**武汉大学计算机学院**

**本科生课程设计报告**

# **实验二：内核printf与清屏功能实现**

专 业 名 称 ：计算机科学与技术

课 程 名 称 ：操作系统实践A

指 导 教 师 ：李祖超 副教授

学 生 学 号 ：2023302111416

学 生 姓 名 ：肖茹琪

二○二五年九月

**郑 重 声 明**

本人呈交的设计报告，是在指导老师的指导下，独立进行实验工作所取得的成果，所有数据、图片资料真实可靠。尽我所知，除文中已经注明引用的内容外，本设计报告不包含他人享有著作权的内容。对本设计报告做出贡献的其他个人和集体，均已在文中以明确的方式标明。本设计报告的知识产权归属于培养单位。



本人签名： 日期： 2025.9.29

摘 要

本次实验旨在深入理解操作系统内核输出系统的设计原理，通过分析xv6的printf实现机制，独立设计并实现一个功能完整的内核级格式化输出系统。实验内容包括理解内核输出特殊性、实现格式化输出、支持清屏清行功能、光标精确定位以及多颜色文本输出，并构建合理的分层架构。通过实现硬件驱动层与格式化层的清晰接口，成功在QEMU模拟的RISC-V平台上实现了支持%d、%x、%s、%c、%%等格式符的printf函数，基于ANSI转义序列实现了终端清屏、清行、光标控制以及8种颜色的文本输出功能。实验结果表明，系统能够稳定运行，输出效果准确美观，具备良好的可扩展性与可维护性，为后续操作系统内核开发提供了重要的输出基础。

**关键词：**RISC-V；操作系统；内核输出；printf；清屏；光标定位；颜色输出

目录

[1 实验目的和意义 5](#_Toc24766)

[1.1 实验目的 5](#_Toc20362)

[1.2 实验意义 5](#_Toc23296)

[2 实验准备 5](#_Toc7103)

[2.1 阅读 kernel/entry.S，回答以下问题： 5](#_Toc12308)

[2.2 理解分层设计： 6](#_Toc31525)

[2.3 深入思考： 7](#_Toc18812)

[3 实验原理与设计 8](#_Toc16406)

[3.1 系统架构图设计 8](#_Toc4484)

[3.2 各层接口定义 8](#_Toc17978)

[3.3 与xv6设计的异同分析 9](#_Toc4871)

[4 实验步骤与实现 1](#_Toc5128)0

[4.1 实验步骤记录 10](#_Toc31237)

[4.2 核心关键代码理解总结 10](#_Toc29256)

[5 实验测试与结果 12](#_Toc380)

[5.1 基础功能测试 13](#_Toc11485)

[5.2 边界情况测试 14](#_Toc7409)

[5.3 颜色输出测试 15](#_Toc15998)

[5.4 光标控制测试 16](#_Toc27433)

[5.5 清屏清行测试 17](#_Toc15651)

[6 思考题 19](#_Toc23415)

[6.1 架构设计 19](#_Toc12063)

[6.2 算法选择 19](#_Toc9710)

[6.3 性能优化 20](#_Toc13090)

[6.4 错误处理 20](#_Toc8658)

[7 实验总结 22](#_Toc4426)

**1 实验目的和意义**

**1.1** **实验目的**

本实验旨在通过动手实现一个RISC-V最小内核的引导过程，深入理解操作系统从硬件上电到内核初始化的完整流程。具体目标是分析xv6系统的启动设计，亲自编写汇编与C代码完成栈设置、BSS段清零和串口初始化等核心步骤，最终在QEMU模拟器上成功引导内核并实现串口输出，验证裸机启动的正确性。

**1.2** **实验意义**

本实验是理解计算机系统底层工作原理的关键实践。通过裸机编程，能够打通软硬件界限，揭示硬件上电后协同工作直至软件获得控制权的完整过程。这一过程为后续学习进程、内存等复杂操作系统机制奠定了坚实基础，同时培养了从系统全局视角分析问题与实现解决方案的综合能力。

**2 实验准备**

在正式开始实验之前，首先完成“任务1：深入理解xv6输出架构”，即阅读理解xv6的相关代码，理解其设计的核心思想。

**2.1** **研读 printf.c 中的核心函数，回答下列问题**

**2．1．1** **printf() 如何解析格式字符串？**

xv6使用状态机方式解析格式字符串，具体解析策略如下：

1. 逐字符扫描，遇到 % 进入格式解析模式
2. 预读后续2个字符（c0, c1, c2）进行最长匹配
3. 使用 va\_arg(ap, 类型) 提取可变参数

