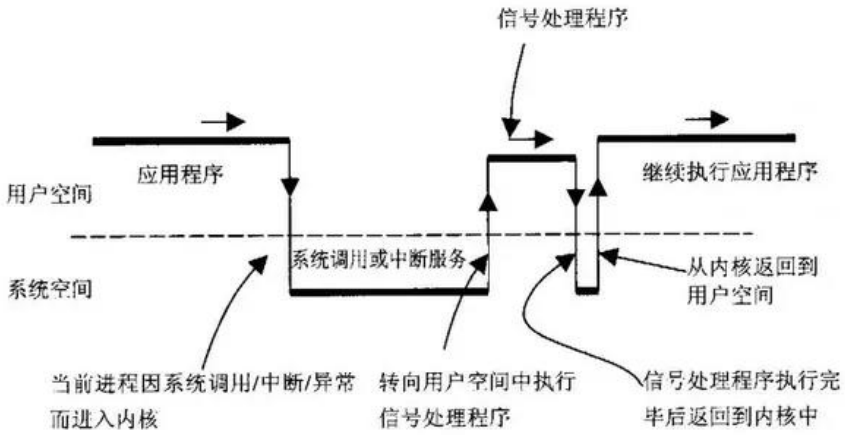
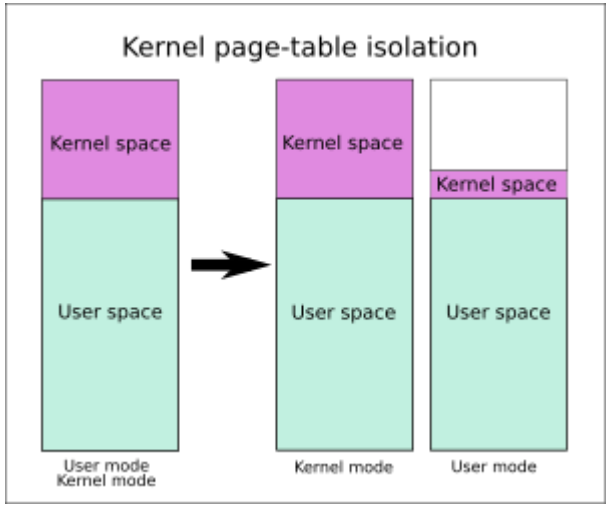
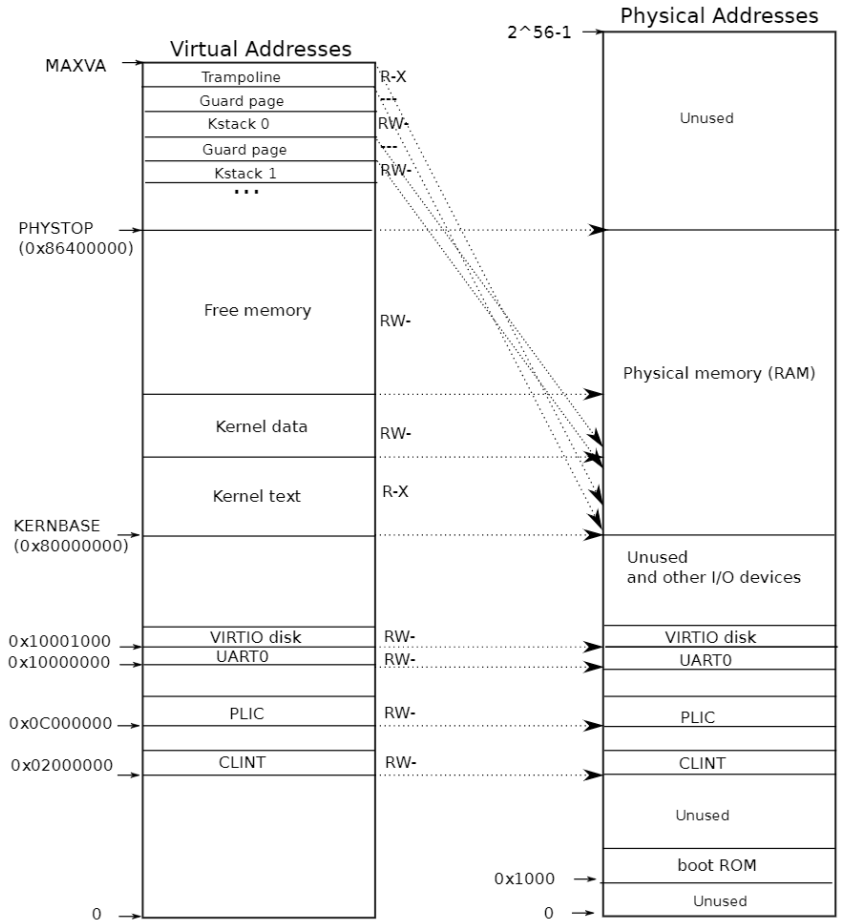
设计背景

