2023 秋季训练营
Hypervisor虚拟化

AARCH64 Hypercraft虚拟化实现

唐诗美

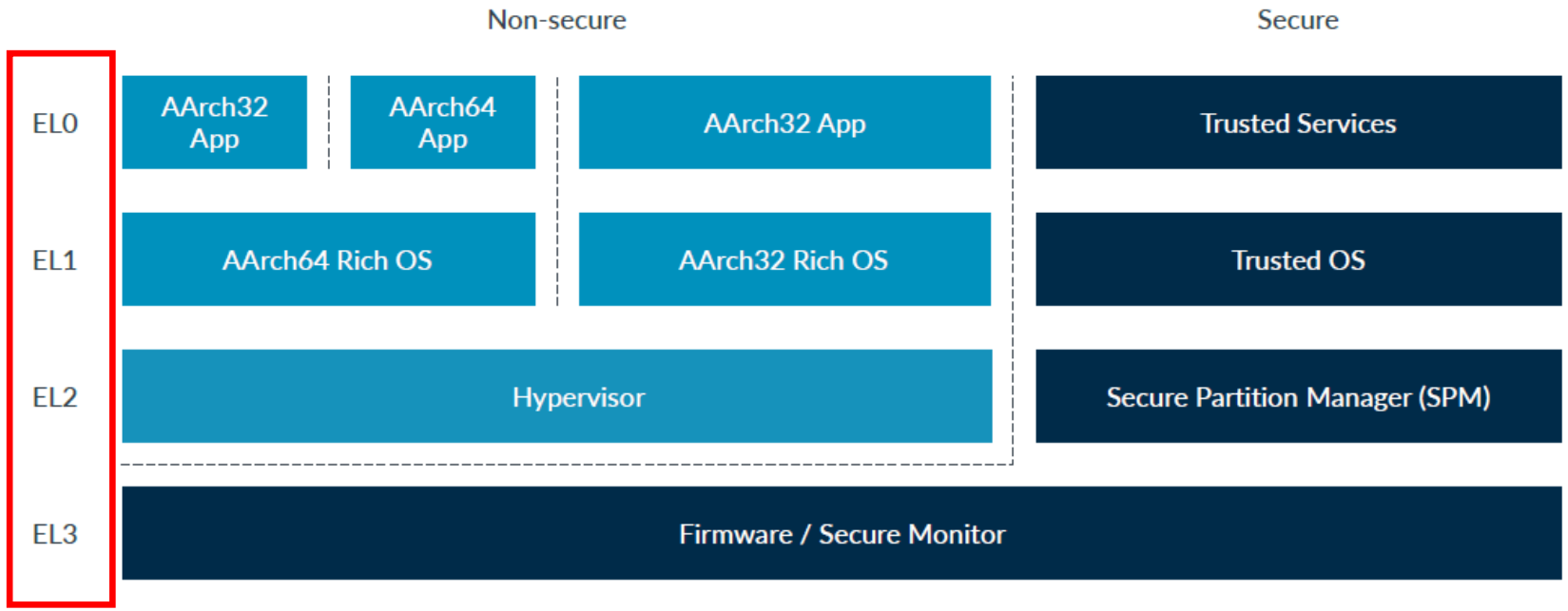
2023.11

整体内容

- AARCH64虚拟化背景知识
- AARCH64 Hypercraft整体概览
- AARCH64 Hypercraft代码解读

AARCH64虚拟化背景知识

AARCH64特权级别

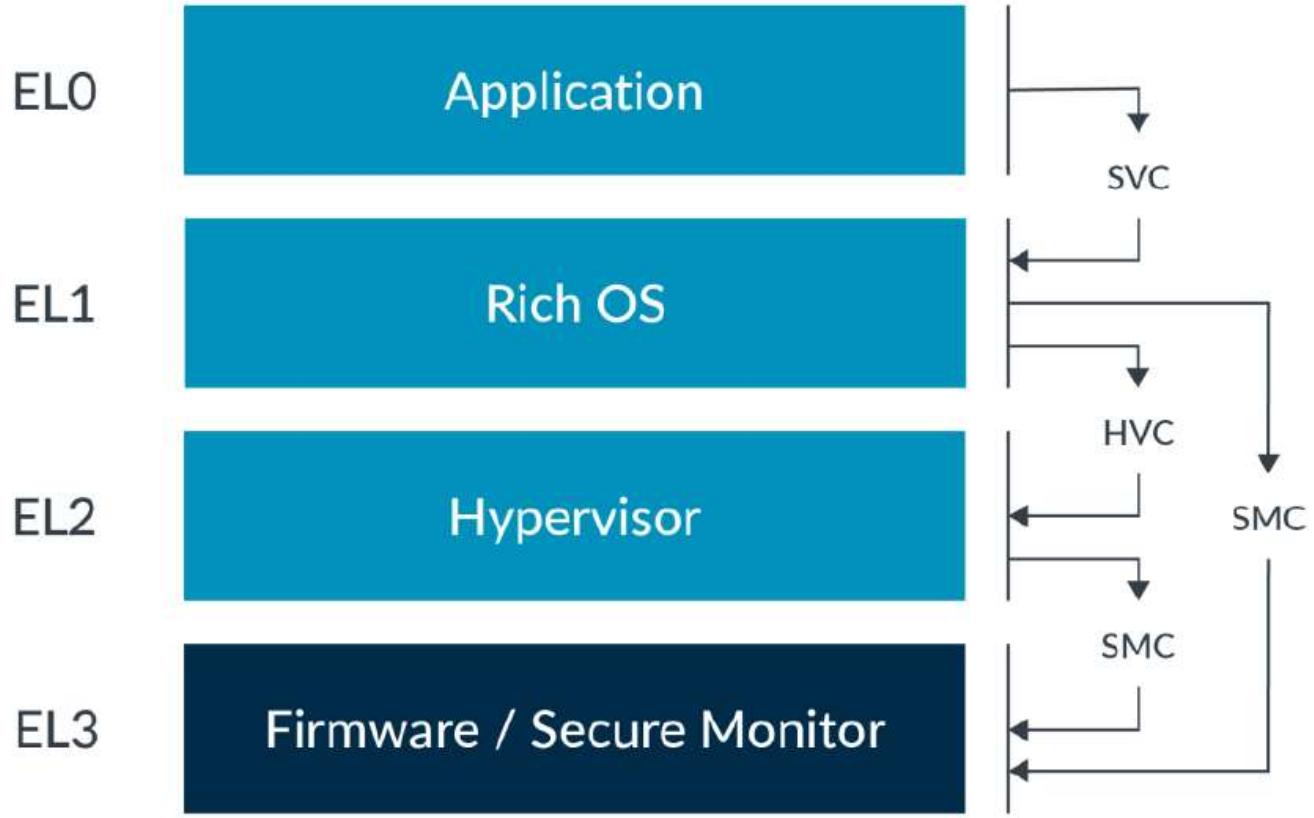


EL0（用户态）： Applications.
EL1（内核态）： OS kernel and associated functions that are typically described as privileged.

EL2： Hypervisor.
EL3： Secure monitor.

AARCH64虚拟化背景知识

AARCH64特权级别切换



- SVC：用于触发一个Supervisor Call Exception，使在EL0特权级的用户程序能够请求EL1特权级的操作系统服务。
- **HVC：用于触发一个Hypervisor Call Exception，使在EL1特权级的操作系统能够请求在EL2特权级的虚拟监控程序提供的服务。**
- SMC：用于触发一个Secure Monitor Call Exception，使在正常世界（Normal world）能够从EL3特权级的固件请求安全世界（Secure world）提供的服务。
- **ERET：用于异常返回，从当前异常等级返回触发异常前的异常等级。**

