ELC-Europe 2009 - Grenoble

Philippe GERUM - SourceTrek
The context

- Linux about to be natively real-time
  - PREEMPT_RT close to mainline
The context

- Linux about to be natively real-time
  - PREEMPT_RT close to mainline
- Legacy applications knocking on Linux's door
  - Traditional, embedded RTOS
  - Non-POSIX core API
  - Flat / physically addressed memory
  - Typically: VxWorks, pSOS, VRTX etc.
The issue

- Porting them to Linux currently means
  - Rebasing on Linux, changing design
  - Keeping design, keeping proprietary RTOS

- How to go the Linux way?
  - Keeping design, using Linux technologies
Possible solution

• Combine existing Linux technologies
  – Native real-time support
  – Linux-native virtualization
  – RTOS emulation
Common porting strategies

- Port to dual kernel
  - Over POSIX API
  - Over API emulator
  - Over *ad hoc* API
Common porting strategies

- Go Linux native
  - Over POSIX API
  - Over API emulator
Common porting strategies

- Introduce virtualization
  - Original RTOS guest
  - Vendor-specific
- Bare metal hypervisor
  - Leverage multi-core
Approaches are challenging

- Dual kernel Linux architecture is complex
Approaches are challenging

- Dual kernel Linux architecture is complex
Approaches are challenging

- Dual kernel Linux architecture is complex
  - Pressure on application design

Diagram showing a comparison between Real-time stack and Regular Linux, with layers of Drivers, Core, Libraries, and Apps.
Approaches are challenging

- Native real-time Linux is complex
Approaches are challenging

- Native real-time Linux is complex

Diagram:

- Bounded latency
- Real-time
- Total unfairness
- Drivers
- Core
- Libraries
- Apps
- Global efficiency
- Regular
- General fairness
Approaches are challenging

- Native real-time Linux is complex
  - Pressure on system configuration
Approaches are challenging

- Proprietary virtualization systems?
Approaches are challenging

• Proprietary virtualization systems?
  – Introduce dependencies on vendor
    • Hypervisor technology
    • Private (PV) Linux kernel
    • Original RTOS as guest
Approaches are challenging

- Proprietary virtualization systems?
  - Introduce dependencies on vendor
    - Hypervisor technology
    - Private (PV) Linux kernel
    - Original RTOS as guest
  - Enable Linux for proprietary RTOS
Approaches are challenging

- Proprietary virtualization systems?
  - Introduce dependencies on vendor
    - Hypervisor technology
    - Private (PV) Linux kernel
    - Original RTOS as guest
  - Enable Linux for proprietary RTOS

BUT,

- Do not help the Linux real-time effort
Teams are challenged

• Inclination to seek 1:1 API mapping
  – Over-emulation of missing calls
  – Pitfalls in mapping common calls
Teams are challenged

- Inclination to seek 1:1 API mapping
  - Over-emulation of missing calls
  - Pitfalls in mapping common calls
- Driver model
  - Weak vs strong
  - Linux kernel API is more complex
Teams are challenged

• Inclination to seek 1:1 API mapping
  – Over-emulation of missing calls
  – Pitfalls in mapping common calls
• Driver model
  – Weak vs strong
  – Linux kernel API is more complex
• Protocol stacks
  – Keep “as is” or offload to Linux?
Legacy issues

- Software architecture
  - BSP code exposed
  - Application and driver code entangled
  - Non-public API sometimes used
Legacy issues

- Software architecture
  - BSP code exposed
  - Application and driver code entangled
  - Non-public API sometimes used

- Programming model
  - Flat / physically addressed memory assumed
  - Supervisor mode assumed
  - CPU architecture assumed
About RTOS emulators
RTOS API emulation?

- A way to mimic the RTOS interfaces
  - Evades the BSP issue
  - Source-level approach
- Has real-time requirements
  - Must run over a deterministic core
  - Must exhibit real-time properties itself
Myths and Reality

• Can (RTOS) API emulation be accurate?
  – Based on public, dependable interfaces
  – Relies on a documented feature set
Myths and Reality

- Can (RTOS) API emulation be accurate?
  - Based on public, dependable interfaces
  - Relies on a documented feature set

Do you trust your vendor documentation?
Myths and Reality

• Can (RTOS) API emulation be accurate?
  – Based on public, dependable interfaces
  – Relies on a documented feature set

Do you trust your vendor documentation? YES
Should your code rely on undocumented features?
Myths and Reality

- Can (RTOS) API emulation be accurate?
  - Based on public, dependable interfaces
  - Relies on a documented feature set

Do you trust your vendor documentation? **YES**
Should your code rely on undocumented features? **NO**
Should your code expect undocumented behavior?
Myths and Reality

• Can (RTOS) API emulation be accurate?
  – Based on public, dependable interfaces
  – Relies on a documented feature set

