黑龙江大学

本科学生毕业论文

论文题目：小型操作系统内核的设计与实现

学　　院：计算机科学技术学院

年　　级：2013 级

专　　业：计算机科学与技术

姓　　名：郭耀文

学　　号：20133104

指导教师：青巴图

2017 年 5 月 10 日

摘要

计算机科学已经成为现代社会非常普遍的科学，互联网深入到人们生活的方方面面。计算机教育也成为了每所高校几乎都要开设的课程，作为计算机四大课之一的计算机操作系统是计算机相关专业学生的必修课程。然而，操作系统课程面临一个尴尬的现象：操作系统本身的特点就是提供对资源的管理，将计算机庞杂的内部细节进行抽象，对用户提供简单的操作接口。这使得使用者对操作系统内部的了解微乎其微。由于代码量的巨大，提高了学习者的进入门槛。本文并避开涉及现有的操作系统的讲解。而是选择操作系统会核心的功能进行实现，简化繁琐的操作，深化操作系统最重要的机制。同时配以代码实现，希望通过最核心的代码让每一个希望了解操作系统的人都能够深入的了解操作系统的运行机制。同时，本文为希望进入操作系统编写的人指出了一套完整的解决方案，包括编译、调试、运行。

本文分五章内容讲解操作系统从无到有，从平台搭建到调试运行，以及开发和调试的全部内容。主要从开发平台搭建、启动分区、加载内核、进入保护模式、段页式内存管理、中断处理、进程调度、输入输出系统、进程间通信、微内核设计内容阐述了一个操作系统的开发过程和其核心的功能。

同时，在开发该系统时，本文给出的不仅是开发的核心技术讲解，也指出了一套完整的项目开发方案，包括自动化编译、运行，以及项目的备份和版本管理。

关键词

操纵系统；微内核；设计与实现；开发环境；开发过程

Abstract

Computer science has become a very popular science of modern society, the Internet goes deep into all aspects of people's lives. Computer education has become almost every university has to set up courses, as one of the four major computer computer operating system is a computer-related professional students compulsory courses. However, the operating system curriculum is facing an embarrassing phenomenon: the operating system itself is characterized by the provision of resources management, the computer will be abstract details of the internal details of the user to provide a simple operating interface. This makes the user's understanding of the internal operating system is minimal. Due to the huge amount of code, the learner's entry threshold is improved. This article and avoid the explanation of the existing operating system. But choose the operating system will be the core function to achieve, simplify the cumbersome operation, deepen the operating system the most important mechanism. At the same time with the code to achieve, hope that through the core of the code so that everyone who wants to understand the operating system can be in-depth understanding of the operating system operating mechanism. At the same time, this article for the people who want to enter the operating system to write a complete set of solutions, including compile, debug, run.

This article is divided into five chapters to explain the operating system from scratch, from the platform to build debugging, as well as the development and debugging of all the content. Mainly from the development platform to build, start the partition, load the kernel, into the protection mode, segment page memory management, interrupt handling, process scheduling, input and output systems, interprocess communication, micro-kernel design describes an operating system development process and its Core function.

At the same time, in the development of the system, this article is not only the development of the core technology to explain, but also pointed out a complete set of project development programs, including automated compilation, operation, and project backup and version management.

Key words

Operating system; Micro kernel; Design and implementation; Development environment; Development process

