

AIGC检测 · 简洁报告单

NO:CNKIAIGC2025SG_202504100115244

检测时间: 2025-04-28 15:26:06

篇名: 自制操作系统的设计与实现
作者: 胡涛涛
单位: 江南大学
文件名: 自制操作系统的设计与实现.docx

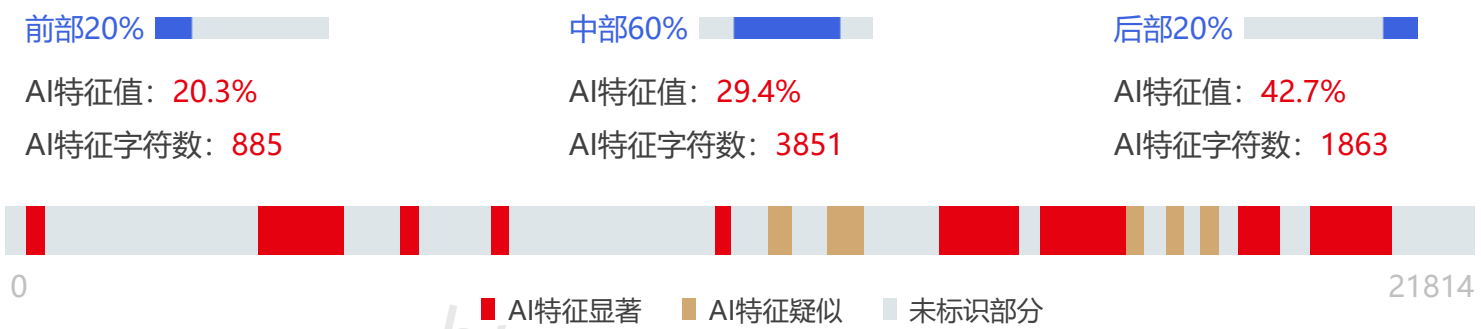
全文检测结果 知网AIGC检测 <https://cx.cnki.net>



AI特征值: 30.3%
AI特征字符数: 6599
总字符数: 21814

- AI特征显著 (计入AI特征字符数)
- AI特征疑似 (未计入AI特征字符数)
- 未标识部分

AIGC片段分布图



分段检测结果

序号	AI特征值	AI特征字符数 / 章节(部分)字符数	章节(部分)名称
1	10.6%	268 / 2518	中英文摘要等
2	42.2%	1513 / 3588	第1章绪论
3	19.8%	266 / 1342	第2章操作系统运行原理简述
4	21.1%	1723 / 8170	第3章系统结构设计与实现

5	41.4%	966 / 2335	第4章SpiderOS功能验证与测试
6	48.3%	1863 / 3861	第5章总结与展望

1. 中英文摘要等

AI特征值: 10.6%	AI特征字符数 / 章节(部分)字符数: 268 / 2518
--------------	---------------------------------

片段指标列表

序号	片段名称	字符数		
1	片段1	268	<div><div></div></div>	10.6%

片段详情

NO.1	片段1	字符数: 268	AI特征: 显著	<div><div></div></div>	10.6%
<p>我们的开发过程从手动编写MBR和加载器开始，逐步实现了从BIOS的实模式到内核的保护模式转换。之后，进行了分页机制的初始化、内存池的划分以及虚拟地址空间的位图构建等关键的内存管理操作。在多任务方面，我们用双向链表管理就绪队列和阻塞队列，结合主动让出和中断驱动的时间片轮转方式，确保了任务的公平调度。文件系统部分，我们完成了硬盘分区扫描、文件的读写操作、目录结构的解析，以及文件描述符的管理。用户可以在Shell命令行中轻松执行目录切换、新建文件和查看目录列表等基本操作。</p> <p>关键词： 操作系统；x86架构；内存管理；线程调度；文件系统</p>					

2. 第1章绪论

AI特征值: 42.2%	AI特征字符数 / 章节(部分)字符数: 1513 / 3588
--------------	----------------------------------

片段指标列表

序号	片段名称	字符数		
2	片段1	1254	<div><div></div></div>	34.9%
3	片段2	259	<div><div></div></div>	7.2%

