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	Lab2 —— 添加内核 printf 与清屏功能实现			实验序号	2	实验日期	
姓名	夏盛宇	学号	2023302111351	专业	计科	年级-班	23-7

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：

（1）理解操作系统内核为什么必须自己实现基本的输出功能（如 printf 和清屏）。

（2）掌握内核中 printf 的设计方法，包括格式解析、数字转换和字符串输出 。

（3）了解 BSS 段清零的重要性，以及启动阶段的最小输出环境搭建 。

（4）通过实现清屏与定位功能，熟悉 ANSI 转义序列或显存写入机制 。

（5）完成综合测试与优化，思考性能瓶颈和改进方向 。

实验内容及原理分析 (架构设计说明)：

本实验基于 RISC-V 架构，在 Lab1 基础上添加格式化输出和终端控制功能 。

（1）内核最小输出环境搭建 ：UART 驱动：实现 uart_putc_sync，保证字符稳定发送到串口 。BSS 清零：在 entry.S 中利用链接脚本导出的 edata/end 清空全局变量区 。多核栈设置：为每个 hart 分配独立的 4 KiB 栈，避免并发冲突 。

（2）printf 的实现 ：

格式支持：支持 %d、%x/%p、%s、%c、%% 。

数字转换：采用非递归除法取余，避免栈过深，并支持负数和 INT_MIN 。

并发安全：通过自旋锁 (spinlock) 保护，保证多核同时输出不交错 。

清屏与控制 ：基于 ANSI 转义序列实现 ——clear_screen() → ESC[2J ESC[H 清屏并复位光标 。goto_xy(row,col) → ESC[row;colH 定位光标 。set_color(fg,bg) / reset_color() → 修改前景/背景色