目录

摘要 I

Abstract II

前言 1

第一章 绪论 2

1.1 研究背景 2

1.2 研究意义 3

1. 3 研究现状 5

1.3.1 国外研究现状 5

1.3.2 国内研究现状 5

第二章 操作系统的技术与知识基础 6

2.2 开发环境搭建 7

2.2.1 开发平台 8

2.2.2 目标处理器 8

2.2.3 汇编器 9

2.2.4 编译器 9

2.2.5 编辑器 9

2.2.6 自动化工具 9

2.2.7 版本控制工具 10

第三章 功能分析及可行性分析 10

3.1 需求分析 10

3.1.1 功能分析 10

3.1.2 非功能分析 11

3.2 可行性分析 11

3.2.1 技术 11

3.2.3 经济 12

第四章 设计 12

4.1 结构设计 12

4.1.2 运行流程设计 13
4.2 启动加载程序 16
4.2.1 开机启动程序 16
4.2.2 内核加载程序 17
4.3 中断程序 17
4.4 任务调度 18
4.5 输入输出系统 18
4.5.1 基本的输入-键盘 19
4.5.2 基本的输出-屏幕 20
4.6 进程间通信(IPC) 21
第五章 实现 22
5.1 启动加载程序 22
5.1.1 开机启动程序 22
5.2.2 内核加载程序 24
5.4 进程 28
5.5 输入输出系统 31
5.5.1 基本的输入-键盘 32
5.5.1 基本的输出-屏幕 33
5.6 进程间通信(IPC) 36
结论 36
参考文献 37
附录 38
致谢 38

前言

操作系统几十年的发展使系统内核本身的代码量级达到了千万行，几亿行，甚至更多。这使得想要完全阅读现代操作系统来完全了解操作系统成为一件极其困难的事。如果阅读操作系统内核的早起版本，由于开发时间的久远，不看看实际的运行效果，只靠代码推断对理解产生阻碍。运行又缺乏运行环境配置指导，对于现在很少接触底层的大学生来说有很大困难。操作系统理论课程要在短短一个学期内完成操作系统重点理论知识的讲授，其对实现方法的抽象使学习者对操作系统的内部运行更加迷惑。再加上操作系统课程和计算机组成原理课程的完全隔离化的课程组合，即便是学过操作系统课程的学生对操作系统的认识也完全只是知道理论的模型。一些操作系统的课程设计要求按照理论的模型使用高级语言完成模型的模拟，希望让学生加深对理论的记忆。然而，这种记忆越是深刻，对学生的误导越大。学生的模拟程序相对于真实的实现方法相差越甚远，甚至有些学生为了模拟操作系统的内部机制，使用图形化界面演示抽象模型的图形表示，最奇怪的是这样竟然能得到很高的分数。我承认使用图形化的方式的代码量更大，对编写者有更高的要求，也能够提高他们的编程能力。但这样却是跟操作系统真实的运作方式几乎没有什么关联，学生根本没有认识到操作系统的运作的本质。如果有人说，你来编写一个可以运行应用程序的操作系统，恐怕所有人都要束手无策。然而这却是操作系统最基本的功能，也是操作系统任务调度的目的。

经常会看到一些资料，记述了计算机发展史上牛人的名人轶事。一不满意就写个操作系统、编程不方便就写个语言、毕业设计工具满足不了需求就写了 Photoshop。恐怕现在没有多少人能够完成这样的工作了，不是因为现在的人知识学的少，也不是因为现在的人没有当初那些人刻苦了。而是因为前人为了更方便的工作，将所有的细节都隐藏在自己巨大的工作成果中。我们站在巨人的肩膀上，用更少的时间就能完成更多的工作，做成更漂亮的界面，做出更人性化的交互程序。对细节的隐藏对与要完成更高的生产效率非常有帮助，却对我们学习计算机只是立起里一道无形的高墙，将我们隔在真理之外。

自己动手写一个操作系统似乎是一个可怕的事情，因为操作系统看起来是这么庞大，完成诸多的事情。最要命的是，我们对操作系统的距离如此之远，只是知道一个模型而已。就像只知道一个推力与反推力的知识，要造一个火箭一样。然而事实却非如此，

我们没有必要，也不可能完成一个现有的操作系统一样完善的操作系统。仅仅实现一个操作系统的雏形还是可以的，想想现在常见的庞大程序，难道是当初刚写出来就如此庞大吗？不也是从头开始一点点积累起来的吗。写一个简陋的操作系统有用吗？这个问题我也在思考，为什么不去阅读别人的程序去了解操作系统的运行。我的结论是，阅读别人的分析能够了解大概，却容易忽视细节。一些高大的理论往往容易接受，反而会是一些边边角角的小问题不自己动手恐怕永远也不会知道。与其花时间浪费在阅读别人的代码的删减上，不如自己动手把时间花在代码设计和结构的设计上。了解了操作系统的设计，再去看看操作系统的理论，和其他系统的实现，将会如鱼得水。

本论文主要研究一学期编写小型操作系统的可行性，为希望了解操作系统运行原理的人提供一个进入操作系统的方向。本系统主要采用C语言编写，C语言无法完成的地方则使用汇编语言编写。主要完成功能：电源键开机即可启动、内存管理，任务调度，输入输出系统，文件系统，简单应用程序编写。

