**编号**

JNDX_M

**本科生毕业设计（论文）**

**题目：** 自制操作系统的设计与实现

理 学院 信息与计算科学 专业

学 号 1131210333

学生姓名 胡涛涛

指导教师 陈蕾

讲师

二〇 二五 年 二 月

设计总说明

SpiderOS 是基于 x86 架构平台独立开发设计的轻量级教学操作系统，主要用途为学习研究，通过从自底向上逐步实现一套完整的操作系统，从而学习理解内核结构，进程调度，内存管理，文件系统，设备驱动，并提升开发者对操作系统体系结构的理解与动手能力。

本系统使用模块化的设计标准，功能模块边界明确，整个操作系统从启动加载到内核运行，最终完成用户交互。并且各模块相互独立，接口规范，且便于维护与扩展，能够良好支撑后续功能完善与特性增强。SpiderOS 的整体架构包括：引导加载模块、内存管理模块、线程与进程调度模块、文件系统模块、系统调用接口、I/O设备驱动模块、用户 Shell 命令解析模块。

本项目的开发流程从MBR和Loader加载开始，逐步的实现了又BIOS的实模式到内核的保护模式的转换。再之后进行了多个模块的初始化如引导加载模块、内存管理模块、线程与进程调度模块、文件系统模块、系统调用接口、I/O设备驱动模块、用户 Shell 命令解析模块。再完成这些模块后进入用户交互页面，用户可以使用一些基本命令来实现与系统的交互。

本次项目开发更注重整体流程框架的搭建，着重于自下而上的实现而非晦涩难懂的代码，能够降低开发者学习的门槛从而循序渐进的学习。

**关键词：** 操作系统；x86架构；内存管理；线程调度；文件系统

General description of the design

SpiderOS is a lightweight teaching operating system developed and designed independently based on the x86 architecture platform. The main purpose of the system is for learning and research, through the bottom-up step-by-step implementation of a complete set of operating system, so as to learn to understand the kernel structure, process scheduling, memory management, file system, device driver, and to improve the developer's understanding of the operating system architecture and hands-on ability.

This system uses modular design standards, with clear boundaries of functional modules, the whole operating system is loaded from startup to kernel operation, and finally complete user interaction. The overall architecture of SpiderOS includes: boot load module, memory management module, thread and process scheduling module, file system module, system call interface, I/O device driver module, and user shell command parsing module.

The development process of this project starts from MBR and loader loading, and gradually realises the conversion from real mode of BIOS to protected mode of kernel. After that, we initialise several modules such as boot loader module, memory management module, thread and process scheduler module, file system module, system call interface, I/O device driver module and user shell command parser module. After completing these modules, we enter the user interaction page, where users can use some basic commands to achieve interaction with the system.

This project focuses more on the overall process framework, focusing on bottom-up implementation rather than obscure code, which can lower the threshold for developers to learn step by step.

**Keywords:** operating system; x86 architecture; memory management; thread scheduling; file system

目录

[设计总说明 2](#_Toc1634291093)

[General description of the design 2](#_Toc1229215156)

[第1章 绪论 5](#_Toc1795950219)

[1.1 研究背景与意义 5](#_Toc61040448)

[1.2 国内外研究现状分析 5](#_Toc1850830639)

[1.2.1 微内核技术的发展 5](#_Toc178978259)

[1.2.2 国内外微内核操作系统 6](#_Toc1003691757)

[1.2 本文研究的主要内容 6](#_Toc831253949)

[第2章 操作系统运行原理简述 7](#_Toc7138075)

[2.1 x86架构计算机硬件简述 7](#_Toc1641580189)

[2.2 操作系统的核心功能   8](#_Toc408072166)

[第3章 系统结构设计与实现 9](#_Toc1487801100)

[3.1 运行环境简述 9](#_Toc273239164)

[3.1.1试验平台概述 9](#_Toc1726218045)

[3.1.2 编译工具链与开发环境 10](#_Toc854643494)

[3.2 引导程序实现 10](#_Toc1307429742)

[3.2.1 引导程序简述 10](#_Toc1423308923)

[3.2.2 启动加载器设计 10](#_Toc1871934547)

[3.2.3 从实模式到保护模式 11](#_Toc1422499605)

[3.3 内核核心模块设计与实现 11](#_Toc1229620158)

[3.3.1 中断管理机制 11](#_Toc1503683951)

[3.3.2 内存管理系统 12](#_Toc2077287256)

[3.3.3 线程与进程调度机制设计 14](#_Toc1760243713)

[3.3.4 输入输出系统 15](#_Toc1846031162)

[3.3.5 用户进程 16](#_Toc107227859)

[3.3.6 文件系统 16](#_Toc363044252)

[3.3.7 系统交互 17](#_Toc1835467900)

[第4章 SpiderOS功能验证与测试 18](#_Toc743304085)

[4.1 内存管理功能验证 18](#_Toc1785848906)

[4.2 线程与进程调度机制验证 18](#_Toc2101436117)

[4.3 输入输出系统功能验证 19](#_Toc691520620)

[4.4 文件系统功能验证 20](#_Toc1272656351)

[4.5 系统交互功能验证 20](#_Toc1183167625)

[第5章 总结与展望   21](#_Toc339987191)

[5.1 研究总结   21](#_Toc1333696799)

[5.2 系统不足与后续改进  21](#_Toc886514616)

[参考文献 22](#_Toc518965450)

[[1] 彭星海.基于x86架构的微内核操作系统的研究与实现.2020.电子科技大学,MA thesis.doi:10.27005/d.cnki.gdzku.2020.000410. 22](#_Toc189904908)

第1章 绪论

* 1. 研究背景与意义

操作系统，也就是我们常说的OS，是电脑里头非常核心的一部分软件，它负责管理硬件资源，比如处理器、内存、硬盘等等，还负责调度各种任务、驱动设备、支持文件系统，甚至还担着连接用户和硬件的桥梁角色。可以说，没有操作系统，电脑基本就没法用。在现代的电脑架构里，操作系统的稳定程度和可靠性直接关系到系统是否顺畅，所以它非常重要。

随着计算机科学与技术的飞速发展，主流的操作系统如Windows、Linux和macOS已变得日益复杂，功能愈发完善，但相应的源码体量和系统结构也越来越庞大，非常不便于阅读和学习。例如Linux第一个相对完整的版本在1991发布，当时的linux 0.11版本的内核才不到200K，而如今最新的4.20.13的Linux内核大小约100M，翻了几百倍[1]。可以体现增长速度可见一斑。而如今我们在看较出名的几款微内核大小，sel4微内核下载宝约有4M左右；hurd微内核大小约有3M左右；mach微内核大小约有4M左右，以及谷歌最新启动的全新操作系统fuchsia开发项目所使用的微内核zircon的下载压缩包大小为9M左右。对比之下我们可以看出无论是哪个内核，在学习上的成本都是巨大的。

所以自主设计一个简单的操作系统很有必要，可以让开发者从零自下而上的逐步熟悉操作系统的核心架构以及运行的原理。并且在实践中深入学习理解原理，并在模块的实现中巩固理论知识。加深计算机体系和操作系统原理的认知理解。

除此之外操作系统的开发对个人能力提升也有很大的进步，独立开发操作系统需要对底层编程能力基础扎实，而且还要有纵观大局的模块化思维能力，并且开发者还需要有良好的代码规范以及异常处理来面对各种可能的问题。在实际的开发中，开发者还得面对很多关键的技术比如模块的划分，接口设计，数据结构选择以及资源同步等问题。这些挑战都能很好的锻炼开发者的个人能力。也可以更深刻的理解软硬件之间复杂的协作关系，把握系统启动和运行的机制。