**2．1．2** **printint() 如何处理不同进制转换？**

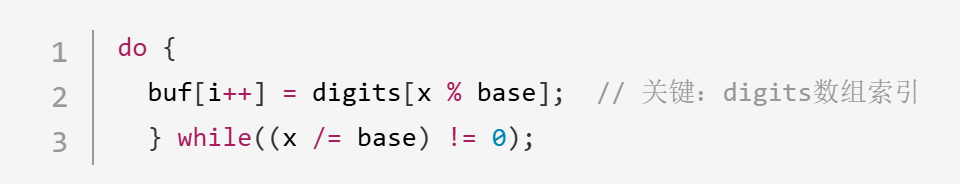


图 1 printf.c文件部分代码1

正如以上代码，预定义digits[] = "0123456789abcdef"包含所有进制字符，再利用x % base得到当前最低位数字、利用x /= base移除已处理的最低位，循环直到x == 0即完成转换。

**2．1．3** **负数处理有什么特殊考虑？**



图 2 printf.c文件部分代码2

正如以上代码，代码中统一将有符号数转为无符号数处理，避免符号位干扰，转换完成后再统一添加负号。此外，支持 sign 参数控制是否进行符号处理（%u 不需要）。

**2.2** **理解分层设计：**

****

图 3 xvv6的分层设计示意图

**2．2．1** **每一层的职责是什么？**

1. **格式化层：**负责解析%d、%s等格式符，将数字转换为字符串，处理可变参数。它完成数据到文本的转换，但不关心输出目标。
2. **控制台层：**作为中间抽象层，提供统一的输出接口，可以管理多个输出设备（串口、显示器等），决定字符路由和设备选择。
3. **驱动层：**直接操作硬件寄存器，负责设备初始化、状态监控和数据传输，是软件与硬件的桥梁。
4. **终端/硬件层：**执行最终的字符显示和ANSI转义序列解释，如清屏(\033[2J)、光标控制等物理操作。

**2．2．2** **这种设计有什么优势？**

1. 模块化清晰：每层职责单一，修改格式化逻辑不影响硬件驱动，更换硬件只需重写驱动层，大幅提升可维护性。
2. 可移植性强：通过硬件抽象层，同一套printf代码可在不同平台（RISC-V、ARM等）和设备（串口、显示器）上运行。
3. 扩展灵活：添加新输出设备只需在控制台层注册，无需修改上层代码，支持输出重定向和多设备并行输出。
4. 健壮性高：错误被隔离在单层内，驱动故障不会导致系统崩溃，可实现优雅降级（如串口故障改用网络输出）。
5. 便于测试：可逐层测试，用模拟驱动替代真实硬件进行自动化测试，调试时能快速定位问题层级。

**2.3** **深入思考：**

**2．3．1** **xv6为什么不使用递归进行数字转换？**

递归可能存在以下问题：

1. 栈溢出风险：数字转换深度有限（最多20位），但内核栈很小
2. 性能开销：函数调用、参数传递、栈帧分配

**2．3．2** **printint() 中处理 INT\_MIN 的技巧是什么？**

NT\_MIN = -2147483648，-INT\_MIN 会溢出。因此，xv6选择使用 unsigned long long，可以安全容纳32位INT\_MIN的绝对值，避免溢出。

**2．3．3** **如何实现线程安全的printf？**

xv6通过自旋锁实现线程安全，具体的线程安全机制包含如下内容：

1. 互斥访问：防止多个CPU同时调用printf导致输出交错
2. panic特殊处理：panic时不加锁，避免死锁
3. 轻量级：自旋锁比互斥锁更适合内核短临界区