第一章 绪论

1.1 研究背景

操作系统是计算机体系中最重要的组成部分之一，担任着管理和调度计算机硬件资源和软件运行，与CPU并称为计算机的两大核心技术。作为信息革命的核心技术，现代生产力的的发展离不开计算机。计算机已经深刻融入到科学计算、国防、工业、医疗、办公等领域，渗透到人们接触到的每一个角落。操作系统更是扮演者计算机系统资源的管理者与人机交互的“中间人”。没有人再愿意在没有操作系统的情况下接触一台计算机，操作系统可以说是计算机的灵魂。

计算机的发展经历了漫长的时间，在电子管发明之前，人们就开始使用机械手段达到数学计算的目的，各种各样设计精巧的机械计算机层出不穷。自1946年人类发明了第一台使用电子管的计算机，计算机才有了本质改变，计算机进入了电子时代，这个时候的计算机还没有操作系统可言，人们是通过拨动开关，或是通过打了孔的纸带直接将

想要运行的操作直接对硬件进行操作的。到 1964 年以后，集成电路技术被应用于计算机，使计算机进入到飞速发展的阶段，飞速发展的计算机硬件能够在完成更多的功能，然而计算机的造价依然很高，人们希望将计算机资源的利用率得到更大的发挥，于是就出现了用于管理用户程序运行的调度程序，这也是计算机操作系统的的雏形。

操作系统操作系统出现之后，经历了从单用户单任务到多用户多任务的分时操作系统的发展。为了将操作系统最核心的功能进行高内聚，衍生出操作系统内核的概念，这也演化出如今的宏内核和微内核两种内核架构设计。微内核将更少的核心功能进程、进程间通信、内存管理等保留在内核中，结构更加紧凑，模块化程度高，较高的代码量意味着微内核更安全。然而由于所有的功能都要通过内核调用，是的微内核运行不如宏内核高效；宏内核则保留了大部分使用频率较高的功能，采用分层的设计方式，这使得完成相同任务的调用层出较少而运行效率较高。关于微内核和宏内核的讨论从没有停止过，论战的双方也都意识到自身的问题，开始向对方的长处借鉴，这也是混合式内核出现的原因。近几年，有人对微内核进行了改进，提出了第二代微内核的概念，其代表就是 L4，L4 将 IPC 消息传递机制的内容进行了简化，从而降低 RPC 的调用开销，从而提高运行效率。

现代操作系统为了满足日益庞大的计算机功能，已经变得非常复杂，这让操作系统的学习者想要完整的学习操作系统的每一部分变得不再现实，人们可以学习一下操作系统的整体架构和核心组件，然后专注于某一功能。

1.2 研究意义

操作系统作为用户与计算机的交互接口，对计算机内部的复杂功能进行了封装，提供给人们简单统一的操作方式。对于普通用户来说，能够熟练使用操作系统就已经足够驾驭计算机的使用。而对于一个计算机研究者或者从事软件研发工作的人来说，不能够深入了解甚至掌握操作系统的运行机制，就像观看魔术表演的观众一样，永远被蒙在鼓里。根本不可能对自己的软件进行深入的改进好优化，更不用说对能够设计一些计算机基础层的软件了。

与生活在操作系统发展早起的人们相比，我们要幸运很多，现在有许多操作系统是开源的，可以供学习者免费的学习研究。同时，也有许多现有的资料可供参考，许多前哲们花费了大量时间积累的资料与书籍，给我们进入操作系统的学习提供了便利，也为

学习者节约了时间。这使得从新写一个操作系统内核似乎没有必要，因为想要了解操作系的统的本质，可以学习现有的操作系统。学习现有的操作系统是必要且必须的，没有人不是站在巨人的肩膀之上完成自己的目标。前人积累的宝贵经验也是本设计参考和借鉴的宝藏。然而，自己动手实现一个模拟的内核确时任何拜读经典都无法代替的宝贵经历。就像进入计算机编程的入门第一课中的“Hello World!”一样，它已经被写了不计其数，然而每个学习编者的人都通过自己的实现才能深刻将其烙印在自己的意识中。操作系统内核的编写也是，通过写一个模拟的微小内核，能够发现那些阅读代码永远无法意识到的问题。并且，通过写自己的模拟内核，能够将作者自己的所思所想加入的内核的设计中，也能够激发作者的疑问，现有的操作系统为什么这样写？能不能不这样写？能不能有更好的实现？