片段详情

尽管第一代微内核在性能方面有一些缺陷，但德国的计算机科学家约亨·利德克（Jochen Liedtke）在90年代初提出了L4微内核。L4大大简化了内核的功能，改善了IPC（进程间通信）机制，系统性能也因此得到了明显提升。特别是，L4采用寄存器传输信息的方法，降低了内存访问的开销，让IPC的速度比Mach快了20多倍。更厉害的是，L4的大小只有12KB，远小于Mach内核的300KB，充分展现了小型化内核在性能和资源使用上的优势。随着人们对信息安全的关注不断增强，第三代微内核在保证高性能的同时，也更重视系统的安全性。seL4是这阶段的代表之一，它在L4的基础上引入了能力空间（capability space）机制，确保每个进程只能访问自己被授权的资源，从而提升了安全等级。更重要的是，seL4还是全球第一个经过正式验证的微内核，使用数学方法验证了其8700行的C代码的正确性和安全性，广泛应用于航空航天和汽车电子等对安全性要求非常高的领域。

1.2.2 国内外微内核操作系统

国外对第三方操作系统的研究可以追溯得很早。20世纪60年代，著名的IBM推出了世界上第一个分时多任务系统CTSS，为现代操作系统里的进程控制和时间片调度提供了基础。后来，随着UNIX的出现和开源，更带动了Linux、FreeBSD等开源内核的发展，形成了如今开放、稳定，广泛应用于服务器、嵌入式系统和云计算的Linux系统。实际上Linux是个非常厉害的操作系统，很多国家和企业的操作系统都借用了它的内核或思想，比如鸿蒙OS、Android、SteamOS等等。过去十几年里，随着移动设备和嵌入式计算的需求不断增加，大家对轻量级操作系统的研究变得更加热烈。开源社区也涌现出了许多适合物联网设备的微内核，比如FreeRTOS和Zephyr，它们已经开始商用。这些系统一般都注重简化功能、提升稳定性，非常适合用来开发嵌入式和实时控制系统。与此同时，谷歌推出的Fuchsia操作系统，采用了全新的Zircon微内核架构，它打破了传统的UNIX框架，为未来的操作系统设计提供了新的思路。

中国的操作系统研发和实践也在不断发展。早期比较有代表性的是中科院计算技术研究所开发的银河操作系统和普华的Linux。近几年，随着国产芯片的普及和新兴公司的支持，涌现出了像麒麟、鸿蒙这样的操作系统，它们面向桌面、移动设备和物联网等多个场景。这些新项目旨在让软硬件能更自主地运作，增强安全性、实现更好的互操作性，还推动相关环境的建设。同时，很多高校也开始推出一些实践性学习项目，比如南京大学的NEMU模拟器，鼓励学生通过编写和调试操作系统模块，培养软硬件协作的能力。如今，操作系统的教学不再只靠理论，而越来越偏重于基于具体项目的实践学习。国外的大学也有一些经典的入门操作系统案例，比如麻省理工学院的开源操作系统“xv6”和哈佛大学的“JOS”。这些课程主要介绍一些核心内容，比如虚拟内存管理、系统调用、文件系统和进程调度，帮助学生打下基础。

（6） 系统调用接口设计

这篇文章介绍了用户空间到内核空间的系统调用机制，涵盖了常见的系统调用类型，比如进程管理、文件操作、内存分配和屏幕显示等。用户程序通过软中断触发内核服务，实现用户态和内核态之间的安全切换，从而保证系统的稳定性和安全性。

（7） Shell命令行交互功能

为了让用户更方便地进行常用操作，我们设计了一个简单的Shell命令行解释器。这个工具支持像ls、pwd、mkdir、cd、rmdir这些基础命令，用户可以用它来创建文件、查看文件内容和切换目录，从而让系统的交互变得更加顺畅和直观。

3. 第2章操作系统运行原理简述

AI特征值: 19.8% AI特征字符数 / 章节(部分)字符数: 266 / 1342

片段指标列表

序号	片段名称	字符数		
4	片段1	266	<div><div></div></div>	19.8%

片段详情

NO.4	片段1	字符数: 266	AI特征: 显著	<div><div></div></div>	19.8%
------	-----	----------	----------	------------------------	-------