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：

开发平台：Ubuntu 24.04 + RISC-V 工具链 (riscv64-linux-gnu-gcc) 。

模拟器：QEMU (qemu-system-riscv64 -machine virt -nographic) 。

代码框架：whu-oslab-lab2 目录结构 。

代码框架：

```
whu-oslab-lab2
├── 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 (实现步骤记录)：

（1）最小环境实现：在 entry.S 中利用 edata/end 符号实现 BSS 清零 。在 start.c 中为主核设置 uart_init 。

（2）printf 核心逻辑实现：在 lib/print.c 中实现数字转字符串的非递归函数 。实现 printf 的格式解析和分派逻辑，支持所有预定格式 。

（3）并发保护实现：在 printf 中加入自旋锁 spinlock 对输出流进行保护 。

（4）终端控制实现：根据 ANSI 转义序列标准，实现 clear_screen()、goto_xy(row,col)、set_color(fg,bg) 等函数 。

综合测试：在 start.c 的主核中调用实现的所有功能进行演示——

（1）调用 clear_screen() 清屏 。

（2）测试 set_color() 输出绿色文字 。

（3）测试边界条件 INT_MIN 和 INT_MAX 。

（4）测试 %d, %x, %s, %c 。

（5）调用 goto_xy(6, 10) 进行光标移动测试 。

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）

调试方法与过程：

分模块调试：首先测试单字符输出 ，然后测试数字转换 ，接着测试字符串处理 ，最后综合格式串 。

错误恢复：遇到未知格式时，直接原样输出 %x，保证内核不会崩溃 40。UART 初始化失败时，调用 panic("uart init failed") 。

遇到的问题与解决方案 (故障分析及排除):

问题	故障分析	解决方案
数字转换错误 (如负数显示为正数)	print_number 中未处理负数情况（直接使用无符号数转换），INT_MIN 转换时溢出 。	在 print_number 中先判断符号 ，转为正数处理 ；INT_MIN 特殊处理，直接硬编码输出 。
清屏或光标定位无效	检查发现是转义序列格式错误（如缺少 [或使用错误参数） 。	使用标准转义序列格式 。
光标跳转后覆盖原有输出	原本输出 Moving cursor to (6,10) and printing there... 的位置被后面的输出覆盖 。	将光标移动位置调整为 goto_xy(6, 10)，避免覆盖。

功能测试结果：

