

第 1 讲：操作系统概述

第五节：OS 实验概述

向勇、陈渝、李国良

清华大学计算机系

xyong,yuchen@tsinghua.edu.cn

2021 年 2 月 21 日

- 设计思路

- 设计 ucore/rcore，覆盖操作系统的关键点，内容如下：
 - 外设：I/O 管理/中断管理
 - 内存：虚存管理/页表/缺页处理/页替换算法
 - CPU：进程管理/调度器算法
 - 并发：信号量实现和同步互斥应用
 - 存储：文件系统 + 磁盘驱动
- 完整代码量控制在 4000 行左右 (希望)
- 提供实验讲义和源码分析文档

OS 实验内容

OS 实验内容

- 1 hello-world OS
- 2 kernel-mode OS
- 3 multi-programming OS
- 4 mem-isolation OS
- 5 multi-process OS
- 6 Inter-Process-Comm OS
- 7 File-System OS
- 8 r/u Core OS



图: OS 实验框架

OS 实验内容

lab1

Lab1: hello-world OS

在裸机上的执行环境，让应用与硬件隔离，简化了应用访问硬件的难度和复杂性。

- 直接与硬件交互的系统程序的编译运行
- 输出字符的方法
- 调试系统程序的方法

Lab2: kernel-mode OS

操作系统利用硬件特权级机制，实现对操作系统自身的保护。应用在用户态通过系统调用得到内核态的内核服务。操作系统批处理机制支持多个程序的自动加载和运行。

- 特权级机制
- 应用程序实现
- 批处理机制
- 特权级切换

Lab3: multi-programming OS

操作系统通过协作机制/抢占机制支持程序主动/被动放弃处理器，提高系统效率。

- 多道程序的放置与加载
- 任务切换
- 协作式调度
- 抢占式调度

Lab4: mem-isolation OS

操作系统通过动态内存分配机制、页表的虚实内存映射机制，加强内存安全，简化应用开发。

- 动态内存分配
- 地址空间 (Address Space) 抽象
- 多级页表

Lab5: multi-process OS

操作系统建立了进程创建、执行、切换和结束的动态管理过程

- 进程 (Process) 抽象
- 进程管理

Lab6: Inter-Process-Comm OS

操作系统通过进程间通信 (Inter-Process-Comm) 机制，让应用之间建立了有效的联系。

- 进程间通信机制
- 管道 (PIPE) 机制

Lab7: File-System OS

操作系统通过文件系统完成对程序和数据的持久保存与灵活的访问

- 文件 (File) 抽象
- 基于 inode 方式的文件系统的设计与实现

Lab8: r/u Core OS

为支持更丰富的应用需求，操作系统需要改进与完善。通过完成一个完整的 OS kernel 的设计与实现，形成面向操作系统的系统思维。

- 操作系统各组成部分关联关系的完善
- 操作系统各组成部分的改进与优化
- 对应用的进一步服务与支持

LabX: 大实验

前提：已经完成基本实验

尝试完成一些有一定挑战性且有趣的 OS 实验。

- 参加 OS 比赛的推荐项目
- 改进与设计 zcore/rcore/acore 操作系统
- 在一个 OS(如 Linux) 实现一个 Hypervisor
- 基于异步机制的新型 OS
- 支持动态更新的 OS
- 驱动程序运行在用户态的 OS

OS 实验内容

labX

选题方向	大实验题目
RISC-V	ucore on RISC-V
RISC-V	简易版 rcore 开发与教学文档编写 && rcore plus 开发
RISC-V	FPGA 上运行 RISC-V rCore 构建路由器
x86_64	对标 Biscuit OS 真实应用真实网卡及性能测试
x86_64	rCore 内核可加载模块和动态链接库
MIPS	第三届全国大学生系统能力培养大赛
Arm	Python (and more) on rCore on RPi
GUI	GUI
GUI	适配 mini GUI