三、 设备管理，操作系统还需要负责设备管理。例如键盘、鼠标、硬盘、显示器这些硬件设备，都会由操作系统统一调度。在硬件插入后，虚拟地址空间会有一块内存被提供用于访问硬件设备的入口。它会借助设备驱动程序，把硬件的复杂接口变得简单，让上层的应用可以以统一的方式访问不同的硬件设备。操作系统还会安排设备的使用顺序，管理缓存，以提高数据传输的效率和稳定性。

五、用户界面最后，操作系统需要提供一个用于交互的页面。这通常是通过命令行界面（CLI）或者图形界面（GUI）实现的。用户可以通过这些界面输入命令、管理文件、运行程序或者调整系统设置。

4. 第3章系统结构设计与实现

AI特征值: 21.1% AI特征字符数 / 章节(部分)字符数: 1723 / 8170

片段指标列表

序号	片段名称	字符数		
5	片段1	230	<div><div></div></div>	2.8%
6	片段2	337	<div><div></div></div>	4.1%
7	片段3	545	<div><div></div></div>	6.7%
8	片段4	1171	<div><div></div></div>	14.3%
9	片段5	322	<div><div></div></div>	3.9%

片段详情

NO.5

片段1

字符数: 230

AI特征: 显著

2.8%

3.3.2 内存管理系统

在设计SpiderOS的时候，内存管理模块扮演着关键的角色。它不仅保证了内核和用户程序的稳定运行，也是操作系统在资源分配和隔离方面的核心所在。

系统启动完成后，进入内核阶段并开启中断后，SpiderOS会立即调用mem_init()这个函数，启动内存管理系统的初始化过程。这个函数会读取存放在0x800地址的BIOS中的信息，确认目前系统的实际物理内存容量，然后把这个信息传递给mem_pool_init()来进行下一步的设置。

NO.6

片段2

字符数: 337

AI特征: 疑似

4.1%

映射关系图3-4

3.3.3 线程与进程调度机制设计

调度机制在实现多个任务同时运行时起着非常关键的作用。它不仅要合理分配CPU资源，确保每个任务都能得到处理，还要让系统在多线程环境下反应迅速，任务切换也要尽可能平滑。为达到这个目标，SpiderOS采用了基于时间片轮转的调度方案，同时结合了线程状态管理、阻塞机制和优先级设定，打造了一个既简单又高效的调度框架。在SpiderOS中，每个任务都作为一个线程存在，而进程则相当于是拥有自己独立虚拟空间的线程扩展。内核用结构体task_struct来描述每个任务的主要信息，包括进程ID (pid)、当前状态 (status)、内核栈指针 (self_kstack)、优先级 (priority) 以及调度用的时间片 (ticks) 等。

NO.7

片段3

字符数: 545

AI特征: 疑似

6.7%

为了让系统资源的使用更加高效，SpiderOS 设计了一个空闲线程，也叫 idle_thread。当就绪队列里没有任何任务时，调度器会自动唤醒这个空闲线程，让它进入休眠状态，然后用 hlt 指令让

CPU 低功耗模式。直到遇到新的中断或任务，空闲线程才会被重新唤醒。

除了基本的时间片轮转调度，SpiderOS 还增加了阻塞和唤醒的机制。当一个线程因为资源还没准备好而无法继续运行时，它可以调用 `thread_block()`，把自己标记为 `BLOCKED`，然后放弃 CPU。等到资源准备好后，再用 `thread_unblock()` 让它回到就绪队列，准备下一次调度。这样，系统既可以聪明地等待资源，又不会陷入忙等状态，从而提高了并发效率。整个调度机制结合了简单直观的队列管理和灵活的状态切换策略，不仅确保了蜘蛛操作系统的公平轮转，还能快速响应各种请求。这套机制不仅是内核多任务管理的基础，也是保证操作系统稳定运行的重要保障。

流程执行图3-5

3.3.4 输入输出系统

SpiderOS 的输入输出（I/O）系统就像是连接用户程序和硬件设备之间的桥梁，直接影响着系统的交互效率和稳定性。这部分负责处理键盘输入、中断驱动、数据缓冲和控制台输出，完成从硬件信号到屏幕显示字符的整个过程。

