HARMAN TECHNICAL NOTES

Device Virtualization for Connected Vehicles

Sharing Memory

Software Version 12.1

Doc. Rev. 1.0 / 556

Reference: DV-0031

Date: 3/24/2022

Table of Contents

Table of Figures

1 Introduction

1.1 This Document

This aims briefly describes the memory sharing mechanism available with the Hypervisor. It compares these mechanisms to help the reader understand which mechanism is relevant for his goal. Detailed description of each communication mechanism is available in the "Hypervisor Reference Manual" and "Virtual Driver Writer's Guide".

1.2 Related Documentation

The Device Virtualization for Connected Vehicles product includes the following documentation:

- Architecture:
  - Product Description
  - System Architecture and Overview
  - Hypervisor Description
- Configuration guides:
  - Hypervisor First Steps
  - Configuring Guest OS and Applications
- User guides:
  - Inter-VM Communication
  - How to Move a Pass-Through Device from System Domain to User Domain
  - Inter-VM Communication for Linux User Space Applications
  - Virtual Driver Writer's Guide
  - Virtual ION Memory Allocator
  - Console Multiplexer
  - Debugging with GDB
  - Hypervisor Awareness in Debugging Environment
  - Suspend to RAM technical report
  - Hypervisor Trace Visualization
  - CPU handover
- Reference manuals:
  - Device Virtualization Reference Manual (Public and/or BSP editions)
  - Virtual Device Driver Reference Manual
- Release Notes
-

1.3 Support

If you have a question or require further assistance, contact HARMAN's customer support at HCS-DL-RB-FR-Virtualization-Support@harman.com.

2 Memory sharing

The reader is required to know the basics of the configuration with device trees and hypervisor interface.

Sharing memory can be achieved:

- by defining a shared memory area in the configuration files (device trees)
- or, at run-time, by using page granting mechanism
- or finally by using communication shared memory area ("the so-called" pmem area). Note we won't describe this method in this document since it is largely covered in other documents (in particular the "Virtual Driver Writer's Guide").

2.1.1 Static Shared Memory Configuration

To share a static region of physical memory containing, you must:

1. Configure this area as "invalid memory", so that the memory is not provided to the Guest OS as "standard" memory.
   "Invalid memory" is a region of memory which should never be used by the the Hypervisor or Guest OS. "invalid memory" is typically a portion of RAM used by external component like DSPs or micro-controller.
2. Configure this memory area as an IO-region in the Guest OSes which need to share this memory region.

2.1.1.1 Configuring invalid memory

You need to edit the Hypervisor device tree and add an "invalid_region" device tree node. For instance,

```
&invalid_region {
reg =  <0 0xd0000000 0x04000000>,
          <0 0xd4000000 0x04000000>;
};
```

The above configuration declares two invalid regions of 64 mega bytes starting at 0xd000_0000 and 0xd400_0000.

Note that the label "&invalid_region" refers to a node inside the Hypervisor device tree which is declared in the "skeleton.dtsi" as follow:

```
/dts-v1/;

/{
    #address-cells = <2>;
    #size-cells = <1>;
    vlm: vlm {

        memory: memory {
            invalid_region: invalid-region {
            };
        };
        ….  /* ignoring the remaining device tree*/
    };
    … }; /* end of device tree */
```

2.1.1.2 Configuring IO region

The second step consists in declaring the corresponding IO-region in the "vplatform" device trees. Vplatform describes the intermediate memory address space (IPA) of a VM. Each VM has a such description. Let's assume we want to share the memory regions between VM2 and VM3, it is enough to add the following description in the "vplatform" device tree of VM2 and VM3.

```
 /dts-v1/;

/ {
    compatible = "vl,vlm", "vl,vlm-exynos8890-aarch64", "vl,vlm-aarch64";
    #address-cells = <2>;
    #size-cells = <1>;
    …
    // first region of shared memory
    vl,io-memory@d0000000-4000000 {
        compatible = "vl,io-memory";
        reg = <0x0 0xd0000000 0x4000000>;
        vl,allow-sharing;
        vl,attributes = <VDT_S2PTE_UPPER_MEM VDT_S2PTE_LOWER_DEV_MEM>;
    };
    …
    // second region of shared memory
    vl,io-memory@d4000000-4000000 {
        compatible = "vl,io-memory";
        reg = <0x0 0xd4000000 0x4000000>;
        vl,allow-sharing;
        vl,attributes = <VDT_S2PTE_UPPER_MEM VDT_S2PTE_LOWER_DEV_MEM>;
    };
    …
};
```

The above "vl,io-memory" nodes defines an identical mapping (the range [0xd4000000-0xd7FFFFFF] of the VM (IPA) will map to the range "[0xd4000000 0xd7FFFFFF]" of physical address space), it is also possible to setup nonidentical mapping as shown below.

