Jailhouse: Lightweight Real-Time Partitioning for Linux
On the Design of the Jailhouse Hypervisor

Agenda

Motivation

Jailhouse introduction & philosophy
Structure & mechanisms
Current status
Summary
Asymmetric Multi-Processing (AMP) & Linux

- Linux
- RTOS / Bare-Metal

Core 1 | Core 2 | Core 3 | Core 4
---|---|---|---
Hardware
AMP Drivers

• Low latency & high throughput

• Hard real-time

• Preexisting software

• Mixed criticality
AMP for Linux?

Linux #1

Linux #2

Linux #3

Linux #n

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

Hardware
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 hardware-assisted virtualization for isolation

• 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
AMP with Jailhouse

Linux

Jailhouse Hypervisor

Core 1  Core 2  Core 3  Core 4

Device A  Device B  Device C  Device D

Hardware

Root Cell

Non-root Cell

RTOS / Bare-Metal

Stahlkocher, CC BY-SA 3.0
Late Partitioning Concept

1. Boot phase
2. Partitioning phase
3. Operational phase
Jailhouse Components

- Jailhouse Image
- Cell Config
- Jailhouse Image
- Cell Config
- System Config
- Jailhouse Management Tool
- /sys/devices/jailhouse
- /dev/jailhouse

- Jailhouse Driver Module

- Linux Kernel

- Jailhouse Hypervisor

- Hardware
Two Management Models

Open Model
- Linux (root cell) is in control
- Cells not involved in management decisions
- Sufficient if root cell is trusted

Safety Model
- Linux controls, but...
- Certain cells are configured to vote over management decisions
- Building block for safe operation
Jailhouse Status – x86

- **Initial focus on x86, first Intel, then AMD**
  - Requirement: VT-x / VT-d, AMD-V / IOMMU
  - AMD interrupt remapping on to-do list

- **It's small!**
  - Currently ~8.8K lines of code (for Intel)

- **Direct interrupt delivery**
  - Zero VM exits, minimal latencies feasible
  - Max. timer IRQ latency (Xeon D-1540): <1 µs

- **Cache Allocation Technology**
  - Intel feature for partitioning caches
  - L3 supported, L2 on to-do list
Jailhouse Status – ARM

• **ARMv7**
  - Runs in FastModel, on Banana-Pi, NVIDIA Jetson TK1
  - WiP: TI AM572x evaluation module
  - SMMU on to-do list

**It's small too!**
  - Currently ~7.1k lines of code

• **ARMv8**
  - Contributed by Huawei (ERC Munich)
  - Merge delayed due to ARM fixes, but now ready
  - Targets: ARMv8 Foundation Model, AMD Seattle, LeMaker HiKey
Summary

- **Jailhouse provides clean AMP for Linux**
  - Full CPU isolation
  - Minimal I/O latency

- **Simplicity and cleanliness rules**
  - Reduced to the minimum (goal: <10k LOC/arch)
  - No emulation, no overcommitment
  - Support for safety scenarios, certification material under preparation

- **Jailhouse is a community project**
  - GPLv2, public development for 3 years
  - Significant contributions enabled
    - AMD64, ARMv7, ARMv8
  - To be proposed as kernel subsystem eventually
Thank you!

https://github.com/siemens/jailhouse

Jan Kiszka <jan.kiszka@siemens.com>