软硬协同的用户态中断机制研究

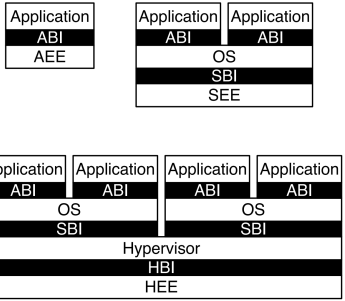
尤予阳

清华大学计算机系

2022 年 6 月 9 日

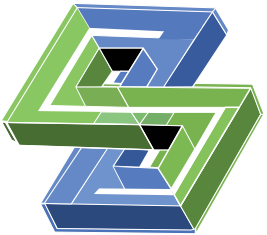
中断与特权级架构

- 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

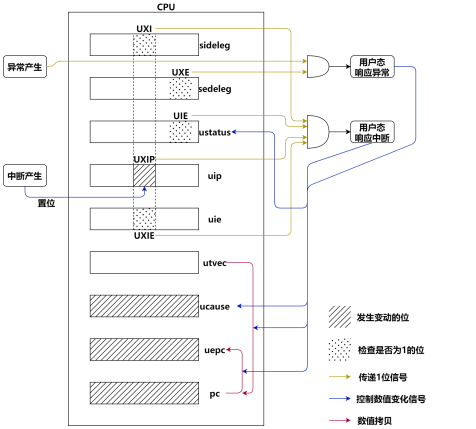
- 硬件驱动需要使用中断来提高响应速度，降低处理器占用
- 跨特权级切换有额外开销，用户态运行的驱动基本只能使用轮询 **速度太快了，OS跟不上**
- 更高效的驱动需要绕过内核直达用户的通知机制——用户态中断



SPDK

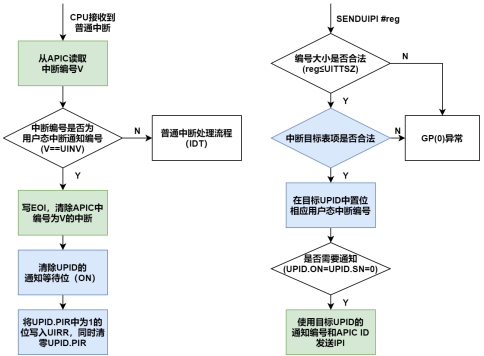
RISC-V 用户态中断扩展

- 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

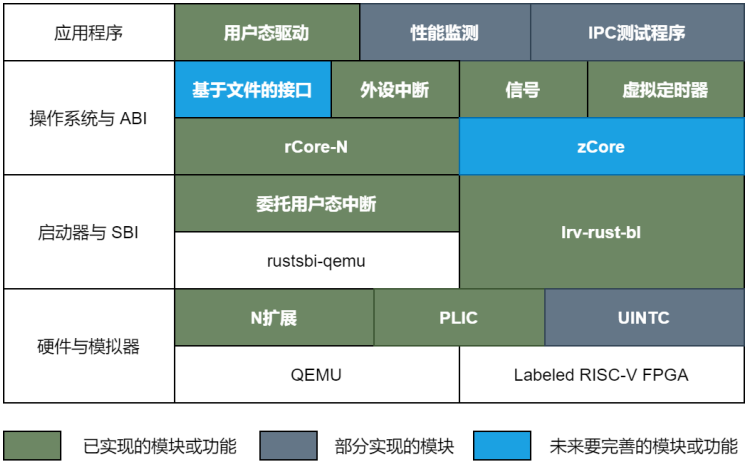


x86 用户态中断扩展

- 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

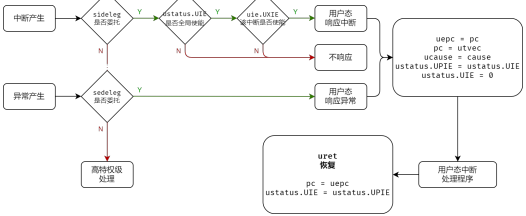

应用程序
用户态驱动
性能监测
IPC测试程序
操作系统与 ABI
基于文件的接口
外设中断
信号
虚拟定时器
rCore-N
zCore
启动器与 SBI
委托用户态中断
rustsbi-qemu
lrv-rust-bl
硬件与模拟器
N扩展
PLIC
UINTC
QEMU
Labeled RISC-V FPGA
已实现的模块或功能
部分实现的模块
未来要完善的模块或功能