因此自制操作系统在学习研究领域有很重要的作用，对于即将从事计算机行业的人来说尤其是系统开发，也有着很重要且积极的促进作用。

1.2 国内外研究现状分析

1.2.1 微内核技术的发展

在操作系统的发展历程中，微内核因其模块化、高安全性和可维护性的优势，逐渐成为研究和应用的热点。

二十世纪八十年代中期的时候，CarnegieMellon大学开发了第一款微内核的操作系统：Mach操作系统。它创新性的把之前内核态的服务都移动到了用户态，来实现更强大的模块化并且更加灵活。当然在那个计算机硬件资源匮乏的年代，硬件性能有很大的限制，所以这一款操作系统在进程间通信(IPC)存在很大的开销，因此导致系统的整体性能不是很好。不过即使如此，它也对后面的诸多内核发展奠定了深远的基础，特别是在多线程支持和多处理器架构方面的探索(也就是现代操作系统)，这也是第一代微内核的创新性尝试。

即使第一代的微内核存在一些性能方面的缺陷，但是德国的科学家约亨·利德克（Jochen Liedtke）在二十世纪九十年代提出了L4微内核。L4表示第二代微内核，它吸收了第一代微内核设计上的一些经验教训，第一代微内核中Mach是最有名的实现之一。Mach和当时的其他微内核实现类似，没有自底向上地思考到底哪些功能应该在内核中实现，哪些不应该在内核中实现。其实它们看上去更像是拿到一个宏内核，然后再尝试将一些内核中的系统服务搞到用户层去。

L4考虑了这些问题，比如哪些服务在用户态运行并且不损失安全性和功能。比如L4内核甚至都没必要引入threads或scheduler的概念，只提供实现进程抢占的系统调用就可以（尽管实际情况是L4支持用户级线程）。微内核就是这样，提供最基础的功能，在不同场景中用户可以执行特定的策略来实现更加复杂的功能。

随后横空出世的seL4在L4的基础上引入了能力空间（capability space）机制，也是目前为止依然是第一个经过形式化验证的内核，形式化验证对 seL4 是个坚持不懈的努力目标，为了验证方便，禁止在内核里并发处理，不允许在内核态里再次发生中断，更重要的是，它成为有史以来第一个具有工业强度的通用操作系统，并正式证明了实现的正确性。

1.2.2 国内外微内核操作系统

国外对操作系统的研究起步非常早。20世纪60年代的时候著名的MIT就推出了世界上第一个分时多任务系统CTSS，允许多个用户通过终端连接到一台计算机并共享资源。CTSS提供了许多现代操作系统具备的功能，开创了许多操作系统领域的先河，为现代操作系统的发展提供了基础。

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，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。即使是现在依旧有大量的操作系统采用的他们的内核，可见其系统领先的地位，再后来谷歌推出了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在这几十年里也在不断发展。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一些大型公司的支持，涌现出了像华为鸿蒙这样的操作系统，他们通过重构内核，保留部分Linux较为优秀的模块，来实现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。除了企业级操作系统的发现，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，致力于培养学生软硬件协作的能力。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。

* 1. 本文研究的主要内容

这个设计主要围绕x86架构平台，目标是实现一个简单的操作系统。开发者可以结合实际经验，深入了解操作系统的完整运行流程，包括引导加载、内核初始化、内存管理和文件系统的工作过程。最终，大家可以自己动手，打造一个具有基本功能的操作系统。以下是具体的研究内容介绍：

1. **操作系统引导与启动机制**

本系统从零开发，首先用汇编语言编写MBR主引导程序和Loader加载程序，在通过Bochs模拟X86环境完成硬盘引导区的读取和内存加载的过程。最后把内核加载到特定的位置并且跳转执行初始化函数，完成从硬件到内核的关键环节。

1. **内核初始化与中断机制设计**

内核初始化包括了GDT全局描述附表的初始化，IDT中断描述附表的初始化以及PIC可编程中断控制器的初始化，实现了软硬件中断的注册和管理。并支持键盘输入，时钟中断等交互。除此之外，内核还设计了统一的中断服务框架，能够打印异常信息。

1. **内存管理模块的实现**

内存管理是操作系统最核心的资源管理系统之一，本文中设计了位图管理的物理内存分配算法，并实现了页级内存的分配和释放。支持内核线程，用户进程的虚拟地址空间管理，还完成了malloc内存申请和释放功能，为进程以及文件系统提供了可靠的内存支持。

1. **线程与进程管理机制**

线程的设计包括完成了内核线程的创建，调度和切换功能。实现了TCB线程控制块和PCB进程控制块的设计，在调度方面，采用了简单的优先级轮转调度算法，支持简单的线程阻塞和唤醒，能够基本实现线程同步。不仅如此，用户进程还可以直接通过系统提供的接口来完成从内核态到用户态的任务切换。并且支持进程的fork和exec加载。

1. **文件系统的设计与实现**  
    文件系统模块可以扫描磁盘的分区并进行挂载，采用自己设计的简单文件系统格式，实现了创建、读取、写入和删除文件的功能。它还支持创建目录和浏览目录结构，形成基本的嵌套。系统通过接口提供给上层程序，让用户可以方便地进行文件操作。
2. **Shell命令行交互功能**  
    本项目设计了一个简单的Shell命令行解释器来供用户与操作系统交互。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，实现了目前主流操作系统该有的功能（如Linux）。
3. 操作系统运行原理简述

2.1 x86架构计算机硬件简述

现代计算机最常见的架构为冯·诺依曼架构。如图2-1所示，这是一个x86计算机系统的硬件组成模型：

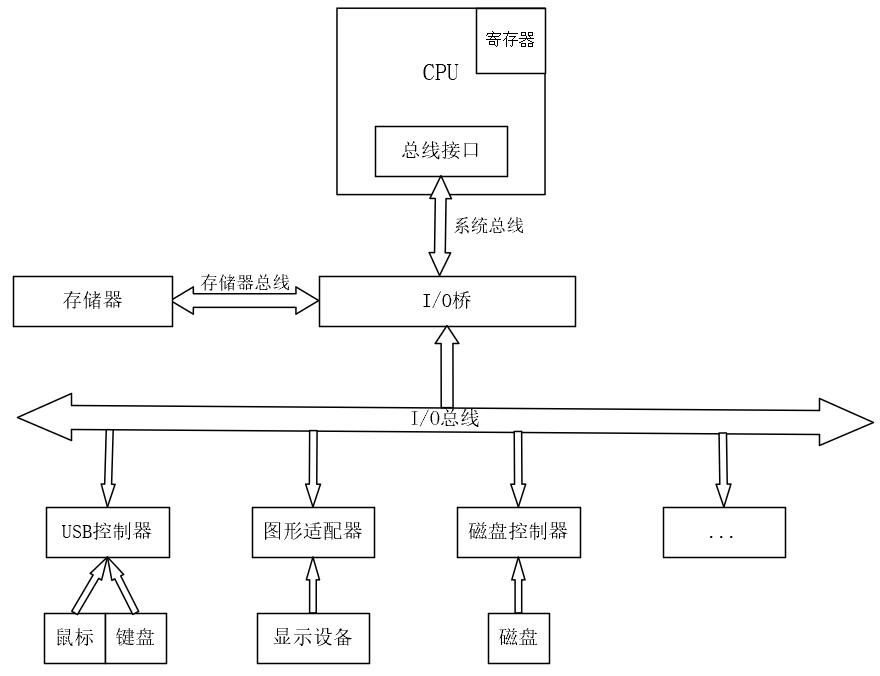


图2-1

传统的X86架构计算机系统中，硬件是由许多不同的子模块来组成，这些模块通过标准化的总线相互连接，相互配合协作来支撑操作系统以及应用程序的运行。如图2-1所示，现代操作系统主要包括了CPU,存储器，I/0桥,I/O总线.USB控制器，输入输出设备，图形适配器，磁盘控制器等等。其中CPU是整个系统的核心，也是大脑，负责指令的执行和数据的运算处理工作。它的内部继承了很多寄存器，用于存储变量和状态信息，并且可以通过总线接口和外部的模块进行通信。CPU通过系统总线连接到I/O桥，系统总线不仅承担着指令和数据的传输，还要负责协调多个模块的同步工作，比如存储器模块就通过存储器总线与I/O桥连接，在系统的启动和运行的时候CPU要频繁访问内存，这时候存储器总线就发挥了至关重要的作用。。I/O总线则与I/O桥相连接，用于连接各种输入和输出设备。不同的设备则需要专门的控制器连接到总线上，例如USB控制器可以连接键盘鼠标，图形适配器则需要连接显示器才可以进行图像的处理和输出，硬盘控制器则可以连接硬盘等设备，进行数据的存储和管理。其他的设备例如网卡，声卡等也可以通过I/O总线连接到整个系统。