**3 实验原理与设计**

**3.1** **系统架构图设计**

本实验设计的简化系统架构图如下：

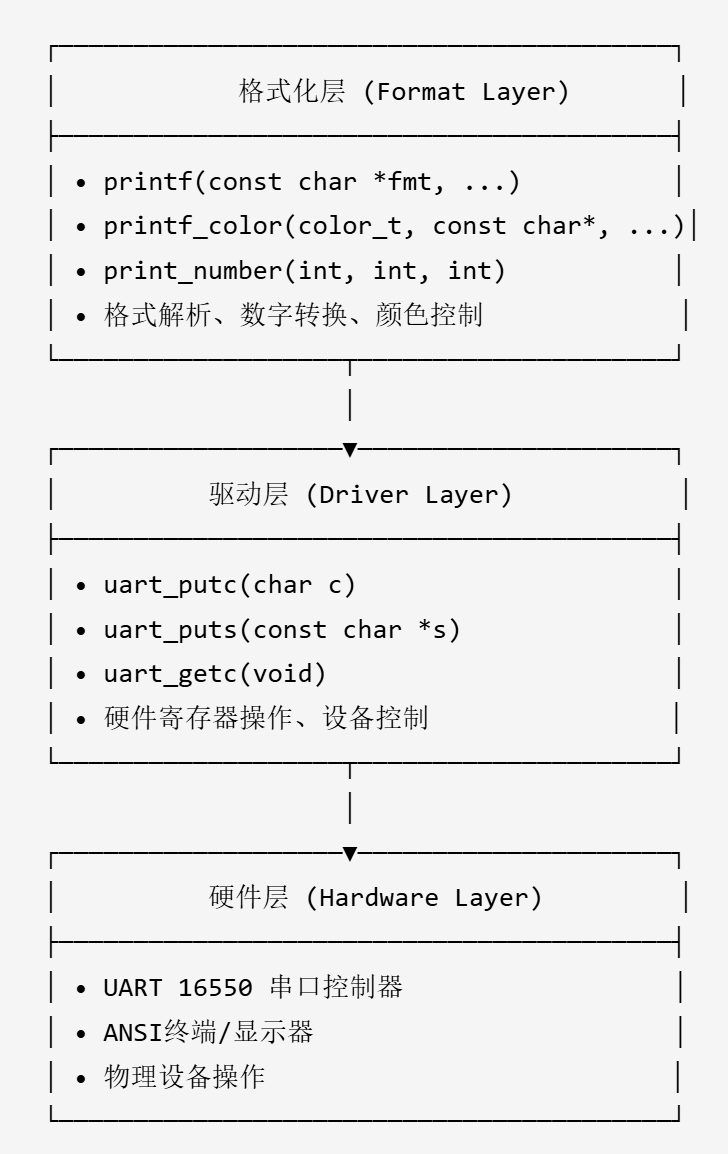


图 4 本实验的系统架构图

**3.2** **各层接口定义**

1. 格式化层接口 (printf.h)：

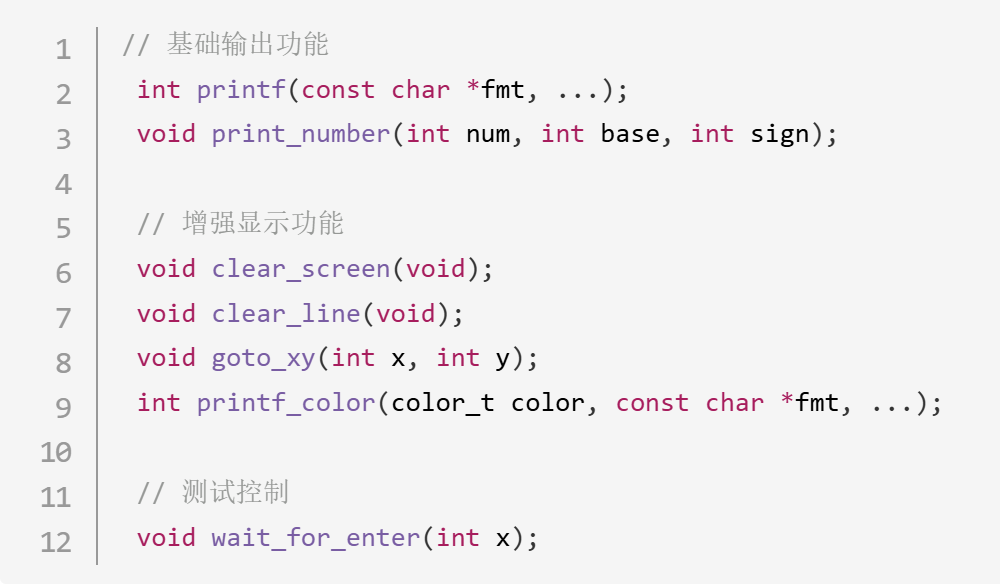


图 5 格式化层接口

1. 驱动层接口 (uart.h)：

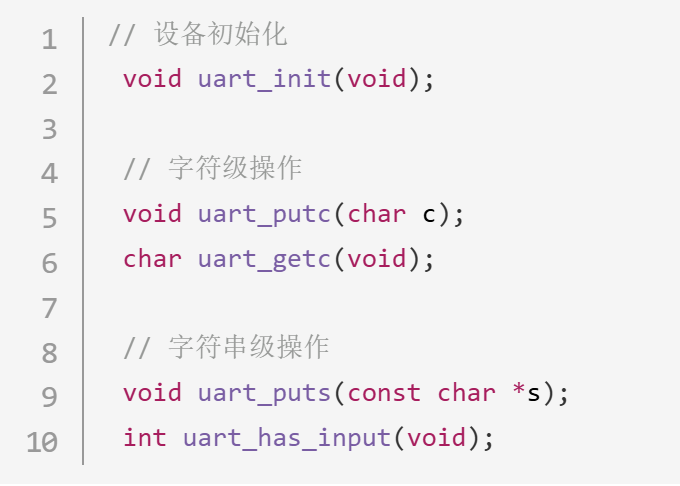


图 6 驱动层接口

**3.3** **与xv6设计的异同分析**

* 相同点：

1. 都采用分层架构，分离格式化逻辑与硬件操作
2. 都使用状态机解析格式字符串
3. 都支持基本的格式符(%d, %x, %s, %c, %%)
4. 都采用迭代算法进行数字转换，避免递归栈溢出

* 不同点：

表 1 本实验与xv6的不同点

|  |  |  |  |
| --- | --- | --- | --- |
| **特性** | **xv6设计** | **本实验设计** | **设计理由** |
| 架构层次 | 三层（格式化+控制台+驱动） | 两层（格式化+驱动） | 简化设计，减少函数调用开销 |
| 并发安全 | 自旋锁保护多核并发 | 无锁，单核假设 | 教学环境简化 |
| 功能扩展 | 基础格式化功能 | 增强显示（颜色+光标） | 提升用户体验 |
| 错误处理 | 复杂panic机制 | 简单空指针保护 | 聚焦核心功能 |
| 缓冲机制 | 有输出缓冲 | 直接字符输出 | 降低实现复杂度 |

**4 实验步骤与实现**

* 1. **实验步骤记录**

**4．1．1** **阶段一：基础框架搭建**

1. 分析xv6源码：深入研究printf.c的状态机解析逻辑和printint的数字转换算法
2. 设计接口规范：定义清晰的层间接口，确保模块化设计
3. 实现UART驱动：完成最基本的字符输出功能，验证硬件通路正常

**4．1．2** **阶段二：核心功能实现**

1. 可变参数处理：使用stdarg.h实现va\_list参数提取，支持不定参数
2. 格式解析状态机：实现%d、%x、%s等格式符的识别和处理
3. 数字转换算法：采用迭代逆序法，重点解决INT\_MIN边界情况

