

操作系统设计与分析文档

—基于课堂实验的微小内核

1. 实验背景与目的

1.1 背景

在我们学校的操作系统教学实验中，学生通常通过高级语言（如 C++）编写模拟文件管理系统。然而，这种高度封装的设计难以让学生理解操作系统底层的实际工作机制，例如文件系统如何与磁盘交互、操作系统如何处理硬件中断等。

Xv6 确实是一个优秀的教学操作系统，但其文件系统功能较为复杂且封装过深，例如使用多级目录结构和系统调用接口，这些特性虽然功能强大，但对初学者不够直观。更重要的是，xv6 将文件管理和其他内核模块整合在一起，缺乏清晰的模块划分，并且目前应该是没有带你从 0 搭建它的课程的。

本实验的设计源于一篇 dev 上的博客 <https://dev.to/frosnerd/writing-my-own-boot-loader-3mld>，在它的框架设计基础上，我们对其修改并添加了文件管理。意在简化操作系统的封装，通过构建一个基于 x86 架构的简化操作系统，帮助学生直接接触底层开发，亲身体验操作系统的基本功能如何实现，文件系统的管理。让学生去理解这种一步步设计 os 的过程，去理解计算机的一个很重要的思想，**层层封装！**

1.2 实验目的

1. **简化学习曲线**：通过一个简化的单级文件系统，让学生能够快

速上手，理解文件系统的基本概念和实现方式。

2. **理解底层机制**：通过直接操作硬件资源(如磁盘、显存、键盘)，帮助学生掌握硬件与操作系统之间的交互机制。
3. **模块化设计思维**：通过清晰的模块划分（如引导加载器、CPU 中断处理、设备驱动和文件系统），让学生理解操作系统的分层设计思想。（这里我就要提到 xv6 了，clone 下来就是一个文件夹里一堆文件，可读性极差。）
4. **教学实用性**：通过构建简单而完整的操作系统，帮助学生从零开始体验操作系统的开发过程，深入理解接口设计、资源管理等核心概念。

2. 系统总体设计

2.1 系统架构

该操作系统通过模块化设计，分为以下主要模块（各自在代码中就是一个文件夹）：

1. **引导加载器 (Boot)**：从磁盘启动系统，加载内核到内存，并切换到保护模式。
2. **CPU 管理 (CPU)**：提供中断管理功能，包括异常处理和硬件中断（如键盘、定时器）。
3. **设备驱动 (Drivers)**：包括显示驱动（字符显示、光标控制、屏幕滚动）和键盘驱动（扫描码转换、键盘中断处理）。
4. **文件系统 (FS)**：提供文件和目录管理功能，支持一级目录结构，实现文件的创建、读取、写入和删除。

5. **内核 (Kernel)**：作为系统的核心，初始化各个模块，并提供用户命令行接口。
6. **工具函数 (Utils)**：提供字符串操作、数据转换和内存管理等辅助功能。
-

2.2 模块划分

模块	主要功能
----	------

Boot	系统引导、内核加载、保护模式切换。
------	-------------------

CPU	中断描述符表 (IDT)、中断处理、定时器和硬件中断管理。
-----	-------------------------------

Drivers	屏幕显示、光标控制、键盘输入管理。
---------	-------------------

FS	文件与目录管理、磁盘扇区分配与释放，仅支持一级目录。
----	----------------------------

Kernel	系统初始化，命令行接口，连接各模块以实现用户交互。
--------	---------------------------

Utils	提供字符串、内存操作和数据转换功能，支持底层开发。
-------	---------------------------

3. 详细模块设计

3.1 引导加载器 (Boot 模块)