OS 实验内容

labX

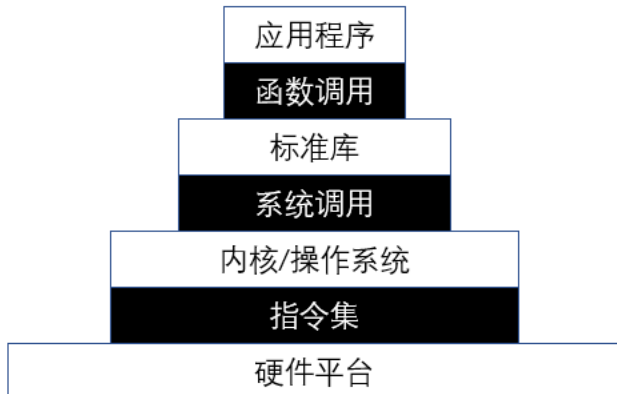
选题方向	大实验题目
驱动	IO 复用
rust	Audio support for rCore
内核语言	编译原理/操作系统综合实验
错误分析	在 ucore 获得稳定触发竞争条件的漏洞样本
行为分析	Program Analysis via Memory Access Patterns
微内核	调研 Fuchsia 的微内核，尝试 rcore 微内核的修改
内核可加载模块	rethink 用户/内核态
模拟器	操作系统中常用算法的性能分析及优化
教学实验设计	对简易版 rcore 的进一步维护和更新

OS 实验内容

lab1: hello-world OS

Lab1: hello-world OS

在裸机上的执行环境，让应用与硬件隔离，简化了应用访问硬件的难度和复杂性。

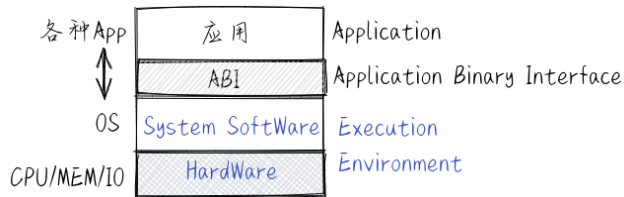


OS 实验内容

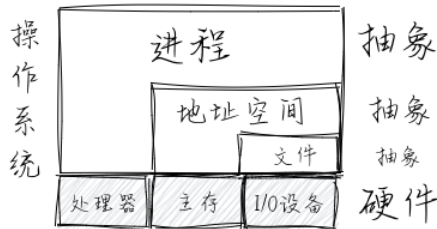
lab1: hello-world OS

Lab1: hello-world OS

在裸机上的执行环境，让应用与硬件隔离，简化了应用访问硬件的难度和复杂性。



运行应用程序

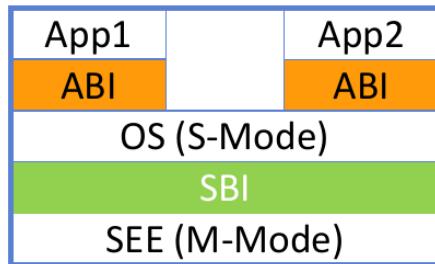
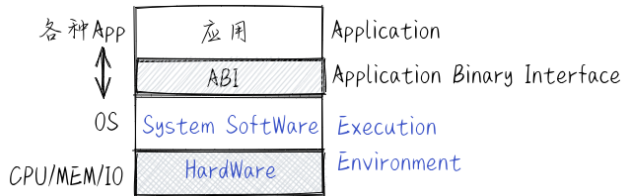


OS 实验内容

lab1: hello-world OS

Lab1: hello-world OS

在裸机上的执行环境，让应用与硬件隔离，简化了应用访问硬件的难度和复杂性。



OS 实验内容

lab1: hello-world OS 硬件

应用所在的硬件环境: QEMU RISC-V 64 虚拟计算机或 K210 RISC-V 64 物理计算机

- 输入输出和存储外设: 16550A UART 和 virtio-block 设备
- 内存: 可参数化的 RAM 内存 (8MB)
- CPU: 可配置的多核 RV64GC M/S/U mode CPU(1 or 2)

OS 实验内容

lab1: hello-world OS 物理内存

```
// qemu/hw/riscv/virt.c
static const struct MemmapEntry {
    hwaddr base;
    hwaddr size;
} virt_memmap[] = {
    [VIRT_DEBUG] = { 0x0, 0x100 },
    [VIRT_MROM] = { 0x1000, 0xf000 },
    [VIRT_TEST] = { 0x100000, 0x1000 },
    [VIRT_RTC] = { 0x101000, 0x1000 },
    [VIRT_CLINT] = { 0x2000000, 0x10000 },
    [VIRT_PCIE_PIO] = { 0x3000000, 0x10000 },
    [VIRT_PLIC] = { 0xc000000, VIRT_PLIC_SIZE(VIRT_CPUS_MAX * 2) },
    [VIRT_UART0] = { 0x10000000, 0x100 },
    [VIRT_VIRTIO] = { 0x10001000, 0x1000 },
    [VIRT_FLASH] = { 0x20000000, 0x4000000 },
    [VIRT_PCIE_ECAM] = { 0x30000000, 0x10000000 },
    [VIRT_PCIE_MMIO] = { 0x40000000, 0x40000000 },
    [VIRT_DRAM] = { 0x80000000, 0x0 },
```

