vfio-ccw: the basic infrastructure

Introduction

Here we describe the vfio support for I/O subchannel devices for Linux/s390. Motivation for vfio-ccw is to passthrough subchannels to a virtual machine, while vfio is the means.

Different than other hardware architectures, s390 has defined a unified I/O access method, which is so called Channel I/O. It has its own access patterns:

- Channel programs run asynchronously on a separate (co)processor.
- The channel subsystem will access any memory designated by the caller in the channel program directly, i.e. there is no iommu involved.

Thus when we introduce vfio support for these devices, we realize it with a mediated device (mdev) implementation. The vfio mdev will be added to an iommu group, so as to make itself able to be managed by the vfio framework. And we add read/write callbacks for special vfio I/O regions to pass the channel programs from the mdev to its parent device (the real I/O subchannel device) to do further address translation and to perform I/O instructions.

This document does not intend to explain the s390 I/O architecture in every detail. More information/reference could be found here:

- A good start to know Channel I/O in general: https://en.wikipedia.org/wiki/Channel_I/O
- s390 architecture: s390 Principles of Operation manual (IBM Form. No. SA22-7832)
- The existing QEMU code which implements a simple emulated channel subsystem could also be a good reference. It makes it easier to follow the flow. qemu/hw/s390x/css.c

For vfio mediated device framework: - Documentation/driver-api/vfio-mediated-device.rst

Motivation of vfio-ccw

Typically, a guest virtualized via QEMU/KVM on s390 only sees paravirtualized virtio devices via the "Virtio Over Channel I/O (virtio-ccw)" transport. This makes virtio devices discoverable via standard operating system algorithms for handling channel devices.

However this is not enough. On s390 for the majority of devices, which use the standard Channel I/O based mechanism, we also need to provide the functionality of passing through them to a QEMU virtual machine. This includes devices that don't have a virtio counterpart (e.g. tape drives) or that have specific characteristics which guests want to exploit.

For passing a device to a guest, we want to use the same interface as everybody else, namely vfio. We implement this vfio support for channel devices via the vfio mediated device framework and the subchannel device driver "vfio_ccw".

Access patterns of CCW devices

s390 architecture has implemented a so called channel subsystem, that provides a unified view of the devices physically attached to the systems. Though the s390 hardware platform knows about a huge variety of different peripheral attachments like disk devices (aka. DASDs), tapes, communication controllers, etc. They can all be accessed by a well defined access method and they are presenting I/O completion a unified way: I/O interruptions.

All I/O requires the use of channel command words (CCWs). A CCW is an instruction to a specialized I/O channel processor. A channel program is a sequence of CCWs which are executed by the I/O channel subsystem. To issue a channel program to the channel subsystem, it is required to build an operation request block (ORB), which can be used to point out the format of the CCW and other control information to the system. The operating system signals the I/O channel subsystem to begin executing the channel program with a SSCH (start sub-channel) instruction. The central processor is then free to proceed with non-I/O instructions until interrupted. The I/O completion result is received by the interrupt handler in the form of interrupt response block (IRB).

Back to vfio-ccw, in short:

- ORBs and channel programs are built in guest kernel (with guest physical addresses).
- ORBs and channel programs are passed to the host kernel.
- Host kernel translates the guest physical addresses to real addresses and starts the I/O with issuing a privileged Channel I/O instruction (e.g SSCH).
- channel programs run asynchronously on a separate processor.
- I/O completion will be signaled to the host with I/O interruptions. And it will be copied as IRB to user space to pass it back to the guest.

Physical vfio ccw device and its child mdev

As mentioned above, we realize vfio-ccw with a mdev implementation.

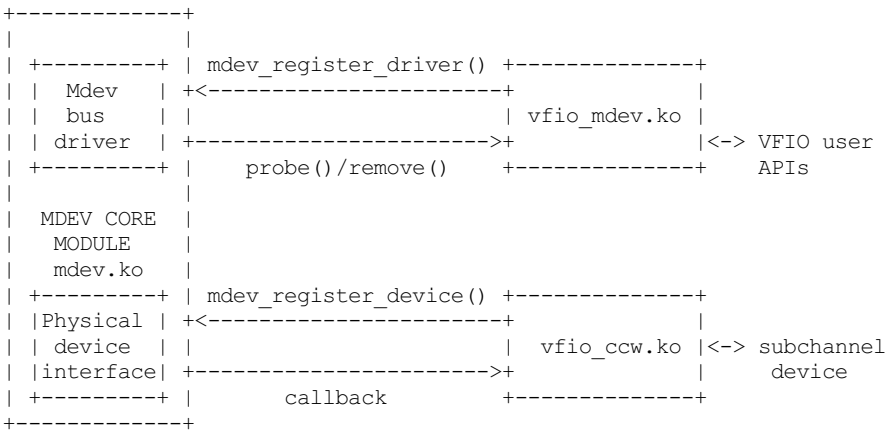
Channel I/O does not have IOMMU hardware support, so the physical vfio-ccw device does not have an IOMMU level translation or isolation.

