KVM in Embedded Requirements, Experiences, Open Challenges

Jan Kiszka, Siemens AG
Corporate Competence Center Embedded Linux
jan.kiszka@siemens.com
Agenda

- Embedded virtualization
  - What does it mean?
  - Why using KVM?

- Use case: KVM-hosted enterprise communication
  - Setup & requirements
  - Virtualization stack experiences

- KVM and real-time
  - Host & guest-side RT
  - Possible enhancements

- Conclusion
“Embedded” means
- Small?
- Limited resources?
- No display?
- Hard real-time?
- ...?

More generic definition
- Designed to perform specific, dedicated tasks
- Integrated part of a larger device
- Not recognizable as individual computer system

Embedded Virtualization
- System uses virtualization transparently
- May involve adaptations to system's task
Embedded Virtualization Benefits

Legacy system migration
- Avoid “divorce” of application and legacy OS
- Single-core software stacks on multicore hosts
- Emulation of discontinued hardware

Consolidation (keeping isolation)
- RTOS aside standard OS
- Multiple virtual boards (or root filesystems) on single silicon

Development environment
- Hardware/software co-development
- Debugging environment
- Virtualization allows speed-up (compared to pure emulation)
Top Requirements on Embedded Hypervisors

- **Hardware support**
  - CPU architecture
  - Board
  - Virtualization extensions (CPU, I/O)

- **Guest OS support**

- **Isolation**
  - Spatial (license barrier, IPR protection, rarely data security)
  - Temporal (provide real-time guarantees)

- **Customizability**

- **Footprint** (volume markets)
“We just need a tiny hypervisor to fully exploit this multicore CPU”
- “A few thousand” lines of hypervisor code
- Minimal hardware emulation
- “A bit” paravirtualization
- Devices are passed through

“But it would be nice to...”
- share some devices
- run upstream Linux and latest Windows
- over-commit resources
- manage power
- backup / migrate guests
- use advanced HA features
- ...

Slide 6  2010-08-10  Jan Kiszka  © Siemens AG, Corporate Technology
## Requirements Match

### Requirement

#### Architecture support
- x86
- PowerPC
- ARM
- Others

#### Board support

#### Guest OS support

#### Customizability

#### Footprint

#### Isolation
- Spatial
- Temporal

#### Future requirements

### KVM support

- ✓
- ✓ (Book E&S, no ISA 2.06 yet)
  - early stage
- ?
- ✓ (Linux...)
- ✓ (broad test bed, virtio drivers, ...)
- ✓
  - depends on use case
- ✓
  - (for most use cases)
  - improvable
  - well prepared
Use Case Example

KVM-hosted Enterprise Communication
Use Case: KVM-hosted Enterprise Communication

The user
Siemens Enterprise Communication (SEN)

The mission
Move proprietary RTOS and application stack from custom hardware to standard x86

Requirements
- Low impact on guest
- Preserve (soft) real-time qualities
- Prefer mainline open source technology

Evaluation ruled out
- Invasive paravirtualization (e.g. Xen's PV mode)
- Pure emulation
- Projects with too small communities
Use Case:  
KVM-hosted Enterprise Communication (2)

The choice: QEMU/KVM
- Early proof of concept using QEMU
- ~2500 LoC for custom hardware bits
- KVM acceleration nicely integrates on top
- Upstreamed generic fixes/enhancements since day 1

The new platform:
- QEMU/KVM hosts...
  - proprietary RTOS (multiple instances)
  - formerly stand-alone application stacks (virtual Linux appliances)
- libvirt as hypervisor interface
- Includes high availability stack

Two possible deployments
- Pre-installed on rack system => virtualization is *embedded*
- On customer server => virtual appliances
**Segmented x86 guests**
- 16-bit mode works quite well (despite uncommon use case)
- Task switching required most patching (few issues may remain)

**Soft real-time is achievable**
- `mlockall()` + `renice -20`
- Most latencies were I/O-related
- Decoupled logging and chardev outputs

**Board model maintenance**
- Out-of-tree enables flexible customizations
- ...but requires custom qemu-kvm package
- Upstream merge appears unrealistic
- 3rd way?
  - Open-Source-only machine plug-ins?
  - Stable API per stable series?
SEN Project Experiences (2)