**4．1．3** **阶段三：增强功能开发**

1. ANSI转义序列：实现\033[2J清屏、\033[x;yH光标定位
2. 颜色输出支持：扩展printf\_color函数，支持8种基本颜色
3. 交互功能完善：添加wait\_for\_enter实现测试流程控制

**4.2** **核心关键代码理解总结**

* **关键代码1：数字转换算法（解决INT\_MIN边界）。** 用无符号运算解决边界溢出，迭代算法保证栈安全。



图 7 数字转换算法

* **关键代码2：格式解析状态机。** 用状态机模式实现格式解析，结构清晰，容易扩展。



图 8 格式解析状态机

* **关键代码3：ANSI清屏与光标控制。** 遵循ANSI标准化实现，确保终端具备兼容性。



图 9 ANSI清屏与光标控制

* **关键代码4：硬件寄存器精确操作。** 状态检查确保传输可靠性，换行符转换提升兼容性。



图 10 硬件寄存器精确操作

**5 实验测试与结果**

终端输入make指令进行编译，之后输入make run运行，进入如下的开始测试界面：

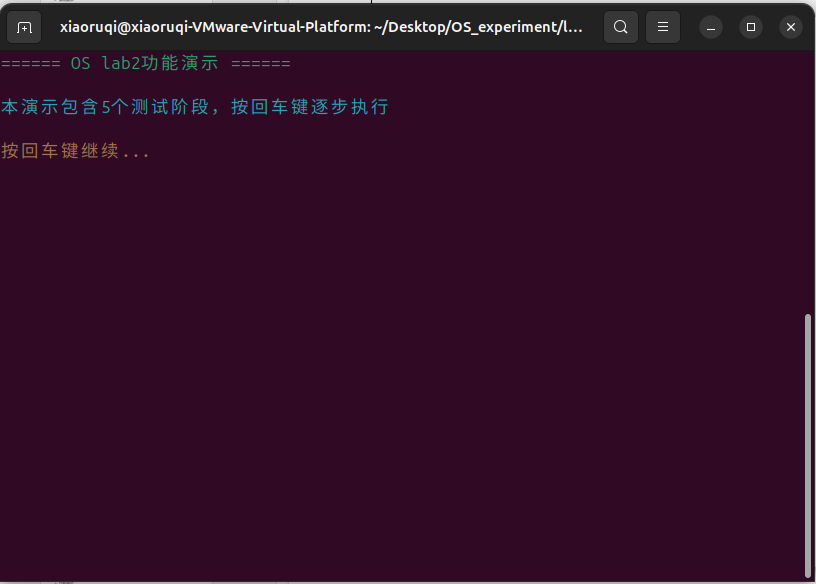


图 11 运行测试初始界面

**5.1** **基础功能测试**

验证printf基本格式化功能，包括整数、字符串、字符、十六进制等格式符的正确解析和输出。测试涵盖正数、负数、零值等常规情况。

输出结果如下图：

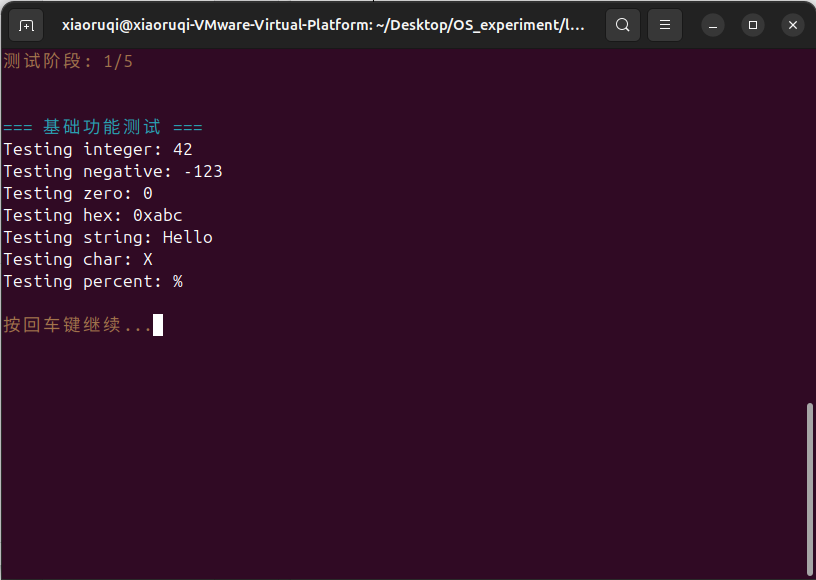


