



RISC-V 多核启动原理介绍

杨德睿·启元实验室

2022/08

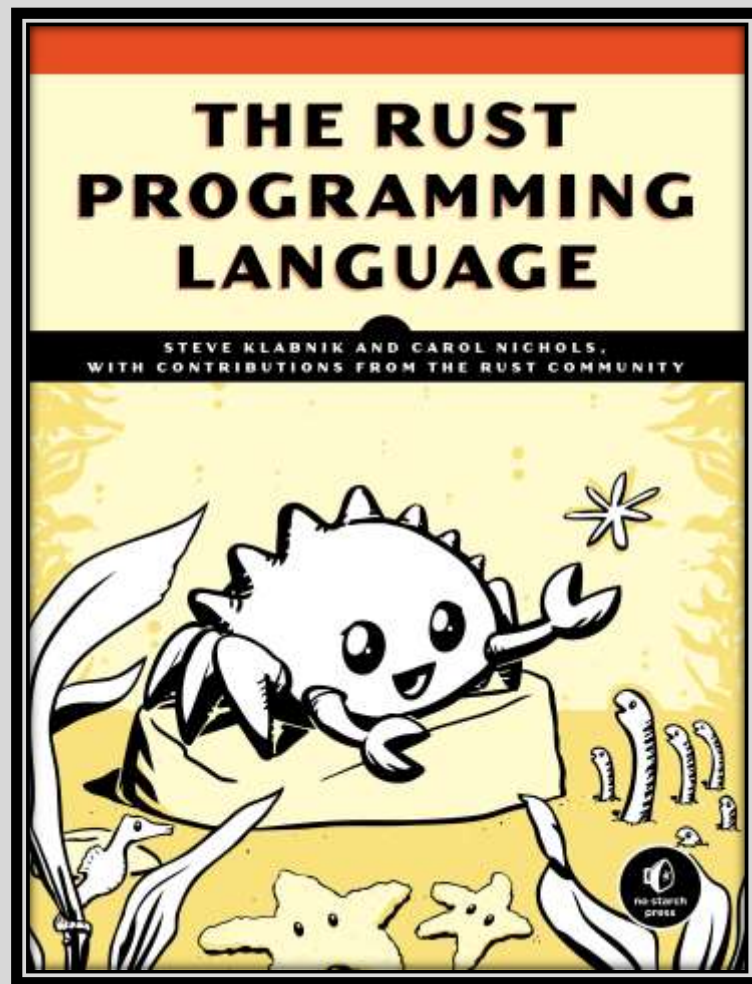
- 北京交通大学通信工程本科
- 多年从事移动机器人软件开发
- 2022 年 3 月入职启元实验室开发 zCore
 - 1 年 Rust 开发经验（做机器人算法实现和驱动开发）
 - 没有操作系统/计算机/RISC-V 经验

自我介绍

我们都一样！
(重在交流)

大纲·TOC

- **zCore 启动流程回顾**
 - 分配启动栈
 - 构造地址空间
 - 多核启动
 - 内核硬件线程用法展望
- **RustSBI-Qemu 多核环境**
 - Qemu-virt ROM 多核环境
 - SBI HSM 扩展支持
 - SBI 多核支持展望



zCore 启动流程

栈

栈是使用高级语言的前提，通常是汇编首先要做的事

地址空间

地址有关程序必须工作在链接地址 (RISC-V Rust 只能地址有关编译)

外设

IO、存储、中断
(MMIO外设也要映射到内核空间)

