KVM/arm64 Architectural Evolutions

Marc Zyngier <marc.zyngier@arm.com>

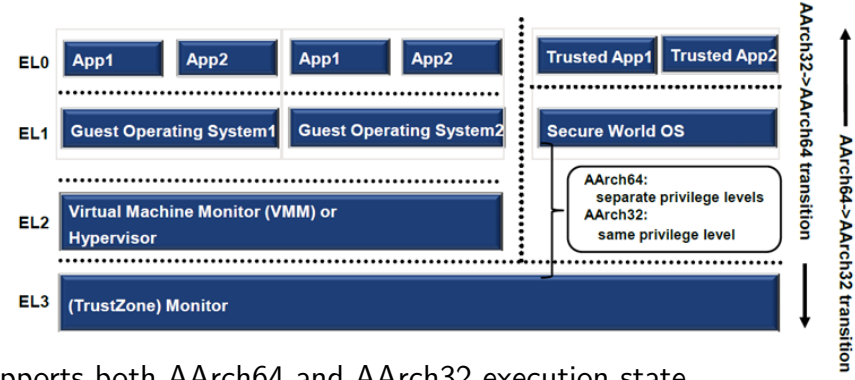
LCA '15

What to Expect in This Presentation

During this presentation, we will discuss:

- The ARM® architecture Virtualization Extensions...
- ... and how KVM/ARM uses them
- How the ARM architecture is evolving...
- ... and what this means for KVM/ARM

ARMv8-A Privilege Model



- Supports both AArch64 and AArch32 execution state
- 32-64bit interworking limited to exception boundaries
- AArch64 always has a higher privilege than AArch32
- AArch64 state is inclusive of the lower-privileged 32bit exception level

ARM

ARM Architecture Virtualization Extensions

- Introduced with the latest revision of the ARMv7 architecture
- New hypervisor execution state (EL2 or HYP)
- Non-secure world, higher privilege than EL1
- Second stage translation
  - Adds an extra level of indirection between guests and physical memory
  - A form of nested page tables, as implemented by many other architectures
  - TLBs are tagged by Virtual Machine ID (VMID)
- Ability to trap access to most system registers
  - The hypervisor decides what it wants to trap
- Can handle IRQs, FIQs and asynchronous aborts
  - The guest doesn't see physical interrupts firing, for example
- Guests can call into HYP mode (HVC instruction)
  - Allows for para-virtualized services
- Standard architecture peripherals are virtualization-aware
  - GIC and timers have specific features to help virtualization

EL2: Not EL1++

HYP is not a superset of NS-EL1, but rather an orthogonal mode allowing for multiple NS-EL1 guests to be multiplexed on the hardware.

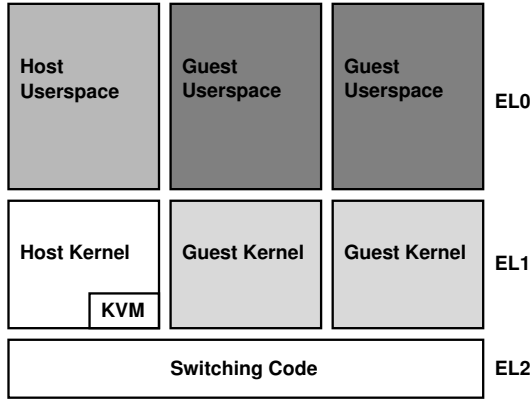
- Own translation regime
  - Separate Stage 1 translation, no Stage 2 translation
  - Broadly follows the LPAE format
  - Only one Translation Table Base Register (TTBR0_EL2)
- It would be very difficult to run Linux in EL2
  - Requires too many changes to be practical
- Instead, the EL2 mode can be used as a "world switch"
  - Between guests (bare metal hypervisor / Type-1)
  - Between host and guests (hosted hypervisor / Type-2).
    This makes the host a form of specialized guest

KVM/ARM: General Architecture

KVM/ARM uses HYP mode to switch between host and guest

Saving and restoring:

- Stage 2 translation table
- Trap configuration
- GP registers
- System control registers
- Floating point
- GIC configuration
- Timers



