RustSBI的设计与实现

蒋周奇（洛佳）

华中科技大学 网络空间安全学院

2020.12 北京

介绍我自己

- 蒋周奇，笔名洛佳，1999年生，华中科技大学网络空间安全学院。热爱Rust嵌入式、操作系统开发。
- RustSBI项目作者。其它身份：社区工作者，《Rust日报》编辑，译《编写Rust语言的操作系统》。
- 3年Rust语言开发经验，曾经创业写手游。

SBI是什么：RISC-V标准

- 引导程序
  - 开发者对这一类软件的第一印象。它将引导启动操作系统，和UEFI比较相似。
- 统一的硬件环境
  - RISC-V架构中，常驻后台的固件，为操作系统提供一系列接口，以便其获取和操作硬件信息。常驻后台是SBI和其它引导程序相比，有特点的地方。
  - 这些统一的接口被称为“监督模式二进制接口”，即“SBI”。
- RISC-V SBI标准
  - 规定了一系列接口，包括重启、跨核软中断、设备树等功能模块。目前的设计目标大致是支持类Unix系统的启动与运行。有多种实现，OpenSBI是一种实现。

SBI是什么：功能和组成部分

- 硬件运行时
  - 处理中断和异常。厂家专有的中断处理器，也需要SBI实现适配和处理。SBI实现将委托一部分的异常到操作系统，这些异常包括缺页等等。
- 硬件环境接口
  - 和系统调用类似，“ecall”指令触发异常而陷入内核，由此实现操作系统向环境的调用。
- 引导启动模块
  - 初始化各个RISC-V寄存器，最终使用“mret”指令下降到S特权级，启动操作系统。
- 兼容模块
  - 捕捉部分新版本指令，使用旧版本指令模拟它。从而达到运行新版程序的目的。

SBI是什么：在操作系统生态中的作用

- 模块化
  - SBI是一个标准，它有多个实现。根据用户的选择，可以自己搭配SBI实现使用。
  - 不同的SBI实现可能拥有不同的功能。
  - 开发操作系统时，可以通过更换SBI实现，来排除可能的故障和问题。
- 兼容性
  - 只需要更新SBI实现，就能支持新版本标准的操作系统。在一段时间内，无需更换硬件。
  - 可以通过SBI软件，模拟硬件上暂未实现的指令集。
  - 延长了RISC-V硬件的生命周期。

欢迎使用RustSBI —— 发布0.1版本

- 适配RISC-V SBI规范v0.2版本
- 对类Unix系统有非常好的支持
- 完全使用Rust语言实现
- 具备OpenSBI的大部分功能
- 支持QEMU仿真器，它使用RISC-V特权级版本v1.11
- 向下兼容RISC-V特权级版本v1.9
- 支持K210芯片，包含内存管理单元（MMU）和S模式

RustSBI的组成模块

- trait抽象
  - 将SBI的功能抽象为trait。通常，每个模块对应一个trait。
  - 只要实现trait，使用函数注册到RustSBI，就可以在ecall中调用想要的功能。
- ecall实现模块
- 平台实现
  - 提供了QEMU、K210等芯片的实现，可以直接安装。
  - 为你想要的平台实现时，可以把RustSBI看作一个库，调用里面的实现函数。
- 文档：https://docs.rs/rustsbi

embedded-hal与Rust嵌入式生态

- 与标准对接
  - embedded-hal是Rust嵌入式开发的标准。
  - SBI的部分功能，和embedded-hal定义的标准比较重合。如果实现了embedded-hal，可以使用RustSBI实现对应的SBI功能。这包括：用串口实现调试控制台。
  - 可以用embedded-hal，简化SBI实现过程。比如：用专用的外设实现硬件重启，或者使用厂家专有的中断外设。
- Rust嵌入式生态
  - 实现embedded-hal支持，需要生成外设访问库“PAC”，然后生成硬件中间层库“HAL”。
  - PAC库使用机器生成，HAL库需要人工编写。这之后，就可以做RustSBI实现了。