2.2 操作系统的核心功能

操作系统是计算机中最基础也最重要的系统软件，它的主要作用是提供一套核心功能，让用户和各种应用程序可以顺利运行。操作系统帮助我们管控计算机的资源分配以及统筹调度。它的主要功能可以大致分为几个方面：

1. 进程管理 进程是计算机在运行中的一个任务或程序，也是是系统资源的基本分配单位。操作系统则负责进程的创建，调度，执行以及终止。他会对每个进程进行分配必要的资源，比如CPU的时间片，内存空间以及文件的访问权限等等。还会根据调度算法来执行多进程并发执行。
2. 内存管理，内存（RAM）是计算机中非常重要的资源，由于ROM的读写速度较慢，而RAM是ROM速度的一百倍左右，所以电脑会把正在执行的资源加载到内存中，再由内存和CPU进行数据交换。而操作系统就负责内存的分配和回收。在现代的操作系统中，一般使用的是虚拟内存技术。把物理内存扩展成虚拟的地址空间，让每个进程都有他独自的内存空间。另外，操作系统还会采用分页和分段的技术，把数据和程序拆分成块再映射到物理内存，来达到数据隔离的效果。不过也可以通过共享内存来让多个进程共享一块内存区域，方便数据的交流。
3. 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，本质上在他们插入计算机后都会受到操作系统的管理。在硬件插入后，虚拟地址空间会有一块内存被映射到硬件设备的入口用于访问，操作系统可以通过设备驱动程序，把硬件的复杂接口变得简单，来达到让上层的应用统一的方式访问不同的硬件设备的目的。操作系统还会统一管理设备的使用顺序，管理缓存，来提供更高效的数据传输以及稳定性。

五、用户界面 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。

第3章 系统结构设计与实现

3.1 运行环境简述

3.1.1试验平台概述

试验用主机配置：

|  |  |
| --- | --- |
| 架构: | x86\_64 |
| CPU: | 13th Gen Intel(R) Core(TM) i7-13700 |
| CPU核心数： | 24核 |
| 内存: | 24GB |
| 操作系统 | Ubuntu 24.04.2 LTS |

模拟运行环境：

Bochs是一款开源的x86架构模拟器，能够完整模拟硬件平台，包括CPU指令集、内存管理单元、I/O设备和硬盘控制器。Bochs在Ubuntu上很方便，因此SpiderOS采用的就是Bochs虚拟机作为主要的开发调试平台。

3.1.2 编译工具链与开发环境

SpiderOS的开发目标平台为x86架构，宿主平台为Linux操作系统，主要工具链如下：

|  |  |
| --- | --- |
| 工具名称 | 作用 |
| nasm | 用于汇编 mbr.S、loader.S、kernel.S 等汇编代码 |
| GNU LD | 用于将编译完成的目标文件链接，设置内核入口地址 0xC0001500 |
| dd | 用于将MBR、Loader和内核映像写入虚拟硬盘镜像文件 |
| Make | 用于自动化构建流程，管理编译与打包过程 |

3.2 引导程序实现

3.2.1 引导程序简述

在我们熟悉的x86架构电脑里，开机启动操作系统的流程基本上可以理解为“固件 → 引导程序 → 操作系统内核”。当电脑刚通电时，主板上的BIOS会先进行一系列的自检和初始化（也就是POST），然后按照预设的启动顺序去读取硬盘、U盘等设备上的主引导扇区（MBR），把它加载到内存的0x7C00地址，然后把控制权传给这个MBR里的引导程序。接下来，引导程序才会启动整个系统。SpiderOS的引导过程也遵循这个经典的流程，因此引导程序主要由两个部分组成：主引导记录（MBR）和加载器（Loader）。MBR的任务其实很简单，就是调用BIOS中断，把磁盘上的Loader加载到内存的0x600位置，然后把控制交给Loader。Loader会执行一些更复杂的初始化，比如检测内存、切换到保护模式、设置分页机制，最后把内核加载到内存并开始运行。

开发操作系统时，除了启动步骤之外，还要根据硬件环境调整，合理安排内存布局，确保CPU能顺利从实模式切换到保护模式。

3.2.2 启动加载器设计

在完成MBR的主引导程序后，Loader顺利的被加载到0x600的位置。然后Loader会进行更复杂的一些初始化流程，比如内存检测，从实模式切换保护模式（详见3.2.3），设置分页机制。最后会把内核加载到内存并开始运行。

对于大部分普通的电脑设备来说，加电后，CPU会先运行主板上的BIOS代码，完成硬件检测和初始化，然后按照预设的启动顺序进行引导。我们用Bochs模拟环境，首先创建一个空白的镜像文件hd40M.img，将主引导扇区（MBR）加载到这个镜像的第一个扇区。为了避免干扰或覆盖，系统会把Loader放在第3个扇区（LBA 2），而内核kernel则加载到第10个扇区。整个hd40M.img的结构可以参考图3-1。

hd40M.img

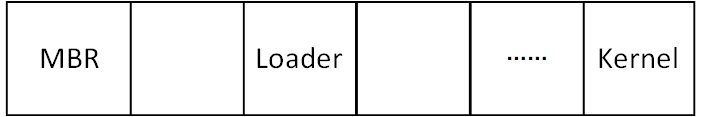


图 3-1

3.2.3 从实模式到保护模式

在使用x86架构的电脑时，一开机，CPU首先进入一种叫做实模式（Real Mode）的状态。在这个状态下，CPU操作的都是物理地址，由于寻址方式为16位的段寄存器的内容乘以16当作段基地址，加上16位的偏移地址形成20位的物理地址，所以它最多只能访问到1MB的物理内存。而且，实模式下还不支持现代操作系统的许多功能，比如内存保护、多任务处理和分页管理。因此，启动过程中，操作系统需要迅速的把CPU从实模式切换到保护模式（Protected Mode），这样才能为内核提供一个稳定可靠的硬件环境。

SpiderOS的启动加载器（Loader）在检测完内存后，会按照一定的步骤，从实模式平稳地切换到保护模式。一开始，BIOS会把主引导扇区（MBR）加载到内存的0x7C00地址，然后CPU会跳转到这个地址开始执行MBR中的代码。经过一番处理后，会调用硬盘读函数rd\_disk\_m\_16，把硬盘上的Loader程序读取到内存中，然后用jmp指令跳转到Loader的内存位置，把控制权交给Loader。

Loader的加载地址通过宏定义LOADER\_BASE\_ADDR控制，在SpiderOS中设置的是0x900。Loader比MBR任务更复杂，它要进行内存检测，确认机器支持多大内存；接下来会设置全局描述符表（GDT），为在切换到保护模式后提供正确的段描述符和属性；最后，它会正式切换到保护模式，完成关键的转换。所有准备工作完成后，Loader会用jmp指令跳转到内核的main函数地址，也就是交还控制权给核心部分，从此进入C语言编写的世界。

3.3 内核核心模块设计与实现

顺利地完成了从实模式到保护模式的切换。当引导程序结束后，CPU的控制权就正式交给了内核代码，此时SpiderOS的内核会进行一系列的初始化。内核就像系统的大脑，主要负责处理用户的各种请求，协调硬件设备，确保整个系统平稳运行。

3.3.1 中断管理机制

中断机制在操作系统内核和硬件之间起着关键的桥梁作用。SpiderOS设计的中断子系统搭建了一个完整的响应体系，让硬件发出信号或出现故障时，CPU能快速而安全地切换到内核进行处理，这样系统才能保持平稳运行。至于x86架构，它为中断管理提供了硬件基础，那就是中断描述符表（IDT）。

