# **操作系统课程设计**

组号：120  
组长姓名：关云兮  
提交日期：2025.8.22  
指导老师：王冬青

**阅读导航 / 目录**

目录

[**操作系统课程设计** 1](#_Toc206796918)

[**阅读导航 / 目录** 1](#_Toc206796919)

[**1 项目简介** 2](#_Toc206796920)

[**1.1** **项目介绍** 2](#_Toc206796921)

[**1.2** **环境搭建** 3](#_Toc206796922)

[**1.3** **项目目标** 3](#_Toc206796923)

[**1.4** **项目特点或亮点** 3](#_Toc206796924)

[**2 系统结构设计（Page 4）** 3](#_Toc206796925)

[**2.1 总体架构图** 4](#_Toc206796926)

[**2.2 模块划分** 4](#_Toc206796927)

[**2.3 模块间关系说明** 6](#_Toc206796928)

[**3 功能模块介绍** 7](#_Toc206796929)

[**3.1 任务管理** 7](#_Toc206796930)

[**3.2 贪吃蛇小游戏与计算器** 8](#_Toc206796931)

[**3.3 文件系统** 9](#_Toc206796932)

[**4 关键代码说明（Page 8）** 10](#_Toc206796933)

[**4.1 操作系统用户交互部分实现流程介绍** 10](#_Toc206796934)

[**5 使用说明/操作指南（Page 10）** 13](#_Toc206796935)

[**5.1 运行环境** 13](#_Toc206796936)

[**5.2 启动方法** 14](#_Toc206796937)

[**5.3 操作方法** 14](#_Toc206796938)

[**5.4 常用命令说明** 14](#_Toc206796939)

[**5.5 注意事项** 14](#_Toc206796940)

[**7 遇到的问题及解决方法（Page 14）** 15](#_Toc206796941)

[**7.1 问题 1：引导程序理解困难** 15](#_Toc206796942)

[**7.2 问题 2：循环鼠标和键盘输入最初无反应** 15](#_Toc206796943)

[**7.3 问题 3：任务切换与内存管理调试困难** 16](#_Toc206796944)

[**8 实验心得或总结（Page 16）** 17](#_Toc206796945)

[**8.1 心得体会** 17](#_Toc206796946)

[**8.2 项目亮点** 17](#_Toc206796947)

[**8.3 遇到的难点** 17](#_Toc206796948)

[**8.4 下一步改进想法** 18](#_Toc206796949)

**1 项目简介**

* 1. **项目介绍**

本项目的选题为 **自制操作系统**，旨在通过独立完成一个简单的操作系统，掌握操作系统核心原理与实现方法。

项目主要任务包括实现引导程序、核心代码、内存管理、多任务管理、文件管理等基本功能模块，同时可以借鉴已有操作系统的设计思想进行优化与扩展。

项目组成员需完成至少一半以上的核心代码，实现一个可独立运行的操作系统环境。

通过本项目的开发，能够加深对多任务管理、内存管理、文件系统设计及用户交互机制的理解，培养独立设计与调试操作系统的能力。参考资料主要包括书籍《一个操作系统的实现》以及其配套示例代码，同时借鉴《30天自制操作系统》中的底层设计思想和实现方法。

* 1. **环境搭建**

本项目开发环境全部搭建在 **Windows 11** 系统下，借助 **MSYS2** 工具完成辅助编译与环境配置。虚拟化运行依赖 **QEMU** 模拟器，通过 QEMU 可以加载自制操作系统镜像并进行测试。使用 **Makefile** 管理编译流程，通过定义编译规则和依赖关系，实现自动化生成操作系统镜像。开发过程中，操作系统源码的编译、调试、运行均在 Windows 环境下完成。底层设计与实验步骤参考《30天自制操作系统》，同时结合个人项目需求进行了适当功能填充和自编代码，以保证系统功能完整性与可运行性。