通过自己来设计实现一个微小的内核，不仅更够提高作者的程序设计能力，而且能够促进写作者掌握操作系统中许多虽然是细枝末节但很有用的知识，对于计算机体系不是很熟悉的学习者也对整个计算机的运行机制和软件的运行原理的认识提高的一个新的层次，甚至有些知识是颠覆原先的认识的。通过编写程序，对于写作者，特别是大学生，能够学到许多程序设计的技巧。同时能够锻炼大型程序的项目管理和组织经验。

通过编写微小的内核，学习者能够掌握操作系统最核心的知识，而且自己的写作让知识的认识和记忆更加深刻。虽然所编写的内核不一定有实用价值，也没有现有的操作系统完善。可能跟现有的操作系统有很大差别。可是，通过自己的编写，能够对操作系统的本质更加了解，知微见著，在阅读的学习大型的操作系统的时候，就能很快将大型操作系统的模块跟自己实现的某模块对应起来，快速的掌握大型操作系统的结构，使学习现有操作系统变得轻松。

同时，编写操作系统是一个完整的过程，包括开发环境、运行条件、调试过程、工程管理，这些都是通过阅读代码这写简单的事情所不能取代的。由于现有的操作系统体系庞大，一开始就进入其中学习，进入门槛高。而巨大的代码量，往往是学习者不能在短时间内在整体上掌握系统的架构，或是只知道架构，花费的时间太久，已经对深入研究失去了兴趣。本文，不仅给想要自己实现操作系统的学习者支出一条编写操作系统的道路，同时也为想快速学操作系统原理的人提供一个最小内核的实现，以便能够快速的学习操作系统的架构及原理、设计与实现。

1.3 研究现状

有许多大学和研究机构专注于研究操作系统的设计和实现方式，他们专注于操作系统最前沿的探索的改进，操作系统是一个庞大的体系，对它的探索和修复都是无止境的。然而，这些研究机构都是专注于最前沿的设计的一些实验，只适合与对操作系统非常熟悉，希望进一步改进现有操作系统的缺陷，或是引入新特性的研究者。对于操作系统从无到有，从零起点开始构建一个操作系统的资料并不多。这很容易理解，操作系统的启始搭建是一个许多人都研究过的不再有任何新意的点，而且这牵涉到许多繁琐的工作。本设计关注于给操作系统学习者提供一个最简的用于学习的内核，同时给入门开发者提供进入操作系统开发的指导。

1.3.1 国外研究现状

国外有几个专注于帮助初学者进入操作系统开发的网站。如 JamesM's kernel development tutorials 给出了一个开发操作系统的指导，而 OSDev.org 则提供了操作系统开发过程中各种资料的整理，以及在操作系统开发构成中可能遇到问题的解决方法。这些往网站的资料都很详实，然而只有文字的描述，而缺乏图标来更好的加以讲解。同时，这些网站都更短的关注于说明该怎样操作，而没有说明为什么这样做。日本一个叫川合秀实的开发者写了一本《30 天自制操作系统》的书，也已经被翻译成中文。该书讲解非常基础，语言也非常生动。是作为入门操作系统非常好的一本书。可能是作者为了降低入门者的困难，书中对操作系统编译的链接的内容提供了许多作者自制的工具，这在提供便利的同时，也让其他一些无法使用其工具的学习者难以继续开发。作者使用自制的工具来隐藏细节，而不是使用现有最通用的工具显然不是一个明智的做法。

1.3.2 国内研究现状

国内开发者于渊老师的《Orange’s 一个操作系统的设计与实现》是一本很好的关于操作系统设计与实现的指导书，也有很多的学习者。本设计正是通过学习该书进行实现的。然而，该书的作者似乎没有对书中的代码进行后期的维护，这使得书中的代码在 64 位机器上无法编译。然而书中的讲解非常到位，是想要学习操作系统编写者不二之选。希望该书的读者在遇到问题时，可以从本设计中找到解决的办法。