设计背景

- 性能问题：
  - 传统的 IPC 需要切换特权级
  - 以 **熔断** 和 **幽灵** 为代表的安全漏洞，引入多页表机制





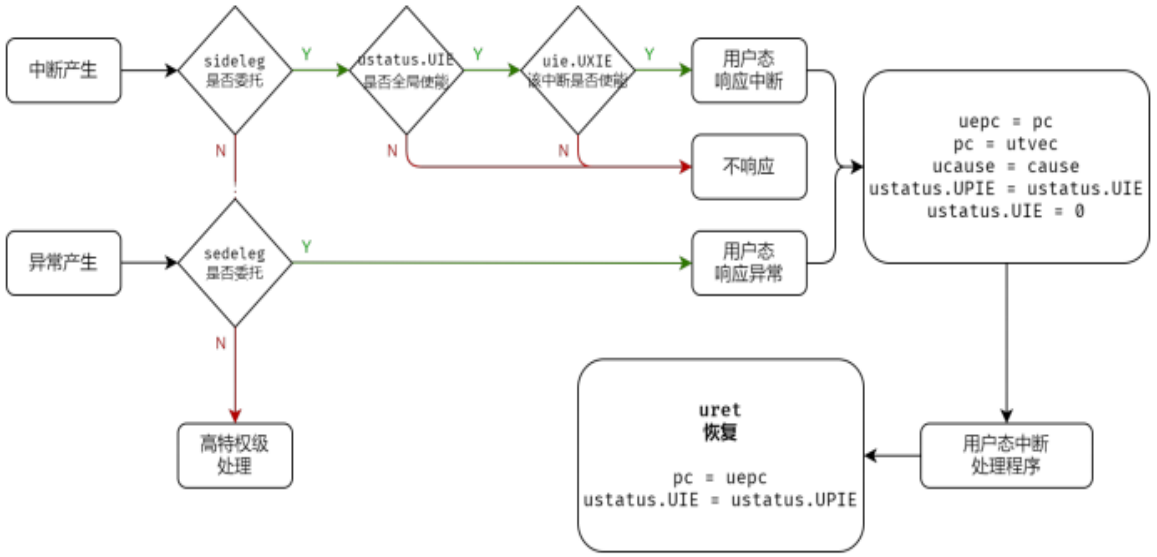


设计背景

- 用户态中断：
  - 软硬件协同方法
  - 进程间的通知和唤醒机制
  - 用户态对中断进行处理
  - 减少特权级切换带来的开销

已有工作

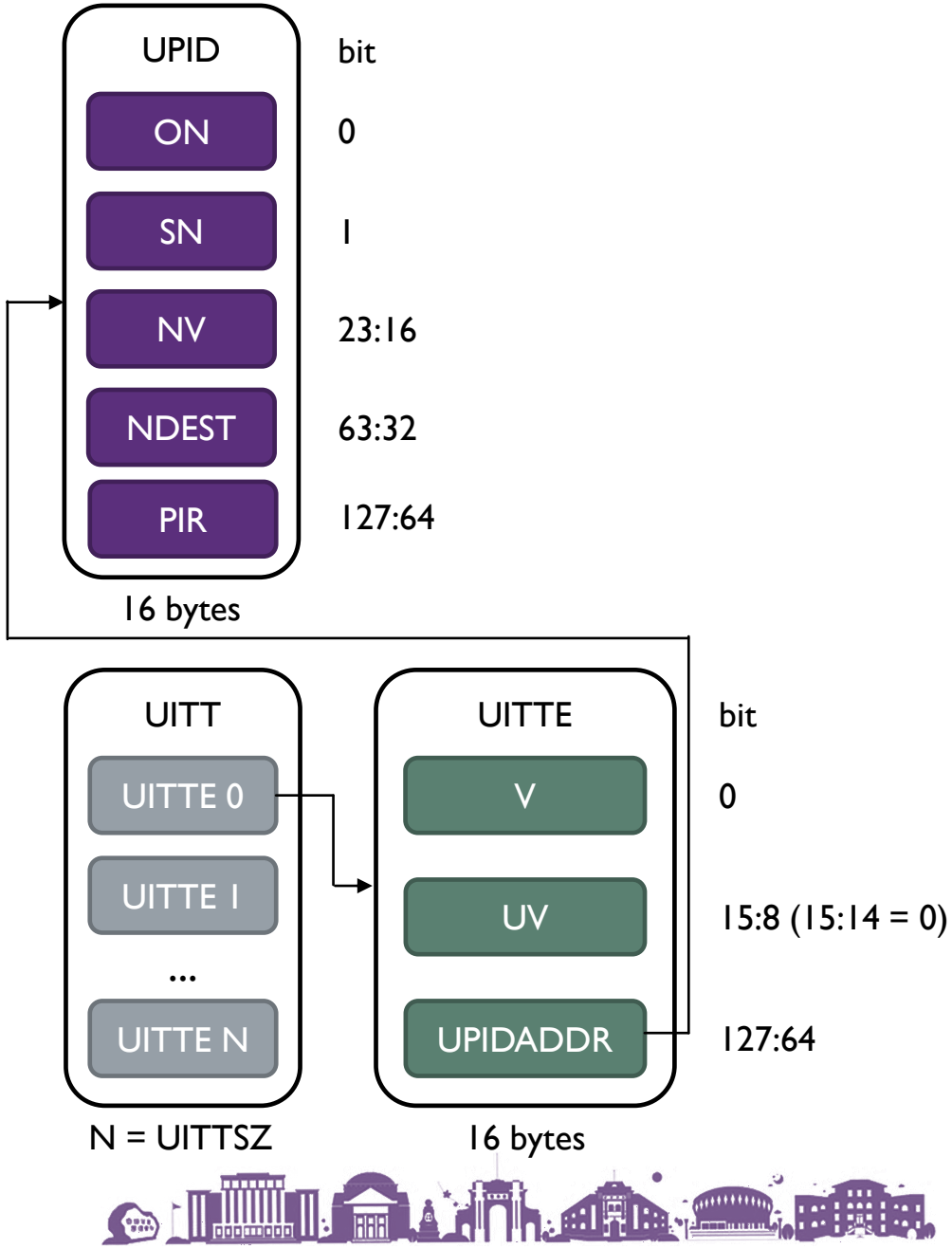
- 围绕 RISC-V N 扩展
  - 硬件开发板 zcu102
  - 外设到用户的中断
  - rCore-N 进行验证分析
  - UINTC （User Interrupt Controller）



已有工作

- Intel 用户态中断扩展
  - QEMU：添加硬件寄存器、指令、中断处理
  - Linux：新增系统调用，两种数据结构：
    - UPID：接收方注册中断处理函数
    - UITT：发送方注册 vector，维护 UPID 的地址
  - 系统调用：