Subchannel I/O instructions are all privileged instructions. When handling the I/O instruction interception, vfio-ccw has the software policing and translation how the channel program is programmed before it gets sent to hardware.

Within this implementation, we have two drivers for two types of devices:

- The vfio_ccw driver for the physical subchannel device. This is an I/O subchannel driver for the real subchannel device. It realizes a group of callbacks and registers to the mdev framework as a parent (physical) device. As a consequence, mdev provides vfio_ccw a generic interface (sysfs) to create mdev devices. A vfio mdev could be created by vfio_ccw then and added to the mediated bus. It is the vfio device that added to an IOMMU group and a vfio group. vfio_ccw also provides an I/O region to accept channel program request from user space and store I/O interrupt result for user space to retrieve. To notify user space an I/O completion, it offers an interface to setup an eventfd fd for asynchronous signaling.
- The vfio_mdev driver for the mediated vfio ccw device. This is provided by the mdev framework. It is a vfio device driver for the mdev that created by vfio_ccw. It realizes a group of vfio device driver callbacks, adds itself to a vfio group, and registers itself to the mdev framework as a mdev driver. It uses a vfio iommu backend that uses the existing map and unmap ioctls, but rather than programming them into an IOMMU for a device, it simply stores the translations for use by later requests. This means that a device programmed in a VM with guest physical addresses can have the vfio kernel convert that address to process virtual address, pin the page and program the hardware with the host physical address in one step. For a mdev, the vfio iommu backend will not pin the pages during the VFIO_IOMMU_MAP_DMA ioctl. Mdev framework will only maintain a database of the iova<->vaddr mappings in this operation. And they export a vfio_pin_pages and a vfio_unpin_pages interfaces from the vfio iommu backend for the physical devices to pin and unpin pages by demand.

Below is a high Level block diagram:



The process of how these work together.

1. vfio_ccw.ko drives the physical I/O subchannel, and registers the physical device (with callbacks) to mdev framework. When vfio_ccw probing the subchannel device, it registers device pointer and callbacks to the mdev framework. Mdev related file nodes under the device node in sysfs would be created for the subchannel device, namely 'mdev_create', 'mdev_destroy' and 'mdev_supported_types'.
2. Create a mediated vfio ccw device. Use the 'mdev_create' sysfs file, we need to manually create one (and only one for our case) mediated device.
3. vfio_mdev.ko drives the mediated ccw device. vfio_mdev is also the vfio device drvier. It will probe the mdev and add it to an iommu_group and a vfio_group. Then we could pass through the mdev to a guest.

VFIO-CCW Regions

The vfio-ccw driver exposes MMIO regions to accept requests from and return results to userspace.

vfio-ccw I/O region

An I/O region is used to accept channel program request from user space and store I/O interrupt result for user space to retrieve. The definition of the region is:

```
struct ccw_io_region {
#define ORB_AREA_SIZE 12
        __u8    orb_area[ORB_AREA_SIZE];
#define SCSW_AREA_SIZE 12
        __u8    scsw_area[SCSW_AREA_SIZE];
#define IRB_AREA_SIZE 96
        __u8    irb_area[IRB_AREA_SIZE];
        __u32   ret_code;
} __packed;
```

This region is always available.

While starting an I/O request, orb_area should be filled with the guest ORB, and scsw_area should be filled with the SCSW of the

Virtual Subchannel.

irb_area stores the I/O result.

ret_code stores a return code for each access of the region. The following values may occur:

`0`
: The operation was successful.

`-EOPNOTSUPP`
: The orb specified transport mode or an unidentified IDAW format, or the scsw specified a function other than the start function.

`-EIO`
: A request was issued while the device was not in a state ready to accept requests, or an internal error occurred.

`-EBUSY`
: The subchannel was status pending or busy, or a request is already active.

`-EAGAIN`
: A request was being processed, and the caller should retry.

`-EACCES`
: The channel path(s) used for the I/O were found to be not operational.

`-ENODEV`
: The device was found to be not operational.

`-EINVAL`
: The orb specified a chain longer than 255 ccws, or an internal error occurred.

vfio-ccw cmd region

The vfio-ccw cmd region is used to accept asynchronous instructions from userspace:

```
#define VFIO_CCW_ASYNC_CMD_HSCH (1 << 0)
#define VFIO_CCW_ASYNC_CMD_CSCH (1 << 1)
struct ccw_cmd_region {
        __u32 command;
        __u32 ret_code;
} __packed;
```

This region is exposed via region type VFIO_REGION_SUBTYPE_CCW_ASYNC_CMD.

Currently, CLEAR SUBCHANNEL and HALT SUBCHANNEL use this region.

command specifies the command to be issued; ret_code stores a return code for each access of the region. The following values may occur:

`0`
: The operation was successful.

`-ENODEV`
: The device was found to be not operational.

`-EINVAL`
: A command other than halt or clear was specified.

`-EIO`
: A request was issued while the device was not in a state ready to accept requests.

`-EAGAIN`
: A request was being processed, and the caller should retry.