分配启动栈

```
9 > /// 内核入口。...
14 #[naked]
15 #[no_mangle]
16 #[link_section = ".text.entry"]
17 0 references
18 unsafe extern "C" fn _start(hartid: usize, device_tree_paddr: usize) → ! {
19     asm!(
20         "csrw sie, zero", // 关中断
21         "call {select_stack}", // 设置启动栈
22         "j {main}", // 进入 rust
23         select_stack = sym select_stack,
24         main = sym primary_rust_main,
25         options(noreturn)
26     )
27 }
28 > /// 副核入口。此前副核被 bootloader/see 阻塞。 now, 从清理。 cleanup。 ...
33 #[naked]
34 1 reference
35 unsafe extern "C" fn secondary_hart_start(hartid: usize) → ! {
36     asm!(
37         "csrw sie, zero", // 关中断
38         "call {select_stack}", // 设置启动栈
39         "j {main}", // 进入 rust
40         select_stack = sym select_stack,
41         main = sym secondary_rust_main,
42         options(noreturn)
43     )
44 }
```

```
98 /// 根据硬件线程号设置启动栈。
99 ///
100 /// # Safety
101 ///
102 /// 裸函数。
103 #[naked]
104 0 references
105 unsafe extern "C" fn select_stack(hartid: usize) {
106     const STACK_LEN_PER_HART: usize = 4096 * STACK_PAGES_PER_HART;
107     const STACK_LEN_TOTAL: usize = STACK_LEN_PER_HART * MAX_HART_NUM;
108
109     #[link_section = ".bss.bootstack"]
110     static mut BOOT_STACK: [u8; STACK_LEN_TOTAL] = [0u8; STACK_LEN_TOTAL];
111
112     asm!(
113         "addi t0, a0, 1",
114         "la sp, {stack}",
115         "li t1, {len_per_hart}",
116         "1: add sp, sp, t1",
117         "addi t0, t0, -1",
118         "bnez t0, 1b",
119         "ret",
120         stack = sym BOOT_STACK,
121         len_per_hart = const STACK_LEN_PER_HART,
122         options(noreturn)
123     )
124 }
```

构造地址空间

```
11 // 启动页表
12 pub(super) struct BootPa
13     root: PageTable<Sv39>
14     sub: PageTable<Sv39>
15 }
16 // 初始化
17 // # Safety
18 // 调用前后位于不同的地址空间，必须内联。
19 #[inline(always)]
20 // 2 references
21 pub unsafe fn launch(&self, hartid: usize) -> usize {
22     use riscv::register::satp;
23     // 启动地址转换
24     satp::set(
25         satp::Mode::Sv39,
26         asid: 0,
27         ppn: self.root.as_ptr() as usize >> OFFSET_BITS,
28     );
29     // 此时原本地址空间还在，所以不用刷快表
30     // riscv::asm::sfence_vma_all();
31     // 跳到高页面对应位置
32     Self::jump_higher(kernel_mem_info().offset());
33     // 设置线程指针
34     asm!("mv tp, {}", in(reg) hartid);
35     // 设置内核可访问用户页
36     let mut sstatus: usize = lusize << 18;
37     asm!("csrrs {0}, sstatus, {0}", inlateout(reg) sstatus);
38     sstatus | (lusize << 18)
39 }
40 // 构造跳板页
41 const SIZE_2HIB:
42 const ASID_2HIB:
43 {
44     let mut p: F
45     let mut v: U
46     while !p.is
47     let entr
48     self.sub
49     p.s
50     v += S12
51     // 向上跳到距离为 'offset' 的新地址然后继续执行。
52     // 跃迁到虚地址
53     // # Safety
54     // 裸函数。
55     // 导致栈重定位，栈上的指针将失效！
56     #[naked]
57     unsafe extern "C" fn jump_higher(offset: usize) {
58         asm!("add sp, sp, a0", "add ra, ra, a0", "ret", options(noreturn))
59     }
60 } fn init
```

1. 探测物理空间

2. 构造启动页表

3. 构造跳板页

4. 跃迁!

