Adjunct Processor (AP) facility

Introduction

The Adjunct Processor (AP) facility is an IBM Z cryptographic facility comprised of three AP instructions and from 1 up to 256 PCIe cryptographic adapter cards. The AP devices provide cryptographic functions to all CPUs assigned to a linux system running in an IBM Z system LPAR.

The AP adapter cards are exposed via the AP bus. The motivation for vfio-ap is to make AP cards available to KVM guests using the VFIO mediated device framework. This implementation relies considerably on the s390 virtualization facilities which do most of the hard work of providing direct access to AP devices.

AP Architectural Overview

To facilitate the comprehension of the design, let's start with some definitions:

- AP adapter

  An AP adapter is an IBM Z adapter card that can perform cryptographic functions. There can be from 0 to 256 adapters assigned to an LPAR. Adapters assigned to the LPAR in which a linux host is running will be available to the linux host. Each adapter is identified by a number from 0 to 255; however, the maximum adapter number is determined by machine model and/or adapter type. When installed, an AP adapter is accessed by AP instructions executed by any CPU.

  The AP adapter cards are assigned to a given LPAR via the system's Activation Profile which can be edited via the HMC. When the linux host system is IPL'd in the LPAR, the AP bus detects the AP adapter cards assigned to the LPAR and creates a sysfs device for each assigned adapter. For example, if AP adapters 4 and 10 (0x0a) are assigned to the LPAR, the AP bus will create the following sysfs device entries:

  ```
  /sys/devices/ap/card04
  /sys/devices/ap/card0a
  ```

  Symbolic links to these devices will also be created in the AP bus devices sub-directory:

  ```
  /sys/bus/ap/devices/[card04]
  /sys/bus/ap/devices/[card04]
  ```

- AP domain

  An adapter is partitioned into domains. An adapter can hold up to 256 domains depending upon the adapter type and hardware configuration. A domain is identified by a number from 0 to 255; however, the maximum domain number is determined by machine model and/or adapter type.. A domain can be thought of as a set of hardware registers and memory used for processing AP commands. A domain can be configured with a secure private key used for clear key encryption. A domain is classified in one of two ways depending upon how it may be accessed:

    - Usage domains are domains that are targeted by an AP instruction to process an AP command.
    - Control domains are domains that are changed by an AP command sent to a usage domain; for example, to set the secure private key for the control domain.

  The AP usage and control domains are assigned to a given LPAR via the system's Activation Profile which can be edited via the HMC. When a linux host system is IPL'd in the LPAR, the AP bus module detects the AP usage and control domains assigned to the LPAR. The domain number of each usage domain and adapter number of each AP adapter are combined to create AP queue devices (see AP Queue section below). The domain number of each control domain will be represented in a bitmask and stored in a sysfs file /sys/bus/ap/ap_control_domain_mask. The bits in the mask, from most to least significant bit, correspond to domains 0-255.

- AP Queue

  An AP queue is the means by which an AP command is sent to a usage domain inside a specific adapter. An AP queue is identified by a tuple comprised of an AP adapter ID (APID) and an AP queue index (APQI). The APQI corresponds to a given usage domain number within the adapter. This tuple forms an AP Queue Number (APQN) uniquely identifying an AP queue. AP instructions include a field containing the APQN to identify the AP queue to which the AP command is to be sent for processing.

  The AP bus will create a sysfs device for each APQN that can be derived from the cross product of the AP adapter and usage domain numbers detected when the AP bus module is loaded. For example, if adapters 4 and 10 (0x0a) and usage domains 6 and 71 (0x47) are assigned to the LPAR, the AP bus will create the following sysfs entries:

  ```
  /sys/devices/ap/card04/04.0006
  /sys/devices/ap/card04/04.0047
  /sys/devices/ap/card0a/0a.0006
  /sys/devices/ap/card0a/0a.0047
  ```

The following symbolic links to these devices will be created in the AP bus devices subdirectory:

```
/sys/bus/ap/devices/[04.0006]
/sys/bus/ap/devices/[04.0047]
/sys/bus/ap/devices/[0a.0006]
/sys/bus/ap/devices/[0a.0047]
```

- AP Instructions:

  There are three AP instructions:

  - NQAP: to enqueue an AP command-request message to a queue
  - DQAP: to dequeue an AP command-reply message from a queue
  - PQAP: to administer the queues

  AP instructions identify the domain that is targeted to process the AP command; this must be one of the usage domains. An AP command may modify a domain that is not one of the usage domains, but the modified domain must be one of the control domains.