AARCH64虚拟化背景知识

AARCH64内存虚拟化

Address translations when EL3 is using AArch64

Secure EL3	VA	Secure EL3 stage 1		PA, Secure or Non-secure
Secure EL1&0	VA	Secure EL1&0 stage 1		PA, Secure or Non-secure
Non-secure EL2	VA	Non-secure EL2 stage 1		PA, Non-secure only
Non-secure EL1&0	VA	Non-secure EL1&0 stage 1	IPA	Non-secure EL1&0 stage 2 PA, Non-secure only

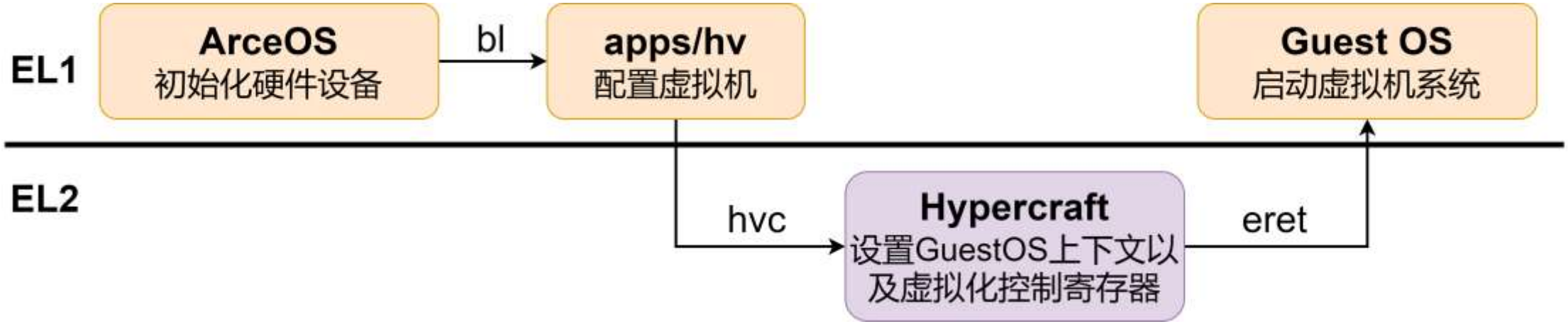
AARCH64虚拟化背景知识

- ARMv8的寄存器
  - 通用寄存器（x0~x30，31个，不包含sp）
  - PSTATE（Process state，程序状态信息的集合，是一组寄存器）
    - CurrentEL、DAIF、SPSel、SP_ELx、SPSR_ELx
  - 系统寄存器
    - HCR_EL2：Hypervisor Configuration Register
    - SCTLR_ELx：System Control Register
    - TTBR0_ELx，TTBR1_EL1：Translation Table Base Register
    - TCR_ELx：Translation Control Register
    - VTTBR_EL2, Virtualization Translation Table Base Register
    - VTCR_EL2, Virtualization Translation Control Register
    - VBAR_ELx：Vector Base Address Register
  - Float Point寄存器（与虚拟化无关）