NO.8 片段4 字符数: 1171 AI特征: 显著  14.3%

通过这种设计实现了用户进程的基本隔离、安全运行与高效调度，并为后续实现系统调用、用户态文件访问、进程创建与终止等功能提供了良好的基础。

3.3.6 文件系统

文件系统模块在整个操作系统中扮演着非常重要的角色，负责数据的存储和管理。为了实现既简洁又可靠的持久化存储，SpiderOS设计了基本的文件系统架构，涵盖了硬盘分区管理、文件的读写接口、目录的解析，以及管理文件信息的控制块（File Control Block, FCB）等关键部分。

在系统启动的早期，SpiderOS通过内核中的IDE设备模块识别硬盘，并用`partition_scan()`函数扫描硬盘上的主分区和逻辑分区，把有效的分区信息记录下来。系统会挂载默认的活跃分区，为后续的文件操作打好基础。每个分区由一个结构体存储，不仅包含起始LBA、扇区数，还内置超级块、块位图、inode位图和根目录信息，形成了对分区资源的管理单元。

文件系统采用一种简化的inode机制来管理每个文件。每次打开的文件都对应一个inode，记录文件的起始块、长度等重要信息。在系统启动时，调用`filesys_init()`函数来初始化整个文件系统。这包括读取存储在磁盘上的超级块、建立内存中的inode表、初始化文件表和目录结构。超级块保存了文件系统的基本信息，比如总的扇区数、inode的数量和数据区的起始位置，是系统正常运行的基础。

在文件操作的接口层面，SpiderOS提供了一系列系统调用，比如`sys_open()`、`sys_read()`、`sys_write()`和`sys_close()`，方便用户程序对文件进行操作。系统内部通过`file.c`模块处理具体的文件控制逻辑。例如，打开文件时会分配文件描述符、维护文件的偏移量，并进行块的映射计算。创建新文件时调用`file_create()`，而打开已有文件则用`file_open()`。

在路径解析方面，系统实现了基本的路径遍历逻辑。用户可以通过输入绝对路径或相对路径来访问

问文件，系统会调用search_file()函数，递归地逐级解析路径中的目录，最终找到目标文件或目录的inode。SpiderOS中的目录项(dir_entry)采用简单的结构，支持基本的目录读取和子文件查找操作。

关于目录管理，SpiderOS支持sys_mkdir()来创建新目录，内部通过调用dir_create()分配磁盘空间，并且会更新父目录的目录项信息。同时，也支持sys_rmdir()删除空目录，以及sys_opendir()打开目录等常用操作。

为了保证读写的稳定性和多线程环境下的安全性，SpiderOS在访问文件或目录时引入了简单的锁机制。尤其是对分区位图和inode表的修改，都采用加锁方式进行保护，以避免多线程同时操作时出现数据冲突或破坏。

NO.9

片段5

字符数: 322

AI特征: 显著

3.9%

命令的解析是由 cmd_parse() 来完成，输入的字符串会被空格拆分成命令和参数两部分。Shell 会检查输入的命令是否属于内置命令（比如 ls、cd、pwd、mkdir 等），如果是，就直接调用相关系统调用实现功能；如果不是，Shell 会打印出” external command”来提示用户命令未知。SpiderOS 的 Shell 支持基础的路径处理，比如使用相对路径或绝对路径切换工作目录，用户可以通过 pwd 查看当前路径，使用 mkdir 创建新目录，使用 ls 查看文件列表，展示了一个简单文件系统的基本操作。

此外，Shell 还能通过 ps 命令查看系统中运行的进程列表，结合内核的调度机制，让用户能够直观了解系统状态。

5. 第4章SpiderOS功能验证与测试

AI特征值: 41.4%

AI特征字符数 / 章节(部分)字符数: 966 / 2335

片段指标列表

序号	片段名称	字符数		
10	片段1	966	<div></div>	41.4%
11	片段2	287	<div></div>	12.3%
12	片段3	259	<div></div>	11.1%
13	片段4	287	<div></div>	12.3%

