软硬协同的用户态中断机制研究

综合论文训练 结题答辩

尤予阳

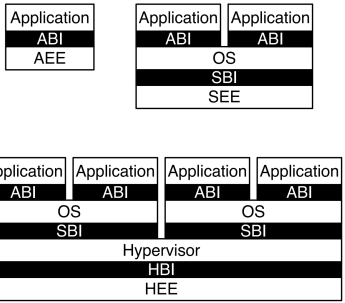
系内导师：马洪兵　交叉导师：陈渝

清华大学电子工程系

2022 年 6 月 9 日

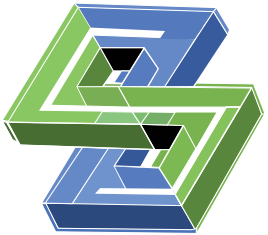
中断与特权级架构

- 处理器通过划分特权级限制软件行为，提供安全保护和隔离
- 中断提示处理器某个特殊事件的产生，通常由较高特权的软件处理，如操作系统内核
- 内核可以通过软件方式为用户程序模拟中断，但效率较低



用户态驱动

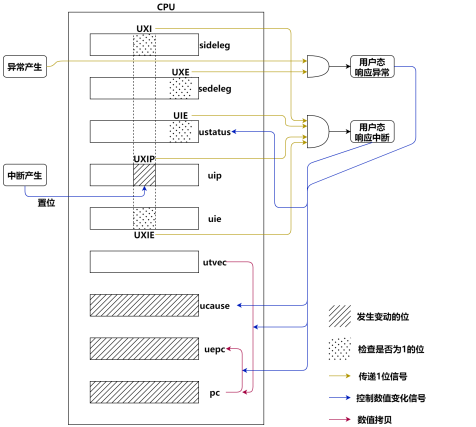
- 硬件驱动需要使用中断来提高响应速度，降低处理器占用
- 跨特权级切换有额外开销，用户态运行的驱动基本只能使用轮询
- 更高效的驱动需要绕过内核直达用户的通知机制——用户态中断



SPDK

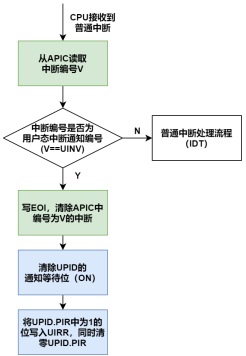
RISC-V 用户态中断扩展

- 也被称作“N 扩展”，v1.12 规范草案阶段提出，正式版本中被移除
- 中断控制寄存器和指令规范
- 未定义软件的跨核中断和外设的中断行为
- 已知有 shakti-c，StarFive 天枢，晶心科技 AX25MP 等处理器核 IP 在硬件上实现了 N 扩展

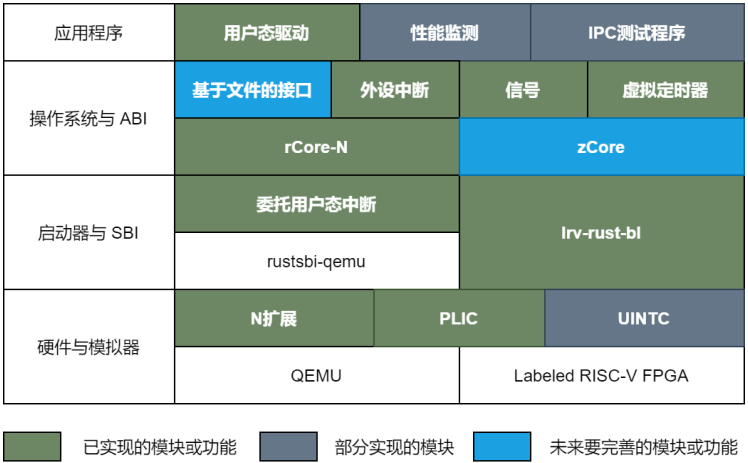


x86 用户态中断扩展

- 在英特尔即将发布的 Sapphire Rapids 处理器中支持
- 已在 Linux 内核中实现了软件接口
- 目前只能用于线程/进程间通信，性能相比软件方式大幅提升
- 尚未实现外设到软件的中断