Nonidentical mapping:

Assuming we want to map the first range of shared memory at "0x800000000" in the address space of one of the VM, we shall use the "ipa-base" property as follow:

```
  // first region of shared memory
    vl,io-memory@800000000-4000000 {
        compatible = "vl,io-memory";
        reg = <0x0 0xd0000000 0x4000000>;
        vl,attributes = <VDT_S2PTE_UPPER_MEM VDT_S2PTE_LOWER_DEV_MEM>;d
        ipa-base = <0x800000000>
        vl,allow-sharing;
    };
```

IO sharing:

By default, sharing IO memory range is prevented by the Hypervisor. To enable the memory sharing "vl,allow-sharing" must be added to the node.

Cache attribute:

“vl,attributes” property defines cache attribute of the IO mapping. In the above example, "<VDT_S2PTE_UPPER_MEM VDT_S2PTE_LOWER_MEM>" enables cached (coherent) access to this memory. The constants defining mapping attributes are declared in nkern/arm/nk/nkvdt.h.

Consideration for the Guest OS

From standpoint of Guest OS this memory might be considered as "IO memory" so the memory allocator might not be aware this is memory and in particular in the case of Linux, this area won't have any page descriptors, this might be an issue or not depending on the device driver.

"Reserved memory" versus "invalid memory":

From the standpoint of the Hypervisor, "Reserved memory" is assigned to a VM (in particular reserved memory described in device tree provided by the bootloader will be assigned to the VM2) and is considered as normal memory. Unlike "invalid" memory which is not assigned to any Virtual Machine.

2.1.2 Sharing memory with page granting

The Hypervisor provides ways for a VM to grant another VM access to some of its memory. Access can be granted for dynamically defined regions (page aligned areas of memory). The access may be granted for reading, writing, and for DMA access.

This may be used by an exporter VM to provide access to some of its memory to an importer VM to perform for instance a disk operation. Once the disk device operation is completed, the exporter VM might revoke (also called deny) the access to this memory.

Note that currently, page granting mechanism only works when both exporter and importer VMs use identical second stage MMU mapping, in other words, when the IPA is equal to the PA.

2.1.2.1 Page Granting Hypervisor Interface

"grant/revoke" operations uses the following Hypervisor interfaces :

```
int hyp_call_vm_mem_grant (NkOsId vmid, unsigned int flags, unsigned
int count, NkMemMap *rgns);
```

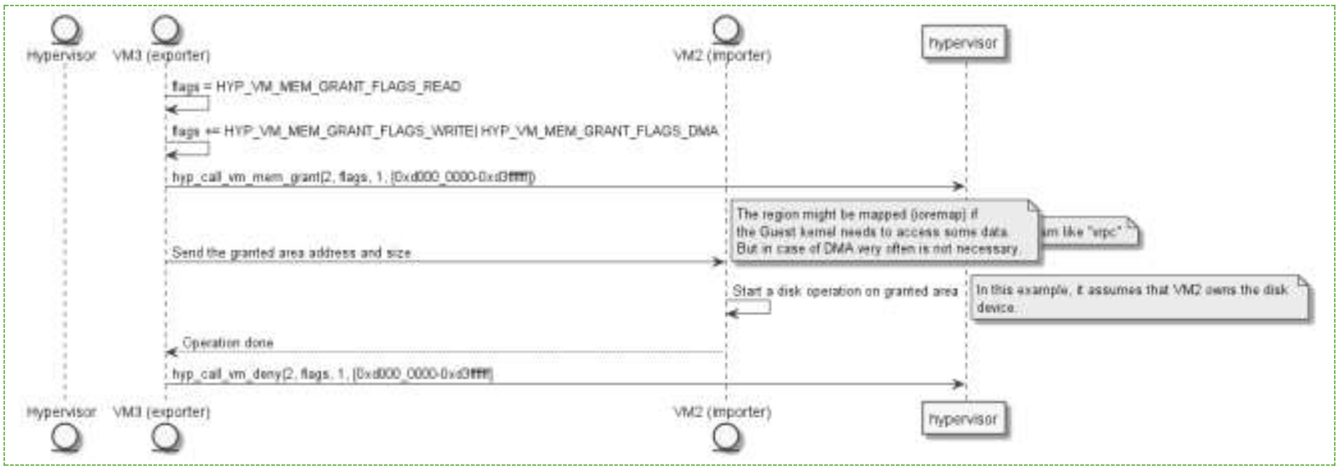
Grant access to the rgns VM memory regions from the destination VM vmid. This operation is initiated by the exporter VMs.

```
int hyp_call_vm_mem_deny (NkOsId vmid, unsigned int flags, unsigned
int count, NkMemMap *rgns);
```

Deny access to the rgns VM memory regions from the destination VM vmid.

