VM-to-VM Communication Mechanisms for Embedded

Stefano Stabellini
Existing VM-to-VM Communication Mechanisms

- **Existing protocols: Xen PV Drivers, VirtIO**
  - discoverable and dynamic
  - create new connections at runtime
  - made for IO virtualization

- A frontend in the guest connects to the backend in Dom0 / hypervisor

- Created to virtualize devices

- Typically based on memory sharing

- VirtIO expects privileged backends
Static Partitioning
Static Partitioning

- Static Partitioning is similar to virtualization with some key differences:
  - No dynamic VMs, a limited number of "partitions" instead
  - Focus on direct assignment of hardware resources
  - Configuration defined at "Build Time"
  - Real-Time, Safety, and Short Boot Times are often key requirements

- Example: Xen Dom0less
Xen Dom0less

U-Boot
loads into memory

Xen
loads into memory

Dom0

DomU 1

DomU 2

CPU

CPU

CPU
Xen Dom0less

U-Boot

boots

Xen

boots

Dom0

DomU 1

DomU 2

CPU

CPU

CPU
Xen Dom0less Current Status

- Static Partitions defined at build time
- Fast boot times, real-time support, easier to safety-certify
- Dom0 is not required
- No "out of the box" communication mechanisms available
Static Partitioning and Communication

- Often only VM-to-VM communication is required, not device virtualization
  - There are enough physical devices to directly assign them to VMs as needed
  - Device virtualization can be interesting for sharing an SD card among multiple VMs

- VM-to-VM communication is different from device virtualization
  - A simple VM-to-VM channel to send and receive raw data
  - It doesn't need "frontends" and "backends"
  - It requires a smaller code base
  - It is faster for exchanging data but it is unwieldy for virtualizing devices

- Static definition of VM-to-VM communication channels
  - Define connections at "build time"
  - Required for safety

- No privileged backends
  - Required for safety

- Support Linux and non-Linux guests (Zephyr, FreeRTOS, WindRiver, QNX, etc.)
Static Partitioning VM-to-VM communication

> No privileged backends
Xen PV Drivers

> Solid and hardened in production for years (AWS)

> Made for device virtualization, can be used for communication:
  >> Network
  >> Block (disks)
  >> Console
  >> 2D graphics, mouse and keyboard
  >> Sound
  >> Etc.

> Pros:
  >> Very Fast Unprivileged Backends
  >> Available for Linux, BSDs and Windows, less common among RTOSes

> Cons:
  >> Might not be available in certain embedded RTOSes (but BSD versions exist for all PV drivers)
  >> Dom0less support is work-in-progress
Xen PV Drivers and Dom0less

- Domains booted in parallel
- PV Drivers connections created after Dom0 is up and running
- Advantages compared to regular non-Dom0less deployments:
  - Domains are still started very quickly
  - Domains can immediately begin to perform critical tasks
  - Overall time to get PV Drivers up and running is shorter (no domain creation needed in Dom0)
- To become available by the end of the year (work by Hipert/Lab @ Unimore)
Frontend Drivers are available in most Operating Systems
"VMM" provides the backends (e.g. QEMU, kvmtools, etc.)

Pros:
- Many virtual device classes

Cons:
- Backends are currently required to be privileged – backends must be in Dom0
  - Security implications
  - Safety implications
- Support for Xen is available, but it requires non-upstream toolstack patches
- No Dom0less support
VirtIO

1. trap
2. ioreq
3. map and emulate
VirtIO

> IOREQ infrastructure upstream in Xen
  >> It enables VirtIO backends and any other emulators to run in Dom0 (e.g. QEMU)
  >> No support in the Xen tools for creating VirtIO frontends/backends yet (patch available)
  >> PoC with virtio-block by EPAM
  >> Requires Backends with full privileges, they have to be in Dom0
  >> Good performance with full privileges

> Support for Unprivileged Backend is work-in-progress by Linaro Project Stratos
  >> Based on memory copies to/from a pre-shared memory region
  >> Performance to be determined (never done before, underlying protocols designed for sharing)

> How to enable VirtIO for Dom0less?
  >> Could VirtIO device hotplug be used to avoid synchronous waiting during boot?
Argo