* 1. **项目目标**
* 独立完成一个可运行的简单操作系统，包括引导程序、任务调度、文件系统和基本控制台等功能。
* 掌握操作系统核心模块的设计与实现原理，如任务管理、内存管理、IO控制和文件操作。
* 提高分析问题和解决问题的能力，能够根据实验需求和参考资料进行自主设计与优化。
* 最终实现一个可在 QEMU 下运行的操作系统镜像，便于展示和答辩。
  1. **项目特点或亮点**
* **自主设计与实现**：操作系统从平台搭建部署到功能实现，均由一人反复调试实现，底层引导参考了《30天自制操作系统资料》，功能实现大多数自行完成。
* **基础功能全面**：系统涵盖了操作系统的基本组成部分，包括缓冲区设置、任务调度、文件管理、输入输出处理和用户交互，形成一个结构完整的简易操作系统。
* **功能拓展丰富**：在基本功能之外，系统内置了**贪吃蛇小游戏**与**计算器程序**，增强了系统的可玩性与实用性，展示了操作系统在用户交互与应用层扩展方面的能力。
* **文件+控制台系统支持**：实现了简易文件系统，支持文件的创建、读取与删除操作，为用户程序和数据存储提供了必要支撑。

**2 系统结构设计（Page 4）**

**2.1 总体架构图**

引导程序 -> 核心模块 -> 各大功能模块 -> 实际用户交互

**2.2 模块划分**

**2.2.1引导程序与底层配置模块**



|  |  |  |
| --- | --- | --- |
| 文件名 | 文件类型 | 功能说明 |
| asmhead.nas | NAS 文件 | 引导程序的头文件，定义启动时段寄存器和内存布局，用于系统引导。 |
| hankaku.txt | 文本文档 | 存储半角字体点阵，用于控制台和窗口文字显示。 |
| ipl10.nas | NAS 文件 | 引导程序，实现从 BIOS 获取硬件信息、加载内核到内存并跳转到核心程序。 |
| Makefile | 文件 | 编译管理文件，定义源码编译规则和依赖关系，自动生成可运行操作系统镜像。 |
| naskfunc.nas | NAS 文件 | 封装底层汇编功能接口，如 I/O 操作、中断处理等，供核心模块和用户程序调用。 |

**2.2.2 核心模块**

核心模块是操作系统的中枢，负责管理系统的较为底层的操作，方便后续各种交互功能的实现，包括内存、中断处理试能等。

内存管理模块负责分配和回收系统内存，为核心模块和用户程序提供运行空间。它实现了简单的内存分配算法，保证在程序运行过程中可以动态申请和释放内存，同时为文件系统和任务管理提供必要的存储支持。

FIFO 队列模块实现了先进先出数据缓冲，用于任务间通信以及键盘和鼠标输入的暂存。该模块提供了初始化、写入、读取和状态查询接口，确保系统能够按顺序处理输入事件和消息，实现稳定的数据流管理。

字体管理模块用于控制台和窗口文本显示，管理字符的点阵信息。系统通过读取 hankaku.txt 中的半角字体数据，将字符绘制到屏幕上，实现文字显示和用户界面信息输出，保证了图形界面和控制台的可视化效果。

中断处理模块负责系统中断的初始化与处理，包括时钟、键盘和鼠标中断。它建立中断向量表并注册中断服务程序，使系统能够响应外部事件，实现任务调度和用户输入的事件驱动机制。

描述符表管理模块负责建立段描述符表（GDT）和中断描述符表（IDT），管理内存段和中断权限。通过设置段选择器和访问权限，该模块为多任务切换、内存保护和系统中断提供了基础保障。

计时器模块负责系统定时功能，通过周期性触发中断为任务调度和时间控制提供基础支持。它允许操作系统按照固定时间间隔执行任务切换、刷新屏幕或更新系统时间，为多任务环境和事件驱动机制提供可靠的时序保障。

