IPC for the Partitioning Hypervisor Jailhouse
IPC for the Partitioning Hypervisor Jailhouse

Agenda

Jailhouse introduction & philosophy

- IPC requirements and status quo
- Inter-partition networking prototype
- Demonstrations
- Summary
What is Jailhouse?

A tool to run
... real-time and/or safety tasks
... on multicore platforms (AMP)
... aside Linux

It provides
• strong & clean isolation
• bare-metal-like performance & latencies
• no reason to modify Linux (well, almost)

... and it's open source (GPLv2)
What makes Jailhouse different?

- Use virtualization for isolation – *ok, nothing new*
- Prefer simplicity over features
  - Resource access control *instead of resource virtualization*
  - 1:1 resource assignment *instead of scheduling*
  - Partition booted system *instead of booting Linux*
  - Do not hide existence of Jailhouse
- **Offload work to Linux**
  - System boot
  - Jailhouse and partition (“cell”) loading & starting
  - Control and monitoring
Asymmetric Multiprocessing with Jailhouse

- **Root Cell**
- **Non-root Cell**

**Linux**

- **Jailhouse Hypervisor**

<table>
<thead>
<tr>
<th>Core 1</th>
<th>Core 2</th>
<th>Core 3</th>
<th>Core 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device A</td>
<td>Device B</td>
<td>Device C</td>
<td>Device D</td>
</tr>
</tbody>
</table>

Hardware

RTOS / Bare-Metal

Stahlkocher, CC BY-SA 3.0
Hard Partitioning of Linux

Linux #1

Linux #2

Linux #3

Linux #n

Jailhouse Hypervisor

Core 1 | Core 2 | Core 3 | Core 4 | Core 5 | Core 6 | Core 62 | Core 63 | Core 64

Hardware
IPC for the Partitioning Hypervisor Jailhouse

Agenda

- Jailhouse introduction & philosophy
- IPC requirements and status quo
  - Inter-partition networking prototype
  - Demonstrations
- Summary
Requirements on Inter-Partition Communication in Jailhouse

- **Local peer-to-peer channels**
  - Hardware independent, portable
  - Focus on two cells, multicast not (yet) in scope

- **Minimal work for hypervisor**
  - Static setups, no dynamic reconfigurations
  - Agnostic to communication protocols

- **Untrusted peers**
  - One side safety-related, the other not
  - Both sides safety-related, but validating each other
  - Secure isolation: one side hides secret from the other

- **Performance matters, but does not rule**
  - Try hard to be fast, low overhead
  - But when in conflict, strict isolation and simplicity win
Adapting ivshmem

- **Inter-VM shared memory (ivshmem) device in QEMU**
  - Designed for shared memory based communication
  - Works between applications inside VMs
  - Multiple peers
  - Doorbell interrupt
  - Supports VMs on different hosts → live migration
  - Modeled as PCI device with 3 resources: shared memory, control registers, MSI registers

- **Jailhouse variant**
  - Shared r/w memory region of two cells at most
  - Local only, no migration
  - Only MSI-based doorbell (currently)
  - Shared memory not relocatable via BAR
Alternative Inter-VM Communication Mechanisms

- **Classic virtio-based**
  - Well established in QEMU/KVM context and beyond
  - Allows networks, consoles and even more
  - Focused on hypervisor ↔ guest setups
    (hypervisor can freely access guest)
  - Requires copying between VMs

- **vhost-user**
  - Replaces hypervisor with separate user-space process
  - Does not resolve the access requirements

- **vhost-pci**
  - Proposal for NFV scenarios, aims at highest performance
  - Builds on top of virtual IOMMUs to reduce copies
  - Very complex because of IOMMU emulation
Alternative Inter-VM Communication Mechanisms (2)

- **Xen grant table**
  - Simpler than vhost-pci
  - Requires runtime remappings for safe / secure operation

- **remoteproc / rpmsg**
  - virtio-derived, focusing AMP scenarios
  - Assumes that co-processor can access whole host-processor address space
  - Currently too asymmetric
  - Pattern to copy: reuse virtio queues!