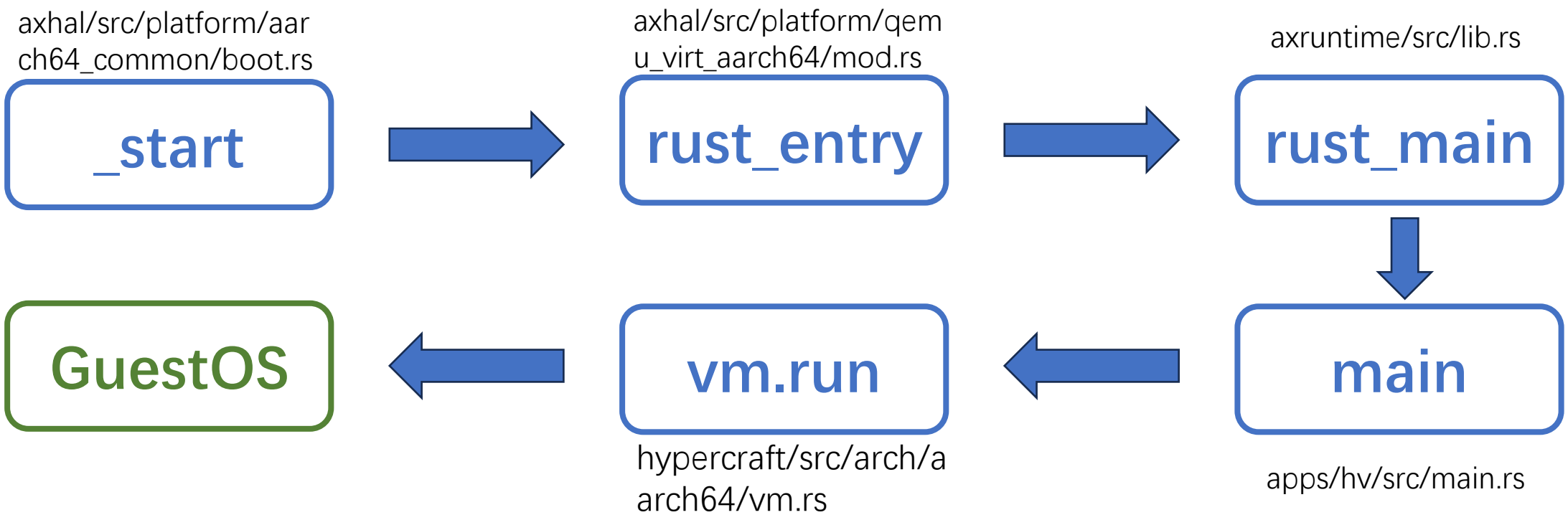
AARCH64 Hypercraft整体概览

- 基于EL2的异常处理
- 虚拟CPU（VCPU）
- 两阶段页表翻译
- 基于设备树的设备透传

目前可运行nimbos与linux
(具体编译方法参见文档)



AARCH64 Hypercraft整体概览



_start: 初始化部分寄存器，开启页表翻译，EL2切换为EL1。

rust_entry: 继续进行初始化工作，如时钟等。

rust_main: 进行内核的初始化工作。

main: 初始化hypervisor与虚拟机的相关设置并启动运行虚拟机。

vm.run: 启动虚拟机，使系统跳转到GuestOS执行。

AARCH64 Hypercraft代码解读

- _start (modules/axhal/src/platform/aarch64_common/boot.rs)

设置EL2的异常向量表

设置EL2页表

```
// set vbar_el2 for hypervisor.
#[cfg(feature = "hv")]
core::arch::asm!("
    ldr x8, ={exception_vector_base_el2}    // setup vbar_el2 for hypervisor
    msr vbar_el2, x8

    mrs     x19, mpidr_el1
    and     x19, x19, #0xffffff     // get current CPU id
    mov     x20, x0                 // save DTB pointer
    adrp    x8, {boot_stack}        // setup boot stack
    add     x8, x8, {boot_stack_size}
    mov     sp, x8

    bl      {init_boot_page_table}
    bl      {init_mmu_el2}
    bl      {init_mmu}              // setup MMU
    bl      {switch_to_el1}         // switch to EL1
    bl      {enable_fp}             // enable fp/neon
```

AARCH64 Hypercraft代码解读

- main (apps/hv/src/main.rs)

设置guest page table

初始化VCPU上下文

启动虚拟机

```rust
// boot cpu
PerCpu::<HyperCraftHalImpl>::init(0, 0x4000);   // change to pub const CPU_STACK_SIZE: usize
= PAGE_SIZE * 128?

// get current percpu
let pcpu = PerCpu::<HyperCraftHalImpl>::this_cpu();

// create vcpu, need to change addr for aarch64!
let gpt = setup_gpm(0x7000_0000, 0x7020_0000).unwrap();
let vcpu: VmxVcpu<{unknown}> = pcpu.create_vcpu(0).unwrap();
let mut vcpus: VmCpus<{unknown}> = VmCpus::new();

// add vcpu into vm
vcpus.add_vcpu(vcpu).unwrap();
let mut vm: VM<HyperCraftHalImpl, GuestPageTable> = VM::new(vcpus, gpt, 0).unwrap();
vm.init_vm_vcpu(0, 0x7020_0000, 0x7000_0000);

info!("vm run cpu{}", hart_id);
// suppose hart_id to be 0
vm.run(0);
```

