安全孤岛RustSBI固件设计简明指南

洛佳

华中科技大学 网络空间安全学院

2022年4月

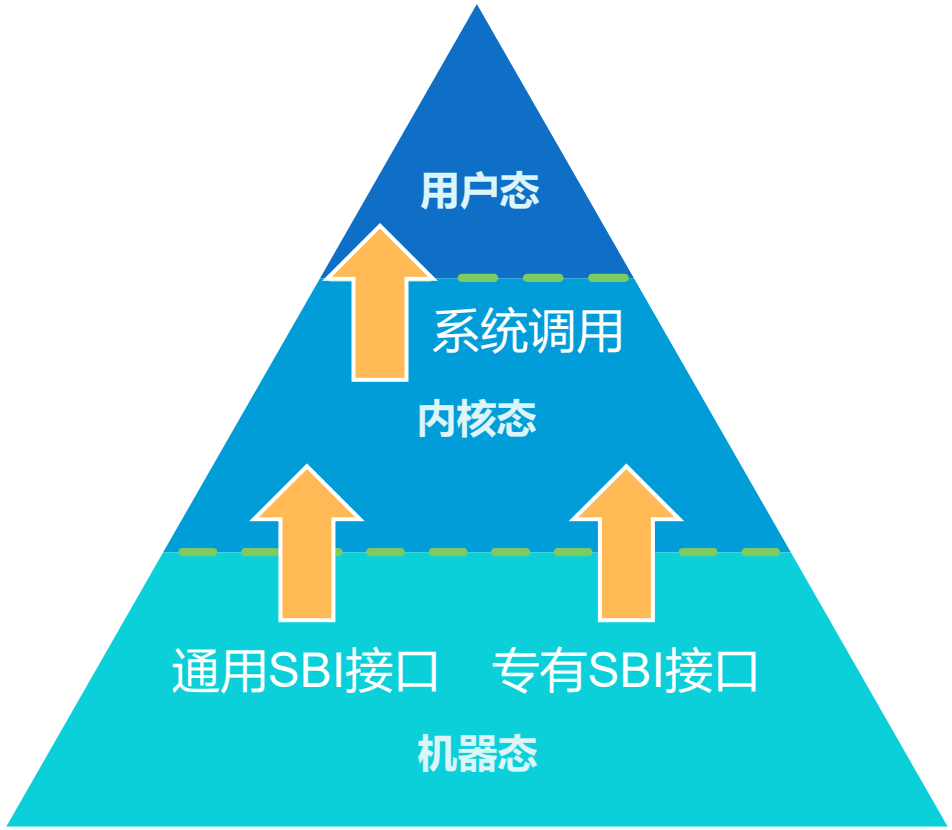
关于我……

- 笔名洛佳
- 社交媒体账号：@luojia65

专有的SBI接口

- RISC-V SBI规范标准v1.0.0版 [SBI] 定义了通用的接口，和专有的接口
- M态和S态是不同的处理器状态，可用于代码隔离
  - 其它架构上类似的系统有TrustZone
- 安全孤岛功能的实现体制：Penglai [SJTU21]，Keystone等等
- 为了调用M态来配置安全孤岛，需要一种环境调用机制，类似于系统调用，恰好可以放置在RISC-V SBI标准预留的专有接口区域中
  - 建议位于：供应者专有的SBI扩展空间，扩展编号从0x09000000到0x09FFFFFF
- 实现方法：检查a7寄存器是否属于定义的扩展空间，然后按照参数处理对应扩展

通用、专有SBI接口的生态位位置



RustSBI的Rust包设计与使用方法

- RustSBI [RUSTSBI] 的包设计
  - 以O开头的SBI实现何种设计和平台能进入上游存在争议，所以RustSBI上游只保留最小的抽象
  - 因此RustSBI本身只是一系列Rust trait的集合
- 引入RustSBI：不应当使用git submodule，建议使用cargo依赖，以减少编译步骤（此时不需要考虑git子模块同步的问题）
- 在代码中使用RustSBI
  - 目前版本的RustSBI相当于单例模型，必须使用init_*函数加载，然后在ecall中断调用rustsbi::ecall函数完成处理过程（未来会有面向虚拟化软件实现的实例模型）
  - 在rustsbi::ecall之前，判断a7寄存器是否是我们定义的固件专有扩展，如果是，使用专有的函数去处理次扩展，然后将mepc循环加4并返回到S态
- 目前设计中处理专有扩展时，绕过了RustSBI库。欢迎大家探讨更好的设计

专有功能与RustSBI同时出现时如何设计?

