- 你的实现比xv6简化了哪些部分？
- 这些简化在什么情况下会成为问题？

4. **错误处理**：
   - 如果UART初始化失败，系统应该如何处理？
   - 如何设计一个最小的错误显示机制？

实验2：内核printf与清屏功能实现

实验目标

通过深入分析xv6的输出系统，理解格式化字符串处理原理，独立实现功能完整的内核printf和清屏功能。

核心学习资料

xv6输出系统架构分析

- `kernel/printf.c` - 格式化输出实现
  - 重点函数：`printf()`，`printint()`，`printptr()`
  - 学习要点：可变参数处理、数字转字符串算法
- `kernel/uart.c` - 硬件抽象层
  - 重点函数：`uartputc()`，`uartinit()`
  - 理解：设备驱动的抽象设计
- `kernel/console.c` - 控制台抽象层
  - 重点函数：`consputc()`，`consolewrite()`
  - 思考：为什么需要这个中间层？

相关技术规范

- **ANSI转义序列**：https://en.wikipedia.org/wiki/ANSI_escape_code
  - 重点：清屏、光标控制、颜色设置
- **C语言可变参数**：stdarg.h的使用方法
  - 关键宏：va_start, va_arg, va_end

任务列表

任务1：深入理解xv6输出架构

分析重点：

1. 研读 `printf.c` 中的核心函数：

- `printf()` 如何解析格式字符串？
- `printint()` 如何处理不同进制转换？
- 负数处理有什么特殊考虑？

2. 理解分层设计：

```
printf() -> consputc() -> uartputc() -> 硬件寄存器
```

- 每一层的职责是什么？
- 这种设计有什么优势？

深入思考：

- xv6为什么不使用递归进行数字转换？
- `printint()` 中处理 `INT_MIN` 的技巧是什么？
- 如何实现线程安全的printf？

任务2：设计你的输出系统架构

设计要求：

1. 画出你的系统架构图
2. 定义各层的接口
3. 说明与xv6设计的异同

关键设计决策：

- 是否需要缓冲区？为什么？
- 如何处理格式错误？
- 是否支持可变宽度格式？

架构建议：

```
// 硬件层
void uart_putc(char c);

// 控制台层
void console_putc(char c);
void console_puts(const char *s);

// 格式化层
int printf(const char *fmt, ...);
int sprintf(char *buf, const char *fmt, ...);
```

任务3：实现数字转换核心算法

学习xv6的printint实现，理解以下问题：

1. 为什么要将负数转为正数处理？
2. 如何避免递归导致的栈溢出？
3. 字符数组的组织方式

实现挑战：

```
// 你需要考虑的边界情况
static void print_number(int num, int base, int sign) {
    // 如何处理 INT_MIN？
    // 如何处理 base=16 的字母输出？
    // 如何实现逆序输出？
}
```

调试策略：

- 先实现十进制正数
- 再处理负数边界情况
- 最后支持十六进制

任务4：实现格式字符串解析

参考xv6的状态机思路：

1. 普通字符直接输出
2. 遇到%进入格式处理状态
3. 解析格式符并调用相应处理函数

实现要点：

```
int printf(const char *fmt, ...) {
    va_list ap;
    va_start(ap, fmt);

    // 你的解析逻辑：
    // 如何区分 %d, %x, %s, %c, %%？
    // 如何提取对应的参数？
    // 如何处理未知格式符？

    va_end(ap);
}
```

测试用例设计：

```
void test_printf_basic() {
    printf("Testing integer: %d\n", 42);
    printf("Testing negative: %d\n", -123);
    printf("Testing zero: %d\n", 0);
    printf("Testing hex: 0x%x\n", 0xABC);
    printf("Testing string: %s\n", "Hello");
    printf("Testing char: %c\n", 'X');
    printf("Testing percent: %%\n");
}

void test_printf_edge_cases() {
    printf("INT_MAX: %d\n", 2147483647);
```

```
    printf("INT_MIN: %d\n", -2147483648);
    printf("NULL string: %s\n", (char*)0);
    printf("Empty string: %s\n", "");
}
```

