Running Linux at EL2

Linaro Connect BKK16
Christoffer Dall
This Talk

• Technical Talk

• Assumes Familiarity with Operating Systems and the ARM architecture

• Make it interactive! Ask Questions!
Virtualization on ARM

Virtualization Extensions

EL ?#@ what?
ARM Virtualization Extensions

EL0 User

EL1 Kernel
ARM Virtualization Extensions

EL0: User
EL1: Kernel
EL2: Hyp
ARM Virtualization Extensions

- EL0: User
- EL1: Kernel
- EL2: Hypervisor

VM 0
- User
- Kernel

VM 1
- User
- Kernel
Type-1 Hypervisor

- Dom0
  - User
  - Kernel

- DomU
  - User
  - Kernel

EL0

EL1

EL2

Xen
KVM

Host

User

Kernel / KVM

KVM

VM

User

Kernel

EL0

EL1

EL2
KVM/ARM VM Exits

Host

EL0
User

EL1
Kernel / KVM

EL2
KVM

VM

User

Kernel
KVM/ARM VM Exits

Host

EL0
User

EL1
Kernel / KVM

EL2
KVM

VM
User

Kernel

Exception
KVM/ARM VM Exits

Host

EL0
User

EL1
Kernel / KVM

Exception

EL2
KVM

VM

User

Kernel

switch state
KVM/ARM VM Exits

EL0
User

Kernel / KVM

KVM

EL1
Host

VM

User

Kernel

EL2

Ret
Exception
switch state
KVM/ARM VM Exits

Host

EL0

User

Kernel / KVM

KVM

EL1

HVC

Ret

Exception

switch state

EL2

VM

User

Kernel
KVM/ARM VM Exits

EL0
User

Kernel / KVM

HVC

EL1
Ret
switch state

KVM

EL2
Exception
switch state

VM
User

Kernel
KVM/ARM VM Exits

EL0

Host

EL1

User

Kernel / KVM

EL2

KVM

HVC

Ret

Exception

switch state

Ret

switch state

Ret
An alternative

EL0
User

EL1
Kernel / KVM

EL2

VM
User

Kernel
A note about VHE

• Virtualization Host Extensions (VHE)

• Available in ARMv8.1

• Allows running the host kernel in EL2 without modifying the software

• Good starting point.
Implementation

• System Registers
• Memory
• Exceptions
• KVM
System Registers Accesses

• Lots of:

```c
#ifndef CONFIG_EL2_KERNEL
msr    tcr_el1, x0
#else
msr    tcr_el2, x0
#endif
```
Implementation

• System Registers
• Memory
• Exceptions
• KVM
EL1 VA Space (39 bits)

0x7f ffffffff

Userspace

0x0

TTBR0_EL1

0xffffffff ffffffff

Kernel

0xffffffff80 00000000

TTBR1_EL1
EL2 VA Space (39 bits)

Where do we put the kernel and userspace?

TTBR0_EL2
Option #1

- Problem A: Page table formats
- Problem B: requires TLB invalidation
- Problem C: address space compression
Option #1: Page tables

- Must share page tables between kernel and user
- Different page table format in EL0/1 and EL2

<table>
<thead>
<tr>
<th>Descriptor bit</th>
<th>EL1/0</th>
<th>EL2</th>
</tr>
</thead>
<tbody>
<tr>
<td>AP[1]</td>
<td>User access</td>
<td>RES1</td>
</tr>
<tr>
<td>UXN/XN</td>
<td>UXN</td>
<td>XN</td>
</tr>
<tr>
<td>PXN</td>
<td>PXN</td>
<td>RES0</td>
</tr>
</tbody>
</table>
**Option #1: Page tables**

<table>
<thead>
<tr>
<th>Descriptor bit</th>
<th>EL1/0</th>
<th>EL2</th>
</tr>
</thead>
<tbody>
<tr>
<td>AP[1]</td>
<td>User access</td>
<td>RES1</td>
</tr>
<tr>
<td>UXN/XN</td>
<td>UXN</td>
<td>XN</td>
</tr>
<tr>
<td>PXN</td>
<td>PXN</td>
<td>RES0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Descriptor bit</th>
<th>User entries</th>
<th>Kernel entries</th>
</tr>
</thead>
<tbody>
<tr>
<td>AP[2]</td>
<td>1/0 user: r/rw kernel: r/rw</td>
<td>1/0 user: none kernel r/rw</td>
</tr>
<tr>
<td>AP[1]</td>
<td>0 user: x/xn kernel: x/xn</td>
<td>0 user: x kernel: x</td>
</tr>
<tr>
<td>UXN/XN</td>
<td>0/1 user: no effect kernel: no effect</td>
<td>0 user: no effect kernel: no effect</td>
</tr>
<tr>
<td>PXN</td>
<td>0 user: no effect kernel: no effect</td>
<td>0 user: no effect kernel: no effect</td>
</tr>
</tbody>
</table>
Option #1: Page tables