高级的SBI设计

- RustSBI可以作为一个库使用
  - SBI开发者可以适配自己的平台。这包括：适配新的平台，增加复杂的实现等。
- 复杂的引导程序
  - 扫描磁盘
  - 从网络引导启动
- 带额外功能的引导程序
  - 带一个调试接口
  - 常驻后台的性能监视软件
  - 验证系统工作行为的单元测试软件

兼容性设计：以特权级1.9到1.11为例

- K210芯片
  - 嘉楠科技自研RISC-V RV64芯片，性价比非常高。
  - 很多资料认为没有MMU的Linux非常适合这款芯片，其实它是有MMU的。
  - 发布于2018年，硬件实现的RISC-V特权级版本是1.9.1。
- 1.9版本RISC-V特权级
  - sstatus.sum位（1.11）和sstatus.pum位（1.9）。
  - 指令sfence.vma（1.11）和指令sfence.vm（1.9）。
  - 1.9版本并没有S特权级的外部中断。
  - 1.11版本有页异常；1.9版本没有单独的页异常，它被包含在访存异常里。

兼容性设计例子：虚拟地址

模拟sfence.vma指令

如果发生非法指令异常，而且触发的是sfence.vma指令，就执行sfence.vm指令。

模拟satp寄存器

保存页表基址，每次页表刷新时模拟。

从1.11版本的“satp”中读内容，取出页表基址，写进1.9版本的sptbr寄存器中。

启用虚拟内存空间

页表配置1.11版本保存在satp中，而1.9保存在mstatus中。

目前只能直接设置mstatus。

```
} else if ins & 0xFE007FFF == 0x12000073 { // sfence.vma instruction
    // There is no `sfence.vma` in 1.9.1 privileged spec; however there is a `sfence.vm`.
    // For backward compability, here we emulate the first instruction using the second one.
    // sfence.vma: | 31..25 funct7=SFENCE.VMA(0001001) | 24..20 rs2/asid | 19..15 rs1/vaddr |
    //               14..12 funct3=PRIV(000) | 11..7 rd, =0 | 6..0 opcode=SYSTEM(1110011) |
    // sfence.vm(1.9):  | 31..=20 SFENCE.VM(000100000100) | 19..15 rs1/vaddr |
    //               14..12 funct3=PRIV(000) | 11..7 rd, =0 | 6..0 opcode=SYSTEM(1110011) |
    // discard rs2 // let _rs2_asid = ((ins >> 20) & 0b1_1111) as u8;
    // let rs1_vaddr = ((ins >> 15) & 0b1_1111) as u8;
    // read paging mode from satp (sptbr)
    let satp_bits = satp::read().bits();
    // bit 63..20 is not readable and writeable on K210, so we cannot
    // decide paging type from the 'satp' register.
    // that also means that the asid function is not usable on this chip.
    // we have to fix it to be Sv39.
    let ppn = satp_bits & 0xFFF_FFFF_FFFF; // 43..0 PPN WARL
    // write to sptbr
    let sptbr_bits = ppn & 0x3F_FFFF_FFFF;
    unsafe { llvm_asm!("csrw 0x180, $0"::"r"(sptbr_bits)) }; // write to sptbr
    // enable paging (in v1.9.1, mstatus: | 28..24 VM[4:0] WARL | ... )
    let mut mstatus_bits: usize;
    unsafe { llvm_asm!("csrr $0, mstatus":"=r"(mstatus_bits)) };
    mstatus_bits &= !0x1F00_0000;
    mstatus_bits |= 9 << 24;
    unsafe { llvm_asm!("csrw mstatus, $0"::"r"(mstatus_bits)) };
    // emulate with sfence.vm (declared in privileged spec v1.9)
    unsafe { llvm_asm!(".word 0x10400073") }; // sfence.vm x0
    // ::"r"(rs1_vaddr)
    mepc::write(mepc::read().wrapping_add(4)); // skip current instruction
} else {
```

兼容性设计例子：系统外部中断

添加一个新的SBI调用

注册系统级的外部中断入口，我们将用软件调用入口，从而处理这个中断。

SBI规范提供了定义专有函数的空间。

当外部中断发生时.....