**2.2.3 图形界面**

图形显示模块负责操作系统界面和鼠标光标的绘制。系统定义了基本颜色常量，如黑色、浅灰、白色、浅蓝、浅绿、浅黄和浅粉，用于窗口、按钮和文本的显示。在鼠标光标初始化函数中，通过一个 16×16 的字符数组定义光标形状，将不同字符映射为对应颜色，实现了彩色鼠标指针的显示。

该模块还提供了将图像块绘制到屏幕的函数，支持将缓冲区中的像素按指定位置渲染到显存上，保证了窗口、控件或图标可以正确显示。系统内置 RGB 调色表，将颜色索引映射到显存颜色值，为界面元素的着色提供了基础。

此外，字体显示功能通过读取 hankaku.txt 中的点阵数据，实现字符在屏幕上的绘制。函数 putfont8 将单个字符的点阵绘制到指定位置，而 putfonts8\_asc 则可连续绘制字符串，为控制台输出和窗口文字显示提供了支持。

同时实现了操作系统的窗口分层机制，为图形界面提供多窗口显示支持。系统通过 SHTCTL 管理所有图层，每个图层对应一个 SHEET 结构，包含缓冲区、尺寸、位置和透明颜色等信息。通过分配函数创建图层，并设置缓冲区与透明色，操作系统可以将不同图层的内容独立绘制到显存上。模块提供图层高度管理功能，通过调整图层的上下顺序实现覆盖显示，同时在图层移动或刷新时，只更新受影响区域，提高显示效率。图层刷新分为刷新映射表和刷新显存两步，映射表记录每个像素属于哪个图层，从而在绘制时判断是否覆盖，保证了窗口叠加、移动和隐藏的正确显示效果。

**2.2.4输入输出模块**

输入输出模块负责键盘和鼠标的数据采集与解码，保证用户交互的正常进行。键盘模块通过定义标准和 Shift 键的扫描码表，将硬件输入转换为可显示字符，大写操作为shift+字母。初始化函数设置键盘控制器工作模式，为后续输入提供可靠基础。

鼠标模块通过 FIFO 队列与系统核心通信，实现鼠标数据的接收与解析。enable\_mouse 函数向键盘控制器发送命令，激活鼠标设备，随后 mouse\_decode 函数将鼠标数据包解码为按钮状态及 x、y 移动增量。通过负增量处理，将鼠标坐标与屏幕显示坐标一致，使光标在屏幕上正确移动。

输入输出模块与核心 FIFO 机制结合，保证键盘和鼠标输入事件按顺序处理，为任务调度、图形界面交互和用户命令输入提供支持。整个模块实现了操作系统基本的用户交互功能，是图形界面和应用程序运行的关键支撑。

**2.2.5 用户交互模块**

控制台命令解析模块负责接收用户输入的文本指令，并将其解析为系统能够执行的操作。它将输入转换为具体的命令调用，支持文件管理、任务控制和系统设置等操作，为用户提供命令行交互的基础功能。

计算器应用模块实现了基本的数学计算功能，包括加减乘除操作。它通过图形界面或控制台显示输入和计算结果，使用户能够直接在操作系统中进行简单运算，提高系统的实用性。

小游戏模块实现了贪吃蛇游戏，包括游戏逻辑、界面绘制和用户输入处理。该模块演示了操作系统对多任务和图形界面支持能力，同时为系统增加了趣味性和交互性。

任务模块负责多任务运行。它协调各任务的调度与资源分配，保证用户输入能够被及时响应，同时支持多个应用程序和任务并行运行，提升系统的稳定性和交互体验。

**2.3 模块间关系说明**

操作系统的最底层是引导程序和硬件配置模块，它们负责系统启动、硬件初始化和内核加载，为核心模块提供稳定的运行环境。

核心模块作为中枢，管理内存、FIFO 队列、中断和计时器等底层资源，同时提供基础服务接口，支撑图形界面和输入输出模块的运行。

图形界面模块和输入输出模块依赖核心模块实现功能。图形界面负责窗口、光标和文字显示的绘制与刷新，并支持多窗口操作；输入输出模块处理键盘和鼠标事件，将用户操作传递给系统和应用程序。

