CMP SCI 635: Modern Computer Architecture

ARM Virtualization: Performance and Architectural Implications

Name: Kunjal Panchal Date: 29th Oct, 2019 Student ID: 32126469

Paper: 14 – Virtualization [Dall16]

Strengths:

1. As we run the core process in all different privilege levels and thereby share one memory view, it seems beneficial to use the same page table set for all privilege levels. As the hypervisor for ARMv8 requires the new page table format anyway, we can implement that within the prior existing kernel. The benefit of this approach would be that we could use the kernel as testbed for validating the new format.

2. The advantage of the architecture where the hypervisor does nothing more than saving and restoring the registers of the Virtual CPU control interface from the VM state is that we can strictly follow the principle of minimizing the common TCB [Trusted Computing Base]. By implementing nearly all virtualization-related functionality into the unprivileged VMM.

Weaknesses:

1. An open issue would be to support for symmetric multi-processing (SMP) in VMs. In general, SMP is already supported by the bare-hardware platform on Cortex A15 CPUs. Having more than one CPU in a VM is mostly a question of enhancing the VMM accordingly. Can we try to represent each virtual CPU by a dedicated VM session? The VM objects implemented in the kernel are almost identical to normal threads with regard to scheduling and can be assigned to different cores. Would we need additional code to be implemented in the kernel? Apart from that, the VMM has to guarantee synchronicity with regard to commonly used device models when dealing with several VM sessions.

2. If more than one VMM might run in parallel to control different VMs and there is only one physical GICV, we cannot grant the VMM direct access to it. Otherwise, different VMMs would interfere. On the other hand, only the VMM knows where the CPU interface shall be placed in guest-physical memory, dependent on the platform it emulates.

Questions/Assertions:

1. Can virtual timer and maintenance interrupts of the interrupt controller cannot be obtained via an IRQ session? Since they are shared between all VMMs. When they occur, I assume, only the VMM of the currently active VM shall receive it. Such an interrupt is signalled by the hypervisor to the VMM implicitly via the VM state. When the VMM recognizes that the VM was stopped due to an interrupt, it reads the interrupt information out of the VM state.

2. The ARM's virtual timer hardware is of little use. We cannot program the timer for an inactive VM and use it for another active VM at the same time. Instead, another time resource needs to be used to monitor time progress of an inactive VM.

3. How to provide direct device access to a virtual machine (VM) for ARM? From what I gathered; we know how to separate device drivers from the rest of the system via IOMMUs of the x86 platform. But on ARM, we have no platforms equipped with such technology yet.

4. How can ARM's IOMMU be integrated in a microkernel-based operating system?

5. In contrast to most other ordinary device interrupts, the ones originating from the SysMMUs are grouped. That means they partially share the same interrupt line. How ARMv8 and x86 distinguish the different sources within one interrupt group?

6. I had read somewhere that giving the VM more virtual RAM than it needs might not be beneficial. Is it some other disadvantage of it rather than just spending more money than required? And can't VMs make some use of memory scaling?