首先，处理与架构有关的代码和内容。比如，对特定的硬件，执行中断的声明过程。

准备完毕后，调用已经注册的中断入口。

最后，要实现中断的收尾过程。

Implementation specific SBI functions

To solve the issue 3 in previous section, RustSBI's current implementation includes a RustSBI specific SBI call as a function.

The K210 supervisor-level external interrupt handler register function is declared as:

```
fn sbi_rustsbi_k210_sext(phys_addr: usize) -> SbiRet;
```

This function registers a device interrupt handler to machine level environment. On any machine-level external interrupt, the RustSBI's K210 environment would call the function provided.

The function's physical address shall be stored in register `a0` before calling this function. RustSBI will regard `a0` as a function without any parameters and any return values, or a `phys_addr: fn()`.

This function will always return `SbiRet` value of zero and error code of `SBI_SUCCESS`.

Function Listing

According to RISC-V SBI specification:

> Firmware Code Base Specific SBI Extension Space, Extension Ids 0x0A000000 through 0x0AFFFFFF
>
> Low bits is SBI implementation ID. The firmware code base SBI extension is the additional SBI extensions to SBI implementation. That provides the firmware code base specific SBI functions which are defined in the external firmware specification.

Since RustSBI has the implementation ID 4, its specific SBI extension is `0x0A000004`. We add the function mentioned above to this specific SBI extension.

Function Name	Function ID	Extension ID
sbi_rustsbi_k210_sext	0x210	0x0A000004

兼容性设计例子：跨级内存访问和页异常

- S层访问U层的内存
  - RISC-V的页表项中，有一个U位，置为1表示这是用户页。操作系统能否访问用户页？
  - 1.9中，默认允许，将PUM设为1可禁用。1.11中，默认禁止，将SUM设为1可允许。
  - 旧版的PUM和新版的SUM在寄存器的同一个位置上，所以暂时需要操作系统做区分。
  - 即使我们可以读写U层的内存，但RISC-V架构中，S层永远不能运行U层的代码，无论是否设置SUM。这个设计可以避免一些安全问题。
- 1.9中不存在独立的页异常
  - 在RustSBI-K210中，M层的运行时不把“页异常”委托到S层。
  - 目前需要操作系统在指令异常中处理情况，比如替换页等等。