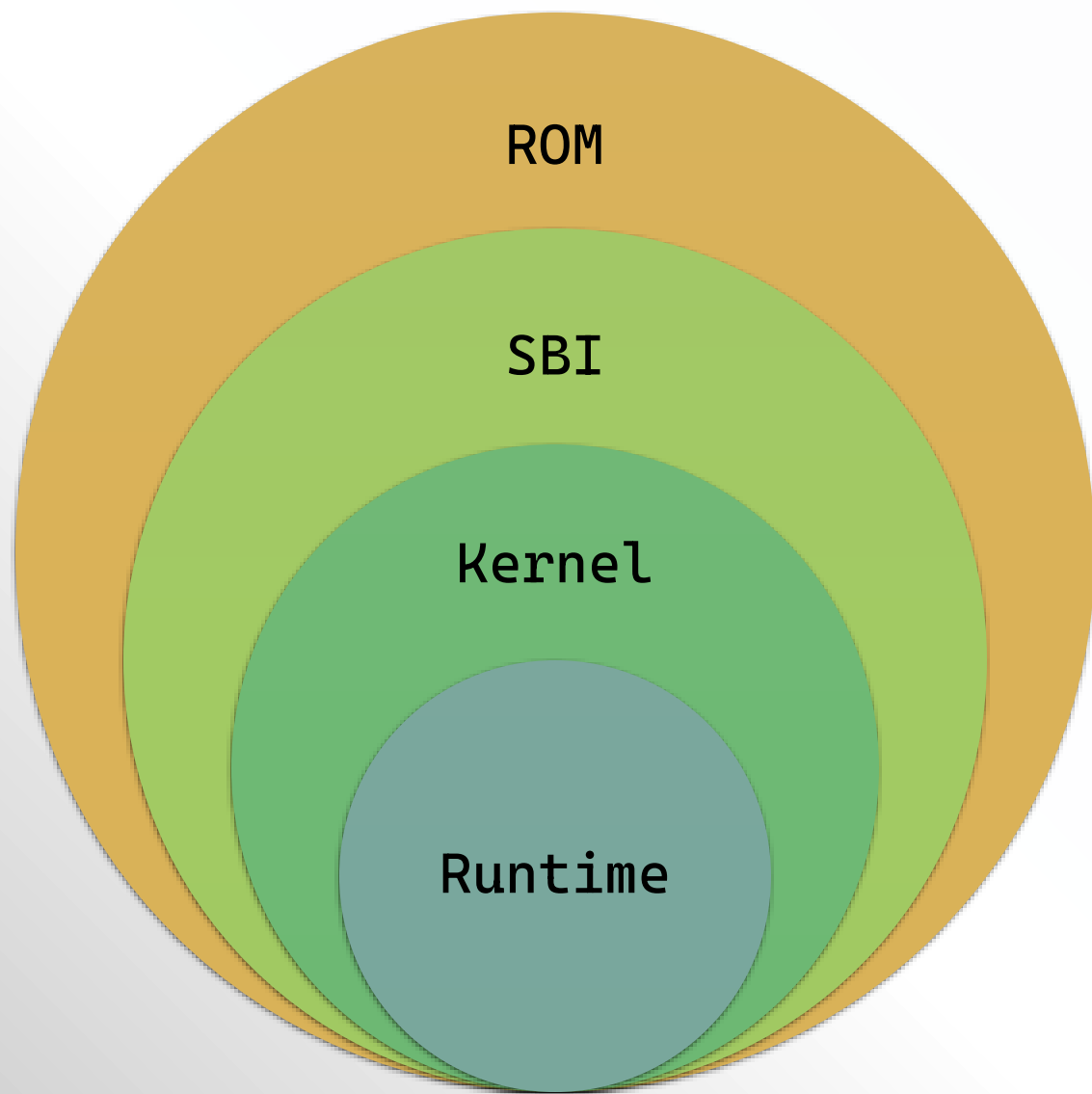
• 跃迁到虚地址

启动副核 (Secondary hart)



内核硬件线程用法展望

现在：硬件线程是特殊的资源	期望：硬件线程是普通外设
有特殊的初始化流程	和普通外设一起（动态）初始化
无状态管理	有状态管理（空闲时休眠/停止）
在一个上下文里循环	有多种工作模式
不支持异构（某些 Linux：异构核作为特殊外设执行特殊任务）	异构核统一管理参与正常调度
特殊的调度器	正常的异步运行时（控制硬件线程组成的线程池）