<table>
<thead>
<tr>
<th>Descriptor bit</th>
<th>EL1/0</th>
<th>EL2</th>
</tr>
</thead>
<tbody>
<tr>
<td>AP[1]</td>
<td>User access</td>
<td>RES1</td>
</tr>
<tr>
<td>UXN/XN</td>
<td>UXN</td>
<td>XN</td>
</tr>
<tr>
<td>PXN</td>
<td>PXN</td>
<td>RES0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Descriptor bit</th>
<th>User entries</th>
<th>Kernel entries</th>
</tr>
</thead>
<tbody>
<tr>
<td>AP[2]</td>
<td>1/0</td>
<td>user: none</td>
</tr>
<tr>
<td></td>
<td></td>
<td>kernel: r/rw</td>
</tr>
<tr>
<td>AP[1]</td>
<td>1</td>
<td>user: x/xn</td>
</tr>
<tr>
<td>UXN/XN</td>
<td>0/1</td>
<td>user: x/xn</td>
</tr>
<tr>
<td>PXN</td>
<td>0</td>
<td>user: no effect</td>
</tr>
<tr>
<td></td>
<td></td>
<td>kernel: no effect</td>
</tr>
</tbody>
</table>
Option #1: Page tables

<table>
<thead>
<tr>
<th>CPU Mode</th>
<th>Memory</th>
<th>AP[1]</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Option #1: Page tables

## User Memory Accesses

<table>
<thead>
<tr>
<th>CPU Mode</th>
<th>Memory</th>
<th>AP[1]</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>EL0</td>
<td>User</td>
<td>1</td>
<td>access</td>
</tr>
<tr>
<td>EL0</td>
<td>Kernel</td>
<td>0</td>
<td>no access</td>
</tr>
</tbody>
</table>
## Option #1: Page tables

### User Memory Accesses

<table>
<thead>
<tr>
<th>CPU Mode</th>
<th>Memory</th>
<th>AP[1]</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>EL0</td>
<td>User</td>
<td>1</td>
<td>access</td>
</tr>
<tr>
<td>EL0</td>
<td>Kernel</td>
<td>0</td>
<td>no access</td>
</tr>
</tbody>
</table>

### Kernel Memory Accesses

<table>
<thead>
<tr>
<th>CPU Mode</th>
<th>Memory</th>
<th>AP[1]</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>EL2</td>
<td>User</td>
<td>1</td>
<td>access</td>
</tr>
<tr>
<td>EL2</td>
<td>Kernel</td>
<td>0</td>
<td>access</td>
</tr>
</tbody>
</table>
Option #1: RES1

- ARMv8.0 hardware must treat non-register RES1 bits as:
  
  reads-as-written with no effect on the behaviour of the CPU
## Option #1: Page tables

<table>
<thead>
<tr>
<th>Descriptor bit</th>
<th>User entries</th>
<th>Kernel entries</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AP[2]</strong></td>
<td>1/0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>user: r/rw</td>
<td>user: none</td>
</tr>
<tr>
<td></td>
<td>kernel: r/rw</td>
<td>kernel r/rw</td>
</tr>
<tr>
<td><strong>AP[1]</strong></td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>user: x/xn</td>
<td>user: x</td>
</tr>
<tr>
<td></td>
<td>kernel: x/xn</td>
<td>kernel: x</td>
</tr>
<tr>
<td><strong>UXN/XN</strong></td>
<td>0/1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>user: x/xn</td>
<td>user: x</td>
</tr>
<tr>
<td></td>
<td>kernel: x/xn</td>
<td>kernel: x</td>
</tr>
<tr>
<td><strong>PXN</strong></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>user: no effect</td>
<td>user: no effect</td>
</tr>
<tr>
<td></td>
<td>kernel: no effect</td>
<td>kernel: no effect</td>
</tr>
</tbody>
</table>
### Option #1: Page tables

