软硬件协同的用户态中断扩展

尤予阳　贺鲲鹏

指导教师: 马洪兵

清华大学 鲲筋烤队

操作系统功能赛答辩，2021 年 8 月 21 日

RISC-V 特权级架构

- MHSU 四层结构
  - H 暂未完全实现与应用
- 特权级不是必需的

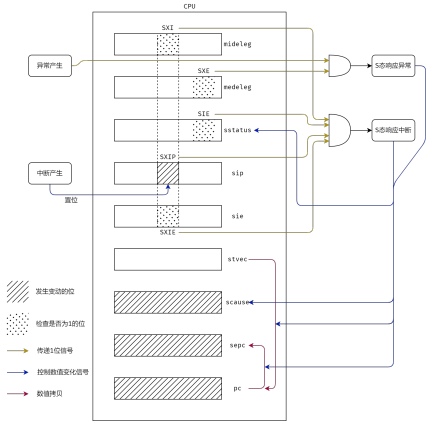
特权级数	特权级	系统
1	M	简单嵌入式系统
2	M U	安全嵌入式系统
3	M S U	类 Unix 操作系统

- 通过中断和异常，陷入高特权级

<table>
<tr><td colspan="3">Machine</td></tr>
<tr><td rowspan="3">bootloader</td><td colspan="2">Supervisor</td></tr>
<tr><td rowspan="2">OS</td><td>User</td></tr>
<tr><td>Application</td></tr>
</table>

RISC-V 中断规范

- 控制寄存器
  - xstatus
  - xie xip
  - xideleg xedeleg
  - xtvec xepc xcause xtval xscratch
- 特权指令
  - MRET SRET URET WFI ⋯

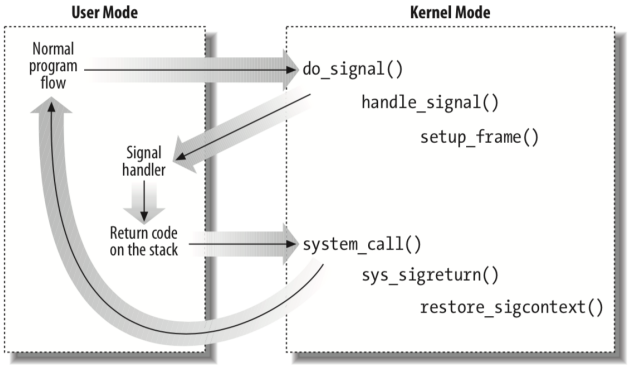


RISC-V N 扩展

- 未完成的用户态中断扩展草案
- 设计初衷主要是为嵌入式系统提供安全性扩展
  - 可信代码运行在 M 态，不可信代码运行在 U 态
  - 允许不可信代码处理中断
- 在最新的 RISC-V 指令集手册 1.12-draft 中被移除
  - 部分原因是认为设计目标可以通过 M+bare S 来实现
  - 还有部分原因是没有人再推动这个扩展草案的完善和实现

Linux 信号

- 一种 IPC 机制
- 传递很短的一段信息
- 从内核态返回用户态时处理
- 内核软件模拟的中断机制
  - 注意与 RISC-V 的“软件中断”（Software Intterupt）区分



用户态驱动

- 由用户态程序直接读写外设，避免系统调用和数据复制
- 以 SPDK 为例，轮询、异步、无锁的用户态 NVMe 驱动
- 为什么不使用中断？
  - 绝大多数硬件不支持将中断交给用户进程处理
  - 性能抖动和上下文切换开销

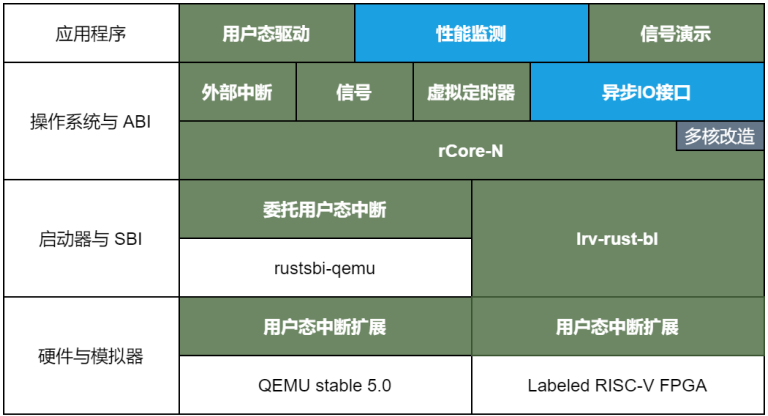
Ref

“SPDK polls devices for completions instead of waiting for interrupts. There are a number of reasons for doing this: 1) practically speaking, routing an interrupt to a handler in a user space process just isn't feasible for most hardware designs, 2) interrupts introduce software jitter and have significant overhead due to forced context switches.”
by `https://spdk.io/doc/userspace.html`