中断描述符表（类似图3-2所示）中存放着129个中断向量，每个向量都对应一个中断门描述符，告诉CPU：当这个中断发生时，应该跳转到哪个地址去执行相关程序。SpiderOS用结构体gate\_desc来抽象这个表项，里面包括中断服务函数的地址（包括低16位和高16位），段选择子，属性等信息。系统启动时，内核会准备好这个中断描述符表，填充每个表项的具体中断处理程序地址，通常由id\_desc\_init()函数完成。填充完毕后，通过lidt指令通知CPU表的起始位置和大小。此时，CPU已经可以处理中断了。当硬件设备（比如键盘输入、硬盘读写完成等）发出中断信号，CPU会根据中断向量自动从IDT中找到对应的中断门描述符，跳转到指定的中断处理程序。通常，我们会用汇编入口进入，然后保存现场、调用C层的general\_intr\_handler来统一处理。在这个过程中，CPU会自动切换堆栈、保护现场，确保用户程序的状态不被破坏。



gate\_desc数据结构图 3-2

SpiderOS的中断管理系统中，有一个叫做register\_handler()的函数接口，它让内核或驱动程序可以把特定的处理函数绑定到某个中断向量上。例如键盘驱动会把keyboard\_handler()注册到键盘中断对应的中断向量上，硬盘驱动则会将intr\_hd\_handler()绑定到硬盘的中断向量。当硬件上发生事件的时候，用户无需直接干预，系统会自动调用对应的处理程序，帮助硬件和内核之间传递数据。为了防止多个中断处理相互干扰，SpiderOS还提供了intr\_disable()和intr\_enable()这两个函数，用来在一些关键操作时临时屏蔽中断。关闭中断后，内核可以安全地修改重要的数据结构，避免被其他中断打断导致的问题，等处理完毕，再恢复中断的状态。

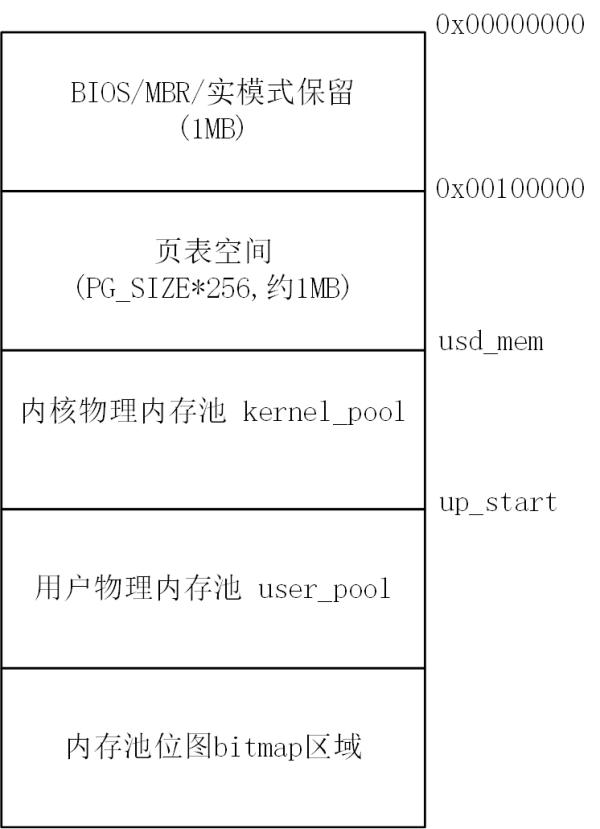
3.3.2 内存管理系统

内存管理模块在操作系统当中扮演着关键的角色。它不仅保证了内核和用户程序的稳定运行，也提供了资源分配和内存隔离的核心功能。

系统启动完成并进入内核后，SpiderOS会调用init\_all()函数进行初始化，并会立即调用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。

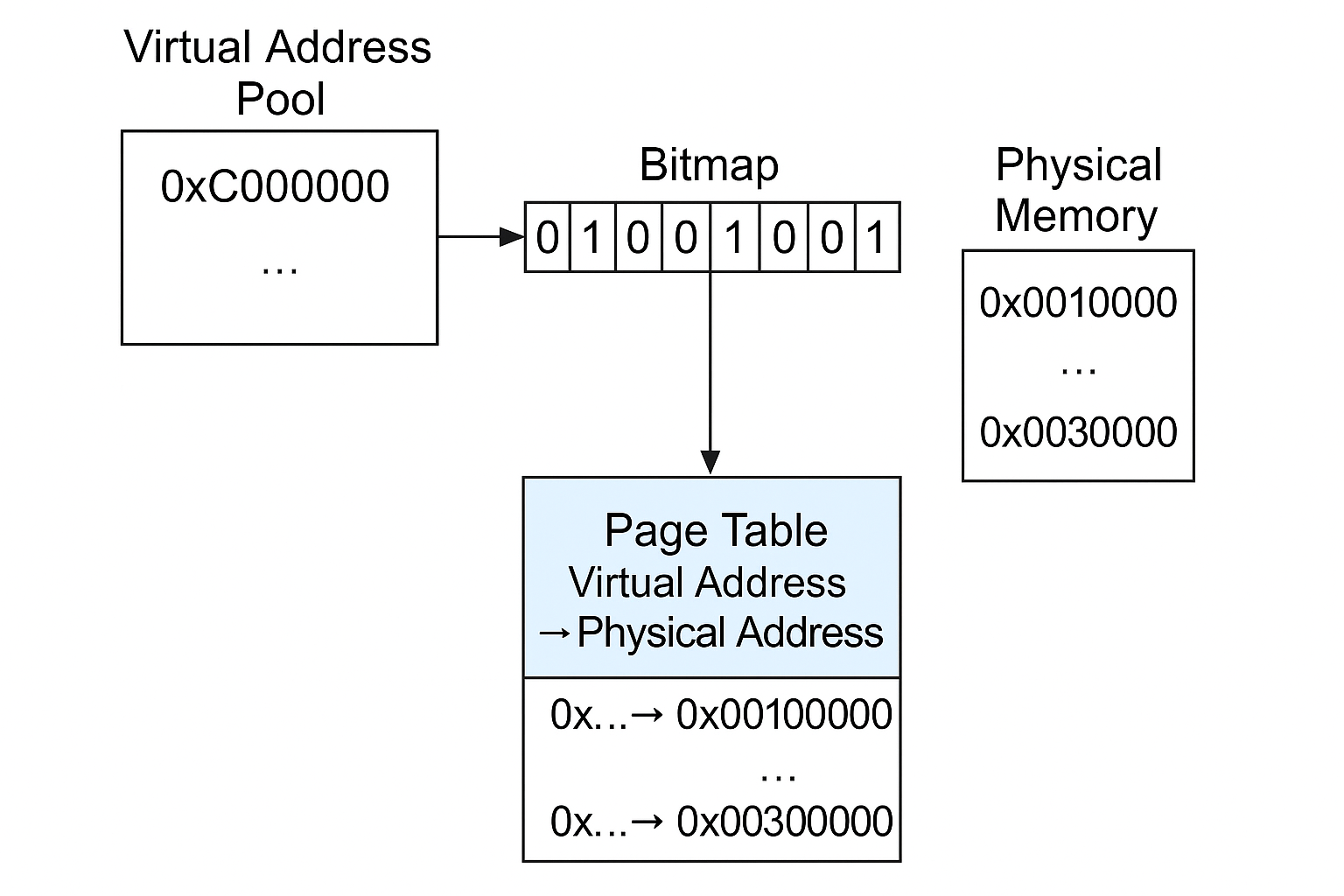
而mem\_pool\_init()是内存池的核心部分，它首先会根据页表的结构计算出用来做分页机制的页表空间大小，默认情况下，操作系统会把低端1MB的空间（0x100000）和供页表使用的空间（256个4KB页）预留出来。这个区域由页表占用，操作系统本身无法直接调度。剩余的空闲物理内存会被划分成两部分：一部分用作内核专用的物理内存池（kernel\_pool），另一部分用作用户空间的内存池（user\_pool），大概各占一半。为了让内存管理更高效，SpiderOS使用一种叫位图（Bitmap）的标记方式，每一位对应一页物理内存。0代表空闲，1代表已被分配。