Do you trust your vendor documentation?  YES
Should your code rely on undocumented features?  NO
Should your code expect undocumented behavior?  NO

Therefore, you don't need the original API implementation to emulate it properly.
Myths and Reality

- Isn't API emulation slower?
Myths and Reality

- Isn't API emulation slower?
  - Traditional RTOS share basic semantics
    - Optimized building blocks can be made
    - Efficient “window-dressing” follows
    - Leveraging single address space helps
Myths and Reality

- Isn't API emulation slower?
  - Traditional RTOS share basic semantics
    - Optimized building blocks can be made
    - Efficient “window-dressing” follows
    - Leveraging single address space helps
  - Naive emulation over POSIX not enough
    - POSIX semantics do not map 1:1
    - POSIX-based building blocks may work better
RTOS emulators shortcomings

- Limited emulation coverage
  - noarch/generic core services
RTOS emulators shortcomings

- Limited emulation coverage
  - noarch/generic core services
- Require Application / Driver split
  - BSP code not accessible from user-space
  - I/O resources live in kernel space
RTOS emulators shortcomings

- Limited emulation coverage
  - noarch/generic core services
- Require Application / Driver split
  - BSP code not accessible from user-space
  - I/O resources live in kernel space
- Restricted by Linux protections
  - No supervisor actions from user-space
Our assets
PREEMPT-RT

- Fully native real-time support
  - Enables real-time virtualization
- Promise of embedded multi-core scalability
  - Sophisticated locking model
  - Sophisticated scheduling
KVM

- Complete sandboxing
- Compatible memory spaces
- Device virtualization through host
  - virtio
- Device emulation through VM
  - Qemu-based modelling
Introducing Xenomai

- Generic RTOS core
- Host abstraction
  - Dual kernel
  - Simulator
  - (Single image *)

(*) Xenomai/SOLO
Introducing Xenomai

- Generic RTOS core
- Host abstraction
  - Dual kernel
  - Simulator
  - (Single image *)
- RTOS personalities

(*) Xenomai/SOLO
Introducing Xenomai

- RTOS building blocks
  - Thread scheduling
  - Synchronization
  - Interrupt handling
  - Memory allocation
  - Timing services
Introducing Xenomai

- RTOS building blocks
  - Thread scheduling
  - Synchronization
  - Interrupt handling
  - Memory allocation
  - Timing services

Building blocks:
- Sched
- Synch
- IRQ
- Memory
- Timing

Libraries:
- taskLib
- semLib
- msgQLib
- wdLib
- tickLib
- sysLib

e.g. VxWorks
What about combining?

- Real-time host kernel
  - PREEMPT-RT
What about combining?

- Real-time host kernel
  - PREEMPT-RT
- Virtualization core
  - KVM
  - QEMU
What about combining?

- Real-time host kernel
  - PREEMPT-RT
- Virtualization core
  - KVM
  - QEMU
- RTOS emulation
  - Xenomai
Virtualization + RTOS emulation

Improvements

• Native real-time
• Original programming model
• Better emulation coverage
• Sandboxing
• Legacy device emulation
Virtualization + RTOS emulation

Restrictions

- No ABI compatibility
- Still not 100% source compatible
- Reworking the device driver layer still required
Virtualize & Emulate
Improved emulation engine

Emulation core

- Xenomai guest
  - Freestanding mode
  - RTOS personality
- QEMU
  - Virtual machine
Improved emulation engine

Handling I/O

- Paravirtualized
  - Common hw
  - High bandwidth
- Emulated
  - Precise emulation
  - Low bandwidth
Improved emulation engine

Native real-time VMM

- PREEMPT-RT host
  - KVM-enabled
TODO list

- Real-time aware KVM
  - Guest scheduling
- Real-time aware QEMU
  - I/O emulation
- Guest mode Xenomai core
- Extended emulation coverage
More applications
Could also be used for...

- Application-specific virtual RTOS
  - Virtual RT appliance (sort of)
Could also be used for...

- Application-specific virtual RTOS
  - Virtual RT appliance (sort of)
- Transition path for in-house RTOS
  - Consolidate & extend via virtualization
Could also be used for...

- Application-specific virtual RTOS
  - Virtual RT appliance (sort of)
- Transition path for in-house RTOS
  - Consolidate & extend via virtualization
- Simulation of complex architectures
  - e.g. modeling Arinc653 systems
Conclusion
Legacy RT application to Linux

**Today**

- Rebase on Linux, change design
- Keep design, keep proprietary RTOS
Legacy RT application to Linux

**Today**
- Rebase on Linux, change design
- Keep design, keep proprietary RTOS

**Tomorrow**
- Combine existing technologies
  - Rely on real-time capable virtualization
  - Couple with accurate RTOS emulation
The End

Thank you for attending