AP and SIE

Let's now take a look at how AP instructions executed on a guest are interpreted by the hardware.

A satellite control block called the Crypto Control Block (CRYCB) is attached to our main hardware virtualization control block. The CRYCB contains three fields to identify the adapters, usage domains and control domains assigned to the KVM guest:

- The AP Mask (APM) field is a bit mask that identifies the AP adapters assigned to the KVM guest. Each bit in the mask, from left to right (i.e. from most significant to least significant bit in big endian order), corresponds to an APID from 0-255. If a bit is set, the corresponding adapter is valid for use by the KVM guest.
- The AP Queue Mask (AQM) field is a bit mask identifying the AP usage domains assigned to the KVM guest. Each bit in the mask, from left to right (i.e. from most significant to least significant bit in big endian order), corresponds to an AP queue index (APQI) from 0-255. If a bit is set, the corresponding queue is valid for use by the KVM guest.
- The AP Domain Mask field is a bit mask that identifies the AP control domains assigned to the KVM guest. The ADM bit mask controls which domains can be changed by an AP command-request message sent to a usage domain from the guest. Each bit in the mask, from left to right (i.e. from most significant to least significant bit in big endian order), corresponds to a domain from 0-255. If a bit is set, the corresponding domain can be modified by an AP command-request message sent to a usage domain.

If you recall from the description of an AP Queue, AP instructions include an APQN to identify the AP queue to which an AP command-request message is to be sent (NQAP and PQAP instructions), or from which a command-reply message is to be received (DQAP instruction). The validity of an APQN is defined by the matrix calculated from the APM and AQM; it is the cross product of all assigned adapter numbers (APM) with all assigned queue indexes (AQM). For example, if adapters 1 and 2 and usage domains 5 and 6 are assigned to a guest, the APQNs (1,5), (1,6), (2,5) and (2,6) will be valid for the guest.

The APQNs can provide secure key functionality - i.e., a private key is stored on the adapter card for each of its domains - so each APQN must be assigned to at most one guest or to the linux host:

```
Example 1: Valid configuration:
-------------------------------
Guest1: adapters 1,2  domains 5,6
Guest2: adapter  1,2  domain 7

This is valid because both guests have a unique set of APQNs:
   Guest1 has APQNs (1,5), (1,6), (2,5), (2,6);
   Guest2 has APQNs (1,7), (2,7)

Example 2: Valid configuration:
-------------------------------
Guest1: adapters 1,2 domains 5,6
Guest2: adapters 3,4 domains 5,6

This is also valid because both guests have a unique set of APQNs:
   Guest1 has APQNs (1,5), (1,6), (2,5), (2,6);
   Guest2 has APQNs (3,5), (3,6), (4,5), (4,6)

Example 3: Invalid configuration:
---------------------------------
Guest1: adapters 1,2  domains 5,6
Guest2: adapter  1    domains 6,7

This is an invalid configuration because both guests have access to
APQN (1,6).
```

The Design

The design introduces three new objects:

1. AP matrix device
2. VFIO AP device driver (vfio_ap.ko)
3. VFIO AP mediated matrix pass-through device

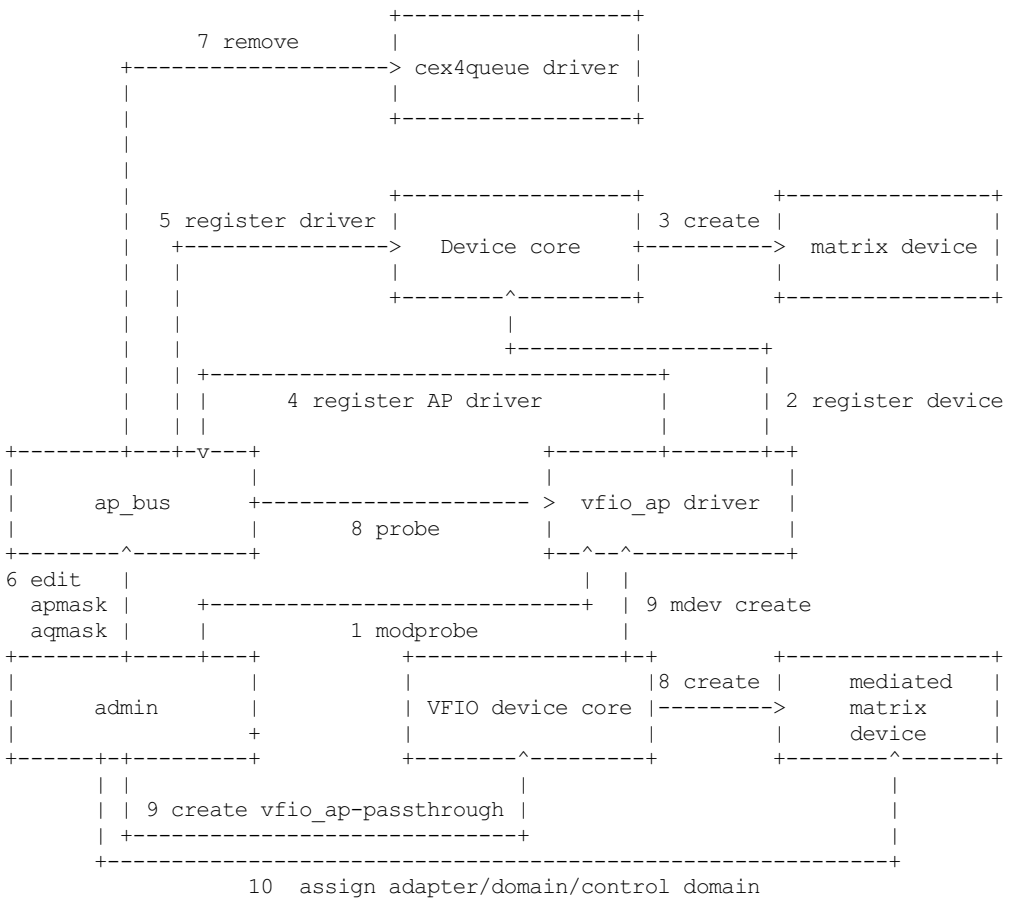
The VFIO AP device driver

The VFIO AP (vfio_ap) device driver serves the following purposes:

1. Provides the interfaces to secure APQNs for exclusive use of KVM guests.
2. Sets up the VFIO mediated device interfaces to manage a mediated matrix device and creates the sysfs interfaces for assigning adapters, usage domains, and control domains comprising the matrix for a KVM guest.
3. Configures the APM, AQM and ADM in the CRYCB referenced by a KVM guest's SIE state description to grant the guest access to a matrix of AP devices

Reserve APQNs for exclusive use of KVM guests

The following block diagram illustrates the mechanism by which APQNs are reserved:



The process for reserving an AP queue for use by a KVM guest is:

1. The administrator loads the vfio_ap device driver
2. The vfio-ap driver during its initialization will register a single 'matrix' device with the device core. This will serve as the parent device for all mediated matrix devices used to configure an AP matrix for a guest.
3. The /sys/devices/vfio_ap/matrix device is created by the device core
4. The vfio_ap device driver will register with the AP bus for AP queue devices of type 10 and higher (CEX4 and newer). The driver will provide the vfio_ap driver's probe and remove callback interfaces. Devices older than CEX4 queues are not supported to simplify the implementation by not needlessly complicating the design by supporting older devices that will go out of service in the relatively near future, and for which there are few older systems around on which to test.
5. The AP bus registers the vfio_ap device driver with the device core
6. The administrator edits the AP adapter and queue masks to reserve AP queues for use by the vfio_ap device driver.
7. The AP bus removes the AP queues reserved for the vfio_ap driver from the default zcrypt cex4queue driver.
8. The AP bus probes the vfio_ap device driver to bind the queues reserved for it.
9. The administrator creates a passthrough type mediated matrix device to be used by a guest
10. The administrator assigns the adapters, usage domains and control domains to be exclusively used by a guest.