- 目前的做法：使用专有功能时绕过RustSBI
  - 如果检测到属于专有模块，由专有模块处理，而不使用RustSBI处理
  - 参考RustSBI-K210中专有的0x0A000004模块和其中的0x210函数
- RustSBI和专有功能并列出现时，不建议修改RustSBI本身来添加专有功能
  - 欢迎讨论更好的设计

读写特权层地址

- 扩展的SBI调用中若出现特权态地址……
  - SBI调用只允许将a0-a6（若存在函数编号，则为a0-a5）作为参数寄存器
  - 参数的数量太多，或者调用者提供的数据（编号、缓冲区或密钥等）太长
- 设置mstatus.MPRV寄存器位，读取S态内存
  - 此时借用（指针）的权限就好像S态去亲自读取它一样，避免手查页表
  - 注意：可能出现缺页或访存异常！
  - 如果选择手查页表，注意RISC-V架构下物理地址宽度可能超过虚拟地址
- 临时切换中断处理函数，检查特定指令（解引用操作）是否能正常运行——“检测指令法”
  - Rust包装代码：struct SupervisorPointer(...);
  - pub unsafe fn try_read<T>(src: SupervisorPointer<T>) -> Result<T, mcause::Exception>
  - 随后，可用少量代码将M态异常转发到S态

检测指令法：某条指令是否能成功运行

```rust
// Detect if hypervisor extension exists on current hart environment
//
// This function tries to read hgatp and returns false if the read operation failed.
pub fn detect_h_extension() -> bool {
    // run detection by trap on csrr instruction.
    let ans = with_detect_trap(0, || unsafe {
        asm!("csrr  {}, 0x680", out(reg) _, options(nomem, nostack)); // 0x680 => hgatp
    });
    // return the answer from output flag. 0 => success, 2 => failed, illegal instructic
    ans != 2
}
// Tries to execute all instructions defined in clojure `f`.
// If resulted in an exception, this function returns its exception id.
//
// This function is useful to detect if an instruction exists on current environment.
#[inline]
fn with_detect_trap(param: usize, f: impl FnOnce()) -> usize {
    // disable interrupts and handle exceptions only
    let (sie, stvec, tp) = unsafe { init_detect_trap(param) };
    // run detection inner
    f();
    // restore trap handler and enable interrupts
    let ans = unsafe { restore_detect_trap(sie, stvec, tp) };
    // return the answer
    ans
}
```

```rust
// Initialize environment for trap detection and filter in exception only
#[inline]
unsafe fn init_detect_trap(param: usize) -> (bool, Stvec, usize) {
    // clear SIE to handle exception only
    let stored_sie = sstatus::read().sie();
    sstatus::clear_sie();
    // use detect trap handler to handle exceptions
    let stored_stvec = stvec::read();
    let mut trap_addr = on_detect_trap as usize;
    if trap_addr & 0b1 != 0 {
        trap_addr += 0b1;
    }
    stvec::write(trap_addr, TrapMode::Direct);
    // store tp register. tp will be used to load parameter and store return value
    let stored_tp: usize;
    asm!("mv  {}, tp", "mv  tp, {}", out(reg) stored_tp, in(reg) param, options(nomem, nostack));
    // returns preserved previous hardware states
    (stored_sie, stored_stvec, stored_tp)
}

// Restore previous hardware states before trap detection
#[inline]
unsafe fn restore_detect_trap(sie: bool, stvec: Stvec, tp: usize) -> usize {
    // read the return value from tp register, and restore tp value
    let ans: usize;
    asm!("mv  {}, tp", "mv  tp, {}", out(reg) ans, in(reg) tp, options(nomem, nostack));
    // restore trap vector settings
    asm!("csrw  stvec, {}", in(reg) stvec.bits(), options(nomem, nostack));
    // enable interrupts
    if sie {
        sstatus::set_sie();
    };
    ans
}
```

源码地址：https://github.com/luojia65/zihai/blob/main/zihai/src/detect.rs

读取位于特权内存的数据结构……

```rust
/// Reads the supervisor memory value, or fail if any exception occurred.
///
/// This function will invoke multiple instructions including reads, write, enabling
/// or disabling `mstatus.MPRV` bit. After they are executed, the value is typically returned
/// on stack or register with type `T`.
pub unsafe fn try_read<T>(src: SupervisorPointer<T>) -> Result<T, mcause::Exception> {
    let mut ans: MaybeUninit<T> = MaybeUninit::uninit();
    if mstatus::read().mprv() {
        panic!("rustsbi-qemu: mprv should be cleared before try_read")
    }
    for idx in (0..mem::size_of::<T>()).step_by(mem::size_of::<u32>()) {
        let nr = with_detect_trap(0, || {
            asm!(
            "li     {mprv_bit}, (1 << 17)",
            "csrs   mstatus, {mprv_bit}",
            "lw     {word}, 0({in_s_addr})",
            "csrc   mstatus, {mprv_bit}",
            "sw     {word}, 0({out_m_addr})",
            mprv_bit = out(reg) _,
            word = out(reg) _,
            in_s_addr = in(reg) src.inner.cast::<u8>().add(idx),
            out_m_addr = in(reg) ans.as_mut_ptr().cast::<u8>().add(idx),
            options(nostack),
            )
        });
        if nr != 0 {
            return Err(Exception::from(nr));
        }
    }
    Ok(ans.assume_init())
}
```