片段详情

第4章 SpiderOS功能验证与测试

4.1 内存管理功能验证

在 SpiderOS 内核的内存管理模块中，物理内存被划分为两个部分：一个是内核内存池（kernel_pool），另一个是用户内存池（user_pool）。这两个内存池通过位图的方式来管理物理页面的分配和回收。为了确保内存池的划分正确、位图能准确反映页面的状态，我们设计了一些测试用例，主要用来验证 `mem_pool_init()` 是否正确完成了内存池的划分，以及 `sys_malloc()` 是否能正确地分配和释放内存，位图是否清楚地显示了这些操作的结果。

测试设计（代码见附录1）：

为了确保 SpiderOS 的内存管理模块在初始化和分配页框方面都能正常工作，这次测试主要是检查内核内存池和用户内存池的分配流程。测试的关键是确认，内核内存池能通过 `get_kernel_pages()` 方法成功分配一段连续的物理内存页。此外，由于用户进程的虚拟地址空间的位图默认没有初始化，我们特别设计了 `user_mem_test()` 函数，手动设置了 `userprog_vaddr.vaddr_bitmap`，确保用户进程可以正常进行内存页的分配操作。

测试结果：成功

内存分配测试结果

4.2 线程与进程调度机制验证

验证 SpiderOS 内核中线程与进程调度模块的稳定性与正确性，设计了一组基于实际代码的功能测试，用来测试线程调度机制的核心功能，确保系统具备基本的多任务并发运行能力。

测试设计（代码见附录2）：

SpiderOS的调度机制采用时间片轮转和阻塞唤醒相结合的策略，线程通过 `thread_start()` 函数创建，并被插入就绪队列 `thread_ready_list`，调度器在每次时钟中断中自动轮换，依次调度各个线程。

测试结果：

4.3 输入输出系统功能验证

在 SpiderOS 操作系统的输入输出系统中，主要涉及对控制台输出的管理和键盘输入的中断处理。输入输出模块不仅承担了用户与内核交互的桥梁作用，还负责在多线程并发访问控制台时保证输出内容的正确性和互斥性。为了验证控制台输出的互斥机制和键盘中断的输入响应，本节设计了简单的测试用例来验证 SpiderOS 的输入输出功能是否按预期运行。

测试设计（代码见附录3）：

本次测试围绕 SpiderOS 的 `console_put_str()` 和 `keyboard_handler()` 两个关键函数展开

，主要测试两部分内容：

控制台输出可行性验证

在SpiderOS启动后，主线程通过`console_put_str()`输出字符串，验证控制台输出是否稳定可靠。

键盘输入中断测试


键盘输入模块通过注册中断处理函数 `keyboard_handler()`，实现用户按键触发中断并将扫描码写入 `ioqueue` 环形缓冲区。测试通过在输入过程中观察是否能够正确捕获按键输入，并在控制台实时回显输入内容，验证中断处理流程和输入缓存逻辑是否稳定可靠。

NO.12 片段3 字符数: 259 AI特征: 疑似  11.1%

测试设计（代码见附录5）：

1. 本次测试首先通过 `sys_mkdir()` 创建一个新的目录 `/testdir`，随后在该目录下调用 `sys_open()` 创建文件 `/testdir/hello.txt`，并使用 `sys_write()` 写入测试数据，最后使用 `sys_read()` 读取内容进行比对，确保数据完整一致，最后通过 `sys_close()` 关闭文件句柄。

此外，为验证目录解析的正确性，测试用例还调用 `sys_open()` 尝试访问不存在的路径，观察返回值是否正确处理错误路径，验证路径解析逻辑是否健壮。

NO.13 片段4 字符数: 287 AI特征: 疑似  12.3%

本次测试的核心目标，是验证 SpiderOS 的 Shell 是否能够正确接收用户输入，并完成命令解析、功能调用和结果回显。测试覆盖以下功能点：

1. 系统是否能正确显示提示符 `[SpiderOS@hutaotao /]:~$`，并等待用户输入。

2. 支持基础命令，如：

`ls` 列出当前目录文件；

`cd` 切换目录；

`mkdir` 创建目录；

`rm` 删除文件；

`pwd` 显示当前路径；

`ps` 查看进程列表。

3. 输入非法命令时，是否能正确回显提示“unknown command”。

测试结果：

根据以下测试图片，所有交互命令均正常运行并能处理非法命令。

6. 第5章总结与展望

片段指标列表

序号	片段名称	字符数		
14	片段1	625	<div><div></div></div>	16.2%
15	片段2	1238	<div><div></div></div>	32.1%