功能：

加载内核到内存指定地址（如 0x1000），并切换到 32 位保护模式。

设置全局描述符表（GDT），为保护模式提供段访问支持。

实现方法：

使用汇编语言编写，通过 BIOS 中断（int 0x13）读取磁盘数据。

设置内核入口点，并调用内核的初始化函数。

特点：

简单但功能完整，从实模式启动到保护模式切换，全程控制系统启动流程。

3.2 CPU 模块

功能：

提供中断管理，包括处理异常（如除零错误）和硬件中断（如定时器、键盘）。

实现方法：

定义中断描述符表（IDT）并加载到 CPU。

实现中断服务例程（ISR）和硬件中断请求（IRQ）。

提供定时器驱动，基于 PIT（可编程中断定时器）生成时钟“滴答”。

（这里更多的是模拟）

特点：

通过简单的中断框架，支持自定义中断处理函数。

集成了键盘和定时器的中断处理。

3.3 设备驱动（Drivers 模块）

3.3.1 显示驱动

功能：

控制屏幕字符显示、清屏、滚屏和光标位置。

实现方法：

直接操作 VGA 显存（0xb8000）显示字符。

通过 I/O 端口（0x3d4 和 0x3d5）控制光标位置。

特点：

支持多行滚动和退格操作，提供简单的屏幕管理功能。

3.3.2 键盘驱动

功能：

管理键盘输入，处理普通按键和特殊按键（如回车、退格）。

实现方法：

使用键盘中断（IRQ1）读取扫描码，并转换为 ASCII。

使用缓冲区记录用户输入内容，支持与内核交互。

特点：

支持字符输入和命令行交互，实时显示用户输入内容。

3.4 文件系统（FS 模块）

功能：

提供文件与目录管理，支持创建、读取、写入和删除文件。

使用磁盘扇区位图管理磁盘空间。

实现方法：

基于 LBA（逻辑块地址）模式实现磁盘的读写操作。

使用位图记录磁盘扇区使用状态，通过简单的根目录表管理文件和目录。

特点：

支持一级目录结构，根目录最多包含 64 个文件或目录。

通过直接操作扇区，帮助学生理解文件系统的基本概念。

3.5 内核 (Kernel 模块)

功能：

负责初始化系统模块，包括中断、文件系统、键盘驱动等。

提供命令行接口，支持用户通过输入命令操作文件系统。

实现方法：

在 `start_kernel()` 中完成模块初始化。

在 `execute_command()` 中解析用户命令并调用相应功能。

特点：

提供简单的命令行接口，支持文件系统和设备驱动的交互。

3.6 工具函数 (Utils 模块)

功能：

提供字符串操作、整数与字符串转换、内存管理等辅助功能。

实现方法：

通过简单的 C 函数实现，例如 `string_length()`、`int_to_string()` 和 `memory_copy()`。

特点：

简单但实用，为其他模块提供基础功能支持。

4. 系统实现环境

开发工具：

使用 VirtualBox 运行 Linux 环境，通过 SSH 连接 VSCode 进行开发。

使用 QEMU 模拟器运行操作系统，支持硬件交互调试。

目标架构： x86 架构。

工具链：

NASM 汇编器、i686-elf-gcc 编译器、i686-elf-ld 链接器。

5. 总结与展望

5.1 系统特点

1. 教学友好性：

系统设计简化，仅支持一级目录结构，减少不必要的复杂性，突出底层概念。

2. 模块化设计：

各模块分工明确，逻辑清晰，便于理解和扩展。

3. 直接硬件交互：

通过直接操作 VGA 显存、I/O 端口和磁盘扇区，帮助理解硬件与软件的关系。

5.2 实验意义

- 1 可以通过该实验掌握从零开发操作系统的基本技能。
- 2 理解模块化设计思想，学习接口封装与实现。
- 3 从硬件交互到文件系统操作，提供完整的实践机会。
- 4 也是我们的总结吧，去理解并实战计算机这种层层封装的这种思维形式。

5.3 改进方向

1. 支持多级目录结构，扩展文件系统功能。

2. 引入分页机制，学习更复杂的内存管理。
3. 实现简单的多任务调度，支持多任务切换。
4. 实现鼠标端口的调用与实现。

6.3 问题与解决

1. 问题：最开始遇到的第一个问题是远程连接的问题，有过跑人工智能大模型的经验，所以应该可以通过 ssh 连接这个系统，来写代码，但是在连接过程中，因为对 linux 的指令的不熟练，端口的映射问题，以及服务器地址的问题搞出来了很多麻烦。

解决：服务器地址应该是 ipconfig 的第二个地址，以及建议直接使用自己电脑的网络配置，不然有些环境装起来可能比较麻烦。

2. 问题：之前也提到过，代码是基于一个博客的 os 系统展开的，实际上做为初学者试过很多的 os 代码，包括 30 天自制操作系统，mit 课程的 xv6 等等，但最后发现很多跑不通，不够规范，或者是版本问题。

解决：也是很幸运的遇到这篇博客吧 <https://dev.to/frosnerd/writing-my-own-boot-loader-3mld>，让以从 0 开始搭建自己的内核。

3.问题：寄存器等配置没有设置好，在 bios 启动后，我的 qemu 会一直闪屏。

解决：需要严格按照 os 的设计流程来设计，很多寄存器，段描述符之类的，都要按照流程来配置。

4.c 语言不能使用，需要进入 32 位保护模式下，才能开始用。

5.这里就是偏计算与函数实现的很多问题了，需要对屏幕的浮标与字

符的大小清楚的认知并计算，在 80X25 的屏幕大小下，如何实现换行。计算字节大小等等。

6.ATA 端口的调用，需要搞清楚软盘与硬盘的相关概念，之前启动内核只靠软盘，然后尝试调用 ATA 端口时，发现不能用，于是便在不断的打印字符来逐步调试错误点，后来发现，软盘没这个端口……

7.也是可以给其他调试者的建议，可以通过直接打印磁盘内容来判断自己的一些语句是否发挥作用，这里 qemu 提供不少语句，我们可以自行查阅。这个解决了我百分之 30 的报错。

8.一个 os 内核对于初学者而言实际上还是很难上手的，非常建议大家去结合大模型来学习里面的部分思想，当然代码里我也加上了很多中文注释。