第二章 操作系统的技术与知识基础

2.1 技术基础与知识准备

本设计所涉及的技术涉及如下几方面，虽然没有这些技术积累，也可以在设计中用到什么知识学习相关内容，然而，更好的知识积累能让设计过程变得轻松愉快。不得不说，计算机操作系统所设计的知识门类确实很多，这也是它实现困难的原因。

2.1.1 汇编语言

汇编语言是硬件编程最直接的符号语言，它的操作指令和机器指令一一对应，所以，汇编语言也是高平台相关的。很多高级编程语言都是先编译成汇编语言，然后再编译为机器语言，即 01 数字串。

汇编语言与开发平台有很高的相关性，因此，不同架构处理器平台上汇编语言的格式也有很大差别。目标的主要汇编语言有 x86 平台的 IMB 格式汇编和 ARM 平台的格式语言。也有为跨平台而设计的 AT&T 格式汇编，由于 AT&T 格式汇编着眼与跨平台的设计，是的它的格式闲的有些怪异。由于汇编语言是面向硬件的，他们的语法结构和指令都很相似，所以，只要有某一个格式的汇编语言编程经历，在使用其他格式的汇编语言都会很轻松。

2.1.2C 语言

完全使用汇编语言来编写程序会使编程速度变得很慢，使用高级语言编程成为明智的选择。本设计使用 C 语言来完成工作，只有在 C 语言无法完成的地方才使用汇编语言。当然，C 语言不是唯一的选择，然而确实做好的选择。C 语言作为通用编程语言，从诞生植入器就着眼于移植性和简洁的语法。20 世纪 60 年代 Ken Thompson 使用汇编语言和 B 语言编写出了 UNIX 系统。然而，UNIX 系统在移植方面遇到了很大困难。后来 Dennis Ritchie 在 B 语言的基础上开发了 C 语言，并用 C 语言完全重写了 UNIX。现在我们使用的 C 语言是美国国家标准局进行了语法规范和改进后的语言，称为 ANSI C。

选择 C 语言并不是因为其它语言无法编写操作系统，而是因为 C 语言除了函数库之外，几乎不依赖于操作系统。C 语言的函数库其实是标准化组织规范好声明，由各个操作系统实现的，是操作系统方便用户编写应用程序而开发的。这也是同样的 C 程序要运

行于不同平台时，是要从新编译的原因。所以在编写操作系统时，所有的函数都需要自己来实现。其他高级语言的许多特性，其实都是系统相关的，使用他们会使程序的实现陷入不必要的麻烦。

另外 C 语言和汇编语言的衔接非常简单、统一，这使得汇编语言和 C 语言的相互调用变得非常简单，只要遵循 C 语言的参数传递机制，你根本感受不到函数背后的实现语言是哪种。这也是主流的操作系统都是使用 C 语言的原因。

2.1.3 计算机组成原理

现在计算机基本上都是冯.诺依曼体系的。本设计的目标运行环境 x86 计算机也是冯.诺依曼体系的代表。具有计算机组成原理的知识将是操作系统的编写变得轻松。进程的调度和保护模式的相关内容都是和平台紧密相关的。虽然对于计算机的组成原理不需要很渗入的掌握，能够简单了解计算机组成的各部分和大致的运行方式显然是必要的。

2.1.4 微型计算机原理和接口技术

操作系统的编写中设计到许多对计算机硬件操作和对芯片进行操作的工作，如中断管理器、时钟发生器等，如果能够对计算机接口有一定的认识，或者用过接口的编程经历，在相应的接口操作时便能够很好的理解和掌握相关内容。当然，本设计不是要开发一个可以广泛应用的操作系统，所以并不会涉及太多的接口内容。在需要相关知识时，再去学习其内容也是可以的。

2.1.5 操作系统原理

如果拥有操作系统的知识，再读本设计能够将知识和理论很好的结合，但没有学习过计算机操作内容的学习者也不不必担心。本设计目的是是实现一个简单的模拟操作系统内核，对于操作系统的高深理论用的并不多，甚至很好提及操作系统理论的相关内容。并且，本设计的目标就是为学习操作系统的学生提供一个实现的模型。希望学习者能够在具体对操作系统有了一个感性的认识之后，再去深入学习操作系统的理论。从而能够将理论和实际相结合，对操作系统的认识更深刻和透彻。