用户交互模块建立在图形界面和输入输出模块之上，实现控制台命令解析、计算器、小游戏等功能，为用户提供直观的操作体验。各模块通过层次分明的接口协作，保证系统的稳定性和可扩展性。

**3 功能模块介绍**

**3.1 任务管理**

**3.1.1 功能描述**

控制台是操作系统的命令行交互界面，提供命令解析和执行功能。用户可以通过控制台执行系统内置命令（如 mem、cls、ver、help 等），管理文件（dir、type、create、del），以及调用其他应用程序或输出文本。系统支持行缓冲输入、退格删除、换行显示和命令回显，并在控制台显示实时反馈。

例如，mem 命令显示系统内存总量；cls 清屏；ver 输出操作系统版本信息；dir 列出磁盘文件列表。

文件操作命令如 type 显示文件内容，create 和 del 对应创建和删除文件，同时在桌面图标上同步显示变化。

**3.1.2 输入/输出**

* 输入：控制台通过 FIFO 队列接收键盘扫描码，判断 Shift 键状态决定字符大小写或符号，处理退格和换行，并将输入行缓存在 line[] 中。
* 输出：使用 console\_putchar 和 console\_putstr 绘制单个字符和字符串到控制台窗口。输出颜色可指定，实现清晰的视觉反馈。
* 文件显示：例如 type 命令通过读取文件缓冲区，将内容逐字符输出到控制台，并处理换行。
* 桌面同步：create 和 del 命令调用 create\_file\_and\_draw 和 delete\_file\_icon 函数，使文件图标与控制台操作保持一致。

**3.1.3 核心流程**

初始化：调用 console\_init 初始化控制台缓冲区和显示窗口，显示欢迎信息和提示符 > 。

事件循环：进入死循环，轮询 FIFO 队列获取键盘事件；若队列为空，则调用 task\_sleep 休眠任务，节省 CPU。

按键处理：

判断 Shift 键状态（按下或释放）。

将扫描码映射为字符，并处理特殊字符：退格删除、换行、普通可打印字符。

命令解析：

输入换行时调用 command\_exec 将输入行拆分为参数数组 argv[]。

遍历命令表 commands[] 查找匹配命令并执行对应函数。

若命令不存在，显示 Unknown command 提示。

输出反馈：命令执行完成后，控制台显示结果，并重新显示提示符，等待下一条输入。

**3.1.4 实现成果**

用户可以在控制台中输入多种命令，并获得实时反馈。系统支持文件管理和桌面图标同步显示，命令解析支持多参数输入。输入输出机制结合 FIFO 队列和图层显示，实现事件驱动和多任务兼容。控制台功能已覆盖基础系统操作和应用扩展，可作为操作系统交互的核心模块。

**3.2 贪吃蛇小游戏与计算器**

**3.2.1 功能描述**

* **计算器**：提供图形化计算界面，支持加、减、乘、除四则运算。用户可以通过点击按钮输入数字和运算符，并在显示区看到输入和计算结果。系统提供清空功能（C 键）和计算执行（= 键），输入错误时显示 Err。
  + 实现方式：
    - 使用独立窗口 SHEET 显示计算器界面，缓冲区保存显示内容。
    - 按钮通过 draw\_button 绘制，点击事件由 calc\_on\_click 处理，将点击位置映射到数字或运算符。
    - 输入字符串缓存在 input[] 中，点击 = 后使用 parse\_expr 解析表达式并计算结果，再调用 calc\_update\_display 更新显示。
    - 清空按钮直接清理缓冲区并重置输入数组。
* **贪吃蛇游戏**：提供带障碍的小游戏，玩家通过键盘方向键控制蛇移动，吃食物增长长度，撞墙或撞障碍物会显示 “GAME OVER”。
  + 实现方式：
    - 游戏窗口独立 SHEET 绘制，蛇体、食物和障碍物使用 boxfill8 渲染不同颜色块。
    - 蛇的状态用数组 snake[] 保存每个节点位置，snake\_len 表示长度。
    - 食物和障碍物位置通过简单算法生成随机坐标，障碍物固定初始位置。
    - 每步移动由 snake\_step 处理，包括蛇身移动、碰撞判定和食物增长。
    - 键盘方向键输入通过 snake\_input 改变 dir，确保不能直接反向。