图 12 测试1-基础功能测试

**5.2** **边界情况测试**

重点测试系统在极端输入下的稳定性，包括INT\_MAX、INT\_MIN等数值边界，空指针、空字符串等异常输入，以及未知格式符的错误恢复能力。

输出结果如下图：

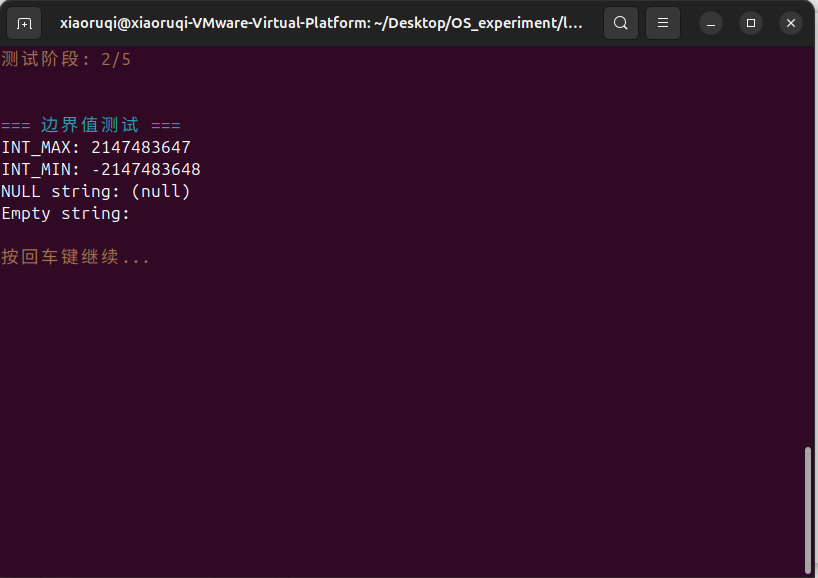


图 13 测试2-边界情况测试

**5.3** **颜色输出测试**

验证ANSI颜色转义序列的实现效果，测试8种基本颜色在终端中的实际显示效果，验证颜色设置和重置功能是否正常。

输出结果如下图，所有颜色正确显示：

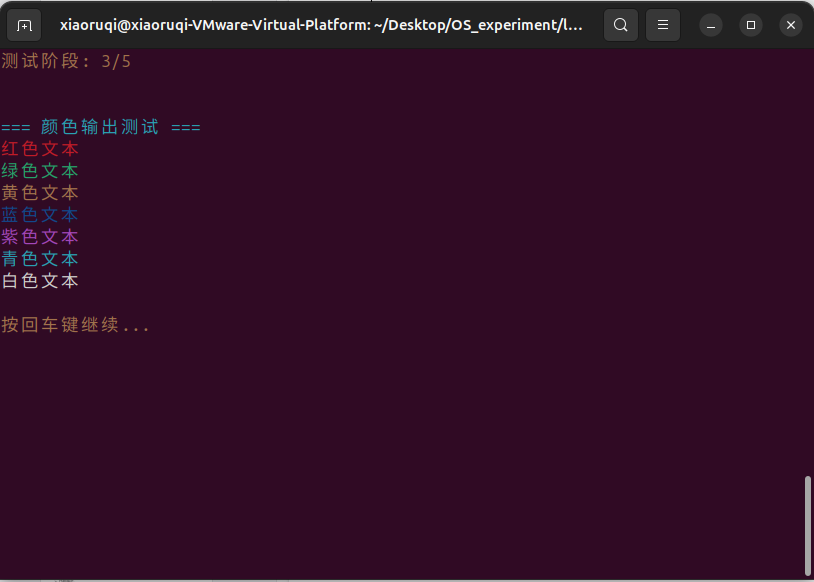


图 14 测试3-颜色输出测试

**5.4** **光标控制测试**

测试光标控制功能的准确性，验证goto\_xy函数能否精确定位到指定行列位置，检查坐标计算的正确性。

输出结果如下图，光标精确定位到第7行开头显示绿色标记线，随后定位到第9行第10列显示蓝色标记，定位准确无偏差。

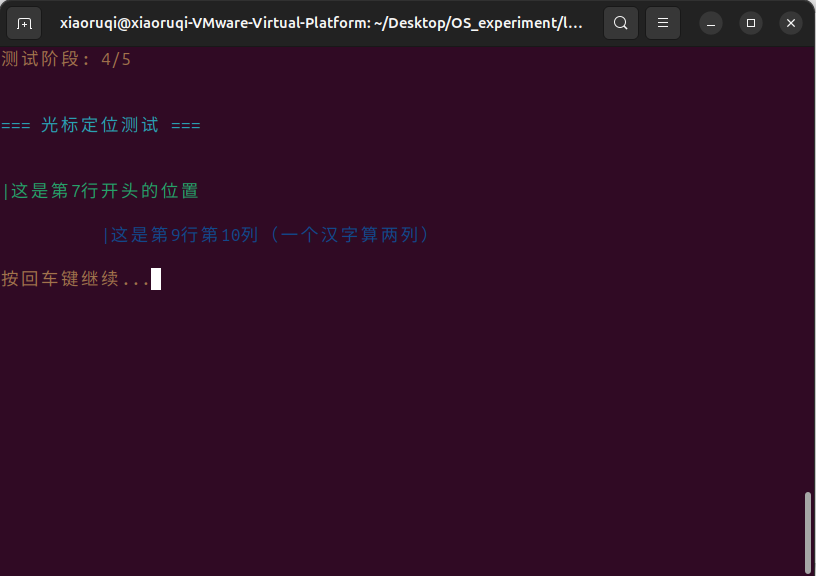


图 15 光标控制测试

**5.5** **清屏清行测试**

由于每一轮测试展示时，都调用了clean\_screen()函数，进行一次清屏，因此这里不再对清屏功能单独测试，这里重点测试清行功能。

1. 首先显示测试内容后提示用户"按回车键清除"，如下图：

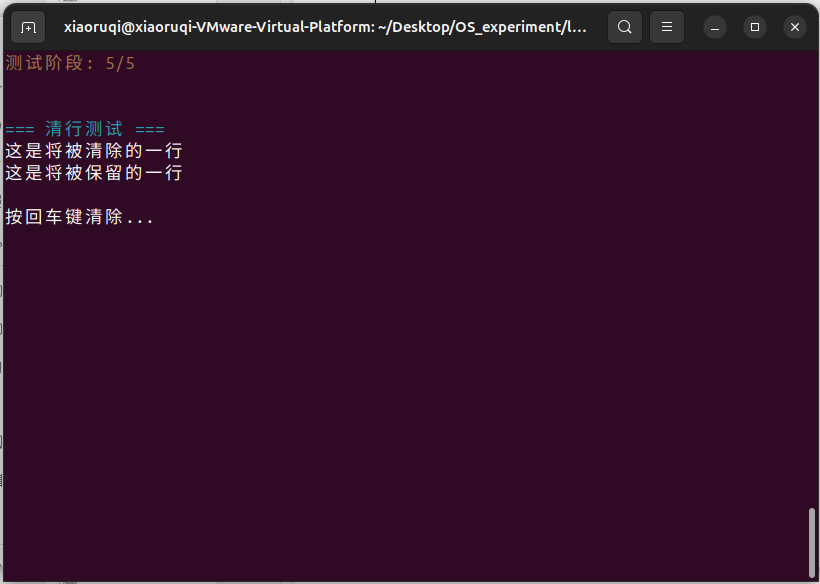


