BMW IDCEvo :: Earlydata Vlink Driver

The HARMAN Hypervisor provides services to support inter-VM communication. These services are available through hypercalls part of the Hypervisor programming interface. These services are made available to Linux Guest OS drivers via the NKDDI library. The full set of API and services is described in the Virtual Device Driver Reference Manual.

Terminology

XIRQ: eXtended Interrupt ReQuests

This service enables a Guest OS running in a Virtual Machine to post a software interrupt to another Virtual Machine. This is commonly used to signal a driver within a Virtual Machine (B) that some data or event produced by a Virtual Machine (A) is available.

PMEM: Persistent memory

Persistent Memory may be shared between different Virtual Machines and mapped into their respective Guest OS address spaces. These PMEM chunks are therefore used to exchange data between front-end drivers and back-end driverswithout copying data between the Virtual Machines.

PDEV: Persistent device repository
PDEV services enable to allocate chunks of memory which are private to a Virtual Machine and which can be used to share information with the Hypervisor. These chunks of memory
are usually used to store device meta-information by Guest OSes.

NKDDI vLink

A vLink is a point-to-point communication channel between 2 end-points. XIRQ, PMEM and PDEV resources are usually associated to a vLink. In a typical usage, one end-point is associated to one Virtual Machine while the second end-point is associated to a different Virtual Machine. This allows to easily set up a communication channel between two Virtual Machines.

Each end-point gets its own local PDEV memory chunk to store the local attributes of the communication channel end-point from the Guest OS standpoint. A chunk of PMEM may also be associated to a vLink to enable transmission of data from one VM to the other. Finally XIRQ may be used to signal the other side that some data is ready to be processed.

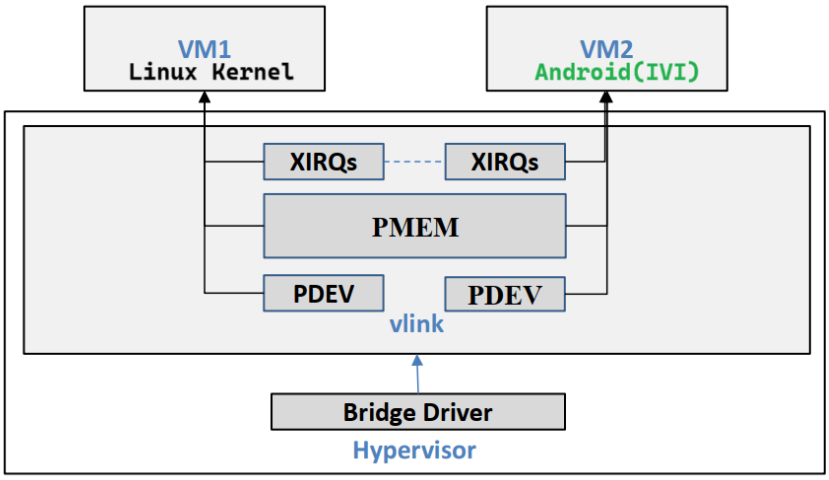
Therefore, vLinks build on the low-level services and are a nice and convenient way to set up interVM communication.

Basic Vlink driver with shared memory

• vlink end-points must be described in the Hypervisor device tree at system build time • Related end-points must use the same vlink “name”

• According to the configuration, each vlink end-point has a set of cross interrupts which can be sent to the VM owning that end-point, and a persistent memory region private to the VM

• Finally, a memory region that can be accessed by the two VMs is associated with the vlink and may be used to share data between these two VMs



Sample Driver

Hypervisor Configuration

```
exynosautov720-evt2/hyp/exynosautov720-evt2-idcevo-hyp-vdevs.dtsi
(Node0)
&vm2_vdevs {

+       vearlydata_be: vearlydata@be {
+               compatible = "vearlydata";
+               server;
+               info = "vearlydata_ctrl";
+              };


};

(IVI)
&vm3_vdevs {


+       vearlydata_fe {
+               peer-phandle = <&vearlydata_be>;
+               client;
+               info = "vearlydata_ctrl";
+       };
```

VM Driver

We can see a new API call, nkops.nk_vlink_lookup(), which allows to find the vlink endpoints. Each vlink end-point is a data structure of type NkDevVlink. It is located in a memory region called PDEV, for persistent device memory. A vlink is a device rather than just memory, because some fields are updated by the Hypervisor.

Passing address of one vlink to nkops.nk_vlink_lookup() returns the address of the next one, until there are no more. Passing address 0 gives the first vlink.4 NkPhAddr is an integer type large

Once the vlink is visible in virtual address space, the code examines two fields inside, vlink- >s_id and vlink->c_id and compares them with the current VM identifier.

The s_id and c_id fields are initialized by Hypervisor from the client and server properties of the vlink nodes in device tree, taking into account which VM the vlink definition is located in. Therefore, if VM 2 contains a vlink node marked server and VM 4 contains a vlink node marked client, then the Hypervisor will create a vlink descriptor where s_id is 2 and c_id is 4.

Both VMs will see the same values. However, each VM gets its own vlink end-point descriptor. The Hypervisor is in charge of propagating changes from one end-point to the other and validating them in the process. The element in charge of this process is called a bridge. Such a bridge is illustrated

```
191 static int earlydata_vlink_link_init(struct early_data *priv,
192                                       NkDevVlink *vlink, NkPhAddr plink,
193                                       void *xirq_handler)
194 {
195         struct device *dev = &priv->pdev->dev;
196         struct earlydata_vlink *earlydata_link;
197
198         earlydata_link = kzalloc(sizeof(*earlydata_link), GFP_KERNEL);
199         priv->vlink = earlydata_link;
200         if (!earlydata_link) {
201                 dev_err(dev, "earlydata_link ERROR");
202                 return -ENOMEM;
203         }
204         earlydata_link->vlink = vlink;
205         earlydata_link->plink = plink;
206         earlydata_link->my.id = nkops.nk_id_get();
207         earlydata_link->server = priv->hpt_be;
208
209         if (earlydata_vlink_is_client(earlydata_link))
210                 priv->vlink->id = 0;
211         else
212                 priv->vlink->id = 1;
213
214         dev_info(dev, "my.id: %d\n", earlydata_link->my.id);
```

Shared Memory

1. Allocate the physical memory for the necessary VM.
2. Map it into the virtual memory.

```
static bool earlydata_vlink_pmem_alloc(struct earlydata_vlink *link)
{
        unsigned int size = earlydata_vlink_pmem_size();
        NkPhAddr paddr = nkops.nk_pmem_alloc(link->plink, 0, size);

        if (!paddr) {
                pr_err("cannot alloc %d bytes of pmem\n", size);
                return 0;
        }
        pr_info("allocate link->pmem:%p\n", (void *)paddr);
        link->pmem = (struct earlydata_vlink_pmem *)nkops.nk_mem_map(paddr, size);
        return 1;
}
```