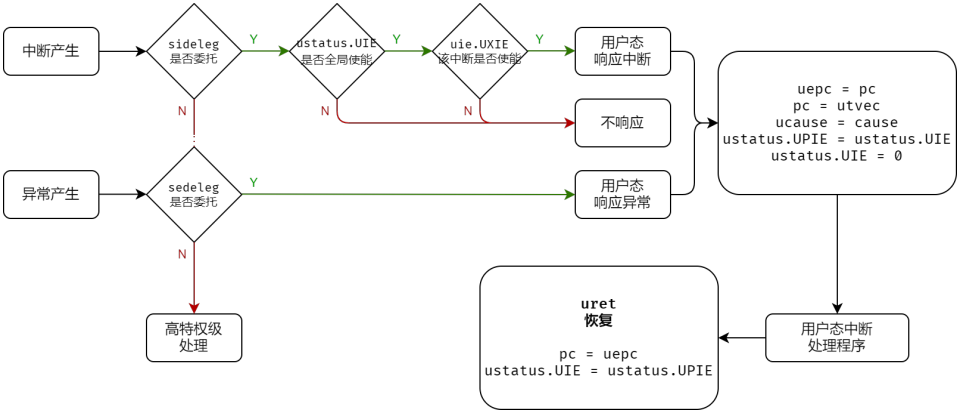
项目意义

- 完善 RISC-V 用户态中断扩展规范
- 提出一种符合该规范的模拟器和 FPGA 实现
- 在操作系统中实现对用户态中断的管理
- 使用该扩展优化操作系统中的若干机制
  - 使用用户态软件中断（USI）实现信号机制，由 URET 指令和跳板代码恢复上下文，省去 sigreturn() 系统调用
  - 为用户态驱动提供硬件中断支持

系统框架


应用程序
用户态驱动
性能监测
信号演示
操作系统与 ABI
外部中断
信号
虚拟定时器
异步IO接口
多核改造
rCore-N
启动器与 SBI
委托用户态中断
rustsbi-qemu
lrv-rust-bl
硬件与模拟器
用户态中断扩展
用户态中断扩展
QEMU stable 5.0
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

用户态陷入的硬件处理流程

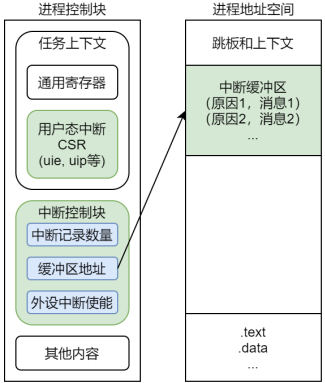


QEMU 和 FPGA 中的实现

- QEMU 基于 5.0 稳定版，FPGA 基于标签化 RISC-V 项目，核心为 Rocket Core
- 添加相关 CSR（ustatus, uie, uip, sideleg, sedeleg, ...）
- 添加处理逻辑（使能，屏蔽，跳转，委托等）
- 添加 URET 指令
- 添加 PLIC 上下文

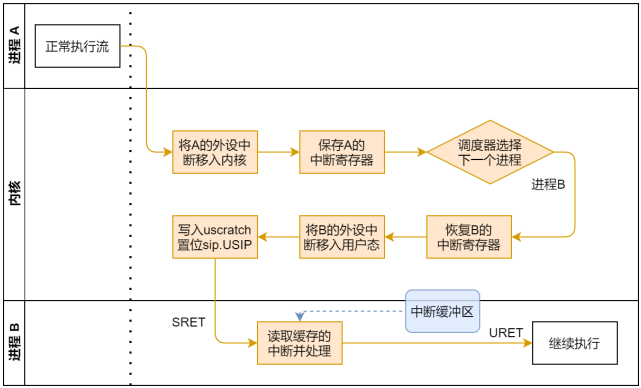
用户态中断上下文

- 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
- 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