<table>
<thead>
<tr>
<th>Descriptor bit</th>
<th>EL1/0</th>
<th>EL2</th>
</tr>
</thead>
<tbody>
<tr>
<td>AP[1]</td>
<td>User access</td>
<td>RES1</td>
</tr>
<tr>
<td>UXN/XN</td>
<td>UXN</td>
<td>XN</td>
</tr>
<tr>
<td>PXN</td>
<td>PXN</td>
<td>RES0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Descriptor bit</th>
<th>User entries</th>
<th>Kernel entries</th>
</tr>
</thead>
<tbody>
<tr>
<td>AP[2]</td>
<td>1/0</td>
<td>1/0 user: none</td>
</tr>
<tr>
<td></td>
<td></td>
<td>kernel: r/rw</td>
</tr>
<tr>
<td>AP[1]</td>
<td>1</td>
<td>0 user: x</td>
</tr>
<tr>
<td></td>
<td></td>
<td>kernel: x</td>
</tr>
<tr>
<td>UXN/XN</td>
<td>0/1</td>
<td>0 user: no</td>
</tr>
<tr>
<td></td>
<td></td>
<td>effect</td>
</tr>
<tr>
<td>PXN</td>
<td>0</td>
<td>0 user: no</td>
</tr>
<tr>
<td></td>
<td></td>
<td>effect</td>
</tr>
</tbody>
</table>
## Option #1: Page tables

<table>
<thead>
<tr>
<th>Descriptor bit</th>
<th>EL1/0</th>
<th>EL2</th>
</tr>
</thead>
<tbody>
<tr>
<td>AP[1]</td>
<td>User access</td>
<td>RES1</td>
</tr>
<tr>
<td>UXN/XN</td>
<td>UXN</td>
<td>XN</td>
</tr>
<tr>
<td>PXN</td>
<td>PXN</td>
<td>RES0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Descriptor bit</th>
<th>User entries</th>
<th>Kernel entries</th>
</tr>
</thead>
<tbody>
<tr>
<td>AP[2]</td>
<td>1/0</td>
<td>1/0 user: none</td>
</tr>
<tr>
<td></td>
<td></td>
<td>kernel r/rw</td>
</tr>
<tr>
<td>AP[1]</td>
<td>0/1</td>
<td>0</td>
</tr>
<tr>
<td>UXN/XN</td>
<td>user: x/xn</td>
<td>user: x</td>
</tr>
<tr>
<td></td>
<td>kernel: x/xn</td>
<td>kernel: x</td>
</tr>
<tr>
<td>PXN</td>
<td>0</td>
<td>user: no effect</td>
</tr>
<tr>
<td></td>
<td>kernel: no effect</td>
<td>user: no effect</td>
</tr>
</tbody>
</table>
Option #1: TLB Invalidation

Kernel accesses to userspace

<table>
<thead>
<tr>
<th>Mode</th>
<th>ASID</th>
<th>VA</th>
<th>PA</th>
</tr>
</thead>
</table>

TLB

EL1 Kernel

task #1

task #2
Option #1: TLB Invalidation

Kernel accesses to userspace

<table>
<thead>
<tr>
<th>Mode</th>
<th>ASID</th>
<th>VA</th>
<th>PA</th>
</tr>
</thead>
<tbody>
<tr>
<td>EL1</td>
<td>2</td>
<td>0xf000</td>
<td>0xba000</td>
</tr>
</tbody>
</table>
Option #1: TLB Invalidation

Kernel accesses to userspace

<table>
<thead>
<tr>
<th>Mode</th>
<th>ASID</th>
<th>VA</th>
<th>PA</th>
</tr>
</thead>
<tbody>
<tr>
<td>EL1</td>
<td>2</td>
<td>0xf000</td>
<td>0xba000</td>
</tr>
<tr>
<td>EL1</td>
<td>45</td>
<td>0xf000</td>
<td>0xbc000</td>
</tr>
</tbody>
</table>
Option #1: TLB Invalidation

Kernel accesses to userspace

TLB

<table>
<thead>
<tr>
<th>Mode</th>
<th>ASID</th>
<th>VA</th>
<th>PA</th>
</tr>
</thead>
</table>

EL2 Kernel

task #1
task #2
Option #1: TLB Invalidation

Kernel accesses to userspace

<table>
<thead>
<tr>
<th>Mode</th>
<th>ASID</th>
<th>VA</th>
<th>PA</th>
</tr>
</thead>
<tbody>
<tr>
<td>EL2</td>
<td>-</td>
<td>0xf000</td>
<td>0xba000</td>
</tr>
</tbody>
</table>
Option #1: TLB Invalidation