Libvirt
- Feature gap required latest & greatest
- Faced few stability issues (resource management...)
- Suboptimal: QEMU wrapper script workaround
- All in all: benefits outweigh current drawbacks

Current open topic: live backup / snapshot
- Block live migration (yet?) too slow
- QEMU snapshots: longer downtime, qcow2-only
- libvirt-managed file-system / block layer snapshots?
Improving KVM

KVM and Real-Time
KVM and Real-Time –
Meeting Host Requirements

Requirement:
Guests must not defer host RT applications

Preemptible KVM
- Problem mostly solved
- The key: preemption notifiers (arch-agnostic concept)
- Keep an eye on preempt/IRQ-disabled paths!
- Known pitfall: wbinvd latencies (x86)

KVM on PREEMPT_RT
- Long supported, but quality varying
- Current 2.6.33.x-rt is fine
- Adoption of raw spinlocks reduced maintenance
- Risk of regressions remain => include in autotest?
KVM and Real-Time – Meeting Guest Requirements

Requirement:
Fulfill guest tasks in a timely manner

Precondition
Sufficient host real-time qualities
(PREEMPT_NONE → PREEMPT → PREEMPT_RT)

Already achievable
- Soft real-time
- Moderate guest reaction times
- Example for <1 ms peak latency:
  Host timer IRQ → in-kernel APIC model → guest RTOS → guest task

Feasible goals
- Standard KVM architecture: < 200 µs (x86)
- “Dedicated” KVM mode: close to hardware limits (<< 50 µs on x86)
What Kills Guest Real-Time?

KVM's MMU emulation
- Can contribute several milliseconds guest latency
- EPT/NPT resolves the issue
- Legacy RTOSes may also run MMU-less

I/O-related priority inversions
- Threaded AIO completions can accumulate long work queues
  => use Linux AIO or lower AIO thread priority
- QCOW2 (contains synchronous write calls)
- SDL graphic output
- Heavy traffic on chardev backends (e.g. virtual serial port)

RT-aware device emulation required
- We already heard about threading it... (→ Anthony's talk)
- No costly synchronous host services in VCPU context!
- Per-device locking will help to avoid priority inversions
- Also relevant for SMP scalability
Managing Priorities

- **Guest A**
  - RT task
  - Time-sharing task
- **Guest B**
  - Time-sharing task

**Black-Box VM Scheduling**

**Paravirtual Scheduling**
Towards Minimal-Latency KVM

KVM as fixed partition hypervisor

Enable migration

- Legacy RTOS
- Windows
- Full KVM
- Linux
- Core 1
- Core n

- Legacy RTOS
- Linux Appl.
- Linux
- Core 1
- Core n
Conclusion

- Embedded Virtualization is a broad domain, today focused on multi-core partitioning
- KVM already meets many of its key requirements
- Well set up for bringing enterprise features to embedded
- More work required
  - Reduce prio-inversions in hypervisor
  - Temporal isolation of guests
  - Paravirtualized scheduling
  - Non-x86 architectures

KVM may never fit all embedded use case, but a significant share
Thank You!

Any Questions?
Paravirtualized Scheduling

Execution model
- Use POSIX scheduling policies
- Per-VCPU policy/priority
- Map guest on VCPU thread priorities:

\[ p = \left\lfloor p_{\text{guest}} \frac{p_{\text{max}}}{99} \right\rfloor \]

- Boost to maximum priority during interrupt
- Nested boosts for NMI support

Host-guest Interface
- Two hypercalls
  - Set Scheduling Parameters (CPU-ID, policy, priority)
  - Interrupt Done

KVM prototype “just” requires rebase and upstream posting
Towards Minimal-Latency KVM (2)

**Step 1: Advanced CPU isolation**
- Single task shall dominate CPU
- Many proposals brought up, none mainline compatible
- Requires iterative approach
  - Migrate timers, disable sched tick
  - Move housekeeping work
  - Exclude CPU from RCU
  - Reduce IPI reasons
- Many folks interested, but no one working on it ATM

Reduce RT-unrelated “noise”

**Step 2: Run KVM VCPUs on isolated CPUs**
- Goals (guest in operation mode):
  - Zero user space VMM exits
  - Zero host task switches
  - In-kernel non-threaded IRQ (re-)injection
  - Adopt guest to avoid user space device emulations