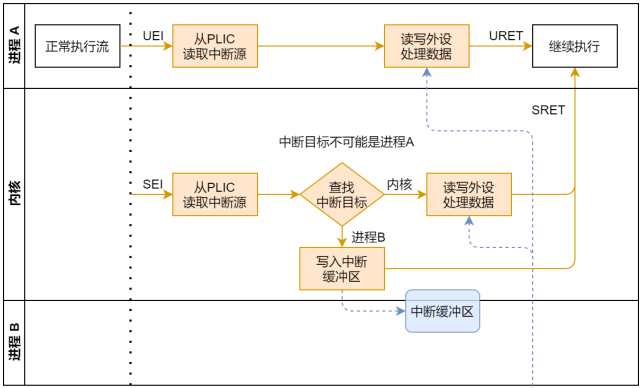
软件中断

- 实现信号机制
- 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



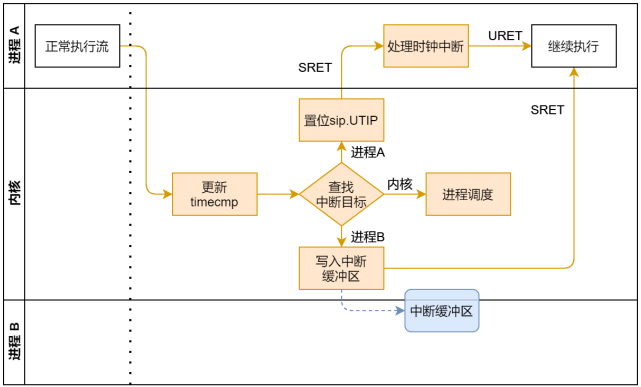
外部中断

- 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

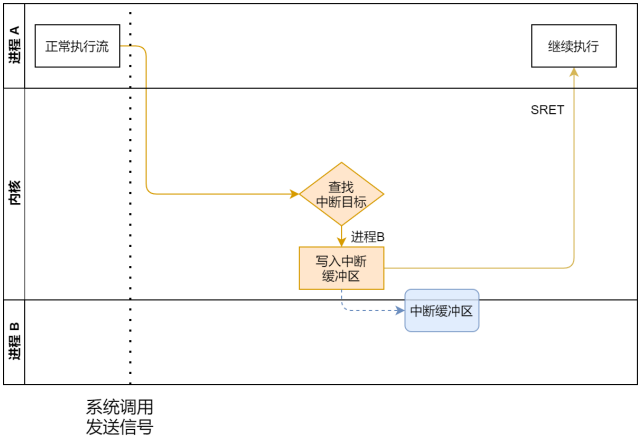
- 内核根据定时器到期时间维护一个有序列表
- 由中断产生的时刻判断是哪个进程设置的定时器



时钟中断

信号发送

- 多核情况下，若目标进程正在另一核上运行，可直接发送跨核中断，无需等待下一次调度



演示平台

- 硬件：ZCU102 FPGA 开发板，四个 Rocket Core，主频 100MHz
- 操作系统：rCore-N，支持多核

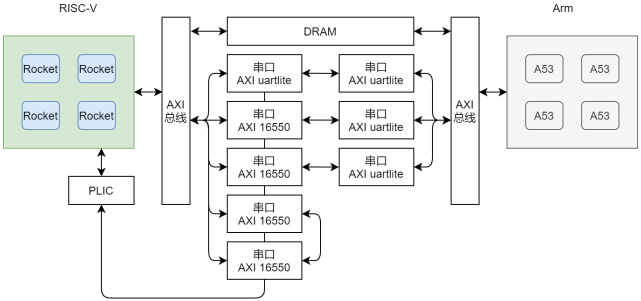


图 1: 硬件平台框图

演示程序

- 用户态的串口驱动
  - 串口接收到输入时产生中断
  - 驱动将用户输入回显到串口
  - 打印自己收到的信号，并在收到 15 时退出
- 信号和定时器演示
  - 设置 10 个间隔 1 秒的定时器
  - 在时钟中断处理函数中，向用户态串口驱动发送一条消息
  - 在最后一次定时器生效时发送 15 (SIGTERM)

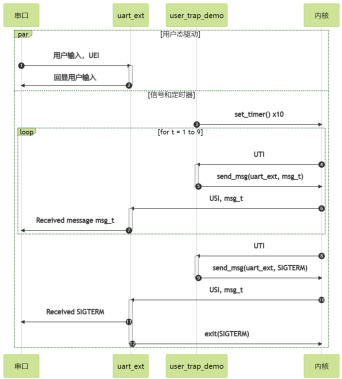


图 2: 演示程序执行流

演示程序


Arm Linux的终端
调试信息输出串口
内核终端串口
用户终端串口
1. 用户程序申请串口外设中断
1. 内核通过中断缓冲区转发的时钟中断
2. 用户输入及回显
3. 用户串口驱动收到外部中断
2. 内核通过直接置位uart.UTIP产生的时钟中断
3. 发送信号
4. 内核转发信号
5. 目标进程接收到信号并输出
6. 发送和接收到SIGTERM目标进程退出

项目信息

- repo：https://github.com/Gallium70/rv-n-ext-impl/
- 镜像：https://gitlab.eduxiji.net/carbon/project325618-89175
- 文档：https://gallium70.github.io/rv-n-ext-impl/

Thanks!