RustSBI-Qemu 多核环境

ROM 准备 SBI 环境

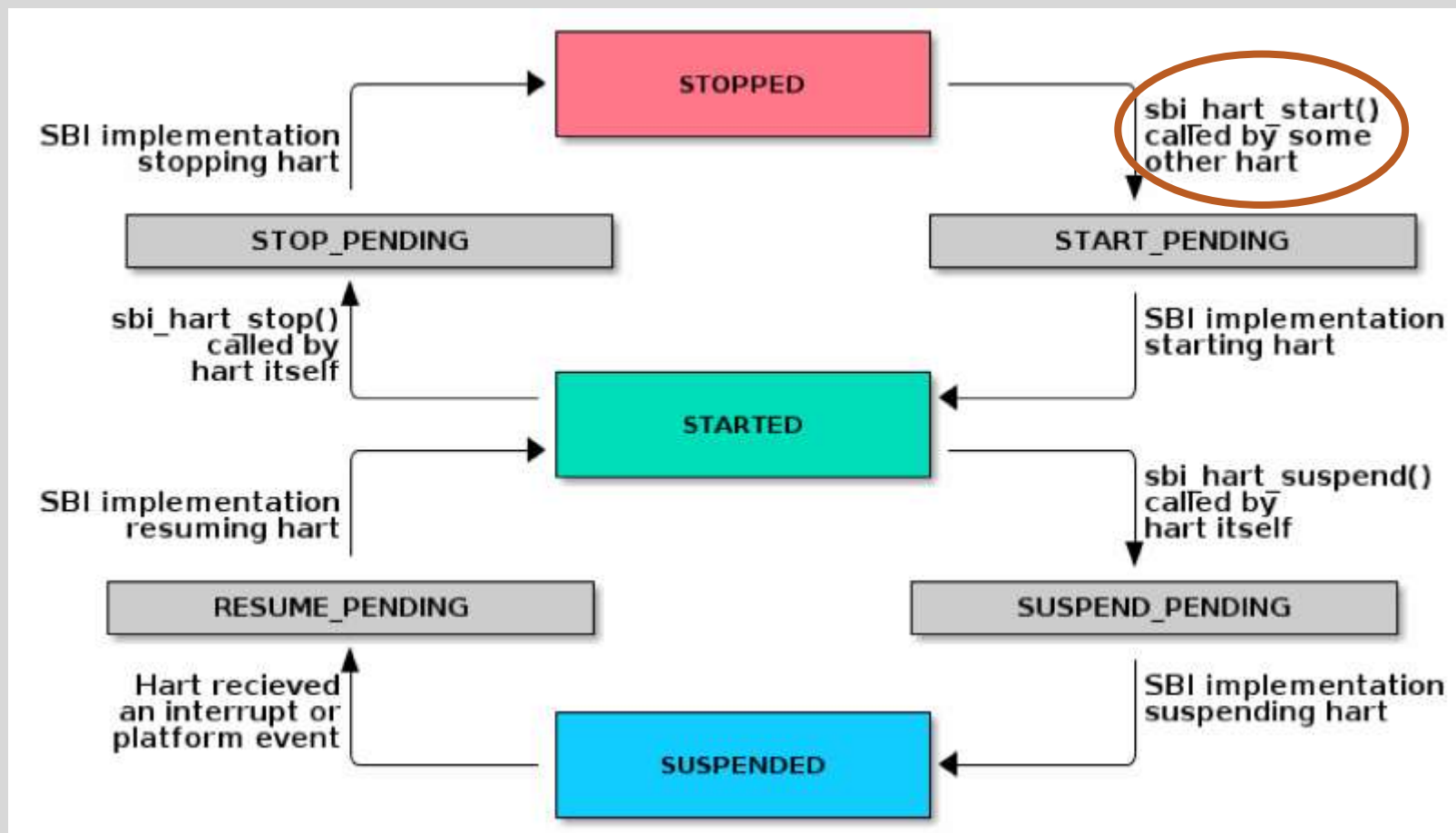
SBI 准备内核环境

内核准备 Runtime 环境

ROM/SBI 环境对比

ROM (不一定)	SBI Legacy (without HSM)	SBI Modern (with HSM)
所有硬件线程独立	一个硬件线程执行全局初始化	一个硬件线程执行全局初始化
所有硬件线程跳到固定地址	所有硬件线程跳到固定地址	每个硬件线程可以跳到不同地址
所有硬件线程并发跳转	所有硬件线程并发跳转	不执行全局初始化的硬件线程关闭
功能不统一，兼容性差	功能统一，要求内核必须接住	功能标准化，对内核要求低