- Hypervisor-Mediated Data Transfers

**HMX: pattern for data delivery**

- **VM: Sender**
  - Message
  - Data
  - Hypervisor invoked to send message

- **VM: Receiver**
  - Receive memory buffer
  - Context
  - Data

**Hypervisor**

Delivery performed by the hypervisor:
- data delivered with context (size, origin)
- writes to the receive buffer, will conform to protocol / structure
Argo

Pros:

- Great performance and very strong security properties
  - Hypervisor checks against malicious data senders
  - Designed and optimized for memory copies
- More lightweight than Xen PV Drivers and VirtIO
  - No Xenstore, no PV backends, no VMM needed
  - Requires Event Channels and Argo drivers (BSD drivers available [here](#) and [here](#))
- Straightforward Dom0less enablement: no need for any kind of "wait"
  - No need to wait for Dom0 to complete booting to communicate with other VMs

Current Status:

- It requires one Linux patch to work with Dom0less
  - Thanks Alec Kwapis from DornerWorks!

Cons:

- Requires Argo driver and Xen event channels for notifications
Static Shared Memory and Interrupts

- U-Boot
- Xen
- Dom0
- DomU 1
- DomU 2
- CPU

ring buffer
Static Shared Memory and Interrupts

- **Plain shared memory region**
  - Configured at "build time"
  - Guests setups ring buffers over shared memory
  - Can use OpenAMP RPMesg or any other communication libraries based on shared memory

- **Interrupt-based notifications, work with any OSes**
  - New hypercall to inject SGIs (patch by Xilinx)

- **Pros:**
  - Very simple
  - Works with any OS
  - Great performance if used correctly

- **Cons:**
  - One non-upstream patch to enable interrupt notifications
  - Require your own communication library
  - No dynamic connections
PL-based communication mechanisms

DomU 1

Network Device

Switch

Network Device

DomU 2
PL-based communication mechanisms

- DomU 1
- DomU 2
- Data Movers
PL-based communication mechanisms

> Create Data Movers in Programmable Logic
  >> From simple Network Devices to optimized Data Movers

> Assign PL resources to VMs

> VMs use PL to send and receive data to/from other VMs

> Pros:
  >> Fastest for larger data sizes
  >> Userspace drivers only
  >> Easy to enable in any OS

> Cons:
  >> Requires PL
<table>
<thead>
<tr>
<th>Solution</th>
<th>Upstream Status for regular Xen</th>
<th>Upstream Status for Dom0less</th>
<th>VM-to-VM Communication vs. Device Virtualization</th>
<th>Compatibility</th>
<th>Performance</th>
<th>Unprivileged Backends</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plain shared memory &amp; interrupts</td>
<td>Patch available for interrupts</td>
<td>Patch available for interrupts</td>
<td>Static VM-to-VM Communication</td>
<td>Can run anywhere</td>
<td>High if implemented correctly</td>
<td>Yes</td>
</tr>
<tr>
<td>Argo</td>
<td>Upstream</td>
<td>One patch for Linux available</td>
<td>Dynamic VM-to-VM Communication</td>
<td>Linux, Windows with a small effort</td>
<td>High</td>
<td>Yes</td>
</tr>
<tr>
<td>Xen PV Drivers</td>
<td>Upstream</td>
<td>Patches available soon</td>
<td>Unprivileged Device Virtualization</td>
<td>Most traditional OSes (Linux, Windows, BSDs)</td>
<td>High</td>
<td>Yes</td>
</tr>
<tr>
<td>VirtIO</td>
<td>Hypervisor: upstream</td>
<td>No</td>
<td>Privileged Device Virtualization</td>
<td>Most traditional OSes (Linux, Windows, BSDs)</td>
<td>High with full privileged Otherwise: ?</td>
<td>No (work in progress)</td>
</tr>
</tbody>
</table>
Conclusions

> Several solutions are already available, but nothing works out of the box yet

> No one-size fits all:
  - Shared memory and notifications: best for OS compatibility
  - Argo: best for VM-to-VM communication
  - Xen PV Drivers: best for virtual devices with unprivileged backends
  - VirtIO: best for virtual device classes available
Demo

By Luca Miccio and Marco Solieri
Demo: Dom0less + PV Drivers

U-Boot

boots

Xen

boots

Dom0

DomU 1

CPU

CPU
Demo: Dom0less + PV Drivers

U-Boot

Xen

NetBack

xl

NetFront

CPU

CPU
Adaptable.

Intelligent.