Kernel accesses to userspace

<table>
<thead>
<tr>
<th>Mode</th>
<th>ASID</th>
<th>VA</th>
<th>PA</th>
</tr>
</thead>
<tbody>
<tr>
<td>EL2</td>
<td>-</td>
<td>0xf000</td>
<td>0xba000</td>
</tr>
</tbody>
</table>

TLB

Hit

task #1

task #2
Option #2:

- **Kernel**
  - Address space: 0x7f ffffffff
  - Problem: How to access userspace (copy_from_user)

- **Userspace**
  - Address space: 0x7f ffffffff
  - Benefit: No EL2 TLB invalidation
  - Benefit: No address space compression
  - Problem: How to access userspace (copy_from_user)
Option #2

Accessing userspace

- get_user_pages walks page tables in software
- access memory via linear address
- real code gets messy
- this is slow!
Option #2

Potential Improvement with AT instruction

```c
user_va_to_phys(va) {
    asm("at s1e1r, %0": "r" (va));
    return par_to_phys(read_par());
}

while true {
    pa = user_va_to_phys(va);
    page = phys_to_page(pa);
    get_page(page);
    _pa = user_va_to_phys(va);
    if (pa == _pa)
        break;
    put_page(page);
}
```
Option #2

Potential Improvement with AT instruction

```c
user_va_to_phys(va)
{
    asm("at s1e1r,
        return par_to_phys(read_par());
}

while true {
    pa = user_va_to_phys(va);
    page = phys_to_page(pa);
    get_page(page);
    _pa = user_va_to_phys(va);
    if (pa == _pa)
        break;
    put_page(page);
}
```
Memory in EL2

Option #1

Benefit: No changes to uaccess
Implementation

- System Registers
- Memory
- Exceptions
- KVM
Exceptions to EL1

EL0

Exceptions from userspace

EL1

User

Kernel

Exceptions from kernel
Exceptions to EL2

EL0

EL1

Exceptions from userspace

EL2

User

Kernel

Exceptions from kernel
Handling IRQs to EL2

- HCR_EL2.IMO: Traps IRQs to EL2
Routing Synchronous Exceptions to EL2

- **HCR_EL2.TGE**: Traps general exceptions to EL2

- Does NOT work because TGE implies SCTLR_EL1.M=0
Routing Synchronous Exceptions to EL2

- Program TTBR1_EL1 with simple pgtable
- Set VBAR_EL1 to address of highest page in VA space
- Implement EL1 exception vector handler
EL1 Shim Handler

```
ventry el1_shim_sync_invalid
ventry el1_shim_irq_invalid
ventry el1_shim_fiq_invalid
ventry el1_shim_error_invalid
ventry el1_shim_sync_invalid
ventry el1_shim_irq_invalid
ventry el1_shim_fiq_invalid
ventry el1_shim_error_invalid
ventry el0_shim_sync
ventry el0_shim_irq
ventry el0_shim_irq_invalid
ventry el0_shim_fiq_invalid
ventry el0_shim_error_invalid
```

```
el0_shim_sync:
  hvc  #0

el0_shim_irq:
  hvc  #1
```
Changes to entry.S

- Exception syndrome and return address are stored in ESR_EL1 and SPSR_EL1
- CONFIG_EL2_KERNEL modifies exception handler to read _EL2 registers
- Check for hypercalls from the EL1 shim and read information from EL1 registers in this case
Exceptions Summary

• Expect low overhead from:
  • one extra exception going through the EL1 shim
  • a few extra assembly instructions in entry.S
Gotcha: Interrupts in EL1

Two instructions in EL1:

```
b el0_shim_sync
```

```
e10_shim_sync:
hvc  #0
```

On exception entry:

- PSTATE.I = 1
- HCR_EL2.IMO==1:
  - masks virtual IRQs
Gotcha: Interrupts in EL1

Two instructions in EL1:

```
b el0_shim_sync
```

```
el0_shim_sync:
hvc #0
```

On exception entry:
- PSTATE.I = 1
- HCR_EL2.IMO==1: masks virtual IRQs

Physical IRQ
Gotcha: solution

clear HCR_EL2.IMO  
set HCR_EL2.IMO

EL0
User

EL1
shim

EL2
Kernel

IRQ
Implementation

- System Registers
- Memory
- Exceptions
- KVM
Changes to KVM

• Rely on VHE code for functionality (tiny tweaks)

• ARM64_HAS_VIRT_HOST_EXTN
  • Currently just checks current CPU mode
  • Change to check VHE is enabled and in EL2

• Have separate ARM64_RUNS_IN_EL2

• Optimizations
KVM Optimizations

• Essentially lazy save/restore of registers
• Move register save/restore to vcpu_put/vcpu_load
• Only touch the vgic when you really have to
• Avoid saving/restoring timer state on every exit
Challenges

Performance?
Challenges

#ifndef CONFIG_EL2_KERNEL
foo
#else
bar
#endif
Challenges

Upstream?
Challenges

Upstream?