源码地址：https://github.com/rustsbi/rustsbi-qemu/blob/main/rustsbi-qemu/src/prv_mem.rs

转发异常与内存页缺页

- 转发异常本身很容易
- SBI标准规定：固件实现访问特权级内存时若发生缺页和权限异常，将回到特权级，并填写sepc寄存器为ECALL指令的地址
  - 特权级重配页表和权限后，将再次调用ECALL指令进入固件
  - 固件中判断是否为缺页和权限异常返回后的情况即可
- 内核设计时，应当考虑固件访问时缺页和权限异常的情况

```rust
#[inline]
pub unsafe fn do_transfer_trap(ctx: &mut SupervisorContext, cause: scause::Trap) {
    // 设置S层异常原因为：非法指令
    scause::set(cause);
    // 填写异常指令的指令内容
    stval::write(mtval::read());
    // 填写S层需要返回到的地址，这里的mepc会被随后的代码覆盖掉。mepc已经处理了中断向量的问题
    sepc::write(ctx.mepc);
    // 设置中断位
    mstatus::set_mpp(MPP::Supervisor);
    mstatus::set_spp(SPP::Supervisor);
    if mstatus::read().sie() {
        mstatus::set_spie()
    }
    mstatus::clear_sie();
    ctx.mstatus = mstatus::read();
    // 设置返回地址，返回到S层
    // 注意，无论是Direct还是Vectored模式，所有异常的向量偏移都是0，不需要处理中断向量，跳转到入口地址即可
    ctx.mepc = stvec::read().address();
}
```

The page and access faults taken by the SBI implementation while accessing memory on behalf of the supervisor are redirected back to the supervisor with `sepc` CSR pointing to the faulting `ECALL` instruction.

特别感谢@dram：https://twitter.com/dramforever/status/1518641518566723584

落实RustSBI到真实硬件上

- RustSBI-Unmatched项目
  - 难点是U740芯片是异构多核的
  - 我在另一篇演讲 [SBI-AMP] 中分析过它的实现细节和注意事项
  - 这个项目的进展：95%完成（需要启动U-Boot第二阶段）
- Oreboot项目
  - 目标是做没有C语言的固件引导程序，从开机到启动，完全代替U-Boot项目。它同时也是一个RustSBI实现。[OREBOOT]
  - 开发进度：刚经历两次大重构，各个平台代码需要调整
- 建议开发方法：分支任何一个具体的RustSBI实现，添加安全孤岛功能
  - 若有可能，在充分讨论后可合并到上游。
- 我做过多篇演讲。欢迎阅读讲稿：https://github.com/rustsbi/slides

在操作系统中使用专有SBI功能

- 欢迎大家各抒己见！
- 我的看法：将额外的SBI模块看作驱动或内核模块，以合适的形式暴露给用户，从而调用SBI提供的安全孤岛或其它专有SBI功能
- 可以做成方便使用的Rust库（这就相当于是SDK了）

致谢

- Open Source Firmware社区
- TUNA社区群友和Rustcc社区群友
- 在RustSBI项目中发挥过重要作用的开源社区成员
- 所有在成长路上帮助过我的老师们和朋友们

引用

- [SJTU21]: Feng E, Lu X, Du D, et al. Scalable Memory Protection in the {PENGLAI} Enclave[C]//15th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 21). 2021: 275-294.
- [SBI]: The RISC-V Foundation, RISC-V SBI specification. Link: https://github.com/riscv-non-isa/riscv-sbi-doc
- [RUSTSBI]: RISC-V Supervisor Binary Interface (RISC-V SBI) implementation in Rust. Link: https://github.com/rustsbi/rustsbi
- [SBI-AMP]: https://github.com/rustsbi/slides/blob/main/2022/非对称多核处理器的RustSBI实现.pdf
- [OREBOOT]: Oreboot is a fork of coreboot, with C removed, written in Rust. Link: https://github.com/oreboot/oreboot