项目架构


应用程序
用户态驱动
性能监测
IPC测试程序
操作系统与 ABI
基于文件的接口
外设中断
信号
虚拟定时器
rCore-N
zCore
启动器与 SBI
委托用户态中断
lrv-rust-bl
rustsbi-qemu
硬件与模拟器
N扩展
PLIC
UINTC
QEMU
Labeled RISC-V FPGA
已实现的模块或功能
部分实现的模块
未来要完善的模块或功能

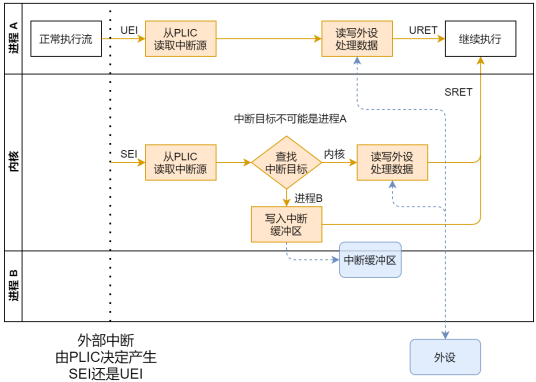
N 扩展的完善与实现

- ustatus, utvec 等寄存器的功能
- uret 指令
- 与中断控制器的连接
- 在模拟器与 FPGA 上实现

```
549     read_mapping += CSRs.mideleg -> reg_mideleg
550     read_mapping += CSRs.medeleg -> reg_medeleg
551+
552+    val read_uie = reg_mie & reg_sideleg
553+    val read_uip = read_mip & reg_sideleg
554+    val read_ustatus = Wire(init = 0.U.asTypeOf(new
    +   MStatus))
555+
556+    read_ustatus.upie := io.status.upie
557+    read_ustatus.uie := io.status.uie
558+
559+    read_mapping += CSRs.ustatus ->  read_ustatus.asUInt
    +   (xLen-1,0)
560+    read_mapping += CSRs.uip -> read_uip.asUInt
561+    read_mapping += CSRs.uie -> read_uie.asUInt
562+    read_mapping += CSRs.uscratch -> reg_uscratch
563+    read_mapping += CSRs.ucause -> reg_ucause
564+    read_mapping += CSRs.utval -> reg_utval.sextTo(xLen)
565+    read_mapping += CSRs.uepc -> readEPC(reg_uepc).
    +   sextTo(xLen)
566+    read_mapping += CSRs.utvec -> read_utvec
567+    read_mapping += CSRs.sideleg -> reg_sideleg
568+    read_mapping += CSRs.sedeleg -> reg_sedeleg
569   }
```

用户态外部中断

- 在平台级中断控制器 (PLIC) 中加入用户态的上下文
- 内核对外部中断的管理
- 基于用户态外部中断的设备驱动



系统调用

- init_user_trap(): 初始化用户程序中断上下文
- send_msg(pid, msg): 向另一个进程发送软件中断
- set_timer(time_us): 设置用户态时钟中断
- claim_ext_int(device_id): 获取对外设地址空间访问权限
- set_ext_int_enable(device_id, enable): 控制外设中断使能

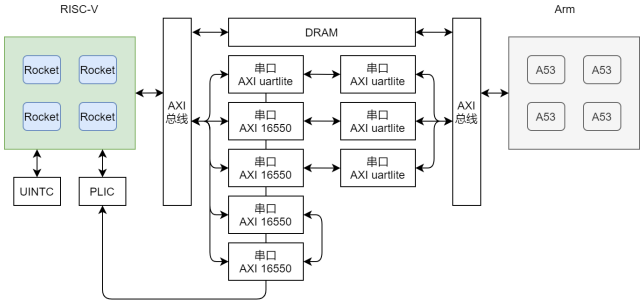
用户中断处理

- 将中断处理函数地址写入 utvec 寄存器
- 系统提供缺省的处理函数和跳板代码
- 用户程序可以覆盖某一个或全部的中断处理函数

```
#[linkage = "weak"]
#[no_mangle]
pub fn user_trap_handler(cx: &mut UserTrapContext) -> &mut
    ↪ UserTrapContext {...}

#[linkage = "weak"]
#[no_mangle]
pub fn ext_intr_handler(irq: u16, is_from_kernel: bool) {...}

#[linkage = "weak"]
#[no_mangle]
pub fn soft_intr_handler(pid: usize, msg: usize) {...}

#[linkage = "weak"]
#[no_mangle]
pub fn timer_intr_handler(time\_us: usize) {...}
```