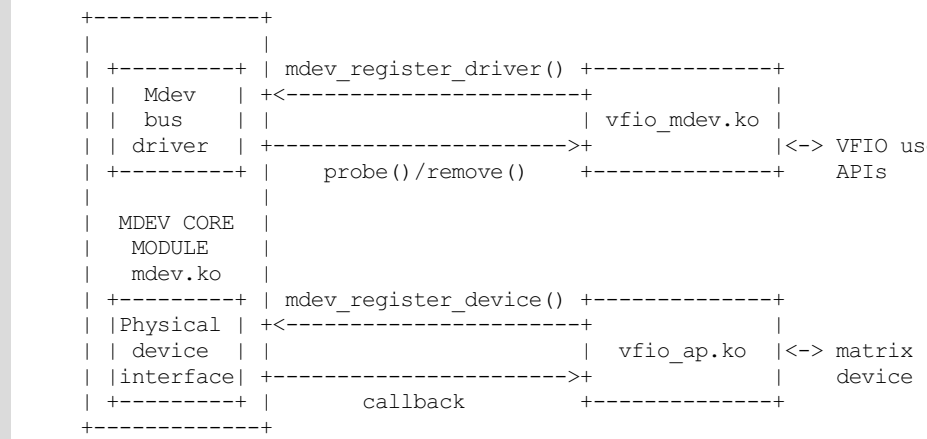
Set up the VFIO mediated device interfaces

The VFIO AP device driver utilizes the common interface of the VFIO mediated device core driver to:

- Register an AP mediated bus driver to add a mediated matrix device to and remove it from a VFIO group.

- Create and destroy a mediated matrix device
- Add a mediated matrix device to and remove it from the AP mediated bus driver
- Add a mediated matrix device to and remove it from an IOMMU group

The following high-level block diagram shows the main components and interfaces of the VFIO AP mediated matrix device driver:



During initialization of the vfio_ap module, the matrix device is registered with an 'mdev_parent_ops' structure that provides the sysfs attribute structures, mdev functions and callback interfaces for managing the mediated matrix device.

- sysfs attribute structures:

  supported_type_groups

  The VFIO mediated device framework supports creation of user-defined mediated device types. These mediated device types are specified via the 'supported_type_groups' structure when a device is registered with the mediated device framework. The registration process creates the sysfs structures for each mediated device type specified in the 'mdev_supported_types' sub-directory of the device being registered. Along with the device type, the sysfs attributes of the mediated device type are provided.

  The VFIO AP device driver will register one mediated device type for passthrough devices:

  /sys/devices/vfio_ap/matrix/mdev_supported_types/vfio_ap-passthrough

  Only the read-only attributes required by the VFIO mdev framework will be provided:

  ```
  ... name
  ... device_api
  ... available_instances
  ... device_api
  ```

  Where:

  - name:
    specifies the name of the mediated device type
  - device_api:
    the mediated device type's API
  - available_instances:
    the number of mediated matrix passthrough devices that can be created
  - device_api:
    specifies the VFIO API

  mdev_attr_groups

  This attribute group identifies the user-defined sysfs attributes of the mediated device. When a device is registered with the VFIO mediated device framework, the sysfs attribute files identified in the 'mdev_attr_groups' structure will be created in the mediated matrix device's directory. The sysfs attributes for a mediated matrix device are:

  assign_adapter / unassign_adapter:
  Write-only attributes for assigning/unassigning an AP adapter to/from the mediated matrix device. To assign/unassign an adapter, the APID of the adapter is echoed to the respective attribute file.

  assign_domain / unassign_domain:
  Write-only attributes for assigning/unassigning an AP usage domain to/from the mediated matrix device. To assign/unassign a domain, the domain number of the usage domain is echoed to the respective attribute file.

  matrix:
  A read-only file for displaying the APQNs derived from the cross product of the adapter and domain numbers assigned to the mediated matrix device.

assign_control_domain / unassign_control_domain:
: Write-only attributes for assigning/unassigning an AP control domain to/from the mediated matrix device. To assign/unassign a control domain, the ID of the domain to be assigned/unassigned is echoed to the respective attribute file.

control_domains:
: A read-only file for displaying the control domain numbers assigned to the mediated matrix device.