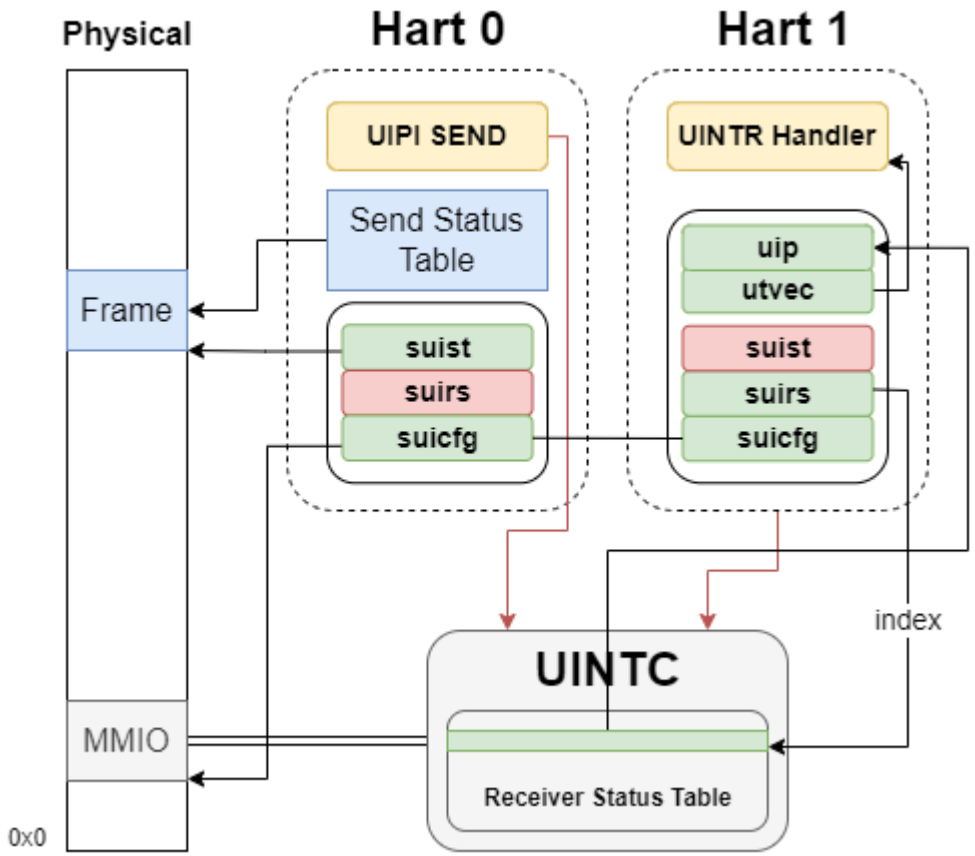
```
uintr_register_handler(handler, flags)
uintr_unregister_handler(flags)
uintr_create_fd(vector, flags)
uintr_register_sender(fd, flags)
uintr_unregister_sender(fd, flags)
```