OS 实验内容: 硬件: RISC-V CPU 启动



- RISC-V CPU 启动过程
 - 初始化 CPU/Regs
 - 初始化内存
 - 初始化基本外设
 - 执行 ROM 中固化的代码
- 出处: <https://github.com/qemu/qemu>

```

/* Default Reset Vector address */
#define DEFAULT_RSTVEC    0x1000

static void riscv_any_cpu_init(Object *obj)
{
    CPURISCVState *env = &RISCV_CPU(obj)->env;
    set_misa(env, RVXLEN | RVI | RVM | RVA | RVF | RVD | RVC | RVU);
    set_priv_version(env, PRIV_VERSION_1_11_0);
    set_resetvec(env, DEFAULT_RSTVEC);
}

static void riscv_cpu_reset(CPUState *cs)
{
    RISCVCPU *cpu = RISCV_CPU(cs);
    RISCVCPUClass *mcc = RISCV_CPU_GET_CLASS(cpu);
    CPURISCVState *env = &cpu->env;

    mcc->parent_reset(cs);
#ifdef CONFIG_USER_ONLY
    env->priv = PRV_M;
    env->mstatus &= ~(MSTATUS_MIE | MSTATUS_MPRV);
    env->mcause = 0;
    env->pc = env->resetvec;
#endif
    cs->exception_index = EXCP_NONE;
    env->load_res = -1;
    set_default_nan_mode( val: 1, &env->fp_status);
}

```

OS 实验内容: 硬件: RISC-V CPU 启动



- RISC-V CPU 启动过程-初始化内存

```
static const struct MemmapEntry {  
    hwaddr base;  
    hwaddr size;  
} virt_memmap[] = {  
    [VIRT_DEBUG] = { .base: 0x0, .size: 0x100 },  
    [VIRT_MROM] = { .base: 0x1000, .size: 0x11000 },  
    [VIRT_TEST] = { .base: 0x100000, .size: 0x1000 },  
    [VIRT_CLINT] = { .base: 0x2000000, .size: 0x10000 },  
    [VIRT_PLIC] = { .base: 0xc000000, .size: 0x4000000 },  
    [VIRT_UART0] = { .base: 0x10000000, .size: 0x100 },  
    [VIRT_VIRTIO] = { .base: 0x10001000, .size: 0x1000 },  
    [VIRT_FLASH] = { .base: 0x20000000, .size: 0x4000000 },  
    [VIRT_DRAM] = { .base: 0x80000000, .size: 0x0 },  
    [VIRT_PCIE_MMIO] = { .base: 0x40000000, .size: 0x40000000 },  
    [VIRT_PCIE_PIO] = { .base: 0x03000000, .size: 0x00010000 },  
    [VIRT_PCIE_ECAM] = { .base: 0x30000000, .size: 0x10000000 },  
};
```

```

static void riscv_virt_board_init(MachineState *machine)
{
    const struct MemmapEntry *memmap = virt_memmap;
    RISCVVirtState *s = RISCV_VIRT_MACHINE(machine);
    MemoryRegion *system_memory = get_system_memory();
    MemoryRegion *main_mem = g_new(MemoryRegion, n_structs: 1);
    MemoryRegion *mask_rom = g_new(MemoryRegion, n_structs: 1);
    char *plic_hart_config;
    size_t plic_hart_config_len;
    target_ulong start_addr = memmap[VIRT_DRAM].base;

    .....
    sifive_clint_create(memmap[VIRT_CLINT].base,
        memmap[VIRT_CLINT].size, smp_cpus,
        sip_base: SIFIVE_SIP_BASE, timecmp_base: SIFIVE_TIMECMP_BASE, time_base: SIFIVE_TIME_BASE);

    .....
    serial_mm_init(system_memory, memmap[VIRT_UART0].base,
        it_shift: 0, irq: qdev_get_gpio_in(DEVICE(obj: s->plic), n: UART0_IRQ), baudbase: 399193,
        chr: serial_hd(i: 0), end: DEVICE_LITTLE_ENDIAN);

    virt_flash_create(s);

