IOMMU Userspace API

IOMMU UAPI is used for virtualization cases where communications are needed between physical and virtual IOMMU drivers. For baremetal usage, the IOMMU is a system device which does not need to communicate with userspace directly.

The primary use cases are guest Shared Virtual Address (SVA) and guest IO virtual address (IOVA), wherein the vIOMMU implementation relies on the physical IOMMU and for this reason requires interactions with the host driver.

Functionalities

Communications of user and kernel involve both directions. The supported user-kernel APIs are as follows:

1. Bind/Unbind guest PASID (e.g. Intel VT-d)
2. Bind/Unbind guest PASID table (e.g. ARM SMMU)
3. Invalidate IOMMU caches upon guest requests
4. Report errors to the guest and serve page requests

Requirements

The IOMMU UAPIs are generic and extensible to meet the following requirements:

1. Emulated and para-virtualised vIOMMUs
2. Multiple vendors (Intel VT-d, ARM SMMU, etc.)
3. Extensions to the UAPI shall not break existing userspace

Interfaces

Although the data structures defined in IOMMU UAPI are self-contained, there are no user API functions introduced. Instead, IOMMU UAPI is designed to work with existing user driver frameworks such as VFIO.

Extension Rules & Precautions

When IOMMU UAPI gets extended, the data structures can *only* be modified in two ways:

1. Adding new fields by re-purposing the padding[] field. No size change.
2. Adding new union members at the end. May increase the structure sizes.

No new fields can be added *after* the variable sized union in that it will break backward compatibility when offset moves. A new flag must be introduced whenever a change affects the structure using either method. The IOMMU driver processes the data based on flags which ensures backward compatibility.

Version field is only reserved for the unlikely event of UAPI upgrade at its entirety.

It's *always* the caller's responsibility to indicate the size of the structure passed by setting argsz appropriately. Though at the same time, argsz is user provided data which is not trusted. The argsz field allows the user app to indicate how much data it is providing; it's still the kernel's responsibility to validate whether it's correct and sufficient for the requested operation.

Compatibility Checking

When IOMMU UAPI extension results in some structure size increase, IOMMU UAPI code shall handle the following cases:

1. User and kernel has exact size match
2. An older user with older kernel header (smaller UAPI size) running on a newer kernel (larger UAPI size)
3. A newer user with newer kernel header (larger UAPI size) running on an older kernel.
4. A malicious/misbehaving user passing illegal/invalid size but within range. The data may contain garbage.

Feature Checking

While launching a guest with vIOMMU, it is strongly advised to check the compatibility upfront, as some subsequent errors happening during vIOMMU operation, such as cache invalidation failures cannot be nicely escalated to the guest due to IOMMU specifications. This can lead to catastrophic failures for the users.

User applications such as QEMU are expected to import kernel UAPI headers. Backward compatibility is supported per feature flags. For example, an older QEMU (with older kernel header) can run on newer kernel. Newer QEMU (with new kernel header) may refuse to initialize on an older kernel if new feature flags are not supported by older kernel. Simply recompiling existing code with newer kernel header should not be an issue in that only existing flags are used.

IOMMU vendor driver should report the below features to IOMMU UAPI consumers (e.g. via VFIO).

1. IOMMU_NESTING_FEAT_SYSWIDE_PASID
2. IOMMU_NESTING_FEAT_BIND_PGTBL
3. IOMMU_NESTING_FEAT_BIND_PASID_TABLE
4. IOMMU_NESTING_FEAT_CACHE_INVLD
5. IOMMU_NESTING_FEAT_PAGE_REQUEST

Take VFIO as example, upon request from VFIO userspace (e.g. QEMU), VFIO kernel code shall query IOMMU vendor driver for the support of the above features. Query result can then be reported back to the userspace caller. Details can be found in Documentation/driver-api/vfio.rst.

Data Passing Example with VFIO

As the ubiquitous userspace driver framework, VFIO is already IOMMU aware and shares many key concepts such as device model, group, and protection domain. Other user driver frameworks can also be extended to support IOMMU UAPI but it is outside the scope of this document.

In this tight-knit VFIO-IOMMU interface, the ultimate consumer of the IOMMU UAPI data is the host IOMMU driver. VFIO facilitates user-kernel transport, capability checking, security, and life cycle management of process address space ID (PASID).

VFIO layer conveys the data structures down to the IOMMU driver. It follows the pattern below:

```
struct {
      __u32 argsz;
      __u32 flags;
      __u8  data[];
};
```

Here data[] contains the IOMMU UAPI data structures. VFIO has the freedom to bundle the data as well as parse data size based on its own flags.

In order to determine the size and feature set of the user data, argsz and flags (or the equivalent) are also embedded in the IOMMU UAPI data structures.

A "__u32 argsz" field is *always* at the beginning of each structure.

For example:

```
struct iommu_cache_invalidate_info {
      __u32   argsz;
      #define IOMMU_CACHE_INVALIDATE_INFO_VERSION_1 1
      __u32   version;
      /* IOMMU paging structure cache */
      #define IOMMU_CACHE_INV_TYPE_IOTLB      (1 << 0) /* IOMMU IOTLB */
      #define IOMMU_CACHE_INV_TYPE_DEV_IOTLB  (1 << 1) /* Device IOTLB */
      #define IOMMU_CACHE_INV_TYPE_PASID      (1 << 2) /* PASID cache */
      #define IOMMU_CACHE_INV_TYPE_NR         (3)
      __u8    cache;
      __u8    granularity;
      __u8    padding[6];
      union {
              struct iommu_inv_pasid_info pasid_info;
              struct iommu_inv_addr_info addr_info;
      } granu;
};
```

VFIO is responsible for checking its own argsz and flags. It then invokes appropriate IOMMU UAPI functions. The user pointers are passed to the IOMMU layer for further processing. The responsibilities are divided as follows:

- Generic IOMMU layer checks argsz range based on UAPI data in the current kernel version.
- Generic IOMMU layer checks content of the UAPI data for non-zero reserved bits in flags, padding fields, and unsupported version. This is to ensure not breaking userspace in the future when these fields or flags are used.
- Vendor IOMMU driver checks argsz based on vendor flags. UAPI data is consumed based on flags. Vendor driver has access to unadulterated argsz value in case of vendor specific future extensions. Currently, it does not perform the copy_from_user() itself. A __user pointer can be provided in some future scenarios where there's vendor data outside of the structure definition.

IOMMU code treats UAPI data in two categories:

- structure contains vendor data (Example: iommu_uapi_cache_invalidate())

- structure contains only generic data (Example: iommu_uapi_sva_bind_gpasid())

Sharing UAPI with in-kernel users

For UAPIs that are shared with in-kernel users, a wrapper function is provided to distinguish the callers. For example,

Userspace caller

```
int iommu_uapi_sva_unbind_gpasid(struct iommu_domain *domain,
                                 struct device *dev,
                                 void __user *udata)
```

In-kernel caller

```
int iommu_sva_unbind_gpasid(struct iommu_domain *domain,
                            struct device *dev, ioasid_t ioasid);
```