AARCH64 Hypercraft代码解读

- setup_gpm (apps/hv/src/main.rs)

创建页表

解析dtb

```
pub fn setup_gpm(dtb: usize, kernel_entry: usize) -> Result<GuestPageTable> {
    let mut gpt: GuestPageTable = GuestPageTable::new()?;
    let meta = MachineMeta::parse(dtb);
```

进行设备内存映射

```
if let Some(pl011) = meta.pl011 {
    gpt.map_region(
        gpa: pl011.base_address,
        hpa: pl011.base_address,
        pl011.size,
        flags: MappingFlags::READ | MappingFlags::WRITE | MappingFlags::USER,
    )?;
}
```

dtb位于apps/hv/src/guest/linux/linux-aarch64.dtb

AARCH64 Hypercraft代码解读

- guest页表相关结构
  - crate/hypercraft/src/arch/aarch64/ept.rs

```rust
use page_table::{PageTable64, PagingMetaData};
use page_table_entry::aarch64::A64PTE;

/// Metadata of AArch64 hypervisor page tables (ipa to hpa).
#[derive(Copy, Clone)]
pub struct A64HVPagingMetaData;

impl PagingMetaData for A64HVPagingMetaData {
    const LEVELS: usize = 3;
    const PA_MAX_BITS: usize = 48;  // In Armv8.0-A, the maximum size for a physical address is 48
    bits.
                                    // The size of the IPA space can be configured in the same way
                                    as the
    const VA_MAX_BITS: usize = 40;  //  virtual address space. VTCR_EL2.T0SZ controls the size.
}
/// According to rust shyper, AArch64 translation table.
pub type NestedPageTable<I> = PageTable64<A64HVPagingMetaData, A64PTE, I>;
```

AARCH64 Hypercraft代码解读

- guest页表接口
  - modules/axruntime/src/gpm.rs

```rust
pub type GuestPagingIfImpl = axhal::paging::PagingIfImpl;

/// Guest Page Table struct\
2 implementations
pub struct GuestPageTable(NestedPageTable<GuestPagingIfImpl>);

impl GuestPageTableTrait for GuestPageTable {
    fn new() -> HyperResult<Self> {…

    fn map(…
    ) -> HyperResult<()> {…

    fn map_region(…
    ) -> HyperResult<()> {…

    fn unmap(&mut self, gpa: GuestPhysAddr) -> HyperResult<()> {…

    fn translate(&self, gpa: GuestPhysAddr) -> HyperResult<hypercraft::HostPhysAddr> {…

    fn token(&self) -> usize {…
} impl GuestPageTableTrait for GuestPageTable
```

AARCH64 Hypercraft代码解读

- vm.init_vm_vcpu (crates/hypercraft/src/arch/aarch64/vm.rs)

```rust
/// Init VM vcpu by vcpu id. Set kernel entry point.
pub fn init_vm_vcpu(&mut self, vcpu_id:usize, kernel_entry_point: usize, device_tree_ipa: usize)
{
    let vcpu = self.vcpus.get_vcpu(vcpu_id).unwrap();
    vcpu.init(kernel_entry_point, device_tree_ipa);
}
```

```rust
/// Init Vcpu registers
pub fn init(&mut self, kernel_entry_point: usize, device_tree_ipa: usize) {
    self.vcpu_arch_init(kernel_entry_point, device_tree_ipa);
    self.init_vm_context();
}
```

(crates/hypercraft/src/arch/aarch64/vcpu.rs)