**3.2.2 使用方法**

* **计算器**：
  1. 点击图标打开窗口（调用 open\_calc\_window）。
  2. 使用鼠标点击数字或运算符按钮输入表达式。
  3. 点击 = 执行计算，结果显示在输入框；点击 C 清空当前输入。
* **贪吃蛇**：
  1. 打开贪吃蛇游戏窗口（调用 snake\_init）。
  2. 使用方向键控制蛇移动，目标是吃食物增长长度。
  3. 游戏结束条件包括撞墙、撞自身或撞障碍物，显示 “GAME OVER”。

**3.2.3 实现成果总结**

* 两个应用都使用操作系统图形界面模块 SHEET 管理窗口和缓冲区，实现多窗口显示与刷新。
* 用户输入处理结合鼠标点击或键盘扫描码，通过事件驱动机制实时响应。
* 逻辑和显示分离：输入和状态存储在缓冲数组中，显示通过绘图函数更新界面，实现简洁、高效的交互体验。
* 这两个模块体现了操作系统对多任务、图层管理和图形界面支持的能力。

**3.3 文件系统**

**3.3.1 功能描述**

**文件创建**  
系统支持创建新文件，用户可以指定文件名和文本内容。文件名采用 8.3 格式统一大写存储，每个文件占用固定簇，便于管理和快速查找。创建成功后，文件会显示在文件管理窗口中，用户可以直观看到新增文件。

 **文件删除**  
删除功能会清空对应文件的 fileinfo 信息和磁盘簇数据，释放空间。文件管理窗口会同步刷新，保证界面和磁盘状态一致，用户操作直观且安全。

 **文件读取与查看**  
用户可以打开文件窗口查看文本内容，系统按行显示文本并处理换行，避免超出窗口范围。如果文件不存在，窗口会显示“File not found!”提示。

 **文件管理可视化**  
文件管理窗口列出所有文件的名称、类型和大小，并提供刷新机制。每次创建或删除文件后，列表自动更新，让用户可以方便地浏览和管理文件。

**3.3.2 实现过程**

**磁盘与文件信息初始化**  
系统在启动时通过 fs\_init 分配模拟磁盘内存，并初始化 fileinfo 数组。通过 fs\_format 将磁盘和文件信息清零，为后续操作创建干净的存储环境，同时提供模拟写入函数 fs\_mock\_write，生成默认文件供测试。

**文件创建与删除操作**  
创建文件时，系统调用 fs\_create，根据用户输入生成 8.3 格式文件名，并将文本写入磁盘对应簇，更新 fileinfo 记录。删除文件时，通过 fs\_delete 清空文件信息并擦除磁盘簇数据，同时刷新文件管理窗口界面，保证操作与显示一致。

**文件查找与读取**  
文件读取通过 fs\_find 查找目标文件，返回文件信息结构。随后 fs\_read 按字节将内容复制到缓冲区，并在窗口中按行显示。系统处理换行和显示长度限制，避免文字溢出窗口。

**文件管理窗口可视化**  
文件管理窗口使用 sheet 分配图层和缓冲区，调用 refresh\_file\_manager 绘制每个文件的名称、类型和大小。创建或删除文件后刷新界面，实现图形化文件浏览和操作，提升用户体验。

**4 关键代码说明（Page 8）**

**4.1 操作系统用户交互部分实现流程介绍**

1. 系统初始化与图层配置

系统启动后，首先完成 CPU、内存和中断初始化，并为图层管理和多任务调度做好准备。初始化包括：

GDT/IDT 和 PIC 设置

PIT 定时器初始化

内存管理模块 memman 初始化

随后创建 桌面背景、鼠标光标和窗口图层：

init\_screen(buf\_back, binfo->scrnx, binfo->scrny); // 初始化背景

init\_mouse\_cursor8(mcursor, 99); // 初始化鼠标

mx = (binfo->scrnx - 16) / 2;

