ARM VIRTUALIZATION

“FOR THE MASSES”

Christoffer Dall <c.dall@virtualopensystems.com>
<cdall@cs.columbia.edu>
Smartphones
Smartphones
Tablets
But now also...

ARM Servers
But now also...

- ARM Servers
- Laptops
Even in cars
... and toasters!
So why virtualize?
So why virtualize?

Server virtualization brings high availability, load balancing, and more..
So why virtualize?

- Phone virtualization
- iOS on Android
- Home-phone Work-phone
- Throw-away virtual phones
So why virtualize?

Rich OS in embedded space with legacy OS support
So why virtualize?

Or debug events leading up to this kernel crash using GDB?
People

- Marc Zyngier from ARM: VGIC + Timers
- Antonios Motakis from Virtual Open Systems: VFP
- Rusty Russell from Linaro: User space register interface
- Peter Maydell from Linaro: QEMU and arch. compliance
- Christoffer Dall from Virtual Open Systems: Maintainer
KVM/ARM
KVM/ARM

• ARM is not virtualizable
KVM/ARM

- ARM is not virtualizable
- Requires Hardware Virtualization Extensions
KVM/ARM

- ARM is not virtualizable
- Requires Hardware Virtualization Extensions
- Runs on Cortex-A15 and Cortex-A7
KVM/ARM upstreaming

- Being upstreamed in mainline
- 14th patch series revision
- Waiting for ARM ack's
- Upstreamed through kvm/next or Russell King (ARM)
ARM Architecture

- RISC-style load/store architecture
- Two instruction modes: ARM and Thumb-2
- Co-processors: An extension to the instruction set
- CP15 holds CPU control state
ARMv7 Processor Modes

PL 0
- User
- System
- SVC
- FIQ
- IRQ
- UNDEF
- ABT

PL 1
## ARM Architecture

### Registers

<table>
<thead>
<tr>
<th>User</th>
<th>System</th>
<th>SVC</th>
<th>ABT</th>
<th>UND</th>
<th>IRQ</th>
<th>FIQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SP_usr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LR_usr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPSR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>User</th>
<th>System</th>
<th>SVC</th>
<th>ABT</th>
<th>UND</th>
<th>IRQ</th>
<th>FIQ</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>SP_svc</td>
<td>SP_abt</td>
<td>SP_und</td>
<td>SP_irq</td>
<td>SP_fiq</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LR_svc</td>
<td>LR_abt</td>
<td>LR_und</td>
<td>LR_irq</td>
<td>LR_fiq</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SPSR_irq</td>
<td>SPSR_fiq</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SPSR_irq</td>
<td>SPSR_fiq</td>
</tr>
</tbody>
</table>
ARM Virtualization Extensions

- Hardware support for virtualization
- Hyp-mode
- Stage-2 memory translation
ARMv7 Processor Modes

PL 0
- User
- System
- SVC
- FIQ
- IRQ
- UNDEF
- ABT

PL 1
Security Extensions (TrustZone)

<table>
<thead>
<tr>
<th>PL 0</th>
<th>Non-Secure</th>
<th>Secure</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td></td>
<td></td>
</tr>
<tr>
<td>System</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SVC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIQ</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IRQ</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UNDEF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ABT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PL 1</th>
<th>Non-Secure</th>
<th>Secure</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td></td>
<td></td>
</tr>
<tr>
<td>System</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SVC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIQ</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IRQ</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UNDEF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ABT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Monitor
Virtualization Extensions

Non-Secure

**PL 0**
- User
- System
- SVC
- FIQ
- IRQ
- UNDEF
- ABT

**PL 1**
- System
- SVC
- FIQ
- IRQ
- UNDEF
- ABT

**PL 2**
- HYP

Secure

**PL 0**
- User
- System
- SVC
- FIQ
- IRQ
- UNDEF
- ABT

**PL 1**
- System
- SVC
- FIQ
- IRQ
- UNDEF
- ABT

Monitor
Hyp mode

• Access to hypervisor control registers
• Controls stage-2 translations
• Entry through exception (ex. HVC instruction)
• Exit by exception return
LPAE

- Large-Physical-Address-Extension (LPAE)
- 40-bit physical addresses
- 64-bit page table descriptors
- Page table format used for stage-2 translations
Memory Translation with LPAE

Virtual Addresses

0

User space application

Kernel

Physical Addresses

0

RAM

Devices

$2^{40}$

Stage-1 Page Tables

MMU
Stage-2 Page Tables

Virtual Addresses

0 2^40

User space application  Kernel

Intermediate Physical Addresses (IPA)