图 16 测试5-清屏清行测试1

1. 待用户按键后特定行内容被精确清除而其他行保留，演示动态界面更新效果。效果如下：

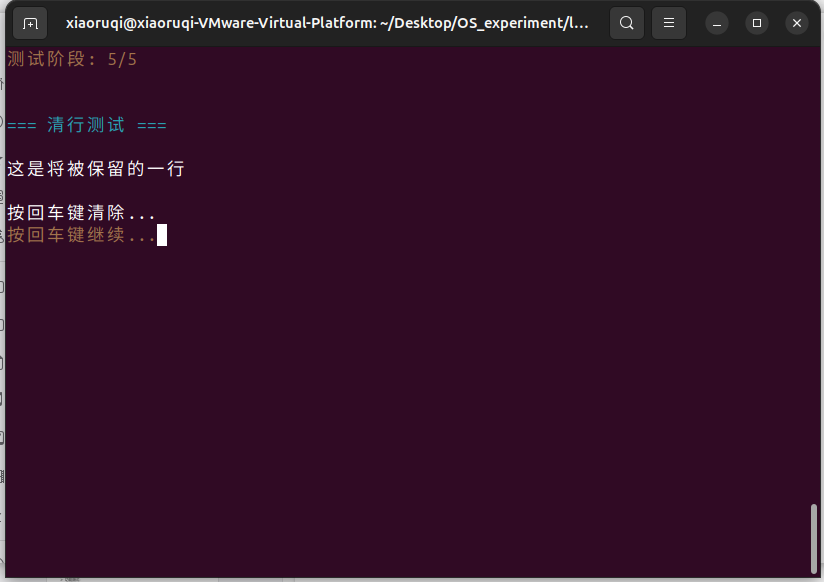


图 17 测试5-清屏清行测试2

**6 思考题**

**6.1** **架构设计**

**6．1．1** **为什么需要分层？每层的职责如何划分？**

本次内核输出系统分层主要解决三个核心问题——硬件差异抽象、功能逻辑分离和代码维护性。通过分层，将易变的硬件操作与稳定的业务逻辑解耦，提高系统可移植性和可维护性。

**6．1．2** **如果要支持多个输出设备（串口+显示器），架构如何调整？**

如需支持串口+显示器双输出，架构可以进行如下调整：

1. 增加设备抽象接口，统一uart和vga的操作方法
2. 控制台层实现输出路由策略（同步/异步、主备切换等）
3. 添加设备管理模块，动态检测设备可用状态

**6.2** **算法选择**

**6．2．1** **数字转字符串为什么不用递归？**

在2.1.3节已讨论过该问题。

**6．2．1** **如何在不使用除法的情况下实现进制转换？**

可以使用查表法——预计算各进制的幂次表，通过比较和减法实现转换：

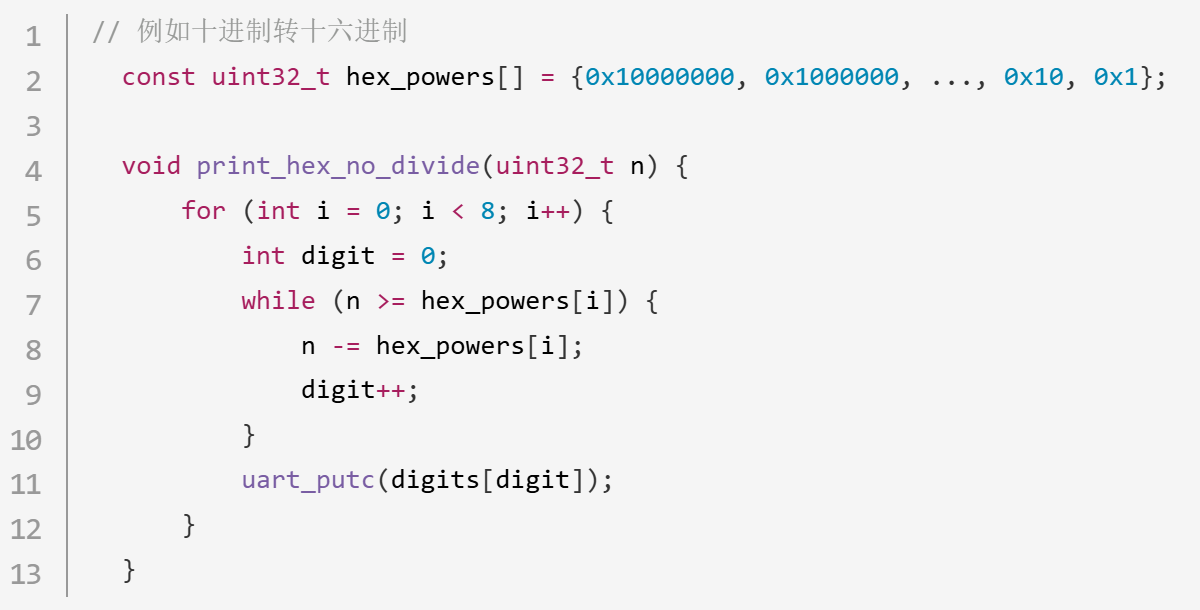


图 18 查表法实现进制转换

**6.3** **性能优化**

**6．3．1** **当前实现的性能瓶颈在哪里？**

1. 字符级传输：每个字符都要单独检查UART状态，大量CPU时间浪费在等待循环