SBI HSM 扩展



RustSBI-Qemu(virt) 实现

```
96  /// rust 入口。
97  0 references
98  extern "C" fn rust_main(
99      #[link_section = ".b"]
100      static GENESIS: Atom
101
102      static SERIAL: Once<
103      static BOARD_INFO: 0
104      static CSR_PRINT: At
105
106      // 全局初始化过程
107  > if !GENESIS.swap(val
149
150      let hsm: &QemuHsm =
151      if let Some(supervis
152          use execute::*;
153          // 设置并打印 pmp
154          set_pmp(board_in
155          if !CSR_PRINT.sw
156              hart_csr_uti
157          }
158          execute_supervis
159      } else {
160          Operation::Stop
161      }
162  } fn rust_main

impl QemuHsm {
1 reference
pub fn new(clint: &static Clint, smp: usize, opaque: usize)
    let state: MaybeUninit<[AtomicU8; NUM_HART_MAX]> = MaybeUninit::uninit();
    let supervisor: MaybeUninit<[Mutex<Option<Supervisor>; NUM_HART_MAX]> = MaybeUninit::uninit();

    let mut state: [AtomicU8; _] = unsafe { state.assume_init_mut() };
    let mut supervisor: [Mutex<Option<Supervisor>; _] = unsafe { supervisor.assume_init_mut() };

    for id: usize in 0..smp {
        state[id] = AtomicU8::new(START_PENDING);
        supervisor[id] = Mutex::new(
            // 执行全局初始化的硬件线程将直通特权软件
            if id == hart_id() {
                Some(Supervisor {
                    start_addr: SUPERVISOR_ENTRY,
                    opaque,
                })
            } else {
                // 否则将在下一个步骤被关闭
                None
            }
        );
    }

    Self {
        clint,
        state,
        supervisor,
    }
} fn new

/// 读取特权态入口地址，转换状态准备跳转。
1 reference
pub fn take_supervisor(&self) -> Option<Supervisor> {
    use core::sync::atomic::Ordering::AcqRel, Acquire];

    // 检查当前状态是启动前的挂起状态
    let state: &AtomicU8 = &self.state[hart_id()];
    let supervisor: Option<Supervisor> = self.supervisor[hart_id()].lock().take();

    let current: u8 = state.load(order: Acquire);
    let new: u8 = match current {
        START_PENDING: u8 => {
            if supervisor.is_none() {
                // 在启动过程中但未设置特权态入口，转入关闭流程
                STOP_PENDING
            } else {
                // 在启动过程中且已设置特权态入口，继续启动
                return supervisor;
            }
        }
        SUSPENDED: u8 => {
            if supervisor.is_none() {
                // 在挂起状态但未设置特权态入口，无法恢复
                panic!("cannot resume without supervisor!");
            } else {
                // 在挂起状态且已设置特权态入口，转入恢复流程
                RESUME_PENDING
            }
        }
        s: u8 => panic!("wrong state {s:?}!"),
    };

    match state.compare_exchange(current, new, success: AcqRel, failure: Acquire) {
        Ok(_) => supervisor,
        Err(unexpected: u8) => panic!("[failed to reboot for a race (current:? => {unexpected:?})"),
    }
} fn take_supervisor
```

SBI 多核支持展望

- SBI (Supervisor Binary Interface) 只能给 Supervisor 提供接口吗?
- SBI 必须作为运行环境占住 M 态吗?

不一定!

- SBI 也可以支持异构核 (小核) 的管理
- SBI 也可以不占满 M 态 (模拟一般 Bootloader)

Enclave-side SBI extension for Penglai enclave

- S+U enclave

- Extension id: 0x100101
- Use the sbi trap of

翻译 复制 搜索 全选 分享

- U enclave

- Extension id: 0x100101
- Use the sbi trap of CAUSE_USER_ECALL, which is undefined in the sbi specification

- Functions:

Macro	Function
SBI_EXIT_ENCLAVE	99
SBI_ENCLAVE_OCALL	98
SBI_ACQUIRE_ENCLAVE	97
SBI_CALL_ENCLAVE	96
SBI_ENCLAVE_RETURN	95
SBI_GET_REPORT	94



欢迎提问!