- functions:

  create:
  : allocates the ap_matrix_mdev structure used by the vfio_ap driver to:
    - Store the reference to the KVM structure for the guest using the mdev
    - Store the AP matrix configuration for the adapters, domains, and control domains assigned via the corresponding sysfs attributes files

  remove:
  : deallocates the mediated matrix device's ap_matrix_mdev structure. This will be allowed only if a running guest is not using the mdev.

- callback interfaces

  open:
  : The vfio_ap driver uses this callback to register a VFIO_GROUP_NOTIFY_SET_KVM notifier callback function for the mdev matrix device. The open is invoked when QEMU connects the VFIO iommu group for the mdev matrix device to the MDEV bus. Access to the KVM structure used to configure the KVM guest is provided via this callback. The KVM structure, is used to configure the guest's access to the AP matrix defined via the mediated matrix device's sysfs attribute files.

  release:
  : unregisters the VFIO_GROUP_NOTIFY_SET_KVM notifier callback function for the mdev matrix device and deconfigures the guest's AP matrix.

Configure the APM, AQM and ADM in the CRYCB

Configuring the AP matrix for a KVM guest will be performed when the VFIO_GROUP_NOTIFY_SET_KVM notifier callback is invoked. The notifier function is called when QEMU connects to KVM. The guest's AP matrix is configured via it's CRYCB by:

- Setting the bits in the APM corresponding to the APIDs assigned to the mediated matrix device via its 'assign_adapter' interface.
- Setting the bits in the AQM corresponding to the domains assigned to the mediated matrix device via its 'assign_domain' interface.
- Setting the bits in the ADM corresponding to the domain dIDs assigned to the mediated matrix device via its 'assign_control_domains' interface.

The CPU model features for AP

The AP stack relies on the presence of the AP instructions as well as two facilities: The AP Facilities Test (APFT) facility; and the AP Query Configuration Information (QCI) facility. These features/facilities are made available to a KVM guest via the following CPU model features:

1. ap: Indicates whether the AP instructions are installed on the guest. This feature will be enabled by KVM only if the AP instructions are installed on the host.
2. apft: Indicates the APFT facility is available on the guest. This facility can be made available to the guest only if it is available on the host (i.e., facility bit 15 is set).
3. apqci: Indicates the AP QCI facility is available on the guest. This facility can be made available to the guest only if it is available on the host (i.e., facility bit 12 is set).

Note: If the user chooses to specify a CPU model different than the 'host' model to QEMU, the CPU model features and facilities need to be turned on explicitly; for example:

```
/usr/bin/qemu-system-s390x ... -cpu z13,ap=on,apqci=on,apft=on
```

A guest can be precluded from using AP features/facilities by turning them off explicitly; for example:

```
/usr/bin/qemu-system-s390x ... -cpu host,ap=off,apqci=off,apft=off
```

Note: If the APFT facility is turned off (apft=off) for the guest, the guest will not see any AP devices. The zcrypt device drivers that register for type 10 and newer AP devices - i.e., the cex4card and cex4queue device drivers - need the APFT facility to ascertain the facilities installed on a given AP device. If the APFT facility is not installed on the guest, then the probe of device drivers will fail since only type 10 and newer devices can be configured for guest use.

Example

Let's now provide an example to illustrate how KVM guests may be given access to AP facilities. For this example, we will show how to configure three guests such that executing the lszcrypt command on the guests would look like this:

Guest1

CARD.DOMAIN	TYPE	MODE
05	CEX5C	CCA-Coproc
05.0004	CEX5C	CCA-Coproc
05.00ab	CEX5C	CCA-Coproc
06	CEX5A	Accelerator
06.0004	CEX5A	Accelerator
06.00ab	CEX5C	CCA-Coproc

Guest2

CARD.DOMAIN	TYPE	MODE
05	CEX5A	Accelerator
05.0047	CEX5A	Accelerator
05.00ff	CEX5A	Accelerator

Guest3

CARD.DOMAIN	TYPE	MODE
06	CEX5A	Accelerator
06.0047	CEX5A	Accelerator
06.00ff	CEX5A	Accelerator