AARCH64 Hypercraft代码解读

- vcpu.VmCpuRegisters (crates/hypercraft/src/arch/aarch64/vcpu.rs)

```rust
#[repr(C)]
#[derive(Clone, Debug)]
pub struct VmCpuRegisters {
    /// guest trap context
    pub guest_trap_context_regs: ContextFrame,
    /// arceos context
    pub save_for_os_context_regs: ContextFrame,
    /// virtual machine system regs setting
    pub vm_system_regs: VmContext,
}
```

```rust
#[repr(C)]
#[derive(Copy, Clone, Debug)]
pub struct Aarch64ContextFrame {
    pub gpr: [u64; 31],
    pub sp: u64,
    pub elr: u64,
    pub spsr: u64,
}
```

(crates/hypercraft/src/arch/aarch64/context_frame.rs)

AARCH64 Hypercraft代码解读

- vcpu.vcpu_arch_init (crates/hypercraft/src/arch/aarch64/vcpu.rs)

```rust
/// Init guest contextFrame
fn vcpu_arch_init(&mut self, kernel_entry_point: usize, device_tree_ipa: usize) {
    self.set_gpr(0, device_tree_ipa);
    self.set_elr(kernel_entry_point);
    self.regs.guest_trap_context_regs.spsr =( SPSR_EL1::M::EL1h +
                                              SPSR_EL1::I::Masked +
                                              SPSR_EL1::F::Masked +
                                              SPSR_EL1::A::Masked +
                                              SPSR_EL1::D::Masked )
                                              .value;
}
```

https://docs.rs/cortex-a/latest/cortex_a/registers/index.html

AARCH64 Hypercraft代码解读

- vcpu.init_vm_context(crates/hypercraft/src/arch/aarch64/vcpu.rs)

```rust
/// Init guest context. Also set some el2 register value.
fn init_vm_context(&mut self) {
    self.regs.vm_system_regs.cntvoff_el2 = 0;
    self.regs.vm_system_regs.sctlr_el1 = 0x30C50830;
    self.regs.vm_system_regs.cntkctl_el1 = 0;
    self.regs.vm_system_regs.pmcr_el0 = 0;
    // self.regs.vm_system_regs.vtcr_el2 = 0x8001355c;
    self.regs.vm_system_regs.vtcr_el2 = (VTCR_EL2::PS::PA_40B_1TB   //0b001 36 bits, 64GB.
                                         + VTCR_EL2::TG0::Granule4KB
                                         + VTCR_EL2::SH0::Inner
                                         + VTCR_EL2::ORGN0::NormalWBRAWA
                                         + VTCR_EL2::IRGN0::NormalWBRAWA
                                         + VTCR_EL2::SL0.val(0b01)
                                         + VTCR_EL2::T0SZ.val(64 - 40)).into();
    //self.regs.vm_system_regs.hcr_el2 = 0x80000001;  // Maybe we do not need smc setting?
    passthrough gic.
    self.regs.vm_system_regs.hcr_el2 = (HCR_EL2::VM::Enable
                                        + HCR_EL2::RW::EL1IsAarch64).into();
    let mut vmpidr = 0;
    vmpidr |= 1 << 31;
    vmpidr |= self.vcpu_id;
    self.regs.vm_system_regs.vmpidr_el2 = vmpidr as u64;

    // self.gic_ctx_reset(); // because of passthrough gic, do not need gic context anymore?
}
```

AARCH64 Hypercraft代码解读

- vm.run (crates/hypercraft/src/arch/aarch64/vm.rs)

```rust
/// Run this VM.
pub fn run(&mut self, vcpu_id: usize) {
    let vcpu = self.vcpus.get_vcpu(vcpu_id).unwrap();
    let vttbr_token = (self.vm_id << 48) | self.gpt.token();
    debug!("vttbr_token: 0x{:X}", self.gpt.token());
    vcpu.run(vttbr_token);
}
```

```rust
/// Run this vcpu
pub fn run(&self, vttbr_token: usize) {
    _ = run_guest_by_trap2el2(vttbr_token, self.vcpu_ctx_addr());
}
```

(crates/hypercraft/src/arch/aarch64/vcpu.rs)