`-EBUSY`
: The subchannel was status pending or busy while processing a halt request.

vfio-ccw schib region

The vfio-ccw schib region is used to return Subchannel-Information Block (SCHIB) data to userspace:

```
struct ccw_schib_region {
#define SCHIB_AREA_SIZE 52
        __u8 schib_area[SCHIB_AREA_SIZE];
} __packed;
```

This region is exposed via region type VFIO_REGION_SUBTYPE_CCW_SCHIB.

Reading this region triggers a STORE SUBCHANNEL to be issued to the associated hardware.

vfio-ccw crw region

The vfio-ccw crw region is used to return Channel Report Word (CRW) data to userspace:

```
struct ccw_crw_region {
```

```
        __u32 crw;
        __u32 pad;
} __packed;
```

This region is exposed via region type VFIO_REGION_SUBTYPE_CCW_CRW.

Reading this region returns a CRW if one that is relevant for this subchannel (e.g. one reporting changes in channel path state) is pending, or all zeroes if not. If multiple CRWs are pending (including possibly chained CRWs), reading this region again will return the next one, until no more CRWs are pending and zeroes are returned. This is similar to how STORE CHANNEL REPORT WORD works.

vfio-ccw operation details

vfio-ccw follows what vfio-pci did on the s390 platform and uses vfio-iommu-type1 as the vfio iommu backend.

- CCW translation APIs A group of APIs (start with *cp_*) to do CCW translation. The CCWs passed in by a user space program are organized with their guest physical memory addresses. These APIs will copy the CCWs into kernel space, and assemble a runnable kernel channel program by updating the guest physical addresses with their corresponding host physical addresses. Note that we have to use IDALs even for direct-access CCWs, as the referenced memory can be located anywhere, including above 2G.
- vfio_ccw device driver This driver utilizes the CCW translation APIs and introduces vfio_ccw, which is the driver for the I/O subchannel devices you want to pass through. vfio_ccw implements the following vfio ioctls:

  ```
  VFIO_DEVICE_GET_INFO
  VFIO_DEVICE_GET_IRQ_INFO
  VFIO_DEVICE_GET_REGION_INFO
  VFIO_DEVICE_RESET
  VFIO_DEVICE_SET_IRQS
  ```

  This provides an I/O region, so that the user space program can pass a channel program to the kernel, to do further CCW translation before issuing them to a real device. This also provides the SET_IRQ ioctl to setup an event notifier to notify the user space program the I/O completion in an asynchronous way.

The use of vfio-ccw is not limited to QEMU, while QEMU is definitely a good example to get understand how these patches work. Here is a little bit more detail how an I/O request triggered by the QEMU guest will be handled (without error handling).

Explanation:

- Q1-Q7: QEMU side process.
- K1-K5: Kernel side process.

Q1.
: Get I/O region info during initialization.

Q2.
: Setup event notifier and handler to handle I/O completion.

... ...

Q3.
: Intercept a ssch instruction.

Q4.
: Write the guest channel program and ORB to the I/O region.

  K1.
  : Copy from guest to kernel.

  K2.
  : Translate the guest channel program to a host kernel space channel program, which becomes runnable for a real device.

  K3.
  : With the necessary information contained in the orb passed in by QEMU, issue the ccwchain to the device.

  K4.
  : Return the ssch CC code.

Q5.
: Return the CC code to the guest.

... ...

K5.
: Interrupt handler gets the I/O result and write the result to the I/O region.

K6.
: Signal QEMU to retrieve the result.

Q6.
: Get the signal and event handler reads out the result from the I/O region.

Q7.
: Update the irb for the guest.

Limitations

The current vfio-ccw implementation focuses on supporting basic commands needed to implement block device functionality (read/write) of DASD/ECKD device only. Some commands may need special handling in the future, for example, anything related to path grouping.

DASD is a kind of storage device. While ECKD is a data recording format. More information for DASD and ECKD could be found here: https://en.wikipedia.org/wiki/Direct-access_storage_device https://en.wikipedia.org/wiki/Count_key_data

Together with the corresponding work in QEMU, we can bring the passed through DASD/ECKD device online in a guest now and use it as a block device.

The current code allows the guest to start channel programs via START SUBCHANNEL, and to issue HALT SUBCHANNEL, CLEAR SUBCHANNEL, and STORE SUBCHANNEL.

Currently all channel programs are prefetched, regardless of the p-bit setting in the ORB. As a result, self modifying channel programs are not supported. For this reason, IPL has to be handled as a special case by a userspace/guest program; this has been implemented in QEMU's s390-ccw bios as of QEMU 4.1.

vfio-ccw supports classic (command mode) channel I/O only. Transport mode (HPF) is not supported.

QDIO subchannels are currently not supported. Classic devices other than DASD/ECKD might work, but have not been tested.

Reference

1. ESA/s390 Principles of Operation manual (IBM Form. No. SA22-7832)
2. ESA/390 Common I/O Device Commands manual (IBM Form. No. SA22-7204)
3. https://en.wikipedia.org/wiki/Channel_I/O
4. Documentation/s390/cds.rst
5. Documentation/driver-api/vfio.rst
6. Documentation/driver-api/vfio-mediated-device.rst