These are the steps:

1. Install the vfio_ap module on the linux host. The dependency chain for the vfio_ap module is: * iommu * s390 * zcrypt * vfio * vfio_mdev * vfio_mdev_device * KVM

   To build the vfio_ap module, the kernel build must be configured with the following Kconfig elements selected: * IOMMU_SUPPORT * S390 * ZCRYPT * S390_AP_IOMMU * VFIO * VFIO_MDEV * KVM

   If using make menuconfig select the following to build the vfio_ap module:

   ```
   -> Device Drivers
      -> IOMMU Hardware Support
         select S390 AP IOMMU Support
      -> VFIO Non-Privileged userspace driver framework
         -> Mediated device driver frramework
            -> VFIO driver for Mediated devices
   -> I/O subsystem
      -> VFIO support for AP devices
   ```

2. Secure the AP queues to be used by the three guests so that the host can not access them. To secure them, there are two sysfs files that specify bitmasks marking a subset of the APQN range as 'usable by the default AP queue device drivers' or 'not usable by the default device drivers' and thus available for use by the vfio_ap device driver'. The location of the sysfs files containing the masks are:

   ```
   /sys/bus/ap/apmask
   /sys/bus/ap/aqmask
   ```

   The 'apmask' is a 256-bit mask that identifies a set of AP adapter IDs (APID). Each bit in the mask, from left to right (i.e., from most significant to least significant bit in big endian order), corresponds to an APID from 0-255. If a bit is set, the APID is marked as usable only by the default AP queue device drivers; otherwise, the APID is usable by the vfio_ap device driver.

   The 'aqmask' is a 256-bit mask that identifies a set of AP queue indexes (APQI). Each bit in the mask, from left to right (i.e., from most significant to least significant bit in big endian order), corresponds to an APQI from 0-255. If a bit is set, the APQI is marked as usable only by the default AP queue device drivers; otherwise, the APQI is usable by the vfio_ap device driver.

   Take, for example, the following mask:

   ```
     0x7dffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff

   It indicates:

     1, 2, 3, 4, 5, and 7-255 belong to the default drivers' pool, and 0 and 6
     belong to the vfio_ap device driver's pool.
   ```

   The APQN of each AP queue device assigned to the linux host is checked by the AP bus against the set of APQNs derived

from the cross product of APIDs and APQIs marked as usable only by the default AP queue device drivers. If a match is detected, only the default AP queue device drivers will be probed; otherwise, the vfio_ap device driver will be probed.

By default, the two masks are set to reserve all APQNs for use by the default AP queue device drivers. There are two ways the default masks can be changed:

1. The sysfs mask files can be edited by echoing a string into the respective sysfs mask file in one of two formats:
   - An absolute hex string starting with 0x - like "0x12345678" - sets the mask. If the given string is shorter than the mask, it is padded with 0s on the right; for example, specifying a mask value of 0x41 is the same as specifying:

     ```
     0x4100000000000000000000000000000000000000000000000000000000000000
     ```

     Keep in mind that the mask reads from left to right (i.e., most significant to least significant bit in big endian order), so the mask above identifies device numbers 1 and 7 (01000001).

     If the string is longer than the mask, the operation is terminated with an error (EINVAL).
   - Individual bits in the mask can be switched on and off by specifying each bit number to be switched in a comma separated list. Each bit number string must be prepended with a ('+') or minus ('-') to indicate the corresponding bit is to be switched on ('+') or off ('-'). Some valid values are:
     - "+0" switches bit 0 on
     - "-13" switches bit 13 off
     - "+0x41" switches bit 65 on
     - "-0xff" switches bit 255 off

     The following example:

     +0,-6,+0x47,-0xf0

     Switches bits 0 and 71 (0x47) on

     Switches bits 6 and 240 (0xf0) off

     Note that the bits not specified in the list remain as they were before the operation.
2. The masks can also be changed at boot time via parameters on the kernel command line like this:

   ap.apmask=0xffff ap.aqmask=0x40

   This would create the following masks:

   ```
   apmask:
   0xffff000000000000000000000000000000000000000000000000000000000000

   aqmask:
   0x4000000000000000000000000000000000000000000000000000000000000000
   ```