my = (binfo->scrny - 28 - 16) / 2;

sheet\_slide(sht\_mouse, mx, my); // 鼠标初始位置

buf\_back 存储桌面像素，鼠标光标独立缓冲区 mcursor 便于覆盖显示。

每个应用窗口使用独立 SHEET，保证图层刷新互不干扰。

2. 任务与窗口创建

操作系统使用 多任务模型，每个窗口或应用对应一个任务（Task）。例如：

task\_a：主任务，负责事件循环和桌面刷新

task\_b：控制台窗口任务，显示系统信息

任务创建时会分配独立的栈空间和寄存器上下文，实现并发操作：

task\_a = task\_alloc();

task\_run(task\_a, 1, 2); // 运行主任务

task\_b = task\_alloc();

task\_run(task\_b, 2, 2); // 控制台任务

窗口初始化包括显示文本和图形元素：

putfonts8\_asc\_sht(sht\_win\_b, 24, 28, COL8\_BLACK, COL8\_LIGHT\_GRAY, s, mstrlen(s));

每个窗口有独立缓冲区 buf\_win，保证刷新局部内容时不会覆盖其他窗口。

3. 输入设备事件处理

系统通过 FIFO 缓冲区 接收键盘和鼠标事件：

if (256 <= i && i <= 511) fifo32\_put(&keyfifo, i); // 键盘

if (512 <= i && i <= 767) fifo32\_put(&mousefifo, i); // 鼠标

键盘事件传递到控制台任务，可用于输入命令或控制应用。

鼠标事件解码后，用于 移动光标、点击图标和拖动窗口：

if (mouse\_decode(&mdec, i - 512)) {

mx += mdec.x;

my += mdec.y;

sheet\_slide(sht\_mouse, mx, my); // 更新鼠标位置

}

4. 桌面图标交互与应用启动

桌面底部设计了多个应用图标，例如 Terminal、计算器、贪吃蛇等。点击逻辑如下：

if (mx >= btn\_x0 && mx <= btn\_x1 && my >= btn\_y0 && my <= btn\_y1) {

open\_console\_shell(shtctl, memman); // 打开控制台窗口

}

if (mx >= icon\_x\_cal && mx <= icon\_x\_cal + icon\_w) open\_calc\_window();

if (mx >= icon\_x\_snake && mx <= icon\_x\_snake + icon\_w) snake\_init();

鼠标位置判断保证点击精确触发。

每个应用窗口独立任务，互不干扰。

5. 窗口拖动与关闭机制

系统支持 窗口拖动、最小化和关闭：

if (y < 21) { // 窗口标题栏被点击

dragging = 1;

drag\_sht = sht;

drag\_x = mx - sht->vx0;

drag\_y = my - sht->vy0;

}

if (dragging) sheet\_slide(drag\_sht, mx - drag\_x, my - drag\_y);

点击右上角关闭按钮可终止任务并隐藏窗口。

窗口拖动时只刷新受影响区域，提高性能。

6. 动态显示与定时器驱动

系统使用 定时器事件 刷新动态内容，例如 CPU 使用率、时钟或游戏动画：

if (i == 1) {

count++;