任务5：实现清屏功能

ANSI转义序列学习：

- `\033[2J` - 清除整个屏幕
- `\033[H` - 光标回到左上角
- `\033[K` - 清除当前行

实现思路：

```
void clear_screen(void) {
    // 方案1：发送ANSI转义序列
    // 方案2：输出足够多的换行符
    // 方案3：直接控制显示硬件（复杂）
}
```

扩展功能：

- 光标定位：`goto_xy(int x, int y)`
- 颜色输出：`printf_color(color, fmt, ...)`
- 清除行：`clear_line()`

任务6：综合测试与优化

功能测试：

1. 基本格式化功能
2. 边界条件处理
3. 性能测试（大量输出）
4. 错误恢复测试

性能优化考虑：

- 字符串输出是否可以批量发送？
- 数字转换是否可以查表优化？
- 格式解析是否可以预编译？

调试建议

分模块调试

1. **底层验证**：先确保单字符输出正常
2. **数字转换**：单独测试各种数字格式
3. **字符串处理**：测试各种字符串边界情况
4. **综合测试**：复杂格式字符串测试

常见问题诊断

输出不完整：

- 检查UART发送是否等待完成
- 验证字符串是否正确终止
- 确认缓冲区大小是否足够

数字输出错误：

- 验证进制转换算法
- 检查负数处理逻辑
- 测试INT_MIN等边界值

格式解析错误：

- 打印解析过程的中间状态
- 验证va_arg的参数类型匹配
- 检查未知格式符的处理

思考题

1. **架构设计：**
   - 为什么需要分层？每层的职责如何划分？
   - 如果要支持多个输出设备（串口+显示器），架构如何调整？
2. **算法选择：**
   - 数字转字符串为什么不用递归？
   - 如何在不使用除法的情况下实现进制转换？
3. **性能优化：**
   - 当前实现的性能瓶颈在哪里？
   - 如何设计一个高效的缓冲机制？
4. **错误处理：**
   - printf遇到NULL指针应该如何处理？
   - 格式字符串错误时的恢复策略是什么？

实验3：页表与内存管理

实验目标

通过深入分析xv6的内存管理系统，理解虚拟内存的工作原理，独立实现物理内存分配器和页表管理系统。

核心学习资料

RISC-V内存管理机制

- **RISC-V特权级规范 第12章： Supervisor-Level ISA**
  - 12.4 Sv39: Page-Based 39-bit Virtual-Memory System
- **在线文档：**https://github.com/riscv/riscv-isa-manual

xv6内存管理源码分析

- `kernel/kalloc.c` - 物理内存分配器
  - 重点函数：`kinit()`，`kalloc()`，`kfree()`
  - 学习要点：空闲页链表管理、简单分配算法
- `kernel/vm.c` - 虚拟内存管理
  - 重点函数：`walk()`，`mappages()`，`uvmcreate()`
  - 学习要点：页表遍历、映射建立、地址转换
- `kernel/riscv.h` - RISC-V相关定义
  - 重点内容：页表项格式、权限位定义、地址操作宏

内存管理理论基础

- **操作系统概念 第9-10章：**内存管理和虚拟内存
  - 在线图书：https://www.os-book.com/OS10/index.html
- **深入理解计算机系统 第9章：**虚拟内存

任务列表

任务1：深入理解Sv39页表机制

学习重点：

1. 分析39位虚拟地址的分解：

```
 38        30 29        21 20        12 11         0
 |  VPN[2]  |   VPN[1]   |   VPN[0]   |   offset   |
```

- 每个VPN段的作用是什么？
- 为什么是9位而不是其他位数？

2. 理解页表项（PTE）格式：
   - V位：有效性标志
   - R/W/X位：读/写/执行权限
   - U位：用户态访问权限
   - 物理页号（PPN）的提取方式

深入思考：

- 为什么选择三级页表而不是二级或四级？