硬件平台

- Rocket Core RV64IMACN @ 100MHz x4
- 2MB L2 Cache, 2GB DRAM
- AXI UART 16550 x4
- PLIC



驱动吞吐率测试

- 所用串口理论吞吐率 625KB/s
- 裸机（无操作系统）环境下的驱动性能优于有操作系统的情形
- 内核态中断模式的驱动性能远低于用户态驱动的性能
- 用户态轮询模式驱动性能最好，但 CPU 占用率高

驱动模式	裸机	rCore-N
内核，中断	396	78
用户，轮询	542	410
用户，中断	438	260

表 1: 吞吐率 (KB/s)

驱动延时测试

- 在代码中插入若干桩函数，在桩函数中读取并记录当时的 CPU 周期，由此计算延时
- 经用户态中断模式驱动向串口读取或写入字符，延时远低于经系统调用访问内核态驱动
- 仅计算驱动逻辑部分，基于用户态中断的驱动延时在均值和散布上也优于内核态
- 用户态驱动消除了特权级切换的开销，访存和代码局域性更好

读取字符延时

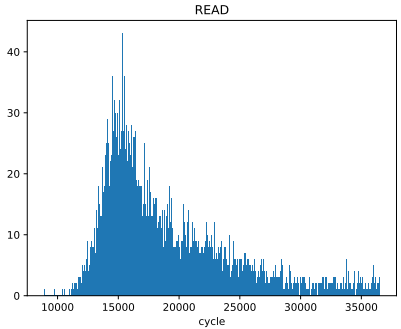


图 1: read 系统调用延时分布

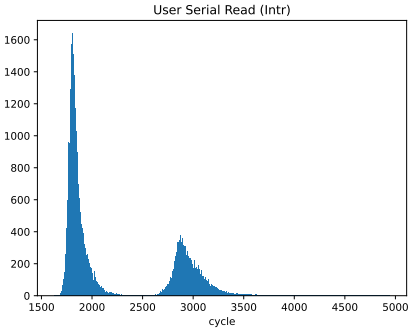


图 2: 用户态驱动读取延时分布

写入字符延时

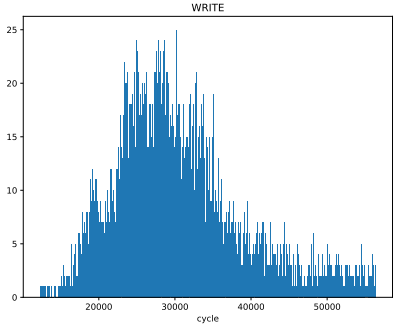


图 3: write 系统调用延时分布

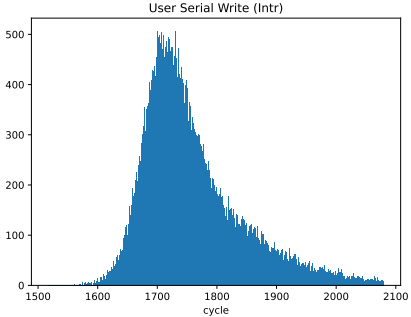


图 4: 用户态驱动写入延时分布

中断处理延时

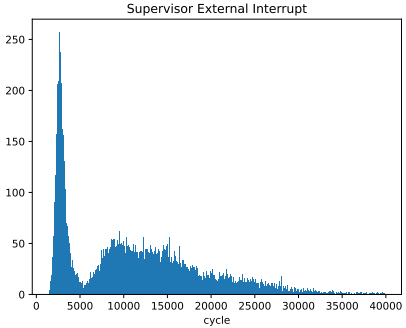


图 5: 内核态外部中断处理延时

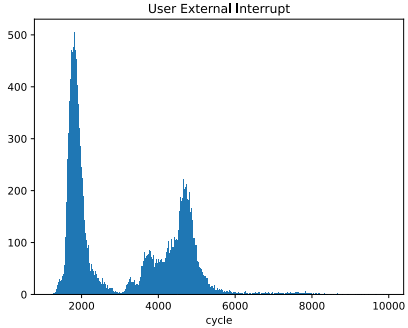


图 6: 用户态外部中断处理延时

Thanks!