The reader can refer to **"VM memory sharing"** chapter in the Hypervisor Reference Manual for further API description.

The below sequence diagram, depicts a typical usage of Page Granting interface


Hypervisor
VM3 (exporter)
VM2 (importer)
hypervisor
flags = HYP_VM_MEM_GRANT_FLAGS_READ
flags += HYP_VM_MEM_GRANT_FLAGS_WRITE| HYP_VM_MEM_GRANT_FLAGS_DMA
hyp_call_vm_mem_grant(2, flags, 1, [0xd000_0000-0xd3ffff])
The region might be mapped (ioremap) if the Guest kernel needs to access some data. But in case of DMA very often is not necessary.
am like "vrpc"
Send the granted area address and size
Start a disk operation on granted area
In this example, it assumes that VM2 owns the disk device.
Operation done
hyp_call_vm_deny(2, flags, 1, [0xd000_0000-0xd3ffff])
Hypervisor
VM3 (exporter)
VM2 (importer)
hypervisor

Using Granting API 1

Configuring Page Granting:

Before using page granting API, the importer VM must be configured to accept granted memory of an exporter VMs. This is achieved by adding a "vl,vm-memory" node both in the "Virtual Platform" device tree and the Guest OS device tree of the importer VM. Adding these nodes allow the Hypervisor to provision the page tables which will be needed to dynamically create mappings in the second stage MMU of the importer VMs.

Example of "vl,vm-memory" node in the "vplatform device tree" :

```
// VPlatform snippet of VM2 (importer VM)
/ {
       vm-memory@3 { // reserve VM3 imported memory
            compatible = "vl,vm-memory";
            vl,vm-id = <3>;
       };
};
```

"vl, vm-id" is a mandatory property which specifies the exporter VM ID (here VM3)

Here is the corresponding "vl,vm-memory" node for the "Guest OS Device Tree" :

```
// Guest OS Device Tree snippet of VM2 (importer VM)
/ {
   reserved-memory {
       #address-cells = <2>;
       #size-cells = <1>;
       ranges;
       vm-memory@3 { // reserve VM3 imported memory
            compatible = "vl,vm-memory";
            reg = <0 0 0>;
            vl,vm-id = <3>;
       };
    };
```

"reg" is populated by the Hypervisor during initialization and will contain VM3 memory description.

The reader can refer to **"Virtual Platform Imported VM Memory Node - vl,vm-memory"** chapter in the Hypervisor Reference Manual for further information regarding "vl,vm-memory" device tree node.

Behavior of Granted Page during Reboot:

When an importer VM reboots or restarts , it automatically unmaps all potentially granted memory . This ensures that the VM has the same memory space (IPA) layout on each VM restart.

When an exporter VM restarts, all granted memory is unconditionally revoked. In other words, all importer VMs will lose the access to the granted memory. Note that operation happens at VM shutdown time (before restart) after the device drivers of the importer Virtual Machine have set to "OFF" all the critical "Vlinks". This mechanism allows the importer to make sure that granted memory is not used anymore when Hypervisor revokes mappings.

2.1.3 Pros and Cons of both Approaches

The benefit of the first method Is the simplicity, but since the configuration is defined statically , any variation of configuration requires to modify several device trees, the second drawback is that the shared memory won't be considered as normal memory (in other words, no page descriptor will be associated to this memory area) in the Guest Operating System and this might prevent device drivers to use this shared memory area for DMA transfers.

The second method is more flexible, you can adapt at run-time the shared memory area to a particular use case. In addition, the granted memory will be considered as standard memory. In consequence, any device drivers of the importer VM will be able to use it. The main drawback is the complexity of the method: you need to handle properly restart of the Guest and you must create an additional communication channel, for transferring the address and size of the "granted" memory.

Terms and Abbreviations

Term	Description
VM	Virtual Machine
IPA	Intermediate Physical Address
DMA	Direct Memory Access