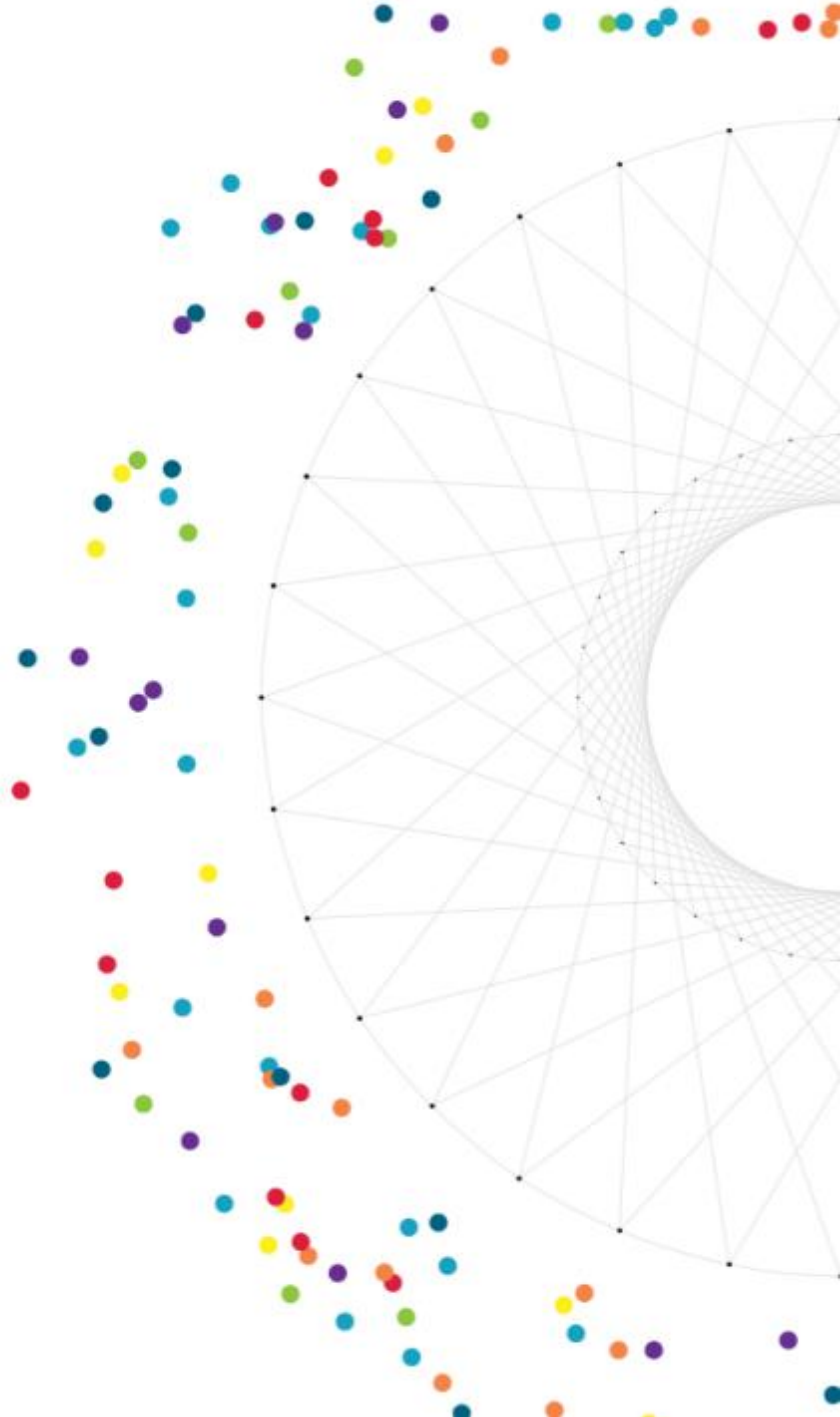
运行时与SBI

- 最简微架构运行时
  - Rust嵌入式中，一系列与微架构有关的运行时。许多架构都有实现，如riscv-rt。
  - 它是能在此架构裸机运行程序最小的抽象。通常包含程序入口点、中断和异常处理函数，还提供方便使用的属性宏。它也将提供参考的内存布局设置。
  - 对SBI来说，这样的运行时最快地验证原型。当然自己用汇编代码做一个也可以。
- RTIC运行时
  - Real-Time Interrupt-driven Concurrency[1]
  - 基于中断和所有权的资源调度运行时。特点：开发简单，运行开销小，生态齐全。
  - 我们需要做好对中断控制器的抽象和包装，这之后RTIC将支持RISC-V架构。
  - 一个很好的支撑SBI实现的运行时。

[1] Eriksson, J., Häggström, F., Aittamaa, S., Kruglyak, A., & Lindgren, P. (2013, June). Real-time for the masses, step 1: Programming API and static priority SRP kernel primitives. In Industrial Embedded Systems (SIES), 2013 8th IEEE International Symposium on (pp. 110-113). IEEE.

致谢

- RustSBI项目得到了很多学长的支持。感谢王润基学长，学长的想法让我受益匪浅。感谢吴一凡学长的意见和建议。
- 项目源于鹏程实验室的rCore Summer of Code 2020活动，感谢向勇教授、陈渝教授的指导，感谢实验室提供参与机会。
- RustSBI得到许多实际项目验证。感谢车春池和更多同学在内核开发上的研究，感谢世界各地的Rust爱好者提供帮助。



感谢各位

RustSBI的设计与实现

蒋周奇（洛佳）

2020.12 北京