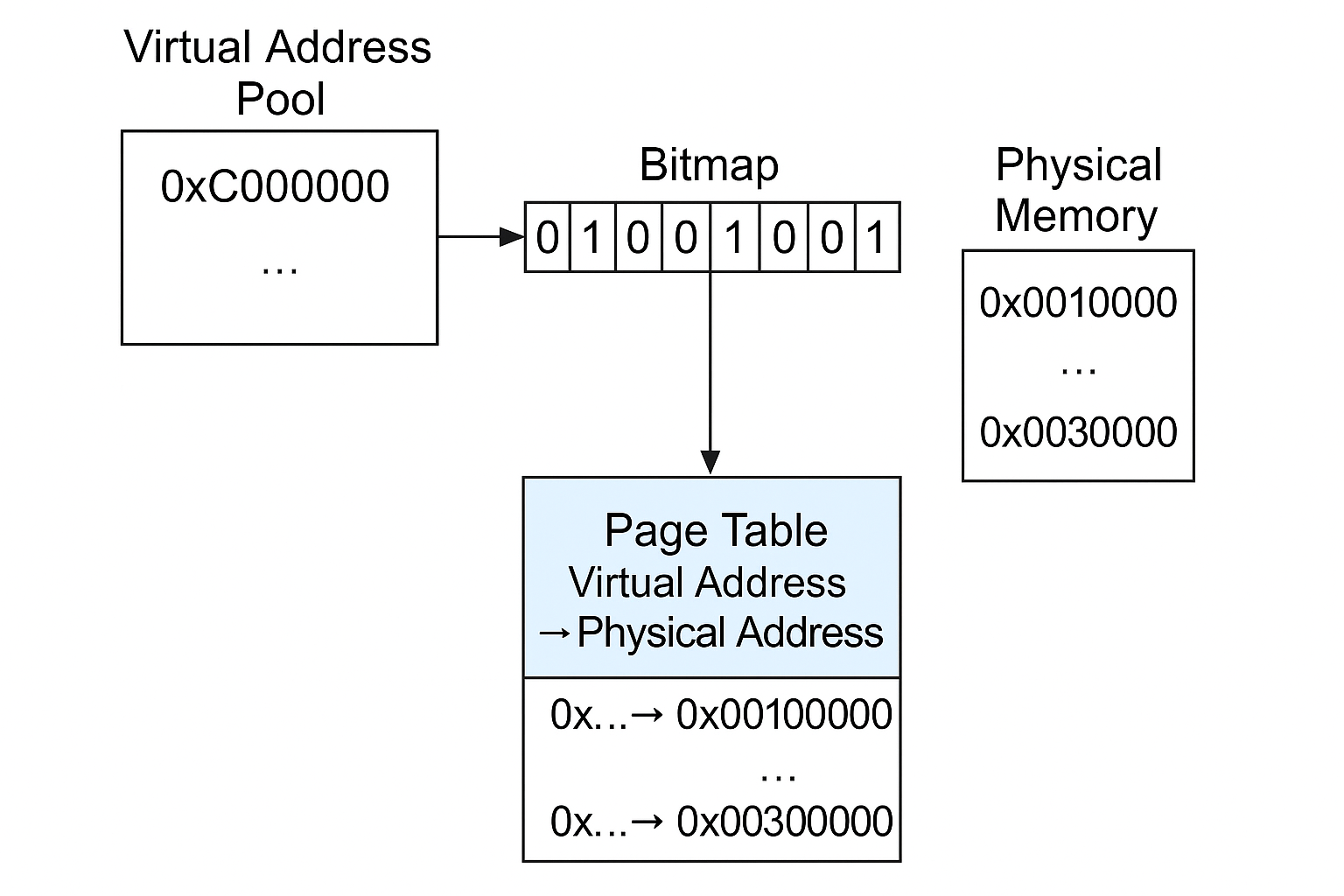
内核和用户的内存池都拥有独立的位图，内核位图存放在MEM\_BITMAP\_BASE的地址，用户空间的位图紧随其后，这样可以节省空间。用这种方式，系统能够用很低的开销，快速准确地跟踪每一页内存的使用情况。



内存池结构图3-3

完成内存池初始化后，操作系统可以通过get\_user\_pages函数申请内存。当调用get\_user\_pages时，内核首先会用vaddr\_get()函数查找虚拟地址。如果找到连续空闲的pg\_cnt个虚拟页，就会循环调用oalloc()函数来分配对应的物理页，然后通过调用page\_table\_add()，在页表中建立虚拟地址与物理地址的映射关系（映射示意如图3-4所示）



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。它要合理分配CPU资源，确保每个任务都能得到处理，并且在多线程环境下还要迅速的响应，任务切换也要尽可能平滑，为达到这个目标本项目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，打造了一个简易的调度框架。在SpiderOS中，每个任务都作为一个线程存在，而进程则拥有自己独立的虚拟空间。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（详细结构图见3-5）。

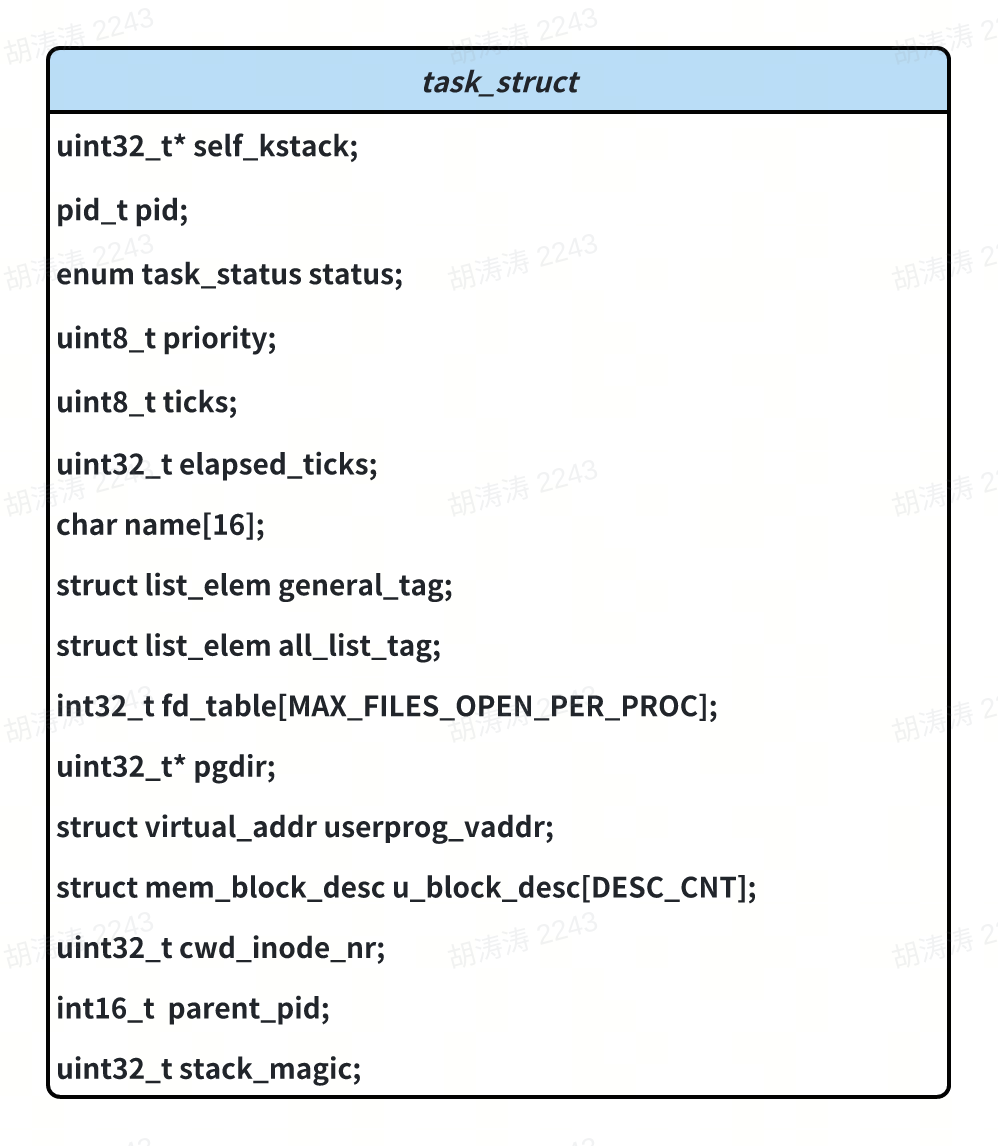
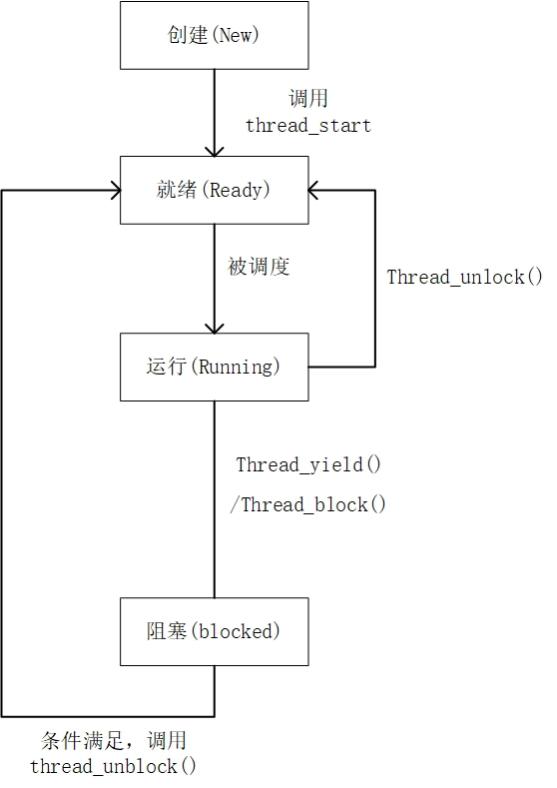