snake\_step(); // 贪吃蛇自动移动

timer\_settime(timer, 50); // 定时器重新设置

}

定时器保证 平滑刷新，避免残影。

游戏与动画应用共享事件循环，实现多任务并行。

7. 总结

通过图层管理、多任务、FIFO 输入缓冲区、定时器驱动和窗口事件处理，实现了一个完整的 可视化交互系统：

用户可点击图标启动应用

支持窗口拖动、关闭和最小化

鼠标和键盘事件可并行处理

定时器保证动态显示流畅

关键设计思想是 任务与图层解耦、事件驱动刷新、局部更新优化，为后续文件管理、游戏和计算器等应用提供了基础框架。

**5 使用说明/操作指南（Page 10）**

**5.1 运行环境**

本操作系统为自制裸机系统，适用于 QEMU 虚拟机或真实 x86 兼容硬件，要求如下：

* **CPU**：x86 架构（支持保护模式）
* **显示器**：VGA 输出
* **输入设备**：标准键盘、PS/2 鼠标
* **开发环境**：GCC / NASM / LD 编译链，可在 Linux 或 Windows（含 WSL）下构建，开发者是纯windows环境。
* **虚拟机推荐**：QEMU

**5.2 启动方法**

系统启动方式如下：

**QEMU 启动**（需自行安装qemu）：

qemu-system-i386 -fda haribote.img

**5.3 操作方法**

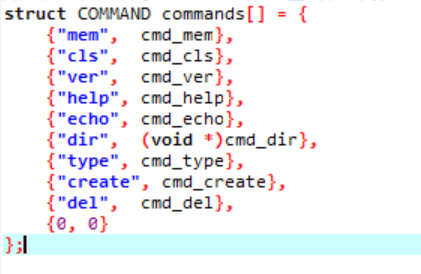
系统界面为 **图形化桌面 + 多窗口模式**，操作方法如下：

* **鼠标操作**：
  + 左键点击图标启动应用（Terminal、计算器、贪吃蛇等）
  + 点击窗口标题栏拖动窗口
  + 点击右上角关闭按钮关闭窗口
* **键盘操作**：
  + Terminal 窗口可输入命令
  + 不同功能模式支持方向键/回车/删除键
* **窗口管理**：
  + 多任务窗口并行显示
  + 鼠标拖动改变窗口位置
  + 活动窗口在最上层显示

由于特殊问题，某些功能可能会出现一次加载不成功情况，可以多次启动img文件进行测试。

**5.4 常用命令说明**

系统内置 Terminal 窗口，支持以下命令：



**5.5 注意事项**

* **鼠标光标和窗口刷新**依赖定时器和图层管理，不建议修改缓冲区指针
* **关闭窗口使用右上角按钮**，拖动上步移动窗口
* **文件系统相关操作命令详情见源代码。**

**7 遇到的问题及解决方法（Page 14）**

**7.1 问题 1：引导程序理解困难**

* **描述**：  
  在开发初期，对于引导程序（Bootloader）从实模式切换到保护模式的过程非常不熟悉。具体包括：
  + GDT、IDT 的配置与加载
  + 段寄存器和栈指针初始化
  + VGA 显示缓冲区初始化  
    由于这些操作涉及底层 CPU 寄存器和二进制指令，调试难度很大。
* **解决方法**：
  + 参考《30天自制操作系统》教材，实现分阶段加载和保护模式切换。
  + 编写 VGA 初始化函数，确保屏幕显示测试信息，以验证系统成功进入保护模式：
  + void init\_palette(void);
  + void init\_screen(char \*vram, int x, int y);
  + 分模块逐步调试，先保证显示，再加载内存管理和任务管理模块。
* **结果**：  
  系统成功进入保护模式，并显示初始桌面界面，为后续窗口和任务管理开发打下基础。

**7.2 问题 2：循环鼠标和键盘输入最初无反应**

* **描述**：  
  在实现窗口交互时，最初鼠标移动和键盘按键无任何响应。尝试直接读取端口或轮询输入均无效。问题主要在于：
  + 输入事件未解码
  + 多任务窗口系统下输入事件容易丢失
  + 屏幕刷新和输入处理未解耦
* **解决方法**：
  + 编写鼠标解码结构体和函数：
  + struct MOUSE\_DEC mdec;
  + if (mouse\_decode(&mdec, data)) {
  + // 更新鼠标位置
  + }
  + 使用 FIFO 队列缓冲键盘和鼠标事件：
  + fifo8\_init(&keyfifo, 32, keybuf);
  + fifo8\_init(&mousefifo, 128, mousebuf);
  + 将输入处理与窗口刷新解耦，通过定时器触发事件处理和画面刷新，使输入及时响应。
  + 增加调试输出，打印检查鼠标位置等等。
* **结果**：
  + 鼠标移动平滑、点击有效
  + 键盘输入可以实时显示在窗口
  + 系统输入处理稳定，多窗口环境下不会丢失事件