片段详情

NO.14

片段1

字符数：625

AI特征：显著

16.2%

5.2 系统不足与后续改进

尽管 SpiderOS 已实现了许多基本功能，但在实际应用中，系统仍存在一些不足之处，主要体现在以下几个方面：

Shell 功能简单

目前的 Shell 仅支持一些基础命令，比如创建文件和目录等，但功能相对单一，无法像主流操作系统那样支持管道、重定向等命令行功能，也不支持执行复杂脚本。因此，用户操作的灵活性和交互性有所局限。

无图形用户界面（GUI）

目前系统仅支持文本模式，没有提供图形用户界面，这让用户操作显得较为繁琐和单调。如果用户操作较多的情况下，界面的复杂度和使用体验会大大降低。

文件系统功能有限

SpiderOS 的文件系统目前还只是支持基本的文件和目录操作，功能上相对简化。比如缺乏高效的文件访问机制，文件的管理和访问速度较慢，不能满足大规模数据处理的需求。

缺少完善的系统调试功能

在调试和错误追踪方面，系统并没有提供完善的工具或机制，用户在进行系统开发或调试时，可能无法快速定位问题，缺乏高效的调试支持。

针对以上不足，未来的改进方向可以包括：

增强 Shell 的功能，支持更多的命令与操作，提高用户操作的灵活性。

设计并实现简单的图形用户界面（GUI），提升用户体验，使操作更直观。

改进文件系统，加入更高效的文件存储和访问机制，优化文件系统性能。

提供更完善的调试工具，帮助开发者在系统开发过程中更高效地排查和解决问题。

附录A： 测试集源码

附录1：

```
cur->userprog_vaddr.vaddr_start = USER_VADDR_START;
uint32_t bitmap_pg_cnt = DIV_ROUND_UP((0xc0000000 - USER_VADDR_START) / PG_SIZE / 8,
PG_SIZE);
cur->userprog_vaddr.vaddr_bitmap.bits = get_kernel_pages(bitmap_pg_cnt);
cur->userprog_vaddr.vaddr_bitmap.btmp_bytes_len = (0xc0000000 - USER_VADDR_START) /
PG_SIZE / 8;
bitmap_init(&cur->userprog_vaddr.vaddr_bitmap);
put_str("[+] userprog_vaddr bitmap init success!\n");
}
```

附录2:

```
void thread_a(void* arg);
void thread_b(void* arg);
int main(void)
{
    init_all();
    intr_enable();
    thread_start("thread_a", 31, thread_a, NULL);
    thread_start("thread_b", 31, thread_b, NULL);
    while (1) {
        put_str("Main thread is running\n");
    }
    return 0;
}

void init(void)
{
    uint32_t ret_pid = fork();
    if(ret_pid)
        while(1);
    else
        my_shell();
    PANIC("init: should not be here");
}
```

```
// 测试用线程A
void thread_a(void* arg) {
while (1) {
put_str("Thread A is running\n");
}
}
```

```
// 测试用线程B
void thread_b(void* arg) {
while (1) {
put_str("Thread B is running\n");
}
}
```

附录3:

```
int main(void)
{
init_all();
intr_enable();
while (1) {
delay(2000);
put_str("Main thread is running,Test input\n");
}
return 0;
}
```

附录4:

```
void file_system_test() {
put_str("\n[+] begin test filesystem...\n");
```

说明:

- 1、支持中、英文内容检测;
- 2、AI特征值=AI特征字符数/总字符数;
- 3、红色代表AI特征显著部分, 计入AI特征字符数;
- 4、棕色代表AI特征疑似部分, 未计入AI特征字符数;
- 5、检测结果仅供参考, 最终判定是否存在学术不端行为时, 需结合人工复核、机构审查以及具体学术政策的综合应用进行审慎判断。



cx.cnki.net

<https://cx.cnki.net>
知网个人AIGC检测服务

<https://cx.cnki.net>
知网个人AIGC检测服务