```

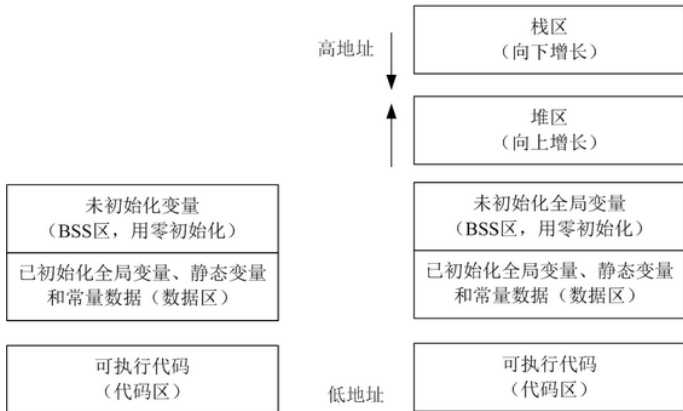
OS 实验内容: 硬件: RISC-V CPU 启动-ROM 初始化代码

```
[VIRT_MROM] = { .base: 0x1000, .size: 0x11000 },
uint32_t reset_vec[8] = {
    0x00000297, /* 1: auipc t0, %pcrel_hi(dtb) */
    0x02028593, /*      addi a1, t0, %pcrel_lo(1b) */
    0xf1402573, /*      csrr a0, mhartid */
#ifdef TARGET_RISCV32
    0x0182a283, /*      lw t0, 24(t0) */
#elif defined(TARGET_RISCV64)
    0x0182b283, /*      ld t0, 24(t0) */
#endif
    0x00028067, /*      jr t0 */
    0x00000000,
    start_addr, /* start: .dword */
    0x00000000,
    /* dtb: */
};
target_ulong start_addr = memmap[VIRT_DRAM].base;
[VIRT_DRAM] = { .base: 0x80000000, .size: 0x0 },
```


OS 实验内容

lab1: hello-world OS 软件

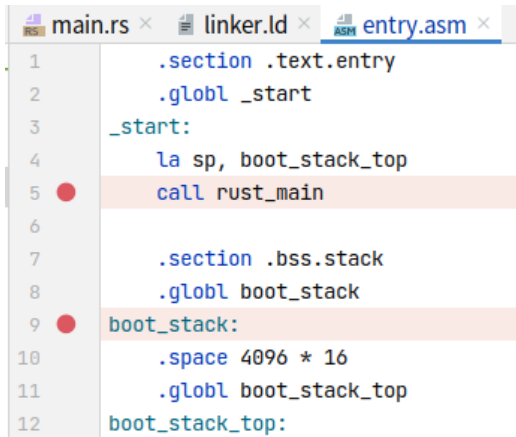
编译后的应用与运行时的应用



OS 实验内容

lab1: hello-world OS 软件

编译后的应用与运行时的应用



```
main.rs × linker.ld × entry.asm ×
1      .section .text.entry
2      .globl _start
3      _start:
4      la sp, boot_stack_top
5      call rust_main
6
7      .section .bss.stack
8      .globl boot_stack
9      boot_stack:
10     .space 4096 * 16
11     .globl boot_stack_top
12     boot_stack_top:
```

OS 实验内容

lab1: hello-world OS 软件

编译后的应用与运行时的应用

```
main.rs x linker.ld x entry.asm x
1 OUTPUT_ARCH(riscv)
2 ENTRY(_start)
3 BASE_ADDRESS = 0x80020000;
4
5 SECTIONS
6 {
7     . = ALIGN(4K);
8     . = BASE_ADDRESS;
9     skernel = .;
10    stext = .;
11    .text : {
12        *(.text.entry)
13        *(.text)
14    }
```

OS 实验内容

lab1: hello-world OS 软件

编译后的应用与运行时的应用



```
1  .....
2
3  #[no_mangle]
4  #[link_section=".text.entry"]
5  extern "C" fn rust_main() {
6      //extern "C" fn _start() {
7          println!("Hello, world!");
8          shutdown();
9      }
```

OS 实验内容

lab1: hello-world OS 软件



```
1  ...
2  pub fn console_putchar(c: usize) {
3      syscall( id: SBI_CONSOLE_PUTCHAR, args: [c, 0, 0]);
4  }
5
6  pub fn shutdown() -> ! {
7      syscall( id: SBI_SHUTDOWN, args: [0, 0, 0]);
8      panic!("It should shutdown!");
9  }
10
11 fn syscall(id: usize, args: [usize; 3]) -> isize {
12     let mut ret: isize;
13     unsafe {
14         llvm_asm!("ecall
15                 : "={x10}" (ret)
16                 : "{x10}" (args[0]), "{x11}" (args[1]), "{x12}" (args[2]), "{x17}" (id)
17                 : "memory"
18                 : "volatile"
19                 );
20     }
21     ret
22 }
```

OS 实验内容

lab1: hello-world OS 软件

问题

- 这个 hello-world 的执行环境包含啥？
- 在这个例子中操作系统是啥？
- 这个软件与我们通常的 app 相比，有哪些相同与不同的地方？