图3-5

通过这套结构，操作系统可以准确地管理每一个正在运行或等待的任务。线程的创建流程也很清楚，主要包括初始化、栈布局配置以及加入调度队列。当一个新线程通过thread\_start()函数创建时，系统会先调用init\_thread()完成PCB（进程控制块）的初始化，然后用thread\_create()设置好线程的栈，最后把它加入到 ready\_list（就绪队列）中，由调度器决定何时运行。而主线程main\_thread则通过make\_main\_thread()进行特殊的初始化，不需要重新配置栈空间，而是直接使用已有的空间，并挂载到就绪队列中等待调度。

调度的核心由 schedule() 函数负责，这个函数会在当前线程耗尽时间片，或者当你主动调用 thread\_yield() 的时候触发。调度器会先把当前正在跑的线程放回就绪队列，然后从 thread\_ready\_list 里挑选下一个准备运行的线程。通过 switch\_to() 来切换线程的执行环境，确保多任务能够顺畅地同时进行。每个线程的状态保存信息（像寄存器和栈）都存放在它的 PCB 里，这样切换时就能保证程序继续从正确的位置运行（看流程图3-5）。



流程执行图3-5

为了更高效地利用系统资源，避免戏院的浪费，SpiderOS 设计空闲线程（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，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，阻塞自身的运行。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，系统既可以避免未就绪的任务持续占用资源而造成浪费，从而提高了并发效率。整个系统的调度机制参考了如今的Linux系统以及主流的调度设计机制，并结合了简单直观的队列管理和灵活的状态切换策略，不仅确保了操作系统的公平轮转，还能快速响应各种请求。

3.3.4 输入输出系统

SpiderOS 的输入输出（I/O）系统是连接用户程序和硬件设备之间的桥梁，影响着系统的交互效率以及稳定性。操作系统最终还是面向用户，因此就需要一套合理的输入输出系统来响应用户的命令。本系统的I/O模块负责处理键盘输入、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。

在 SpiderOS 里，I/O 系统主要由 keyboard.c、ioqueue.c 和 console.c 这几个模块组成。 键盘模块通过设置中断向量，接收来自硬件的扫描码，然后把它们放到 ioqueue 的环形队列中进行缓冲，这样就避免了硬件输入速度和处理速度不一致的问题。这样一来，用户输入就可以以异步的方式稳定传输。当用户程序调用相关接口时，系统会从 ioqueue 取出字符，保证输入的顺序。而输出部分由 console.c 负责。它实现了把字符显示到屏幕上的功能，底层通过直接操作显存，把字符写入到内存地址 0xb8000。同时，它还会用 set\_cursor() 动态更新光标位置，让终端的表现更自然，更顺畅。此外，I/O 系统在设计时也考虑了同步问题，采用信号量和锁机制，确保在多个任务同时进行输入和输出的操作时，数据不会出错。在多线程环境下，SpiderOS 的 I/O 子系统通过对缓冲区和打印操作加锁，有效避免了因为线程抢占带来的字符混乱。

3.3.5 用户进程

在完成以上的工作后，终于可以进入用户进程了，用户进程的设计实现充分体现了操作系统内核与用户空间的隔离思想，保障了系统稳定性与进程独立性。用户进程作为用户程序运行的基本载体，是内核资源的调度和分配的核心单元。通过进程的引入我们可以实现了多用户程序并发运行的目标。

用户进程的创建主要由 process.c 文件中模块完成，函数 process\_execute() 作为用户进程入口，负责完成用户态程序的内存空间的分配，进程控制块的初始化，页表设置以及一些内核线程的准备工作。在进程创建阶段，SpiderOS 会首先调用内存管理模块，为用户进程单独申请用户态虚拟地址池，来确保用户程序在运行时拥有独立的地址空间。同时，系统会分配一套独立的页目录，隔离用户态与内核态的地址映射关系，实现了基本的地址保护。

每一个用户进程都通过task\_struce数据结构和线程共享基础调度的框架，但是区别在于pgdir成员指向了单独的页表，而内核线程该项为NULL,这样就可以实现用户进程的虚拟地址空间与内核线程的物理内存安全隔离。除此之外用户进程的userprog\_vaddr结构体用来标记用户态虚拟地址的分配状态，来实现用户态内存的高效管理。在用户进程的调度与运行过程中，SpiderOS 保持了统一的线程调度机制，用户进程与内核线程同属于 thread\_ready\_list，通过时间片轮转方式公平竞争 CPU 使用权。当用户进程被调度时，process\_activate() 函数会加载其页表，确保 CPU 访问的是用户进程自己的虚拟空间，而不是其他进程或内核的内存区域。

3.3.6 文件系统

文件系统是操作系统用于明确存储设备（如磁盘或固态硬盘）上的文件的方法和数据结构。它是软件系统的一部分，通过特定的方式组织和管理数据，使得应用可以方便地使用抽象命名的数据对象和大小可变的空间。简单来说，文件系统负责在存储设备上创建、删除、存储、检索和修改文件，同时保证数据的完整性和安全性。SpiderOS设计了基本的文件系统架构，包括了硬盘分区管理、文件的读写接口、目录的解析，以及管理文件信息的控制块等关键部分，即使整个文件系统比较简易，但麻雀虽小五脏俱全，在SpiderOS中可以完整运行。

在系统初始化的后期，SpiderOS会调用filesys\_init()函数进行初始化文件系统。我们重点来研究filesys\_init()函数：首先系统会调用sys\_malloc()分配一个扇区大小的内存空间用于临时村吃从磁盘分区读取的超级快数据，接着扫描IDE通道的adb从盘，并一次检查其中的主分区和逻辑分区。并让每个分区的起始地址+1扇区内容，即该分区的超级块。然后会判断是否格式化并格式化未初始化的分区，建立空白的文件系统结构，如果已经存在就会跳过。扫描完成后系统会遍历分区链表，查找sdb1分区并挂载，作为系统文件操作的默认分区。接着调用open\_root\_dir()函数打开当前挂载分区的根目录，并建立内存中的根目录结构，以便于支持后续目录查找和文件创建等操作。最后会初始化全局文件描述附表file\_table[]，它的大小为MAX\_FILE\_OPEN，用来管理系统可以同时打开的文件数量，并且初始化的时候所有表项的fd\_inode均被设置为NULL表示未使用。