(crates/hypercraft/src/arch/aarch64/hvc.rs)

```rust
pub fn run_guest_by_trap2el2(token:
usize, regs_addr: usize) -> usize {
    // mode is in x7. hvc_type:
    HVC_SYS; event: HVC_SYS_BOOT
    hvc_call(token, regs_addr, 0, 0,
    0, 0, 0, 0)
}
```

AARCH64 Hypercraft代码解读

- hvc_call (crates/hypercraft/src/arch/aarch64/hvc.rs)
  - 调用hvc指令，使执行流触发异常，跳转到EL2对应的异常向量表处理。
- exception_vector_base_el2 (modules/axhal/src/arch/aarch64/trap_el2.S)
  - 处理流程：
    - 硬件处理部分（执行HVC）
      - 设置异常处理的上下文（如寄存器ELR_ELx、SPSR等）
    - 软件处理部分
      - 保存现场（上下文）
      - 根据异常类型执行对应的异常处理函数（此处HVC对应定义在hypercraft中的lower_aarch64_synchronous）
      - 恢复现场 (上下文)
      - ERET
    - 硬件处理部分（执行ERET）
      - 令下一条指令跳转到ELR_ELx对应的地址执行

AARCH64 Hypercraft代码解读

- lower_aarch64_synchronous (crates/hypercraft/src/arch/aarch64/exception.rs)

```rust
/// deal with lower aarch64 synchronous exception
#[no_mangle]
pub extern "C" fn lower_aarch64_synchronous(ctx: &mut ContextFrame)
{
    info!("lower_aarch64_synchronous exception class:0x{:X}",
    exception_class());
    // current_cpu().set_context_addr(ctx);

    match exception_class() {
        0x24 => {
            // info!("Core[{}] data_abort_handler", cpu_id());
            data_abort_handler(ctx);
        }
        0x16 => {
            hvc_handler(ctx);
        }
```

AARCH64 Hypercraft代码解读

- hvc_handler (crates/hypercraft/src/arch/aarch64/sync.rs)
  - 流程
    - 解析HVC_TYPE与HVC_EVENT
    - 调用hvc_guest_handler(crates/hypercraft/src/arch/aarch64/hvc.rs)根据传入的HVC_TYPE参数调用对应的handler
    - 调用HVC_TYPE对应的handler，根据传入的HVC_EVENT执行具体操作。以当前实现的HVC_SYS_BOOT事件为例，最终会调用实现定义在hvc.rs中的init_hv函数
  - **注意：在if hvc_type == HVC_SYS && event == HVC_SYS_BOOT这个条件语句中，会保存目前arceos触发这个异常时的寄存器，同时会把当前栈上的上下文覆盖为guest trap context。因为当异常处理完毕用eret返回时，我们需要直接跳转到guest kernel entry执行，所以此处需要将原本的上下文修改为之前vcpu初始化的guest上下文**

AARCH64 Hypercraft代码解读

- init_hv (crates/hypercraft/src/arch/aarch64/hvc.rs)

参数:
guest 页表基址,
vm上下文地址

初始化虚拟化相关寄存器

初始化虚拟机内部寄存器

```rust
#[inline(never)]
/// hvc handler for initial hv
/// x0: root_paddr, x1: vm regs context addr
fn init_hv(root_paddr: usize, vm_ctx_addr: usize) {
    // cptr_el2: Condtrols trapping to EL2 for accesses to the CPACR, Trace functionality
    //           an registers associated with floating-point and Advanced SIMD execution.
    unsafe {
        core::arch::asm!("
            mov x3, xzr           // Trap nothing from EL1 to El2.
            msr cptr_el2, x3"
        );
    }
    msr!(VTTBR_EL2, root_paddr);
    unsafe {
        core::arch::asm!("
            tlbi    alle2         // Flush tlb
            dsb nsh
            isb"
        );
    }

    let regs: &VmCpuRegisters = unsafe{core::mem::transmute(vm_ctx_addr)};
    // set vm system related register
    regs.vm_system_regs.ext_regs_restore();
}
```

Q&A