N 扩展寄存器和指令

- ustatus: 用户态中断全局使能
- utvec: 陷入处理函数入口
- uip & uie: 待处理中断和各类中断使能
- uepc: 陷入时的程序指针地址
- ucause: 陷入原因
- utval: 陷入辅助值
- sideleg & sedeleg: 陷入委托
- uscratch: 自由使用
- uret 指令: 从用户态陷入上下文中返回

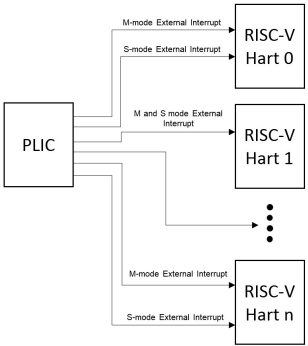
用户态陷入的基本处理流程

- 硬件处理陷入相关寄存器，跳转到 utvec
- 软件保存通用寄存器等上下文，处理陷入
- 软件处理完成，恢复上下文，调用 uret 返回
- 硬件恢陷入相关寄存器，跳转回 uepc 继续执行



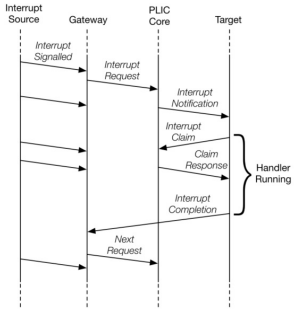
平台级中断控制器 (PLIC)

- RISC-V 系统中常见的外部中断控制器，在多个中断源和中断目标之间建立通路
- 硬件中断上下文是（硬件线程，特权级）的组合
- 加入用户态对应的中断上下文即可



用户态外部中断处理

- 需要内核将部分 PLIC 和外设对应的地址段映射到用户地址空间
- 读取 PLIC 的领取/完成寄存器，拿到对应的中断源编号
- 执行针对具体外设的处理逻辑
- 写入 PLIC 领取/完成寄存器，通知 PLIC 中断处理完成



用户态中断控制器 (UINTC)

- 多个用户态实体之间发送（跨核）中断，这些实体不一定运行在在特定的硬件线程上
- 现有的 (A)CLINT 架构无法满足需求，提出一种新的中断控制器设计——UINTC
- 设发送方数量为 S，接收方数量为 R，硬件上下文数量为 C
- 每个发送方和接收方具有一个用户态中断 ID（UIID）

UINTC 的寄存器

- enable(S, R): 第 s 个发送方是否可以给第 r 个接收方发送中断
- pending(S, R): 是否有第 s 个发送方发给第 r 个接收方的待处理的中断
- listen(C): 第 c 个上下文监听的接收方编号（不是 UIID）

UINTC 的寄存器

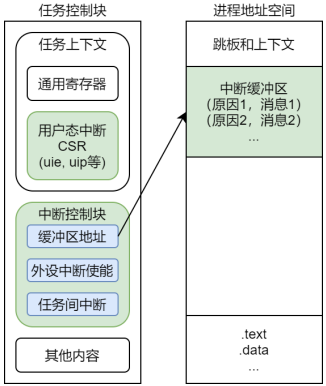
- sender_uiid(S): 第 s 个发送方对应的 UIID
- send/status(S): 写入接收方的 UIID 来发送中断，读出上一次发送的结果
- receiver_uiid(R): 第 r 个接收方对应的 UIID
- claim(R): 领取一个待处理中断，读出对应发送方的 UIID

UINTC 硬件处理流程

- 收到发送方软件 s 写入的接收方 UIID，查找是否存在接收方 r 使得 receiver_uuid(r)==UIID
- 若存在，且 enable(s,r)==1，则将 pending(s,r) 置为 1
- 若存在硬件上下文 c 满足 listen(c)==r，则将对应上下文的 xip.XSIP 位置 1，触发软件中断
- 接收方软件 r 读取 claim(r) 时，若存在 s 使得 pending(s,r)==1，则返回 sender_uuid(s)，否则返回 0