   Resulting in these two pools:

   ```
   default drivers pool:    adapter 0-15, domain 1
   alternate drivers pool:  adapter 16-255, domains 0, 2-255
   ```

Securing the APQNs for our example

To secure the AP queues 05.0004, 05.0047, 05.00ab, 05.00ff, 06.0004, 06.0047, 06.00ab, and 06.00ff for use by the vfio_ap device driver, the corresponding APQNs can either be removed from the default masks:

```
echo -5,-6 > /sys/bus/ap/apmask

echo -4,-0x47,-0xab,-0xff > /sys/bus/ap/aqmask
```

Or the masks can be set as follows:

```
echo 0xf9fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff \
> apmask

echo 0xf7ffffffffffffffefffffffffffffffffffffefffffffffffffffffffffffe \
> aqmask
```

This will result in AP queues 05.0004, 05.0047, 05.00ab, 05.00ff, 06.0004, 06.0047, 06.00ab, and 06.00ff getting bound to the vfio_ap device driver. The sysfs directory for the vfio_ap device driver will now contain symbolic links to the AP queue devices bound to it:

```
/sys/bus/ap
... [drivers]
...... [vfio_ap]
```

```
......... [05.0004]
......... [05.0047]
......... [05.00ab]
......... [05.00ff]
......... [06.0004]
......... [06.0047]
......... [06.00ab]
......... [06.00ff]
```

Keep in mind that only type 10 and newer adapters (i.e., CEX4 and later) can be bound to the vfio_ap device driver. The reason for this is to simplify the implementation by not needlessly complicating the design by supporting older devices that will go out of service in the relatively near future and for which there are few older systems on which to test.

The administrator, therefore, must take care to secure only AP queues that can be bound to the vfio_ap device driver. The device type for a given AP queue device can be read from the parent card's sysfs directory. For example, to see the hardware type of the queue 05.0004:

```
cat /sys/bus/ap/devices/card05/hwtype
```

The hwtype must be 10 or higher (CEX4 or newer) in order to be bound to the vfio_ap device driver.

3. Create the mediated devices needed to configure the AP matrixes for the three guests and to provide an interface to the vfio_ap driver for use by the guests:

```
/sys/devices/vfio_ap/matrix/
--- [mdev_supported_types]
------ [vfio_ap-passthrough] (passthrough mediated matrix device type)
--------- create
--------- [devices]
```

To create the mediated devices for the three guests:

```
uuidgen > create
uuidgen > create
uuidgen > create

or

echo $uuid1 > create
echo $uuid2 > create
echo $uuid3 > create
```

This will create three mediated devices in the [devices] subdirectory named after the UUID written to the create attribute file. We call them $uuid1, $uuid2 and $uuid3 and this is the sysfs directory structure after creation:

```
/sys/devices/vfio_ap/matrix/
--- [mdev_supported_types]
------ [vfio_ap-passthrough]
--------- [devices]
------------ [$uuid1]
---------------- assign_adapter
---------------- assign_control_domain
---------------- assign_domain
---------------- matrix
---------------- unassign_adapter
---------------- unassign_control_domain
---------------- unassign_domain

------------ [$uuid2]
---------------- assign_adapter
---------------- assign_control_domain
---------------- assign_domain
---------------- matrix
---------------- unassign_adapter
-----------------unassign_control_domain
-----------------unassign_domain

------------ [$uuid3]
---------------- assign_adapter
---------------- assign_control_domain
---------------- assign_domain
---------------- matrix
---------------- unassign_adapter
-----------------unassign_control_domain
-----------------unassign_domain
```

4. The administrator now needs to configure the matrixes for the mediated devices $uuid1 (for Guest1), $uuid2 (for Guest2) and $uuid3 (for Guest3).

This is how the matrix is configured for Guest1:

```
echo 5 > assign_adapter
echo 6 > assign_adapter
echo 4 > assign_domain
echo 0xab > assign_domain
```

Control domains can similarly be assigned using the assign_control_domain sysfs file.

If a mistake is made configuring an adapter, domain or control domain, you can use the unassign_xxx files to unassign the adapter, domain or control domain.