- ...
IPC for the Partitioning Hypervisor Jailhouse

Agenda

Jailhouse introduction & philosophy
IPC requirements and status quo
Inter-partition networking prototype
Demonstrations
Summary
Enabling PCI-free Targets for ivshmem

• How to bring ivshmem to ARM?
  • Often, there is no physical PCI host bridge, thus no place to inject ivshmem virtual devices
  • ivshmem as platform device?
  • Or rather add a virtual PCI host bridge?

```
core: pci: Add virtual host controller
...

hypervisor/include/jailhouse/cell-config.h | 2 +- 
hypervisor/pci.c | 8 ++++----
2 files changed, 5 insertions(+), 5 deletions(-)
```
Enabling PCI-free Targets for ivshmem (2)

• **How plug in the virtual PCI host bridge?**
  - Not yet available during boot-up, only after Jailhouse is enabled → cannot be part of device tree
  - Linux does not expect such bridges to be hot-plugged

• **What about those device tree overlays?**
  - Invented to address reconfigurable hardware like FPGAs, extension boards (for the Pi or BeagleBone)
  - Support already in upstream – well, almost...

• **What is missing?**
  - No overlay-aware DTC in upstream → use Pantelis Antoniou's DTC branch
  - No easy way of injecting DTB overlay blobs → use Pantelis' configfs patch
ivshmem-net – Networking over shared memory

- **Build upon ivshmem support in Jailhouse**
  - Register on PCI device
  - Use shared memory as transport
- **Reuse virtio**
  - Not the device layer
  - ...but the queues: mature data structures and access protocols
- **Early prototype by Måns Rullgård is working**
  - `linux/drivers/net/ivshmem-net.c`: 470 line of code
  - `iperf3` throughput room ↔ non-root cell: 2.2 Gbit/s
  - Round-trip latency: ~20 µs
  - To optimize: too many interrupts under high load
  - Life-cycle management lacking: what if one side restarts?
  - Zero-copy RX?
### Managing virtio Queue Access More Strictly

#### Virtqueue View

- **receiveq**
  - Desc Table (RX)
  - Avail (RX)
  - Used (RX)
  - Queue Data (RX)

- **transmitq**
  - Desc Table (TX)
  - Avail (TX)
  - Used (TX)
  - Queue Data (TX)

#### Window View

- **Read Window**
  - Desc Table (RX)
  - Avail (RX)
  - Used (TX)
  - Queue Data (RX)

- **Write Window**
  - Desc Table (TX)
  - Avail (TX)
  - Used (RX)
  - Queue Data (TX)
IPC for the Partitioning Hypervisor Jailhouse

Agenda

- Jailhouse introduction & philosophy
- IPC requirements and status quo
- Inter-partition networking prototype

Demonstrations

Summary
Live Demonstration

Jailhouse booting Linux

- QEMU/KVM
- Inter-Cell Comm
- Jailhouse Hypervisor
  - Core 0
  - Core 1
  - Core 2
  - Core 3
- Devices
  - UART 1
IPC for the Partitioning Hypervisor Jailhouse

Agenda

Jailhouse introduction & philosophy
IPC requirements and status quo
Inter-partition networking prototype
Demonstrations

Summary
Summary

• Jailhouse needs guest-to-guest communication
  • Simpler (for the hypervisor) than usual
  • Still fast enough to address common needs
  • Strict separation mandatory

• ivshmem-based networking prototype
  • Reasonable but not yet optimal throughput
  • Ongoing work to enable it also on ARM

• Outlook
  • Strict isolation via read/write / read-only split of shared memory
  • Full life-cycle management, likely via some “ivshmem 2.0”
Any Questions?

Thank you!

https://github.com/siemens/jailhouse

Jan Kiszka <jan.kiszka@siemens.com>