_size).

The flow for creating and configuring a virtio-FS device is as follows:

1. `doca_devemu_vfs_dev_create` – Create a new DOCA DevEmu Virtio-FS device instance.
2. `doca_devemu_vfs_dev_set_tag` – Set a unique tag for the device according to the virtio specification.
3. `doca_devemu_vfs_dev_set_num_request_queues` – Set the number of request queues for the device.
4. `doca_devemu_vfs_dev_set_vfs_req_user_data_size` – Set the user data size of the virtio-FS request. If set, a buffer with this size is allocated for each DOCA DevEmu Virtio-FS on behalf of the user.
5. Configure virtio-related parameters as described in [DOCA Virtio configurations](#).
6. Configure PCIe-related parameters as described in [DOCA DevEmu PCI configurations](#).

**Note**

doca_devemu_virtio_dev_set_num_queues should be equal to the number of request queues +1 (for the hirio queue) since DOCA does not currently support the virtio-FS notification queue.

6. Configure PCIe-related parameters as described in [DOCA DevEmu PCI configurations](#).
7. doca_ctx_start – Start the virtio-FS device context to finalize the configuration phase.

- The virtio-FS device object follows the DOCA context state machine as described in **DOCA Core context state machine**

- The virtio-FS device context moves to **running** state after the initial number of virtio IO contexts is bound to it and turns to **running** state, as described at **DOCA DevEmu Virtio configurations**

At this point, the DOCA Devemu Virtio-FS context is fully operational.

**Mandatory Configurations**

The following are mandatory configurations:

- `doca_devemu_vfs_dev_set_tag` – set a unique tag for the device

**Optional Configurations**

The optional configurations are as follows:

- `doca_devemu_vfs_dev_set_num_request_queues` – set the number of request queues for the device. If not set, the default value is taken from the virtio-FS type configuration.

- `doca_devemu_vfs_dev_set_vfs_req_user_data_size` – set the user data size of the virtio-FS request. If not set, user data size defaults to 0.

**Virtio-FS Device Events**

DOCA DevEmu Virtio-FS device exposes asynchronous events to notify about changes that happen out of the blue, according to the DOCA Core architecture.

Common events are described in **DOCA DevEmu Virtio device events**, **DOCA DevEmu PCI device events** and in **DOCA Core event**.
Virtio-FS IO

The virtio-FS IO context extends the Virtio IO Context. To start using the DOCA DevEmu Virtio-FS IO it is recommended to read the guidelines of DOCA DevEmu Virtio IO and DOCA Core context configuration phase.

This section describes how to create, configure and operate the virtio-FS IO context.

Virtio-FS IO Configurations

The flow for creating and configuring a virtio-FS IO context should be as follows:

1. `doca_devemu_vfs_io_create` – Create a new DOCA DevEmu Virtio-FS IO instance.

2. `doca_devemu_vfs_io_event_vfs_req_notice_register` – Register event handler for incoming virtio-FS requests.

3. `doca_ctx_start` – Start the virtio-FS IO context to finalize the configuration phase. The virtio-FS IO object follows the DOCA Core context state machine. The virtio-FS device context moves to running state after the initial number of virtio-FS IO contexts is bound to it and moves to running state (as described at DOCA DevEmu Virtio configurations).

Mandatory Configurations

The following are mandatory configurations:

- `doca_devemu_vfs_io_event_vfs_req_notice_register` – Register event handler for incoming virtio-FS requests is mandatory

Virtio-FS Request

The virtio-FS request object serves as an abstraction for handling requests arriving on virtio-FS queues, including high-priority, request, or notification queues. These requests are initially generated by the device driver through created virtio queues and then routed to the user via a registered event handler, which is set up using `doca_devemu_vfs_io_event_vfs_req_notice_register`, on the associated virtio IO context. This event
The following APIs operate a virtio-FS request:

1. `doca_devemu_vfs_req_get_datain` – Get a DOCA buffer representing the data-in of the virtio-FS request. This DOCA buffer represents the host memory for the device-readable part of the request according to the virtio specification.

2. `doca_devemu_vfs_req_get_dataout` – Get a DOCA buffer representing the data-out of the virtio-FS request. This DOCA buffer represents the host memory for the device-writable part of the request according to the virtio specification.

3. `doca_devemu_vfs_req_complete` – Complete the virtio-FS request. The associated virtio-FS IO context completes the request toward the device driver according to the virtio-FS specification.

### Discovery

Emulated virtio-FS PCIe functions are represented by a `doca_devinfo_rep`. To find the suitable `doca_devinfo_rep` that is used as the input parameter for `doca_devemu_vfs_dev_create`, users should first discover the existing device representors using the below:

1. `doca_devinfo_create_list` – Get a list of all DOCA devices.

2. `doca_devemu_vfs_is_default_vfs_type_supported` – Check whether the device can manage device associated to virtio-FS type.

3. If supported:
   1. `doca_dev_open` – Get an instance of the DOCA device that can be used as virtio-FS emulation manager.
   2. `doca_devemu_vfs_find_default_vfs_type_by_dev` – Get the default virtio-FS device type.
   3. `doca_devemu_vfs_type_as_pci_type` – Cast virtio-FS type to PCIe type.
4. `doca_devemu_pci_type_rep_list_create` – Create a list of all available representor devices for the virtio-FS type.