2. 无缓冲机制：频繁的小数据量传输，无法利用硬件批量传输优势
3. 格式解析开销：每次printf都要重新解析格式字符串

**6．3．2** **如何设计一个高效的缓冲机制？**

可以采用双缓冲+批量传输方案。通过这种方案，可以减少90%以上的状态检查开销；同时支持DMA批量传输，能够释放CPU资源并提升吞吐量。代码如下：



图 19 双缓冲+批量传输方案

**6.4** **错误处理**

**6．4．1** **printf遇到NULL指针应该如何处理？**

采用分级处理方案：



图 20 NULL指针的分级处理方案

**6．4．2** **格式字符串错误时的恢复策略是什么？**

采用状态机容错机制：



图 21 格式字符串状态机容错机制

**7 实验总结**

通过本次实验，我深入掌握了内核级输出系统的设计与实现，从xv6的复杂架构分析到自主设计简化的两层模型，成功实现了支持格式化输出、颜色显示和清屏功能的完整系统。实验过程中遇到的INT\_MIN边界处理、ANSI转义序列兼容性、状态机解析等实际问题，锻炼了我底层编程和调试能力。更重要的是，这次实践让我深刻理解了内核开发与用户态编程的本质差异——内核必须自给自足，不能依赖外部库，所有功能都需要从硬件寄存器操作开始逐层构建。这种从零搭建系统组件的经验，不仅巩固了操作系统理论知识，更为后续更深层次的内核开发奠定了坚实基础。

教师评语评分

评语：

评分：

评阅人：

年 月 日

（备注：对该实验报告给予优点和不足的评价，并给出百分之评分。）