运行 make qemu 后，终端成功显示出清屏的功能。

终端历史输出记录显示字体颜色发生改变，成功输出了绿色的字符串 。

边界测试（INT_MIN、INT_MAX）结果成功输出 。

四、实验结果总结

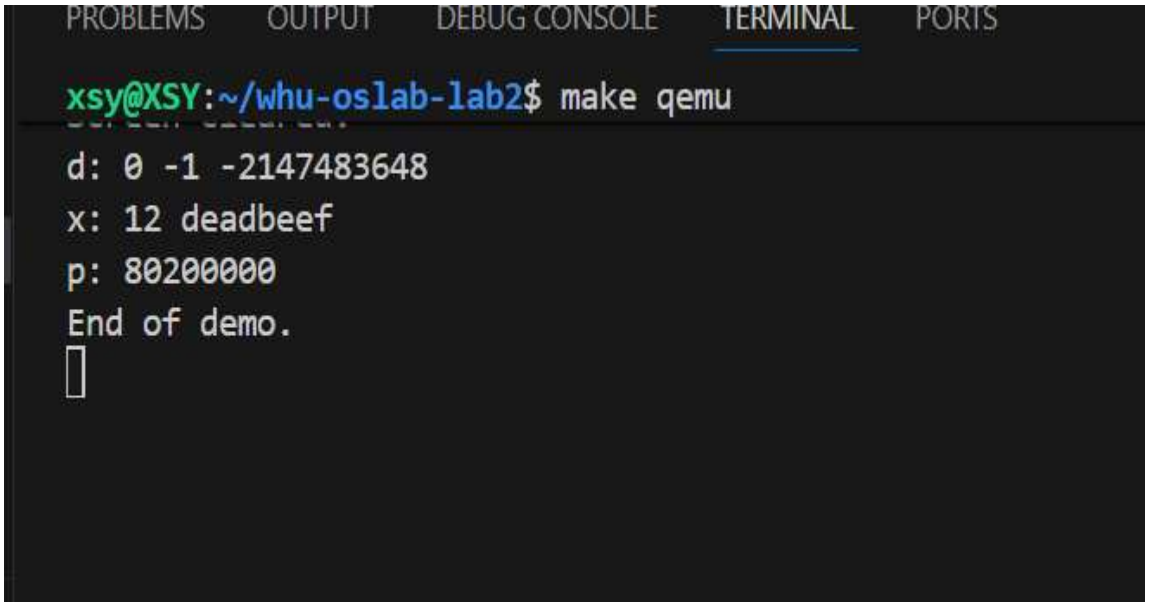
（对实验结果进行分析，完成思考题目，总结实验的新的体会，并提出实验的改进意见）

实验总结：

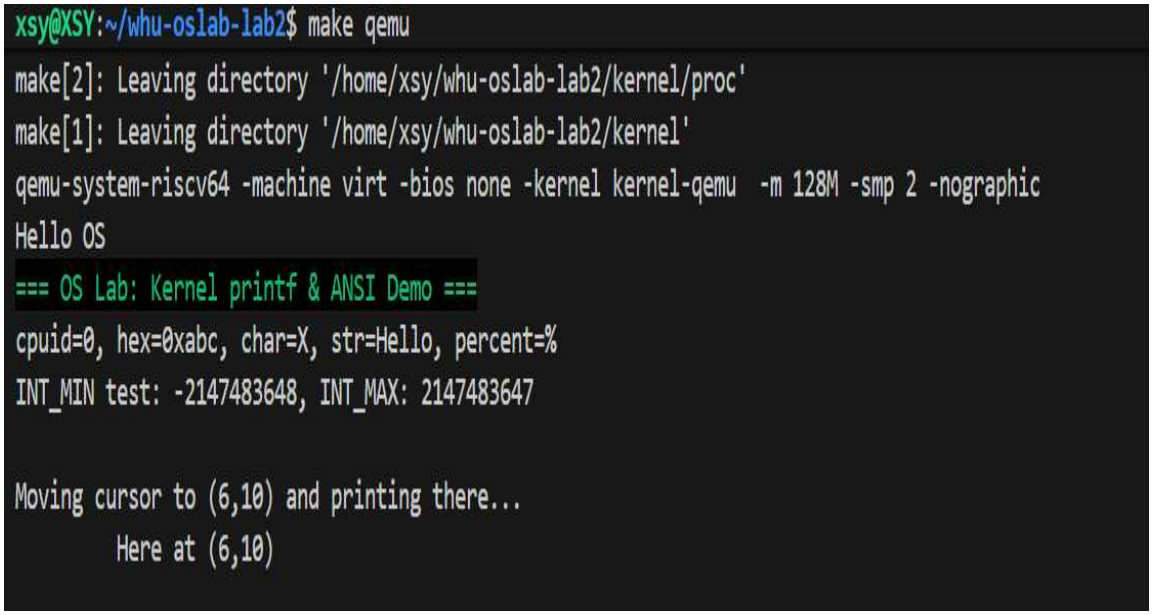
本实验成功地在内核早期搭建了最小输出环境，并用 C 语言实现了一个轻量级的、并发安全的 printf 。通过实现清屏和定位功能，掌握了 ANSI 转义序列的应用 。实验过程中，认识到 BSS 清零、锁保护和分层设计的重要性 。本实验为后续进程调度和系统调用实验提供了必要的调试工具 。

展示如下：

运行 make qemu 后出现，后面的 d，x，p 是在清屏功能后重新输出的内容，之前输出的已经被清屏。可以看到成功达到清屏效果。



像上滑动查看历史输出显示如下：



可以看见 OS Lab 字体变成绿色，同时光标位置移动到（6，10），INT_MAX 和 INT_MIN 测试也都成功。

思考题回答：

（1)为什么内核要自己实现 printf?

在内核启动或出错时，用户态和标准库尚未建立，必须有最小的输出机制用于调试和诊断。内核需要直接控制硬件（如 UART、显存），而用户态 printf 依赖于系统调用，这在早期启动阶段是不可用的 。

（2）为什么需要清屏？

避免多次输出堆叠影响观察，尤其是教学/调试时更直观 。清屏功能是实现更复杂交互界面（如菜单系统、日志查看器）的基础 。

（3）架构设计：为什么分层？

分层目的：驱动层（UART）只负责单字符输入/输出 ；上层（print.c）实现格式化 ；调用层（start.c）演示逻辑 。

分层益处：便于替换或扩展输出设备（如 VGA），代码更易于测试和维护，每一层职责明确，耦合度低 。

（4）算法选择：数字转字符串不用递归的原因及不用除法实现进制转换

不用递归：避免早期内核栈消耗过大 ，同时迭代实现效率更高 。递归在栈空间有限的内核环境中容易导致栈溢出，且调用开销较大 。

不用除法实现：可以使用查表法或减法替代法（通过重复减法模拟除法运算）进行进制转换 。

（5）性能优化：瓶颈与改进？

瓶颈：主要在循环调用 uart_putc_sync 。

改进：引入缓冲区、批量写、甚至中断驱动 。可以针对常用输出路径进行内联优化，减少函数调用开销 。

（6）错误处理策略

printf 遇到空指针时，先判断，然后输出字符串 "(null)" 。

格式字符串出错时遇到未定义的字符直接输出 。

实验的改进意见 (优化思路)：

批量输出：把字符串先缓存在临时数组中，一次性写 UART FIFO，以提高效率 。

查表优化：十六进制转换可用预定义查表，减少计算 。

格式解析优化：简单场景下可用有限状态机代替复杂 switch，提高解析速度 。