2.2 开发环境搭建

要想开发操作系统内核，一个现有的开发平台是必不可少的，便利的工具能让开发变得轻松愉悦，一致的开发环境能让开发变得如沐春风。没有人愿意回到通过给卡片打孔编程的年代，那也不可能在短时间内编写一个微小的操作系统内核，况且本设计还有许多保护机制和文件系统的内容需要完成，无疑即便是使用 C 语言，其工作量也是巨大的。

2.2.1 开发平台

用于开发目标操作系统的操作系统，称之为开发平台。在开发平台方面，本设计使用 Linux 系统，这里所说的 Linux 是 GUN/Linux 的发行版，因为单指 Linux 是指 Linux 内核。Linux 操作系统有许多优良的特性，衍生与 Unix 体系的 Linux 从开始就支持多用户分时功能，同时也是开放源代码的，这意味着你可以随时将其拿来学习并添加自己的创意。它同时继承了 Unix 一切接文件的设计哲学。同时，最重要的一点是，大多数 Linux 发行版都有免费版可以使用。

当然，在其他系统也是能够完成该工作，Linux 上的软件基本都是跨平台的，你可以在其他平台上安装本文所提到的软件。甚至，你可以选择其他能够完成相同功能的软件工具，毕竟，其目的不是学会使用某一软件，而是达到设计操作系统。选择 Linux 不仅是因为 Linux 系统开源免费，而是许多开源软件的工具都是 Linux 原生支持的，而且有了远程仓储的支持。这些软件的安装和卸载都非常方便。更重要的是，在该平台上有许多现有的操作系统开发经验可以借鉴，你会发现，即便是在其他平台上，依旧使用的是 Linux 平台上的开发经验，甚至，你需要安装为 Linux 体系设计的软件。不如直接使用 Linux 平台来开发，可以拥有更完整一直的开发体验。

关于 Linux 系统的安装和使用，本文不做过多的介绍，本文中涉及的许多软件都不会介绍其安装和使用方法，以免本末倒置。

2.2.2 目标处理器

由于 x86 处理器在个人电脑的占有率很高，同时大学的教学基本都是以 x86 处理器为基础的。本设计将以 x86 处理器作为目标运行环境。

2.2.3 汇编器

操作系统是和 CPU 架构紧密相关的，本文要介绍的内核将以 x86 架构的 CPU 为开发基础，这是因为该平台的市场占有率极高，基本上我们的个人电脑都是该架构的。使用该架构，也是因为作者手中的电脑就是该架构。另外一点不得不提，就是在大学的教学课程中，所有涉及到平台内容，也都是围绕 x86 架构。这样的知识积累，也让人自然而然的选择 x86 平台作为开发目标。

本设计选择了 NASM 汇编器，他是为 x86 而设计的可移植模块化汇编器，同时使用 IBM 语法，如果你使用过 MASM 汇编，使用 NASM 对于你来说不会有任何困难。同时，它的语法格式更加简单一致。

2.2.4 编译器

本文使用 GCC 作为开发的编译器。它是 GNU 计划的一部分，是有开源软件基金会 GNU 为 GNU 操作系统而设计的编译器。GCC 是 C 语言编译器的事实标准，你很难找到比它更优秀的编译器。由于其出色的表现，大多数 UNIX 系统已将其作为标准编译器。

2.2.5 编辑器

要想高效率的开发，一个顺手的编辑器是非常重要的，虽然一个记事本就可以完成所有的工作，但有语法高亮和自动补全功能的编辑器显然能够提高开发的效率，你可以随意选择自己喜欢的开发工具，甚至你可以使用集成开发环境，只要你能够很好的控制其编译的过程和编译输出文件。

2.2.6 自动化工具

在代码增多后，为了结构的清晰，会将代码分成不同的文件进行存放，处理代码的依赖关系，编译顺序以及修改后的重新编译都变得杂乱。这使得编译和调试都变得繁琐，于是，一个自动化的工具来管理整个过程变得非常重要。本设计使用 Unix 体系中的 makefile 来管理整个过程。它是 Linux 中项目自动化管理的规范，虽然使用它你需要付出一些额外的劳动，然而在后期的使用过程中你会发现，这是值得的。