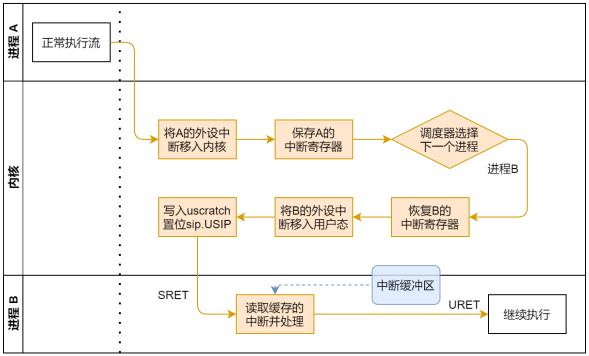
用户态中断上下文

- 记录每个进程的用户态中断 CSR、中断缓冲区和中断记录数目
- 中断缓冲区为一个内存页
- 一条中断记录包括原因和附加消息
  - 时钟中断和外部中断原因分别为 4 和 8，与 xcause 寄存器编码保持一致
  - 外部中断附加消息为中断外设号
  - 信号的中断原因为源进程 PID « 4



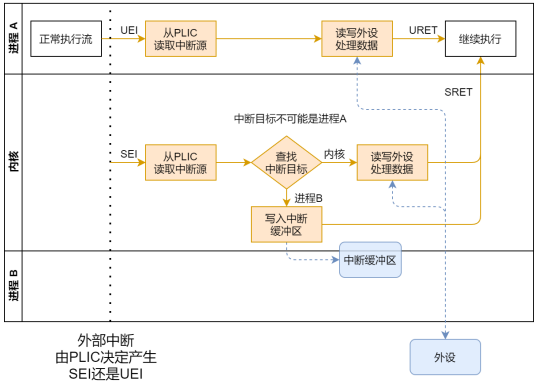
进程切换

- 进程切换时，保存当前进程的中断 CSR，恢复下一进程的中断 CSR，以及外设中断使能配置
- 从内核返回用户态时，将缓存的中断数量写入 uscratch 寄存器
- URET 返回正常执行流，无需系统调用



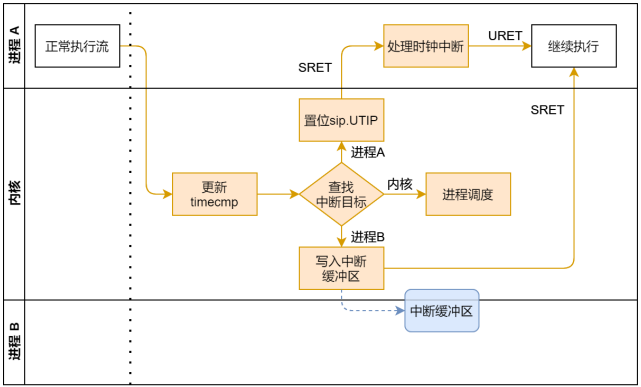
外部中断

- 内核记录每个外设对应的进程编号
- 如果外设对应的驱动进程正在 CPU 上运行，PLIC 直接产生 UEI，无需经过内核
- 否则产生 SEI，由内核转发



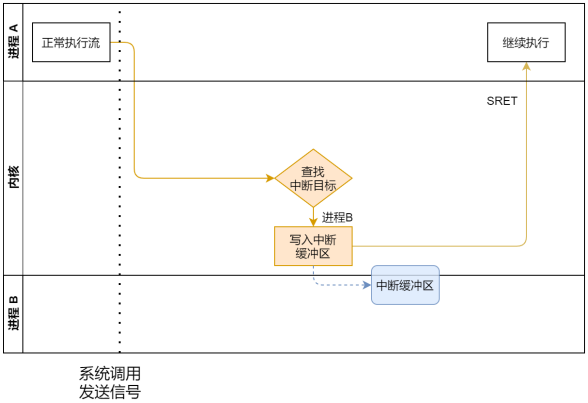
时钟中断

- 内核根据定时器到期时间维护一个有序列表
- 产生中断时传递给到期时间最早的申请者



软件间中断

- 在没有 UINTC 的情况下，需由内核转发，在目标任务的中断缓冲区中写入一条记录
- 有 UINTC 的情况下，发送过程不经过内核，中断记录保存在 UINTC 中，内核只需在切换时维护 listen 寄存器