**7.3 问题 3：任务切换与内存管理调试困难**

* **描述**：  
  系统支持多任务运行，每个任务需要独立栈和寄存器保存。最初任务切换不稳定，出现：
  + 堆栈溢出或覆盖
  + 窗口刷新异常
  + FIFO 输入数据丢失  
    另一个挑战是内存分配和回收管理不完善，容易出现内存泄漏。
* **解决方法**：
  + 采用多任务管理结构体：
  + struct TASK {
  + int flags;
  + int priority;
  + int tss\_sp;
  + struct FIFO8 fifo;
  + };
  + 在任务切换时保存/恢复寄存器和堆栈：
  + farjmp(taskA->tss\_eip, taskB->tss\_eip);
  + 使用内存管理模块 MEMMAN 管理内存分配与释放：
  + memman\_free(memman, addr, size);
  + memman\_alloc(memman, size);
* **结果**：
  + 多任务切换稳定，每个任务独立运行
  + 窗口刷新和输入事件均正常
  + 内存管理可靠，减少内存碎片和泄漏

**8 实验心得或总结（Page 16）**

**8.1 心得体会**

本次实验通过从零构建类操作系统，深刻理解了底层计算机系统的工作原理。实践过程中，我对以下方面有了深刻体会：

* **硬件与软件协作**：VGA 显示、键盘和鼠标中断处理，让我理解了硬件驱动程序的工作方式。
* **任务管理与多任务调度**：通过学习和实现任务切换机制，掌握了 TSS、堆栈和寄存器保存与恢复的原理。
* **内存管理**：使用 MEMMAN 模块管理内存，了解了分配、回收、碎片化问题，以及如何在裸机环境下实现可靠的内存操作。
* **调试技巧**：通过在交互输入（鼠标位置、键盘按键）部分打印调试信息，掌握了排查底层问题的方法，尤其是在没有标准库和调试工具的情况下。

**8.2 项目亮点**

* **模块化设计**：将显示（graphic.c）、字体（font.c）、系统主逻辑（main.c）、汇编接口（naskfunc.nas）分离，实现清晰的代码结构。
* **窗口与交互系统**：支持多窗口显示、鼠标拖拽、任务栏按钮点击，窗口可打开终端执行基本命令（如 mem、cls）。
* **多任务与FIFO输入**：实现了任务间切换与独立输入事件队列，保证不同任务独立响应鼠标和键盘事件。
* **稳定的内存管理**：通过 memman\_alloc\_4k 与 memman\_free\_4k 实现内存分配和回收，减少内存泄漏和碎片化问题。

**8.3 遇到的难点**

* **引导程序理解**：底层二进制与引导加载程序的逻辑复杂，一开始难以完全理解系统启动流程。
* **循环鼠标和键盘输入处理**：最初循环读取输入事件时，系统没有响应，调试打印鼠标位置后才发现逻辑问题，修改后交互才稳定。
* **任务切换与窗口刷新**：任务切换不稳定会导致窗口刷新异常或FIFO数据丢失，需要精确控制堆栈和寄存器保存与恢复。

**8.4 下一步改进想法**

* **扩展终端功能**：增加更多系统命令，支持文件操作、进程管理等。
* **完善多任务交互**：实现任务优先级调度、抢占式多任务，增强任务管理能力。
* **增强UI体验**：增加窗口缩放、最小化、拖拽优化，提升交互流畅性。
* **调试工具集成**：在操作系统内置简单调试打印和错误日志，提高调试效率。