本设计使用的 makefile 规则解析软件是 GNU make。GNU make 是 make 的一个实现版本，却是 makefile 的事实标准。

2.2.7 版本控制工具

在现代，很难想像有人在没有代码风险控制的情况下开始一个项目。以前的开发经历也告诉我代码备份是多么重要。一旦代码丢失，或者电脑出现问题，甚至是代码改动引起的风险都将是让你追悔莫及。现有的版本控制工具很多，这里使用 git，同时使用 github 作为备份站点。不仅因为 git 是现在版本控制的翘楚，同时 git 和 github 都是免费使用的。

2.6.8 模拟器

即使有 x86 的电脑，一个模拟器也是必要的，不可能每启动一次都从新开关机一次电脑，使用模拟器的好处是，软件的启动和重启非常快速，甚至能够在运行过程中对开发的系统进行调试，这些都是物理机无法做到的。一些模拟器可以模拟出其他平台的环境，如果你没有 x86 计算机，那么一个模拟器显得更加必要。

本文使用 Bochs 模拟器，这个一个开源免费的模拟器，它厉害的地方就是能够根据需要的配置模拟出不同平台的环境，同时 Bochs 具有调试能力，这我系统开发提供了许多方便。甚至，你可使用 Bochs 和 gdb 联合调试，使程序的调试工作变得简单。

第三章 功能分析及可行性分析

本章将对模拟操作系统需要完成的目标，功能，以及实现的可行性进行分析。其实这显得多余，因为在没有操作系统编写经理的前提下是难以对短时间内编写模拟操作系统内核的可行性进行准确把握的。并且，即使是同样的工作量，相对于不同知识掌握水平和不同执行执行能力的人来完成，所完成的时间和可行性也是不同的。本分析将在具有第二章所涉及知识的条件下进行可行性分析。

3.1 需求分析

3.1.1 功能分析

1. 功能

开机引导：本部分完成操作系统最显著的特征，在电脑开机时即由 BIOS 引导程序加载进内存。在 BIOS 将控制权交给引导程序后，引导程序完成操作系统内核加载进内

存的任务，并完成内存和其他资源的初始化工作。然后将控制权交给操作系统内核。

内存管理：主要完成段页式内存管理，内存分配和回收。虚拟内存地址转换计算。

任务调度：完成操作系统任务的创建，等待，执行，阻塞的调度。

输入输出系统：键盘输入的捕获，标准输出屏幕的输出。

文件系统：实现一个最小型文件系统，能够创建文件，保存和写

2. 界面

由于本设计专注于操作系统内核的设计与实现，在用户界面方面只提供命令行的操作和交互方式。同时为了向现有的 UNIX 系统致敬，本设计将提供七个终端接口。

3.1.2 非功能分析

由于本设计的目的是帮助学习者学习操作系统的设计与实现，并且偏向于入门的基础架构搭建。所以对操作系统的性能、可靠性、移植性都不做要求，这些内容都是对操作系统的架构和实现深入掌握之后才能处理的内容。

同时，本操作系统内核模拟程序只是用于学习操作系统的设计实现的入门，不会牵涉到更多的社会问题，所以也不再对其他外部需求进行展开分析。

3.2 可行性分析

3.2.1 技术

操作系统的设计与实现是一项高深且牵涉面极广的浩大工程，即便是现在主流的操作系统也才不断的修复不断发现的新问题和添加新的功能。同时，庞大的代码量也以为这一个人根本无法实现主流操作系统同样量级的完善功能。以 Linux 为例，在 2.6 版本的内核代码量就达到了百万行级别。

然而本设计的目标并不是实现像现在主流的操作系统那样完善的功能。而是一个模拟的操作系统微内核，该内核具有操作系统的基本特征：能够安装到物理机上，实现加电启动，提供内核的保护，具有内存管理的管理功能，实现任务调度和进程间通信。得益于软件自由运动发展，可以有许多现有的操作系统和关于操作系统实现的资料供我们参考。所以，实现一个简单的操作系统内核是可以实现的。

