

武汉大学计算机学院教学实验报告

课程名称	操作系统实践			成绩		教师签名	
实验名称	Lab1 —— RISC-V 引导与裸机启动			实验序号	1	实验日期	
姓名	夏盛宇	学号	2023302111351	专业	计科	年级-班	2023-7

一、实验目的及实验内容

(本次实验所涉及并要求掌握的知识；实验内容；必要的原理分析)

实验目的：通过参考 xv6 的启动机制，理解并实现最小操作系统的引导过程，最终在 QEMU 中输出 "Hello OS"。

具体目标包括：掌握 RISC-V 裸机启动流程。学会编写启动汇编、链接脚本。理解 BSS 段清零、栈初始化的重要性。实现最小串口驱动并输出字符串。熟悉 QEMU + GDB 调试方法。

实验内容及原理分析 (架构设计说明)：

主要模块包括：

boot：引导代码，负责栈初始化、BSS 段清零、跳转到 C 语言入口。

lib：基础库，提供 打印输出、自旋锁 等功能实现。

dev：外设驱动，如 UART 串口输出。

proc：进程与 CPU 抽象，提供 mycpu / mycpuid。

kernel.ld：链接脚本，规定内存布局并导出符号。

完成情况：

- 在 `entry.S` 中实现 per-hart 栈、BSS 清零与跳转逻辑。

- 补全 `kernel.ld`、`start.c`、`uart.c`、`print.c` 等核心文件，并为 printf 系列加锁。

- QEMU virt 平台多次运行 `make qemu` 均稳定输出 "Hello OS"，无未完成项。

二、实验环境及实验步骤

(本次实验所使用的器件、仪器设备等的情况；具体的实验步骤)

实验环境：

- 硬件：x86_64 主机（2 核）
- 操作系统：Ubuntu 24.04 LTS
- 交叉工具链：riscv64-unknown-elf-gcc 12.2.0
- 模拟器：QEMU System riscv64 8.2.2
- 调试器：gdb-multiarch 15.0.50.20240403

代码框架：

```
whu-oslab-lab1
├── include
│   ├── uart.h
│   └── lib
│       ├── print.h
│       └── lock.h
└── proc
    ├── cpu.h
    └── proc.h
```

```

common.h
memlayout.h
riscv.h

kernel
├── boot
│   ├── main.c
│   ├── start.c
│   ├── entry.S
│   └── Makefile
├── dev
│   ├── uart.c
│   └── Makefile
└── lib
    ├── print.c
    ├── spinlock.c
    └── Makefile
├── proc
│   ├── pro.c
│   └── Makefile
└── Makefile
└── kernel.ld

picture
└── *.png
Makefile
common.mk
README.md
Report.md

```

实验步骤:

环境搭建: 安装 Ubuntu、QEMU、交叉编译工具链 (riscv64-unknown-elf-gcc), 使用 git 初始化仓库并整理目录结构。

修改 entry.S: 添加 BSS 清零循环, 确保全局变量初始化。

编写 kernel.ld: 导出 edata、end 符号供汇编清零使用。

补全 print.c: 定义 panicked 全局变量, 实现 panic、puts、assert, 并参考 xv6 加入自旋锁保护。

参考 xv6 实现 spinlock.c: 编写 acquire/release, 保证多核同步。

实现 proc.c / cpu.h: 定义 struct cpu 和 mycpuid, 封装 tp 寄存器读取。

修改 start.c: 初始化 UART 并调用 puts("Hello OS")。

系统架构设计:

整体结构按“加载→汇编引导→C 初始化→驱动输出”四个阶段划分, 各模块与 xv6 对应部分一一映射, 方便扩展。

- boot: 设置 per-hart 栈、清零 BSS, 跳转到 C 入口。

- lib: 提供 print、自旋锁与 assert, 保证串口输出线程安全。

- dev: 实现 16550A UART 驱动, 统一寄存器访问。
 - proc: 定义 `struct cpu` 与 `mycpuid()`, 为多核调度预留接口。
- 与 xv6 对比: 精简了中断、调度和分页支持, 但保留 per-hart 栈与锁语义, 后续扩展时无需推翻现有设计。

关键数据结构

核心数据结构兼顾了硬件抽象与同步需求:

...

```
struct cpu {
    int id;
    int started;
};

typedef struct spinlock {
    int locked;
    char *name;
    int cpuid;
} spinlock_t;

__attribute__((aligned(16))) uint8 CPU_stack[4096 * NCPU];
extern int panicked;
...
```

- struct cpu: 通过 tp 寄存器获取 hartid, 用于锁归属检查。
- spinlock_t: 记录锁状态与持有者, panic 时可打印锁名, 便于定位死锁。
- CPU_stack/panicked: 放在 BSS 段并由 `_entry` 清零, 确保所有核初始状态一致。

核心算法与流程

1. `_entry` 读取 `mhartid`, 为每个 hart 分配 4KB 栈, 并循环清零 `[edata,end)`。
2. 跳转到 `start()` 后只允许 hart0 初始化串口、输出 “Hello OS”, 其余核立即 `wfi`。
3. `print.c` 中 `puts/printf` 通过自旋锁保护, `uart_putc_sync` 轮询 LSR 寄存器, 保证裸机串口输出可靠。
4. 若未来加入中断/调度, 只需在 `start()` 中扩展 S 模式切换和 trap 向量, 不必重写引导流程。