设计简述

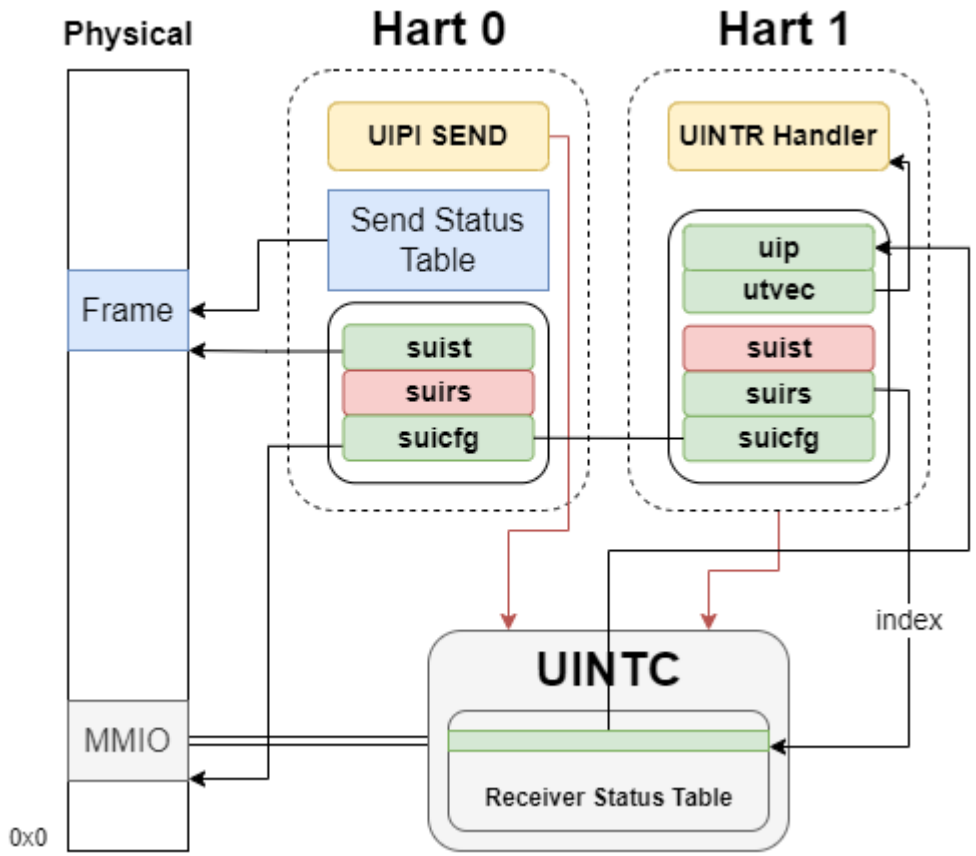
RISC-V 用户态中断扩展

- CSR 控制寄存器：
  - **suicfg:** 保存 **UINTC** 基址
  - **suist**：发送方使能，状态表基址、大小
  - **suirs**：接收方使能，状态表 index
- 用户态中断指令 **UIPI**
- 用户态中断控制器 **UINTC**