4. At this point, the user can choose the preferred representor device, open it using `doca_dev_rep_open`, and proceed with the flow described in section "Virtio-FS Device Configurations".

**Initialization**

This section describes the initialization flow of a DOCA DevEmu Virtio-FS device and one or more DOCA DevEmu Virtio-FS IO contexts (4 in this example). In this procedure, the user sets up and prepares the environment before starting to receive control path events (from the virtio-FS device context) and IO requests (from the virtio-FS IO contexts). During initialization, the user should configure various essential components to ensure correct behavior.

The user should perform the following:

1. Choose 4 Arm cores to run the application threads on.
2. Create 4 DOCA Core progress engine (PE) objects (`pe1`, `pe2`, `pe3`, `pe4`).
3. Find the suitable representor device according to the Discovery flow or any other method.
4. Create, configure, and start a new virtio-FS device according to the virtio-FS device configuration flow. Assume `pe1` is associated with the virtio-FS device and `doca_devemu_virtio_dev_set_num_required_running_virtio_io_ctxs` is set to 4.
5. Create, configure, and start 4 new virtio-FS IO contexts according to the virtio-FS IO configuration flow. Assume `pe1`, `pe2`, `pe3`, and `pe4` are associated with each of the 4 virtio-FS IO contexts respectively.
6. At this point, the 4 virtio-FS IO contexts transition to running state, followed by the virtio-FS device context transitioning to running state.

**Note**

During the initialization flow, it is guaranteed that no virtio/PCIe control path or IO path events are generated until the virtio-FS device
Teardown

This section describes the teardown flow of DOCA DevEmu Virtio-FS device and one or more DOCA DevEmu Virtio-FS IO contexts (4 in this example). In this procedure, the user cleans all the resources allocated in the initialization flow and all the outstanding events and requests.

The user should perform the following:

1. Start the teardown flow by calling `doca_ctx_stop`. This causes the DOCA Virtio-FS device context to transition to stopping state. It is guaranteed that no virtio/PCIe control path events is generated during this state.

2. Call `doca_ctx_stop` for any DOCA Virtio-FS IO context. This causes the DOCA Virtio-FS IO context to transition to stopping state. It is guaranteed that no IO path events are generated during this state.

3. Flush all outstanding virtio-FS requests to the associated virtio-FS IO contexts by calling `doca_devemu_vfs_req_complete`. Upon completing all the requests associated with a virtio-FS IO context, the DOCA Virtio-FS IO context transitions to idle state.

4. At this point, it is safe to destroy the virtio-FS IO context by calling `doca_devemu_vfs_io_destroy`. Destroying a virtio-FS IO context not in idle state will fail.

5. Once all 4 virtio-FS IO contexts associated with the virtio-FS device transition to idle state, the DOCA Virtio-FS device context transitions to idle state as well.

6. At this point, it is safe to destroy the virtio-FS device context by calling `doca_devemu_vfs_dev_destroy`. Destroying a virtio-FS device context not in idle state will fail.

Execution Phase

This section describes execution on BlueField Arm cores using several DOCA Core PE objects (one per core):

1. Choose 4 Arm cores to run the application threads on.
2. Create 4 DOCA Core PE objects. The application threads should periodically call `doca_pe_progress` to advance all DOCA contexts associated with the PE.

3. Create, configure, and start the DOCA Virtio-FS device.

4. Create, configure, and start 4 DOCA Virtio-FS IO contexts.

The progress of DOCA Virtio-FS objects is illustrated by the following diagram:

### Control Path

The DOCA Virtio-FS device context extends the DOCA Virtio device context (which extends the DOCA PCIe device context). This means that the DOCA Virtio-FS device control path is comprised by all the object it extends (i.e., DOCA Context, DOCA DevEmu PCI device, and DOCA DevEmu Virtio device).

The following events can be triggered by a virtio-FS device context:

- DOCA context state change events as described in DOCA Core context state machine and in DOCA DevEmu PCI state machine
- DOCA DevEmu PCI FLR flow
- DOCA DevEmu Virtio reset flow
The DOCA Virtio-FS IO context extends the DOCA Virtio IO context (which extends the DOCA core context). This means that the DOCA Virtio-FS IO context control path is comprised by all the object it extends (i.e., DOCA Context and DOCA DevEmu Virtio IO).

The following events can be triggered by a Virtio-FS IO context:

- DOCA context state change events as described in DOCA Core context state machine

In addition to the control path events, the DOCA DevEmu Virtio-FS IO context also produces IO path events as described in IO path.

**IO Path**

This section describes the flow for a single virtio-FS request sent by the device driver until its completion.

It is assumed that the user properly configured an event handler for an incoming virtio-FS request as explained in section "Virtio-FS IO Configurations".

It is also assumed that the user is familiar with the virtio-FS specification and has the ability to perform DMA operations to/from the host using DOCA DMA or any other suitable method.

The DOCA virtio-FS flow is illustrated in the following diagram: