Reliable Recovery in Partitioned Embedded Systems using IVSHMEM and Jailhouse Hypervisor

Paresh Bhagat

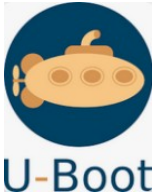
Gyan Gupta

About us: TI Processors and Open Source

Upstream FIRST!

Open Source

- ✔Upstream FIRST mentality!
- ✔Decades of contribution and collaboration
- ✔Focus on long term, sustainable and quality products
- ✔Upstream & opensource ecosystem in device architecture
- ✔Ingrained culture to give back to the community



Introduction

Gyan Gupta

Senior Engineering Manager, TI

Working as Senior Engineering Manager for RTOS, Safety diagnostic, with experience in Linux (kernel, user space), hypervisor, RTOS, Safety.

Paresh Bhagat

Software Engineer, TI

Working as Software Engineer with experience in Linux (kernel, userspace), hypervisor, tools.

Disclaimers

- This is a technology presentation, not product-readiness or roadmap commitment
- Opinions presented here are that of the speakers and may not reflect that of Texas Instruments Inc.

Table of Contents

Slide
Challenges in Real-Time and Mixed-Criticality Systems

Slide 02
Existing Solutions

Slide 03
Use of Hypervisor in Mixed Criticality Systems.

Slide 04-05
What is Jailhouse?

Slide 06-07
Linux, RTOS, Baremetal Cell

Slide 08
Crash Detection and Recovery

Slide 09
Use Cases

Slide 10
Advantages

Slide 11
Summary

Slide 12
References

Challenges in Real-Time and Mixed-Criticality Systems

Performance

Need to ensure strict isolation without impacting performance

Reliability and Uptime

Crucial in real-time and mixed criticality systems

Fault Isolation

Faults in one application can propagate to other subsystems

Delayed Recovery

Detecting failures late increases system downtime

Selective Recovery

Instead of full system reset, need selective recovery of faulty subsystem

Inadequate Crash Data

No stack traces, logs or memory dumps after failure (ex. baremetal app)

Existing Solutions

Bootloader

Ex. U-Boot Watchdog Integration

Kernel

Ex. Panic Reboot Timeout

User-Space

Using systemtd restart/ Busybox init or SysVinit

System-Wide Recovery

Hardware or Software Watchdog Timers

Use of Hypervisor in Mixed Criticality Systems

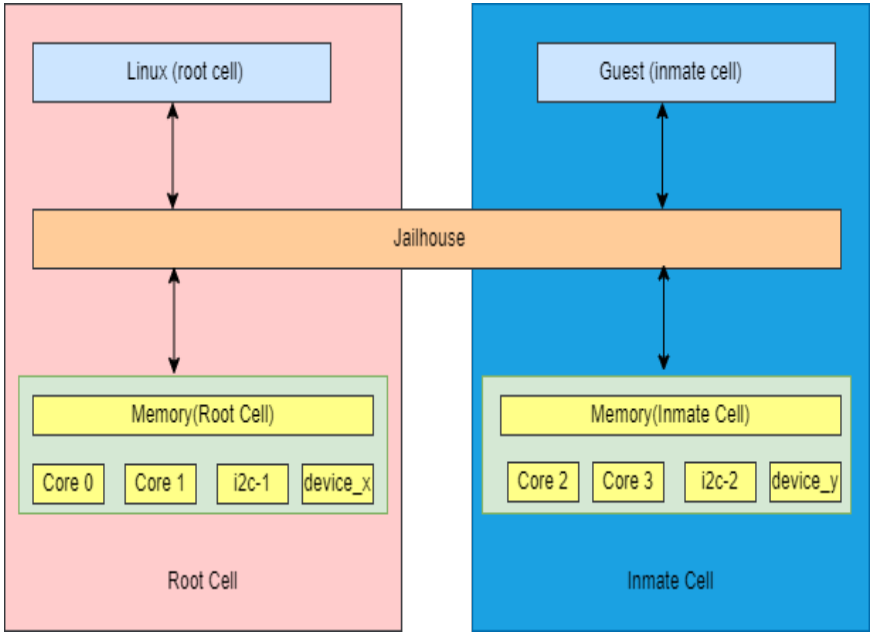
- To safely run safety-critical and non-critical workloads on the same hardware
- Strong isolation and deterministic resource partitioning are essential.
- A hypervisor ensures faults in one domain do not affect others.
- Enables reliable operation and faster recovery.

What is Jailhouse?

1. **Static Partitioning Hypervisor**, splits hardware into isolated compartments or cells.

2. Jailhouse **does not emulate resources**. VMs or inmates directly access hardware for high performance.

3. **Linux Based Root Cell:** Linux Instance used for configuration and management.



Jailhouse running state

Root Cell

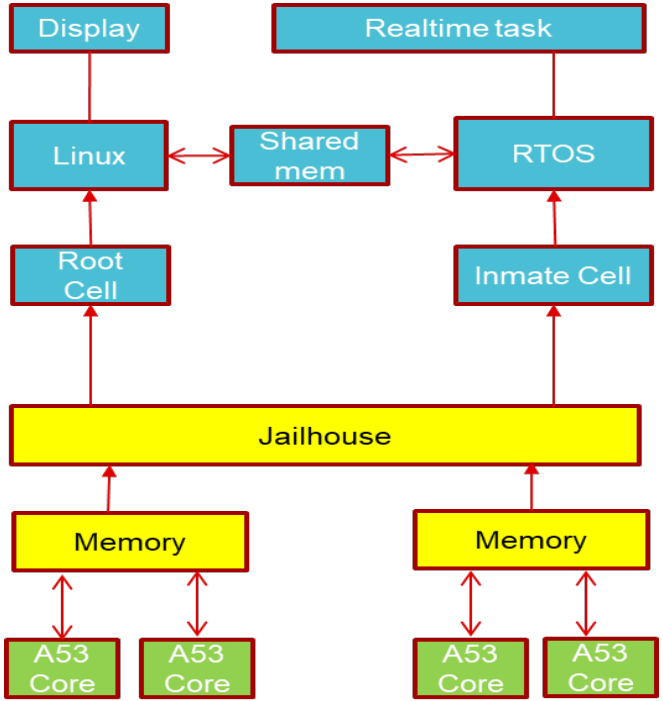
- Runs a minimal Linux kernel.
- Handles system initialization and management.

IVSHMEM

Creates a shared memory region accessible by multiple cells/VMs.

Non-root Cells (Inmate Cells)

- Independent partitions running different applications ex. real-time or specialized workloads.



Jailhouse running state

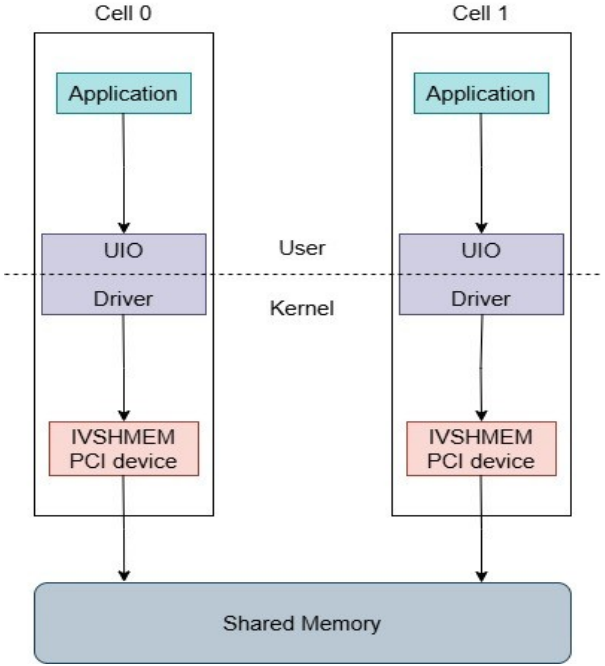
Linux Cell

1. Jailhouse **IVSHMEM** device is presented to Linux cell (root or inmate) as a PCI device and is accesses via /dev/uioX.

2. A **UIO (User Space Input/Output)** driver (e.g. uio_ivshmem) maps the shared memory and interrupt registers into user space.

3. User Space applications can then **directly read/write** the shared memory region and handle interrupts notifications.

4. Doorbell interrupts are used to notify peer cells about new data availability in shared memory.



IVSHMEM

Image Credit: The Optimization of IVSHMEM Based on Jailhouse | SpringerLink

RTOS or Baremetal

RTOS Inmate

Can directly map the IVSHMEM memory region in its address space.

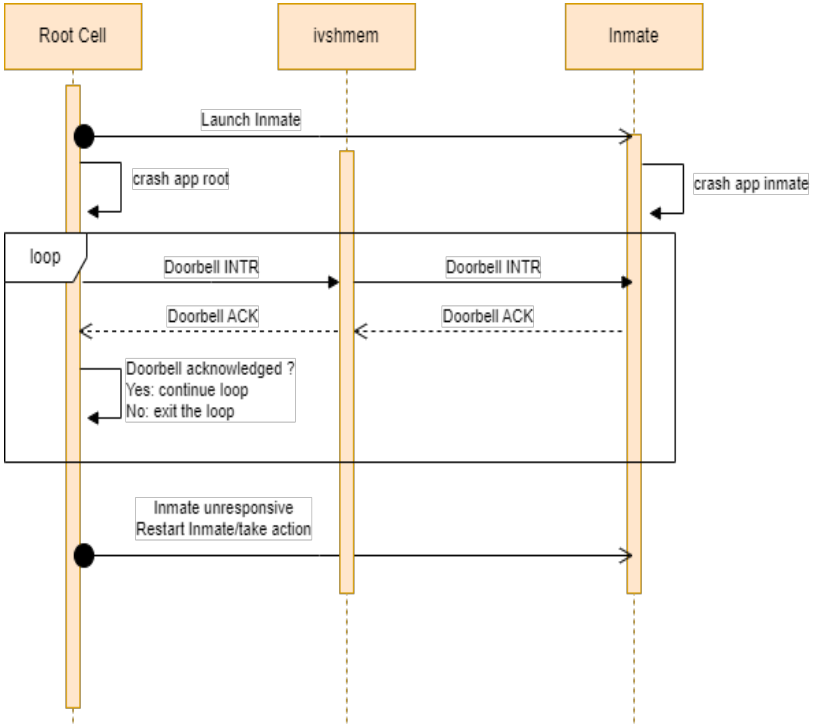
Bare-Metal Inmate

Same as RTOS. Directly access the shared memory region and any interrupt registers provided by Jailhouse

```
.mem_regions = {
    /* IVSHMEM shared memory regions for 00:00.0 (demo) */
    {
        .phys_start = 0xdfa00000,
        .virt_start = 0xdfa00000,
        .size = 0x10000,
        .flags = JAILHOUSE_MEM_READ | JAILHOUSE_MEM_ROOTSHARED,
    },
    {
        .phys_start = 0xdfa10000,
        .virt_start = 0xdfa10000,
        .size = 0x10000,
        .flags = JAILHOUSE_MEM_READ | JAILHOUSE_MEM_WRITE |
            JAILHOUSE_MEM_ROOTSHARED,
    },
    /* Peer 0 */ {
        .phys_start = 0xdfa20000,
        .virt_start = 0xdfa20000,
        .size = 0x10000,
        .flags = JAILHOUSE_MEM_READ | JAILHOUSE_MEM_ROOTSHARED,
    },
    /* Peer 1 */ {
        .phys_start = 0xdfa30000,
        .virt_start = 0xdfa30000,
        .size = 0x10000,
        .flags = JAILHOUSE_MEM_READ | JAILHOUSE_MEM_ROOTSHARED,
    },
    /* Peer 2 */ {
        .phys_start = 0xdfa40000,
        .virt_start = 0xdfa40000,
        .size = 0x10000,
        .flags = JAILHOUSE_MEM_READ | JAILHOUSE_MEM_WRITE |
            JAILHOUSE_MEM_ROOTSHARED,
    },
    /* IVSHMEM shared memory region for 00:01.0 */
    JAILHOUSE_SHMEM_NET_REGIONS(0xdfb00000, 1),
```

Cell config for Shared Memory

Crash Detection and Recovery



Use Cases

Heterogenous Systems with Mixed Criticality

Bare Metal Supervision

Safety Critical Isolation

Central Crash Analysis and Logging

Advantages

- Selective Recovery
- Recovery without Full Reboot
- Minimal Downtime
- Scalable
- Supplement Hardware Watchdog
- Isolation
- OS Agnostic

Summary

1. Hypervisor-based crash detection is not a **replacement** but a **powerful addition.**

2. **Strict Isolation** between different domains (ex. Linux, RTOS, baremetal).

3. **Recover specific parts** of the system without full reboot, with **fine-grained control** over restart and logging.

References

- Jailhouse Source
- TI Sitara AM62x SK EVM Jailhouse Doc
- TI Sitara AM62Px SK EVM Jailhouse Doc
- TI Sitara AM62Lx SK EVM Jailhouse Doc
- Siemens-jailhouse doc
- Understanding the Jailhouse hypervisor
- UIO: user-space drivers [LWN.net]
- The Userspace I/O HOWTO — The Linux Kernel documentation
- The Optimization of IVSHMEM Based on Jailhouse | SpringerLink

Q&A

- Contact Information:
  - Paresh Bhagat <p-bhagat@ti.com>
  - Gyan Gupta <g-gupta@ti.com>
- Also on IRC @ libera.chat #linux-ti

Learn more about TI products

- https://www.ti.com/linux
- https://www.ti.com/processors
- https://www.ti.com/edgeai

Why choose TI MCUs and processors?

Scalability

Our products offer scalable performance that can adapt and grow as the needs of your customers evolve.

Efficiency

We design products that extend battery life, maximize performance for every watt expended, and unlock the highest levels of system efficiency.

Affordability

We strive to make innovation accessible to all by creating cost-effective products that feature state-of-the-art technology and package designs.

Availability

Our investment in internal manufacturing capacity provides greater assurance of supply, supporting your growth for decades to come.