介绍完文件系统的初始化流程，我们来看具体实施的细节。本项目的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了一个inode节点，这个节点记录文件的起始块、长度等重要信息。而上文提到的超级块则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。

在参考linux系统内核的时候，文件操作会提供一些接口，因此为了学习这一巧妙设计，SpiderOS也实现了这一功能，比如sys\_open()、sys\_read()、sys\_write()和sys\_close()，方便用户程序对文件进行操作。调用这些接口的时候内核会通过file.c模块处理具体的文件控制逻辑。比如打开文件时会分配文件描述符、维护文件的偏移量等等。创建新文件时调用file\_create()，而打开已有文件就会调用file\_open()。并且本系统也提供sys\_mkdir()接口来创建新目录，内部则是通过调用dir\_create()来分配磁盘空间，并更新父目录的目录项信息，同时sys\_rmdir()支持删除空目录。

操作系统必定离不开路径的解析，本系统也同样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节点。

为了保证读写的稳定性和多线程环境的安全性，比如防止两个线程同时打开一个目录并写入数据这种情况。SpiderOS引入了简单的锁机制，在访问文件或者目录的时候会加一层锁来保护，来避免多线程同时操作对数据冲突产生的破坏。

3.3.7 系统交互

用户和操作系统的互动主要靠命令行终端（Shell）来实现。Shell 就像是系统的一个指挥官，不仅是用户输入指令的地方，也能直接展现出内核的功能。它负责解析用户的命令、执行任务以及显示结果，是连接用户和核心系统的重要桥梁。SpiderOS 的 Shell 模块以简洁实用为目标，主要围绕一个叫做 my\_shell() 的主循环函数来设计。当系统启动完毕后，init 进程会调用这个 my\_shell()，把它绑定到用户的终端上，从而开始等待用户的指令。Shell 一开始会显示一个提示符 [SpiderOS@hutaotao /]:~$，然后等待用户输入命令，解析后执行相应的操作。

而命令的解析会由cmd\_parse()来完成，解析的逻辑就是会把用户输入的字符串通过空格拆解成命令和参数两部分。Shell会检查命令是否属于系统内置的命令(比如实现的ls,cd,pwd,mkdir等等）,如果是，就调用到相关的命令来实现功能。如果不是，Shell就会打印出”external command”来提醒用户命令未知。SpiderOS的shell还支持基础的路径处理，比如使用相对路径或者绝对路径来切换工作目录，例如cd /workspace/test1 这种。同时还实现了ps命令来查看系统正在运行的进程列表，输出的进程格式参考了经典的Linux结构。

第4章 SpiderOS功能验证与测试

4.1 内存管理功能验证

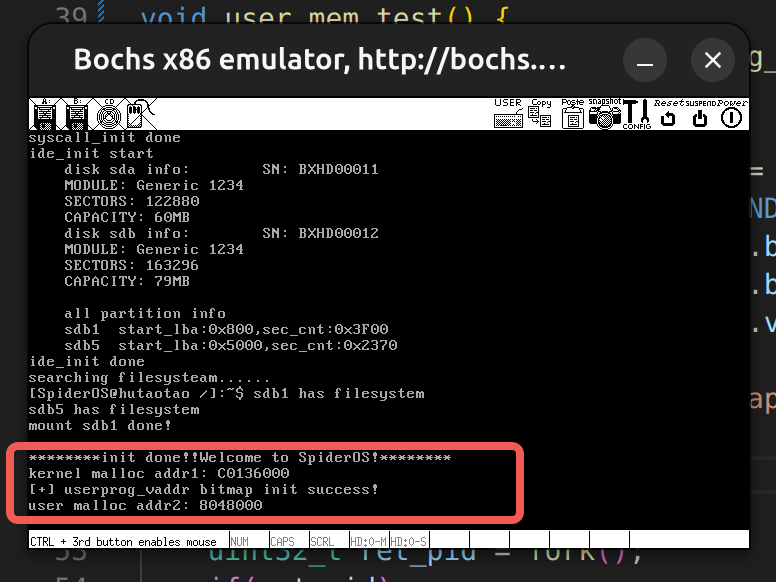
在 SpiderOS 内核的内存管理模块中，物理内存被划分为两个部分：一个是内核内存池（kernel\_pool），另一个是用户内存池（user\_pool）。这两个内存池通过位图的方式来管理物理页面的分配和回收。为了确保内存池的划分正确、位图能准确反映页面的状态，下面给出测试用例。

被测试功能：内存管理功能

测试目的：内核空间以及用户空间的内存分配功能

期望结果：分配成功，能够返回分配的空间地址

测试结果描述：结果与需求相符，通过。结果如图4-1所示。

  
 图4-1 内存分配测试结果

4.2 线程与进程调度机制验证

线程与进程调度机制是操作系统多任务并发运行的核心机制，下面给出测试用例，通过两个线程A,B循环打印来模拟用户的工作。

被测试功能：线程与进程调度功能

测试目的：测试多线程同时运行模式下是否每个线程可以正常工作。

期望结果：A,B两个线程可以交替打印运行LOG。

测试结果描述：结果与需求相符，通过。结果如图4-2所示。

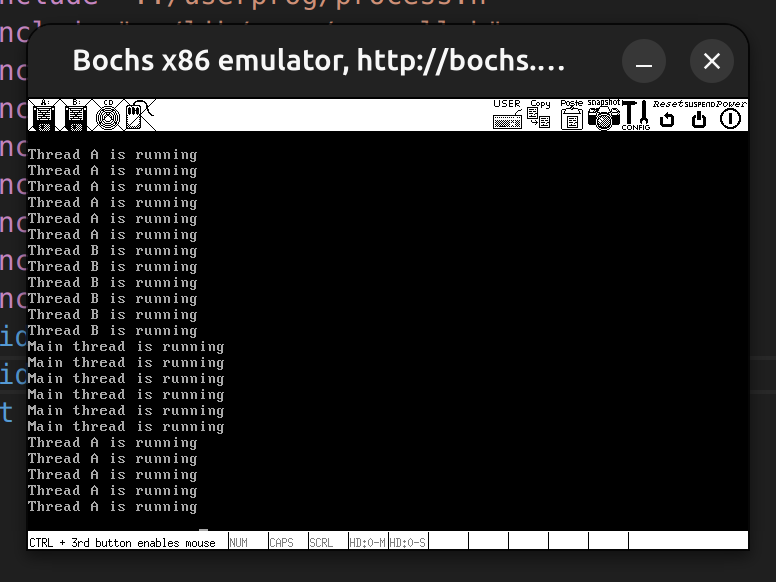


图4-2

4.3 输入输出系统功能验证

输入输出系统涉及用户的交互，是操作系统关键的功能之一，因此本测试用例围绕主线程输出和用户输入两个功能进行测试验证。测试用例如下：

被测试功能：输入输出系统

测试目的：测试主线程运行（模拟现象为主线程循环打印）的同时可以通过终端输入自定义指令。

期望结果：主线程打印结果的同时用户可以输入命令并通过SHELL回显。

测试结果描述：如下图4-3所示，可以看到主线程在循环输出"Main thread is running,Test input\n"的同时，用户可以输入hutaotao，控制台正确的实时回显输入内容。测试结果正常。