0 2^40

RAM  Devices

Stage-1 Page Tables

MMU

Physical Addresses

0 2^40

RAM  Devices

Stage-1 Page Tables

MMU
Compared to x86

• No concept of VMCS or VMRESUME / VMEXIT
• Instead, control registers must be programmed in Hyp mode
• Hyp mode is not a superset of normal privileged mode!
• We cannot run the host kernel in Hyp mode.
• Hyp mode was designed with classic embedded hypervisor in mind
void *mem;

kvm_fd = open("/dev/kvm");
vm_fd = ioctl(kvm_fd, KVM_CREATE_VM, 0);

posix_memalign(&mem, PAGE_SIZE, 0x20000000);
ioctl(kvm_fd, KVM_SET_USER_MEMORY_REGION, mem);

vcpu1_fd = ioctl(vm_fd, KVM_CREATE_VCPU, 0);
vcpu2_fd = ioctl(vm_fd, KVM_CREATE_VCPU, 0);

thread1_start(run_vcpu(vcpu1_fd));
thread1_start(run_vcpu(vcpu2_fd));

while (1)
    process_io();
KVM and Hyp mode

- Running VMs have to go through Hyp mode
- VM data structures must be mapped in Hyp mode
Owning Hyp mode

- HVC instruction enters Hyp mode
- CPU jumps to HVBAR + offset
- Host kernel must control HVBAR and HTTBR
- Boot the kernel in Hyp mode, not SVC!
KVM/ARM Architecture

- **QEMU**
  - PL0

- **VM0**
  - Guest Kernel
  - PL0 + PL1

- **VM1**
  - Guest Kernel
  - PL0 + PL1

- **Host Kernel**
  - PL1

- **KVM**
  - PL1 + PL2

- **Hardware**
KVM/ARM Architecture

- KVM virtualizes CPU and Memory
- QEMU virtualizes platform and devices
- Two exceptions: VGIC and Timer
CPU virtualization

- Context switching guest accessible registers
- Trap access to other registers and emulate in software
Memory Virtualization

- Built on hardware support
- VMID tagged caches and TLB
- Support swapping through MMU notifiers
- Support freeing stage-2 page tables
- Support hugetlbfs and THP (waiting on ARM support)
Handling stage-2 aborts
Handling stage-2 aborts

1. Assign a blank stage-2 PGD to the VM
Handling stage-2 aborts

1. Assign a blank stage-2 PGD to the VM

2. Run vcpu
Handling stage-2 aborts

1. Assign a blank stage-2 PGD to the VM
2. Run vcpu
3. VM generates stage-2 fault to Hyp mode
Handling stage-2 aborts

1. Assign a blank stage-2 PGD to the VM
2. Run vcpu
3. VM generates stage-2 fault to Hyp mode
4. Capture fault information
Handling stage-2 aborts

1. Assign a blank stage-2 PGD to the VM
2. Run vcpu
3. VM generates stage-2 fault to Hyp mode
4. Capture fault information
5. Create stage-2 mapping
Handling stage-2 aborts

1. Assign a blank stage-2 PGD to the VM
2. Run vcpu
3. VM generates stage-2 fault to Hyp mode
4. Capture fault information
5. Create stage-2 mapping
6. Run vcpu
I/O Virtualization

- All I/O is MMIO
- A stage-2 abort not in RAM considered I/O
- Emulation hints in HSR
Load/Store instruction decoding
Load/Store instruction decoding

!ldr sp, {r0, #4}!
Load/Store instruction decoding

`ldr sp, {r0, #4}!

addr = reg[0] + 4;
reg[sp] = mem[addr];
reg[0] = addr;`
Load/Store instruction decoding

• No decode hints in HSR!

```c
addr = reg[0] + 4;
reg[sp] = mem[addr];
reg[0] = addr;
```
Load/Store instruction decoding

- No decode hints in HSR!
- Avoided in newer kernels

```c
addr = reg[0] + 4;
reg[sp] = mem[addr];
reg[0] = addr;
```

```c
ldr sp, {r0, #4}!
```
Hardware Interrupts

- Configured to trap to Hyp mode
- World-Switch back into host
- Re-enable IRQs
Generic Interrupt Controller

Devices

GIC

CPU 0

CPU 1
Generic Interrupt Controller

Devices

Dist.

CPU Interface

GIC

CPU Interface

CPU 0

CPU 1
Generic Interrupt Controller

Devices

CPU Interface

Dist.

CPU Interface

GIC

CPU 0

CPU 1
Generic Interrupt Controller

Devices

CPU Interface

Dist.

CPU Interface

GIC

CPU 0

CPU 1
Generic Interrupt Controller

Devices

Dist.

GIC

CPU Interface

CPU Interface

CPU 0

CPU 1
Generic Interrupt Controller

Devices

CPU 0

CPU 1
Generic Interrupt Controller

Devices

Dist.

Virtual Control Interface

CPU Interface

Virtual CPU Interface

CPU Interface

GIC

CPU 0

CPU 1
Virtual Interrupts

- Generated by QEMU devices
- KVM_IRQ_LINE
- Programmed in VGIC list registers
- No virtual distributor interface
Architected Generic Timers

- Defines physical + virtual timer and counter
- Virtual counter = Physical counter + offset
- Counter reads from guest without trap
- Timer interrupts do trap
Voodoo Bug
Voodoo Bug

page = __get_user_pages(...);
...
if (mapWritable)
  SetPageDirty(page);
stage2_set_pte(page, ...);
put_page(page);
Voodoo Bug

```c
page = __get_user_pages(...);
...
SetPageDirty(page);
stage2_set_pte(page, ...);
put_page(page);
```
Clues?
Questions?