To display the matrix configuration for Guest1:

```
cat matrix
```

This is how the matrix is configured for Guest2:

```
echo 5 > assign_adapter
echo 0x47 > assign_domain
echo 0xff > assign_domain
```

This is how the matrix is configured for Guest3:

```
echo 6 > assign_adapter
echo 0x47 > assign_domain
echo 0xff > assign_domain
```

In order to successfully assign an adapter:

- The adapter number specified must represent a value from 0 up to the maximum adapter number configured for the system. If an adapter number higher than the maximum is specified, the operation will terminate with an error (ENODEV).
- All APQNs that can be derived from the adapter ID and the IDs of the previously assigned domains must be bound to the vfio_ap device driver. If no domains have yet been assigned, then there must be at least one APQN with the specified APID bound to the vfio_ap driver. If no such APQNs are bound to the driver, the operation will terminate with an error (EADDRNOTAVAIL).

  No APQN that can be derived from the adapter ID and the IDs of the previously assigned domains can be assigned to another mediated matrix device. If an APQN is assigned to another mediated matrix device, the operation will terminate with an error (EADDRINUSE).

In order to successfully assign a domain:

- The domain number specified must represent a value from 0 up to the maximum domain number configured for the system. If a domain number higher than the maximum is specified, the operation will terminate with an error (ENODEV).
- All APQNs that can be derived from the domain ID and the IDs of the previously assigned adapters must be bound to the vfio_ap device driver. If no domains have yet been assigned, then there must be at least one APQN with the specified APQI bound to the vfio_ap driver. If no such APQNs are bound to the driver, the operation will terminate with an error (EADDRNOTAVAIL).

  No APQN that can be derived from the domain ID and the IDs of the previously assigned adapters can be assigned to another mediated matrix device. If an APQN is assigned to another mediated matrix device, the operation will terminate with an error (EADDRINUSE).

In order to successfully assign a control domain, the domain number specified must represent a value from 0 up to the maximum domain number configured for the system. If a control domain number higher than the maximum is specified, the operation will terminate with an error (ENODEV).

5. Start Guest1:

```
/usr/bin/qemu-system-s390x ... -cpu host,ap=on,apqci=on,apft=on \
   -device vfio-ap,sysfsdev=/sys/devices/vfio_ap/matrix/$uuid1 ...
```

7. Start Guest2:

```
/usr/bin/qemu-system-s390x ... -cpu host,ap=on,apqci=on,apft=on \
   -device vfio-ap,sysfsdev=/sys/devices/vfio_ap/matrix/$uuid2 ...
```

7. Start Guest3:

```
/usr/bin/qemu-system-s390x ... -cpu host,ap=on,apqci=on,apft=on \
   -device vfio-ap,sysfsdev=/sys/devices/vfio_ap/matrix/$uuid3 ...
```

When the guest is shut down, the mediated matrix devices may be removed.

Using our example again, to remove the mediated matrix device $uuid1:

```
/sys/devices/vfio_ap/matrix/
   --- [mdev_supported_types]
```

```
      ------ [vfio_ap-passthrough]
      --------- [devices]
      ------------ [$uuid1]
      --------------- remove

   echo 1 > remove
```

This will remove all of the mdev matrix device's sysfs structures including the mdev device itself. To recreate and reconfigure the mdev matrix device, all of the steps starting with step 3 will have to be performed again. Note that the remove will fail if a guest using the mdev is still running.

It is not necessary to remove an mdev matrix device, but one may want to remove it if no guest will use it during the remaining lifetime of the linux host. If the mdev matrix device is removed, one may want to also reconfigure the pool of adapters and queues reserved for use by the default drivers.

Limitations

- The KVM/kernel interfaces do not provide a way to prevent restoring an APQN to the default drivers pool of a queue that is still assigned to a mediated device in use by a guest. It is incumbent upon the administrator to ensure there is no mediated device in use by a guest to which the APQN is assigned lest the host be given access to the private data of the AP queue device such as a private key configured specifically for the guest.
- Dynamically modifying the AP matrix for a running guest (which would amount to hot(un)plug of AP devices for the guest) is currently not supported
- Live guest migration is not supported for guests using AP devices.