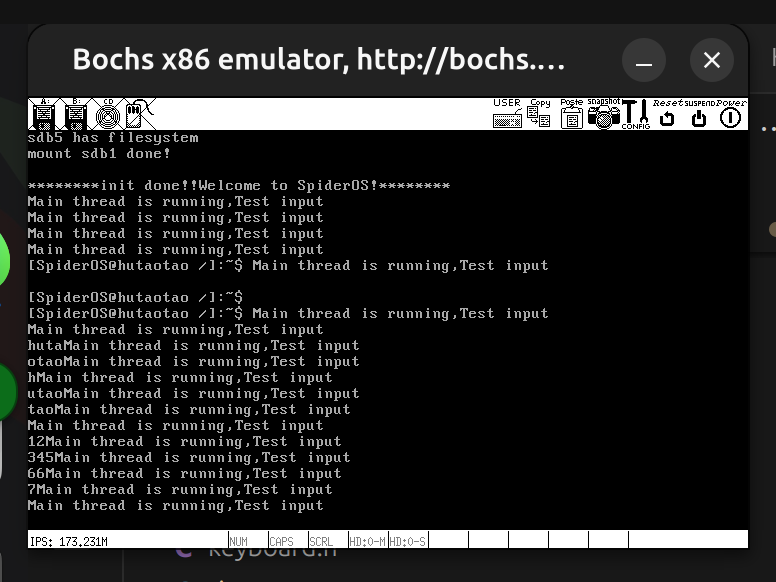


图4-3

4.4 文件系统功能验证

为了验证文件系统模块的正确性，主要围绕文件创建、读写、目录解析等几个核心功能进行测试，确保 SpiderOS 的文件系统在正常条件下能够完成文件的基本操作。

被测试功能：文件系统

测试目的：测试文件系统的创建，读写，切换目录，删除，以及异常处理功能是否正常。

期望结果：功能可以正常运行并通过shell显示运行结果，同时未知命令可以打印报错。

测试结果描述：结果与需求相符，通过。结果如图4-4所示

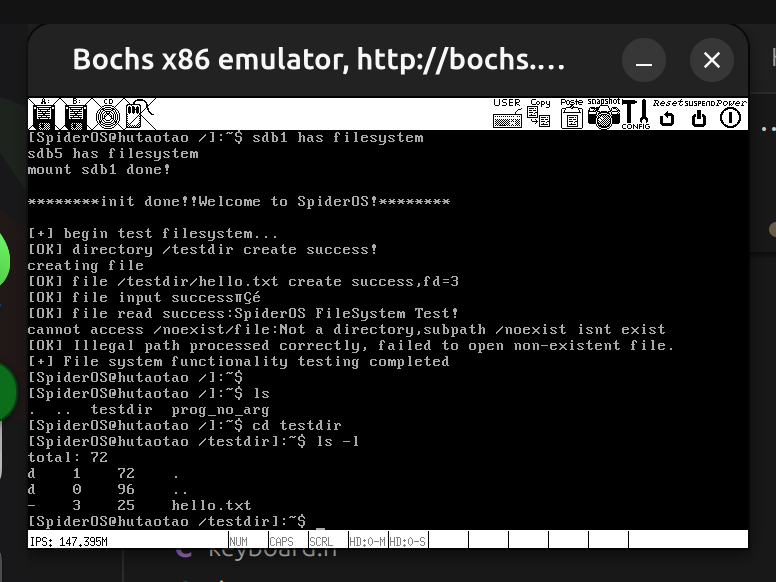


图4-4

4.5 系统交互功能验证

被测试功能：交互功能

测试目的：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。

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

测试结果描述：结果与需求相符，通过。结果如图4-5&4-6所示

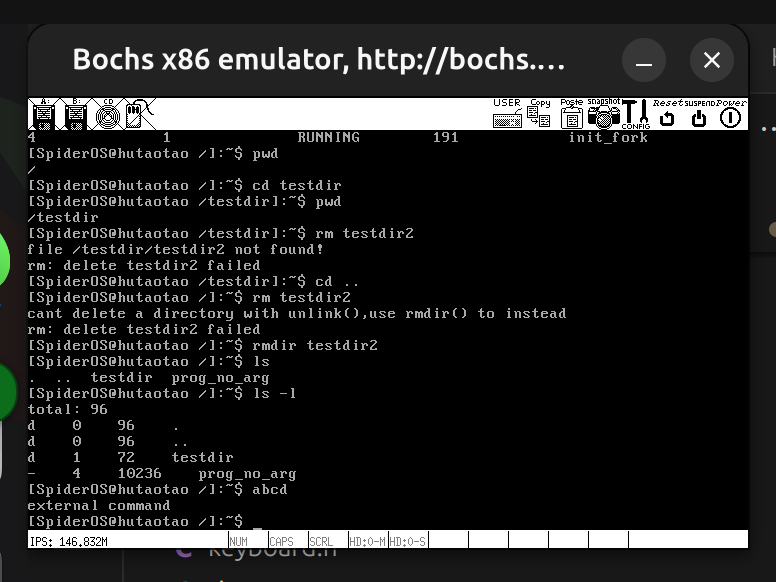
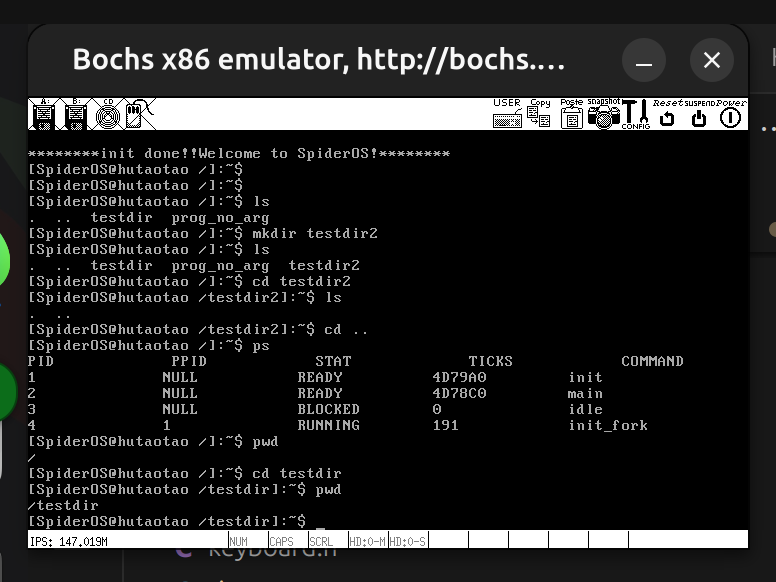


图4-5 图4-6

第5章 总结与展望

5.1 研究总结

本项目基于X86架构自制了一个小型的操作系统SpiderOS,围绕着操作系统的核心机制和功能展开开发研究。通过实践来深入理解操作系统各个方面的整体运行机制。

在系统启动部分，本系统实现了从机器上电到加载MBR再到保护模式切换，加载内核等完整流程，来确保系统可以顺利的进入多任务的操作环境。除此之外通过手动设计中段模块，内存管理模块，线程和进程模块，文件系统模块来让我深入学习操作系统底层的相关知识，也为我后续从事的系统研发工作奠定了技术基础。至此虽然操作系统的设计告一段落，但是相关源码已上传至Github开源并在后续会持续完善。

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，系统仍存在许多的不足之处，本系统只是从最底层自下而上的搭建了完整的框架，能够完整的运行整套流程，但是实际SpiderOS实现非常基础，仅具有操作系统应该具备的核心功能，并没有使用和生产力价值，仅可用于学习研究操作系统框架和原理型研究的用途，不具备图形界面，网络通信等功能。并且系统的稳定性和健壮性还有待提升。目前偶尔会有系统异常崩溃的现象。并且从应用层面看，系统并未实现用户态的应用，仅可通过在main.c文件中修改测试用例来模拟用户的交互。因此功能还非常的匮乏。

因此针对这些系统的不足，未来若有进一步的开发计划，SpiderOS会进行进一步的改进和拓展，包括引入图形化用户界面与驱动支持，完善文件系统功能，完善网络功能以及拓展Shell命令等。

参考文献

1. 彭星海.基于x86架构的微内核操作系统的研究与实现.2020.电子科技大学,MA thesis.doi:10.27005/d.cnki.gdzku.2020.000410.