学习链接：<https://github.com/chen-p/xv6-lab> <https://github.com/riscv/riscv-gnu-toolchain> https://github.com/haiboshi2003/Mit-xv62020

一、实验环境搭建

1. 工具链准备

RISC-V 工具链：从 [riscv-gnu-toolchain](https://github.com/riscv/riscv-gnu-toolchain) 安装 `riscv64-unknown-elf-gcc` 编译器。

QEMU 模拟器：编译支持 `riscv64-softmmu` 的版本，用于运行 xv6。

- 验证安装：

```bash

$ make qemu

# 成功启动后显示 xv6 命令行

```

二、实验核心内容与收获

实验 1：系统调用（System Calls）

1. 目标：添加自定义系统调用（如 `trace`，跟踪进程的系统调用）。

2. 实现步骤：

注册系统调用号：在 `kernel/syscall.h` 定义 `SYS\_trace`。

实现调用逻辑：修改 `kernel/sysproc.c`，添加 `sys\_trace()` 函数。

用户接口：在 `user/user.h` 声明 `int trace(int);`，并在 `user/usys.pl` 生成汇编入口。

3. 难点与解决：

参数传递：通过 `argint()` 从用户态读取参数。

进程状态保存：在 `struct proc` 中添加 `trace\_mask` 字段，记录跟踪标记。

4. 收获：

理解用户态与内核态交互机制。

掌握系统调用表（`syscalls[]`）的注册与调度流程。

实验 2：进程调度（Scheduling）

1. 目标：实现多级反馈队列（MLFQ）调度算法。

2. 关键修改：

进程控制块：在 `struct proc` 中增加优先级字段 `priority` 和时间片 `timeslice`。

调度器逻辑：修改 `kernel/proc.c` 的 `scheduler()`，按优先级选择进程。

3. 调试技巧：

打印日志：在 `kernel/printf.c` 中添加调试输出，观察进程切换顺序。

QEMU 监视器：使用 `Ctrl+a c` 进入 QEMU 控制台，查看寄存器状态。

4. 收获：

深入理解上下文切换（`swtch()`）和时钟中断（`timer interrupt`）的作用。

实践调度算法对系统性能的影响。

实验 3：虚拟内存（Virtual Memory）

1. 目标：实现懒分配（Lazy Allocation）优化，延迟物理页分配至实际访问时。

2. 实现步骤\*\*：

修改 `sbrk()`：在 `kernel/sysproc.c` 中取消立即分配，仅更新进程的堆顶地址 `p->sz`。

3. 边界条件：

非法地址检测：检查访问地址是否在 `[0, p->sz)` 范围内。

COW 优化：结合写时复制（Copy-on-Write）减少内存拷贝。

4. 收获：

掌握页表（`pagetable\_t`）的遍历与修改方法。

理解缺页异常处理与物理内存管理的协同机制。

三、关键调试工具与技巧

1. GDB 调试：

```bash

$ make qemu-gdb # 启动 QEMU 并等待 GDB 连接

$ gdb-multiarch kernel/kernel

(gdb) target remote localhost:26000

(gdb) b syscall # 在系统调用入口设断点

```

四、学习总结

1. 操作系统核心概念\*\*：

隔离性：通过虚拟内存与进程模型隔离用户程序。

并发控制：锁（`spinlock`）、信号量（Semaphore）的实现与应用。

持久化：文件系统（`inode`, `log`）的数据一致性保障。

2. 能力提升：

代码阅读：熟悉大型 C 项目（如 xv6）的模块化结构。

系统思维：从硬件抽象（RISC-V）到软件策略（调度算法）的全栈视角。

典型问题与解决

问题 1：`make qemu` 编译失败，提示 `riscv64-unknown-elf-gcc not found`。

解决：检查工具链路径，在 `Makefile` 中设置 `TOOLPATH=/opt/riscv/bin`。

问题 2：系统调用返回错误值 `-1`。

解决：在 `kernel/syscall.c` 的 `syscall()` 函数中打印 `num` 和参数，验证调用号与参数传递。

通过 xv6-lab，我不仅掌握了操作系统的核心原理，更培养了从代码到设计的系统性工程能力。这种“动手学内核”的实践，是理解现代计算机系统不可或缺的一课。