Dead Code?
Challenges

Upstream?

Dead Code?

Maintenance Burden?
# Implementation Complexity

<table>
<thead>
<tr>
<th>File Path</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>arch/arm/include/asm/kvm_host.h</td>
<td>4 +</td>
</tr>
<tr>
<td>arch/arm/kvm/arm.c</td>
<td>7 ++</td>
</tr>
<tr>
<td>arch/arm64/Kconfig</td>
<td>13 +++</td>
</tr>
<tr>
<td>arch/arm64/include/asm/cpufeature.h</td>
<td>3 +--</td>
</tr>
<tr>
<td>arch/arm64/include/asm/esr.h</td>
<td>2 +--</td>
</tr>
<tr>
<td>arch/arm64/include/asm/kernel-pgtable.h</td>
<td>5 +</td>
</tr>
<tr>
<td>arch/arm64/include/asm/kvm_hyp.h</td>
<td>5 +</td>
</tr>
<tr>
<td>arch/arm64/include/asm/memory.h</td>
<td>15 ++--</td>
</tr>
<tr>
<td>arch/arm64/include/asm/mmu_context.h</td>
<td>23 ++++</td>
</tr>
<tr>
<td>arch/arm64/include/asm/percpu.h</td>
<td>9 ++</td>
</tr>
<tr>
<td>arch/arm64/include/asm/pgtable-hwdef.h</td>
<td>5 +</td>
</tr>
<tr>
<td>arch/arm64/include/asm/pgtable.h</td>
<td>12 +--</td>
</tr>
<tr>
<td>arch/arm64/include/asm/suspend.h</td>
<td>4 +</td>
</tr>
<tr>
<td>arch/arm64/include/asm/tlbflush.h</td>
<td>32 ++++--</td>
</tr>
<tr>
<td>arch/arm64/include/asm/virt.h</td>
<td>52 +++++</td>
</tr>
<tr>
<td>arch/arm64/kernel/Makefile</td>
<td>1 +</td>
</tr>
<tr>
<td>arch/arm64/kernel/cpufeature.c</td>
<td>10 ++</td>
</tr>
<tr>
<td>arch/arm64/kernel/ell-vectors.c</td>
<td>107 ++++++</td>
</tr>
<tr>
<td>arch/arm64/kernel/entry.S</td>
<td>195 ++++++</td>
</tr>
<tr>
<td>arch/arm64/kernel/head.S</td>
<td>150 ++++++</td>
</tr>
<tr>
<td>arch/arm64/kernel/setup.c</td>
<td>22 +++</td>
</tr>
<tr>
<td>arch/arm64/kernel/smp.c</td>
<td>6 +</td>
</tr>
<tr>
<td>arch/arm64/kernel/traps.c</td>
<td>6 +</td>
</tr>
<tr>
<td>arch/arm64/kernel/vmlinux.lds.S</td>
<td>9 ++</td>
</tr>
<tr>
<td>arch/arm64/kvm/hyp.S</td>
<td>5 +</td>
</tr>
<tr>
<td>arch/arm64/kvm/hyp/switch.c</td>
<td>21 +++--</td>
</tr>
<tr>
<td>arch/arm64/mm/init.c</td>
<td>2 +</td>
</tr>
<tr>
<td>arch/arm64/mm/mmu.c</td>
<td>2 +</td>
</tr>
<tr>
<td>arch/arm64/mm/pgd.c</td>
<td>23 +++--</td>
</tr>
<tr>
<td>arch/arm64/mm/proc.S</td>
<td>69 +++++--</td>
</tr>
<tr>
<td>drivers/clocksource/arm_arch_timer.c</td>
<td>2 ++</td>
</tr>
</tbody>
</table>

31 files changed, 792 insertions(+) 29 deletions(-)
It works

EL2 Kernel boot CPU started in EL2
CPU features: detected feature: Runs at EL2
CPU: All CPU(s) started at EL2
Conclusions

• It is possible to run Linux in EL2

• It’s a bit invasive, but not overly so

• Potential performance benefits, remains to be measured

• To Upstream or not to Upstream; that’s the question