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, enable): 控制外设中断使能

系统调用

- uipi_sender_ctl(flags, sender, buf): 控制用户态软件中断发送方信息
- uipi_receiver_ctl(flags, receiver, buf): 控制用户态软件中断接收方信息
- uipi_connection_ctl(sender, receiver, connected): 建立或取消两个任务间的中断连接

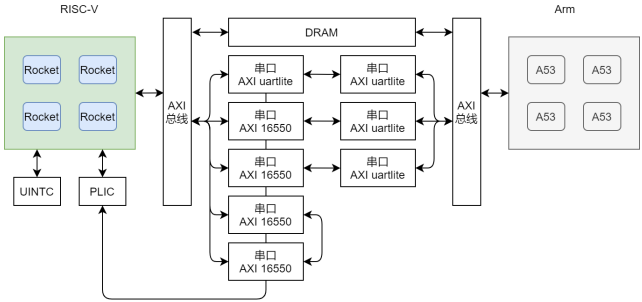
用户中断处理

- 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

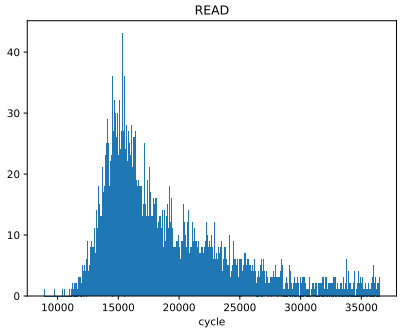


图 1: read 系统调用延时分布

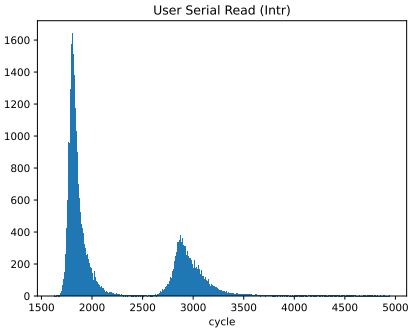


图 2: 用户态驱动读取延时分布

写入字符延时

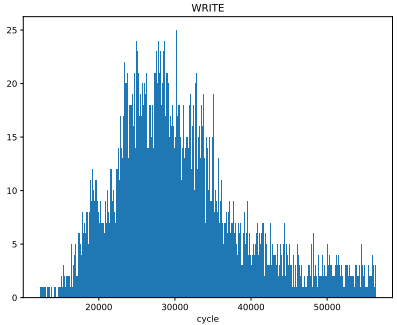


图 3: write 系统调用延时分布

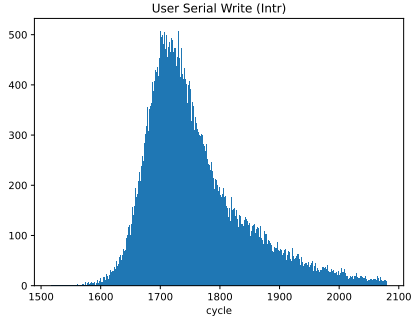


图 4: 用户态驱动写入延时分布

中断处理延时

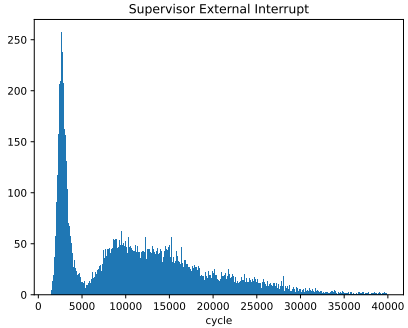


图 5: 内核态外部中断处理延时

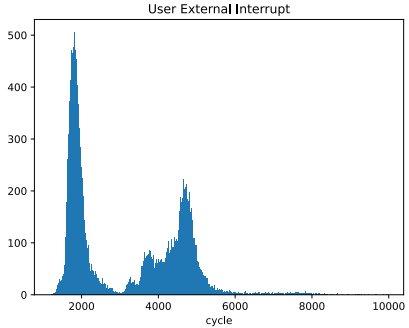


图 6: 用户态外部中断处理延时

后续工作

- 完成 UINTC 的 FPGA 实现（目前已有 QEMU 实现和 rCore 内核支持）
- 基于用户态软件驱动的任务间通信框架
- 基于用户态外部中断的异步外设驱动
- 更丰富可靠的性能评估

Thanks!