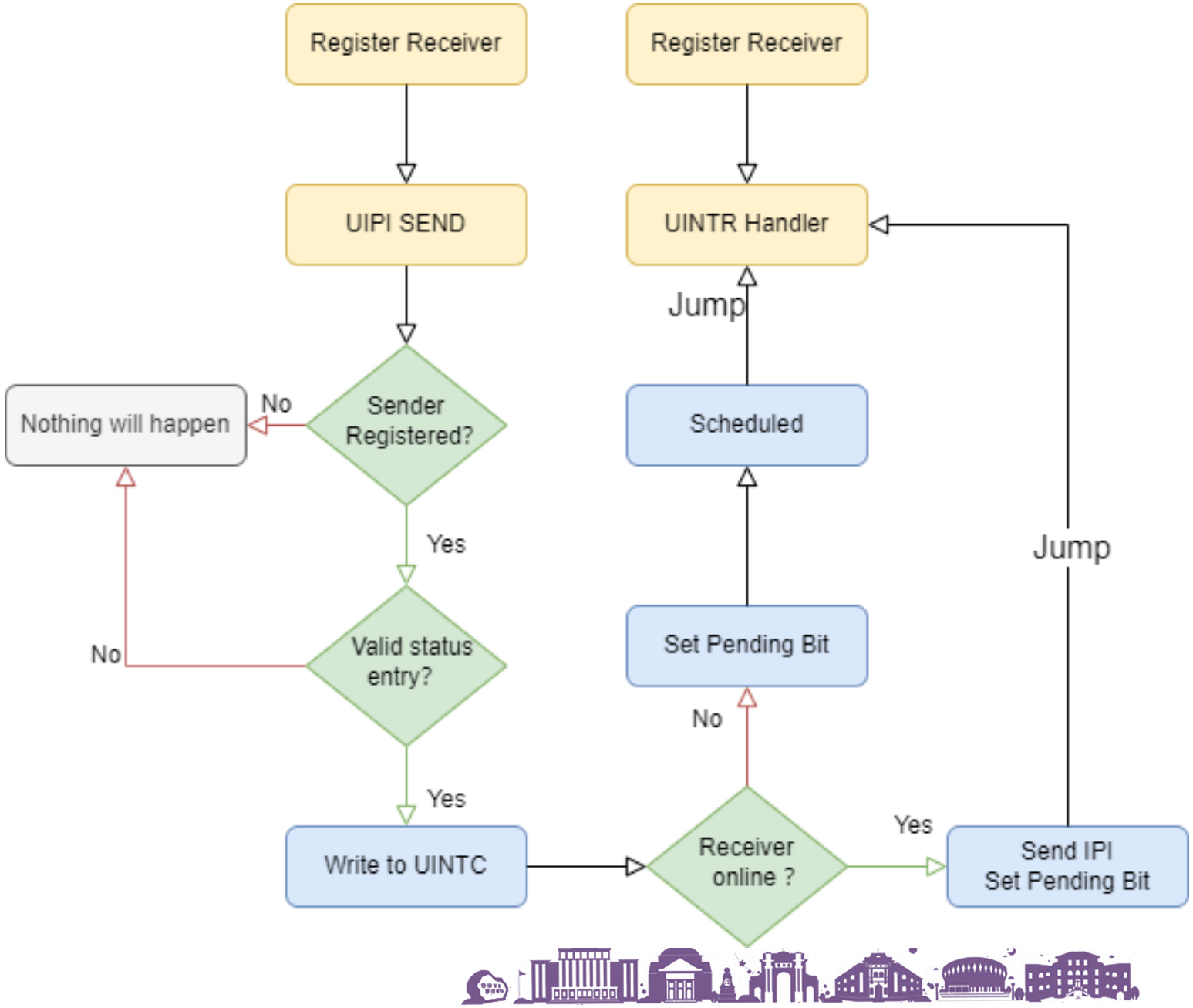
RISC-V 用户态中断扩展

- x86：接收方和发送方的状态都保存在内存中
  - 一次 SEND 操作需要两次读内存和一次写内存
- RISC-V：接收方状态保存在外设中，发送方状态保存在内存中：
  - 一次 SEND 操作需要一次读内存和一次写外设
  - UINTC 维护接收方状态，向对应核发送中断



RISC-V 用户态中断扩展

- 接收方：
  - 注册中断处理函数
  - 注册中断向量
- 发送方：
  - 根据 fd 注册状态表项
  - 根据 index 发送用户态中断



目前进展

QEMU

- **指令翻译**：
  - UIPI，URET 指令译码和执行
- **CPU 状态**：
  - CSR 寄存器
  - 用户态中断触发和响应
- **内存读写**：
  - UIPI 指令直接发起物理内存读写请求
- **核间中断**：
  - UINTC 用户态中断控制器
  - virt 模拟硬件中添加外设信息

内核实现

- **tCore** Kernel：
  - Rust 编写，代码行数 12 k
  - 使用 OS 比赛测例进行测试和开发
  - 参考 maturin，linux 2.6 等内核
  - 支持多核，FAT 文件系统，COW
  - 支持常见系统调用如 clone，mmap 等
  - 支持 RISC-V 用户态中断扩展

LIBC 实现

- 参考 x86 用户态中断测例 uipi_sample.c 进行功能测试
- 提供 libc 系统调用支持，对上下文处理进行封装：
  - uintr_register_receiver (handler)
  - uintr_create_fd (vector)
  - uintr_register_sender (fd)

```
extern void __handler_entry(struct __uintr_frame* frame, void* handler) {
  uint64_t irqs = uipi_read();

  uint64_t (*__handler)(struct __uintr_frame * frame, uint64_t) = handler;
  irqs = __handler(frame, irqs);

  uipi_write(irqs);
  csr_clear(CSR_UIP, MIE_USIE);
}

static uint64_t __register_receiver(void* handler) {
  // set user interrupt entry
  csr_write(CSR_UTVEC, uintrvec);
  csr_write(CSR_USCRATCH, handler);

  // enable U-mode interrupt handler
  csr_set(CSR_USTATUS, USTATUS_UIE);
  csr_set(CSR_UIE, MIE_USIE);

  int ret;
  ret = __syscall0(__NR_uintr_register_receiver);

  // enable UINTC
  uipi_activate();

  return ret;
}
```

未来计划

未来计划（本学期）


在 Linux 6.0 中加入对 RISC-V 用户态中断的支持
性能测试，对比传统 IPC
（可能）在 Rocket Chip 中加入对 RISC-V 用户态中断的支持