三、实验过程分析

(详细记录实验过程中发生的故障和问题, 进行故障分析, 说明故障排除的过程及方法。根据具体实验, 记录、整理相应的数据表格、绘制曲线、波形等)

启动流程总结:

QEMU 加载 kernel → _entry 设置栈 → 清零 BSS → start() → 初始化 UART → puts("Hello 05")。

调试流程:

打开一个终端，执行 make qemu-gdb。

打开另外一个终端，执行 riscv64-unknown-elf-gdb kernel.elf 或 gdb-multiarch kernel.elf。

连接到 QEMU: target remote :1234。

设置断点、运行等：b _start、b main、c、si (单步执行)、info registers (查看寄存器)、x/16x 0x80000000 (查看内存)。

遇到的困难和解决方案:

问题	解决方案
找不到交叉编译器 riscv64-linux-gnu-gcc	安装 gcc-riscv64-unknown-elf 并修改 Makefile。
多核同时打印导致输出乱序	只允许 hart0 打印，或使用 spinlock 保护 printf 等打印输出函数。
entry.S 文件名后缀错误 (小写 's' 导致 make 报错)	将文件名后缀改为大写的 'S'。
字符串常量地址太远导致编译器报错 (relocation 溢出)	在链接脚本中将 .rodata 紧跟 .text，使地址更近，避免溢出。
GDB 调试时 (gdb) c The program is not being run.	发现是由于输入 target remote :1234 未连接成功，排查发现是输入指令格式问题（多了一个换行符）。

四、实验结果总结

(对实验结果进行分析，完成思考题目，总结实验的新的体会，并提出实验的改进意见)

功能测试结果：

运行 make qemu 后，成功输出 "Hello OS"。

```
xsy@XSY:~/whu-oslab-lab1$ make qemu
make build --directory=proc/
make[2]: Entering directory '/home/xsy/whu-oslab-lab1/kernel/proc'
riscv64-linux-gnu-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -gdwarf-2
-MD -mcmodel=medany -ffreestanding -fno-common -nostdlib -mno-relax -I. -fno-stack-protector -fno-pie -no-pie -I ../../include -c proc.c
make[2]: Leaving directory '/home/xsy/whu-oslab-lab1/kernel/proc'
ls: cannot access './/*/*/*.o': No such file or directory
riscv64-linux-gnu-ld -z max-page-size=4096 -T kernel.lds -o ./kernel-qemu ./boot/entry.o ./boot/main.o ./boot/start.o ./dev/uart.o ./lib/print.o ./lib/spinlock.o ./proc/proc.o
riscv64-linux-gnu-ld: warning: ./kernel-qemu has a LOAD segment with RWX permissions
make[1]: Leaving directory '/home/xsy/whu-oslab-lab1/kernel'
qemu-system-riscv64 -machine virt -bios none -kernel ./kernel-qemu -m 128M -smp 2 -nographic
Hello OS
Hello OS
```

异常测试及分析：

BSS 段清零：发现去掉 entry.S 中的 BSS 清零也能正常输出，这是因为 QEMU 的 ELF loader 自动完成了 .bss 段的清零。但是，从 OS 启动的正确性来说，清零 .bss 仍是必须的，否则更换加载方式（裸 bin / 真机）会出问题。

多核打印：去除 start.c 中 if(hartid == 0) 的判定后，会重复输出 "Hello OS"，次数取决于核的数目。因为在打印中使用了锁保护机制，输出的字符没有混乱。

思考题解答：

(1) 启动栈的设计：

栈大小：考虑到函数调用深度，采用 4kb 大小的栈。

栈溢出：溢出会造成程序崩溃。

检测：可在栈底添加一个魔数，在主循环里轮询，若魔数改变则 panic("stack overflow")。

(2) BSS 段清零：

不清零现象：全局变量的值是随机的。

可省略情况：每一个全局变量均进行了初始化，或 ROM 引导加载器（如 OpenSBI）将其全部初始化为 0。

(3) 与 xv6 的对比：

简化部分：本实现比 xv6 简化了对多核、中断以及内存管理的支持。Lab1 仅支持单核/双核和串口输出。

简化带来的问题：一旦加入中断/异常/系统调用，必须补全页表、栈隔离、锁，否则可能出现串口输出乱码、寄存器被覆盖。若想运行用户程序，需加入 sstatus.SPP 切换、页表隔离。若想支持多核，单栈模型会崩溃，必须使用 per-hart stack。

(4) 错误处理：

UART 初始化失败处理：裸机无返回地址，不能 return -1，只能原地死循环 + 闪灯/蜂鸣。

最小错误显示机制：由于 UART 初始化失败，不能依赖 UART，此时可以依据一个 LED 灯，通过其闪烁状态去识别错误。

实验总结：

通过本实验，掌握了 RISC-V 裸机启动流程，学会了如何从 _start 设置栈、清零 BSS，

再跳转到 C 函数，并通过串口打印输出验证结果。通过 QEMU + GDB，可以精确调试每一步。最终成功实现了最小 OS 输出 "Hello OS"。

在实验中，我通过向 AI 提问解决了如 entry.S 文件后缀大小写错误、编译器 relocation 溢出错误，以及 GDB 连接问题。这加强了对交叉编译环境和裸机内存布局的理解，特别是了解到 QEMU 的 ELF loader 可能会“隐藏” BSS 清零的重要性。