ARM

The Evolution of ARMv8

The ARM architecture is in constant evolution to better fit the software ecosystem.
The upcoming ARMv8.1 revision contains a number of points of interest:

- New atomic instructions
- New SIMD instructions
- Memory management improvements
- Performance monitoring improvements
- VMIDs expanded to 16 bits
- New Virtualization Host extensions, designed for Type-2 hypervisors

Unsurprisingly, this last item is very interesting for KVM

ARMv8.1: an Improved EL2

The Virtualization Host Extensions (VHE) expand the capabilities of EL2:

- Designed to improve the support of Type-2 hypervisors
- Allows the host OS to be run at EL2
- The host OS requires minimal changes to run at EL2
- User-space still runs at EL0
- Guest OSes run at EL1
- Host has no software running at EL1
- AArch64 specific

EL2 becomes a strict superset of EL1

VHE: Software compatibility

Major design goals for VHE:

- Make architecture features discoverable
- Allow EL1 software to run at EL2 transparently
- Put the burden of the change on virtualization software

When VHE is in use:

- Most EL2 system registers are accessed with their EL0/EL1 counterpart
  - `TCR_EL1` access at EL2 really accesses `TCR_EL2`
- EL0/EL1 system register access from EL2 requires a specific accessor
  - To access `TCR_EL1`, you must use `TCR_EL12`

As a consequence of the above:

- The kernel should run mostly unmodified
- Virtualization software has to be modified

VHE: Impacts on the Linux Kernel

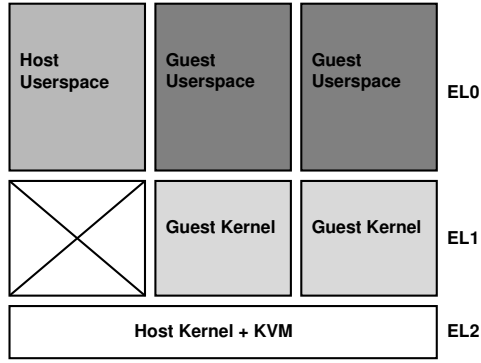
With VHE enabled, the Linux kernel can now be run at EL2.

Changes required for the kernel itself:

- Prevent the kernel from switching to EL1
- New debug configuration
- Use HYP timer interrupt instead of the Guest's
- And that's about it.

The KVM changes are more invasive:

- New method to enter the switch code
- New way to address guest's system registers
- Vectors are switched depending on the role (host kernel or hypervisor)



ARM

VHE: Impacts on the KVM/arm64 Architecture

- Host system register save/restore is reduced
  - Almost no system register sharing
  - Down to four registers (tpidr*, actlr)
- Guest system register lazy save/restore
  - Can be deferred until actually required by the host
  - Or until scheduling another VM
  - Could significantly reduce system register churn
- Reduced physical interrupt latency
  - Only a minimal state has to be restored before handling the interrupt
- No trap to enter the KVM code
  - Just a normal function call
- No HYP mappings anymore
  - The whole kernel is there
- Heavily re-written switching code

VHE: One Kernel to Rule Them All

- We want one single kernel binary
- We want it to support all revisions of ARMv8
- That does include KVM as well
- We need to be able to pick the right code at runtime

Two complementary approaches:

- Using feature bits to make runtime decisions
  - `if (is_kernel_in_hyp_mode()) {...} else {...}`
  - Simple, but only on slow path
- Using runtime kernel patching to rewite critical code
  - `ifvhe "mrs x2, TCR_EL1", "mrs x2, TCR_EL12"`
  - Very efficient, good for fast path
  - Less readable, harder to maintain
  - Mostly for assembly code

VHE: News from the Coding Front

This is still a work in progress:

- Prototype code based on 3.15-rc3
  - A nasty ball of hacks
  - Only used for architecture evaluation
  - Unmaintainable, and unmaintained
  - Will hopefully be deleted soon
- Proper support in progress
  - Based on *Mainline of the Day*
  - Using code patching extensively
  - Tested on ARM Fast Models
  - Hopefully public soon (-ish)

Thank You