3.2.2 时间

本设计是对操作系统内核的极大简化实现，并不要求达到实用级别，更关注与操作系统设计和对操作系统本质的学习。所以在五个月的时间内是可以实现的。

3.2.3 经济

如今，拥有一台个人电脑是再平常不过的事情了，只需要一台个人电脑就可以完成本设计的所有内容。

第四章 设计

本章主要介绍本设计的整体架构和设计，以及每个模块所用到的技术和知识。

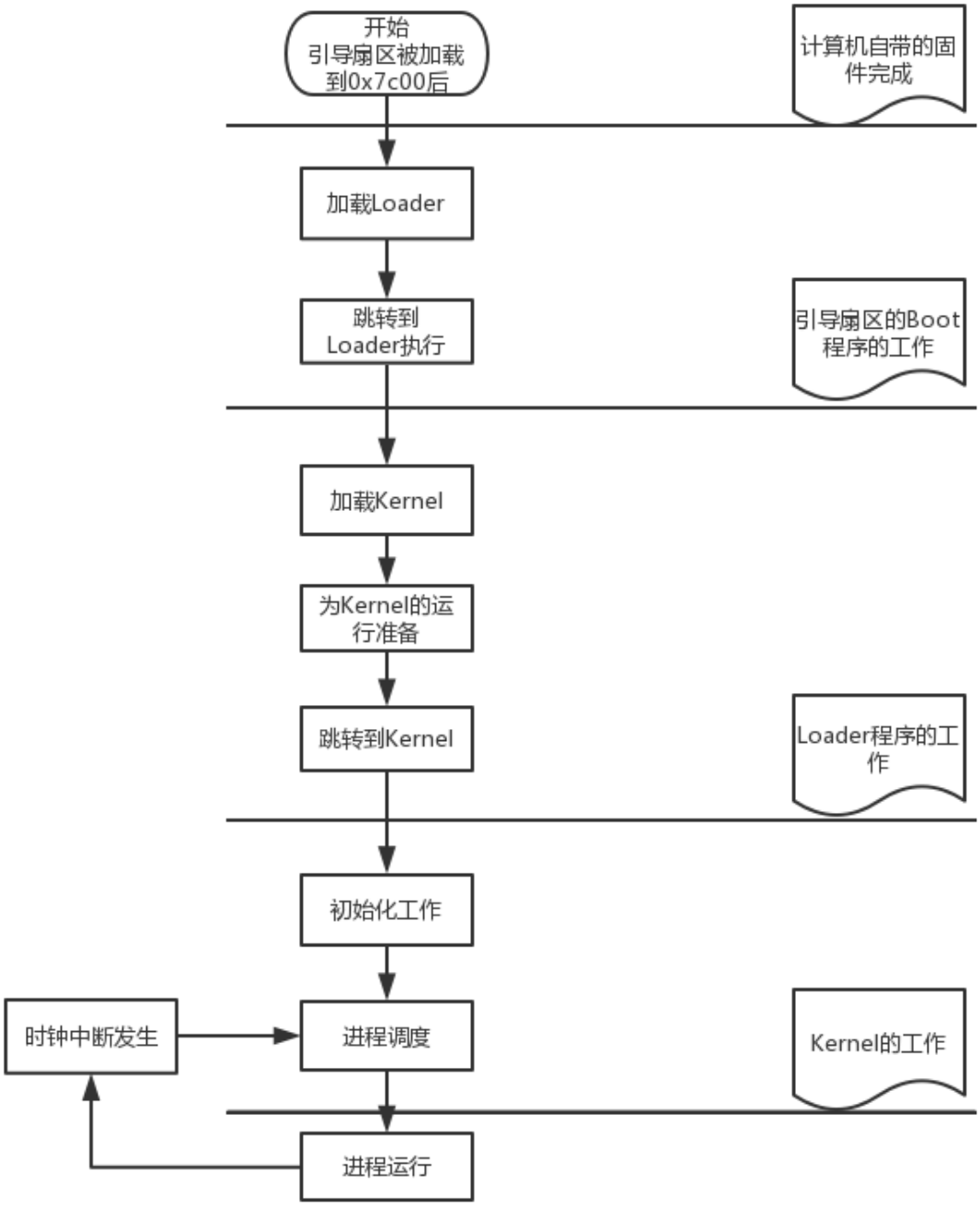
4.1 结构设计

4.1.1 内核架构

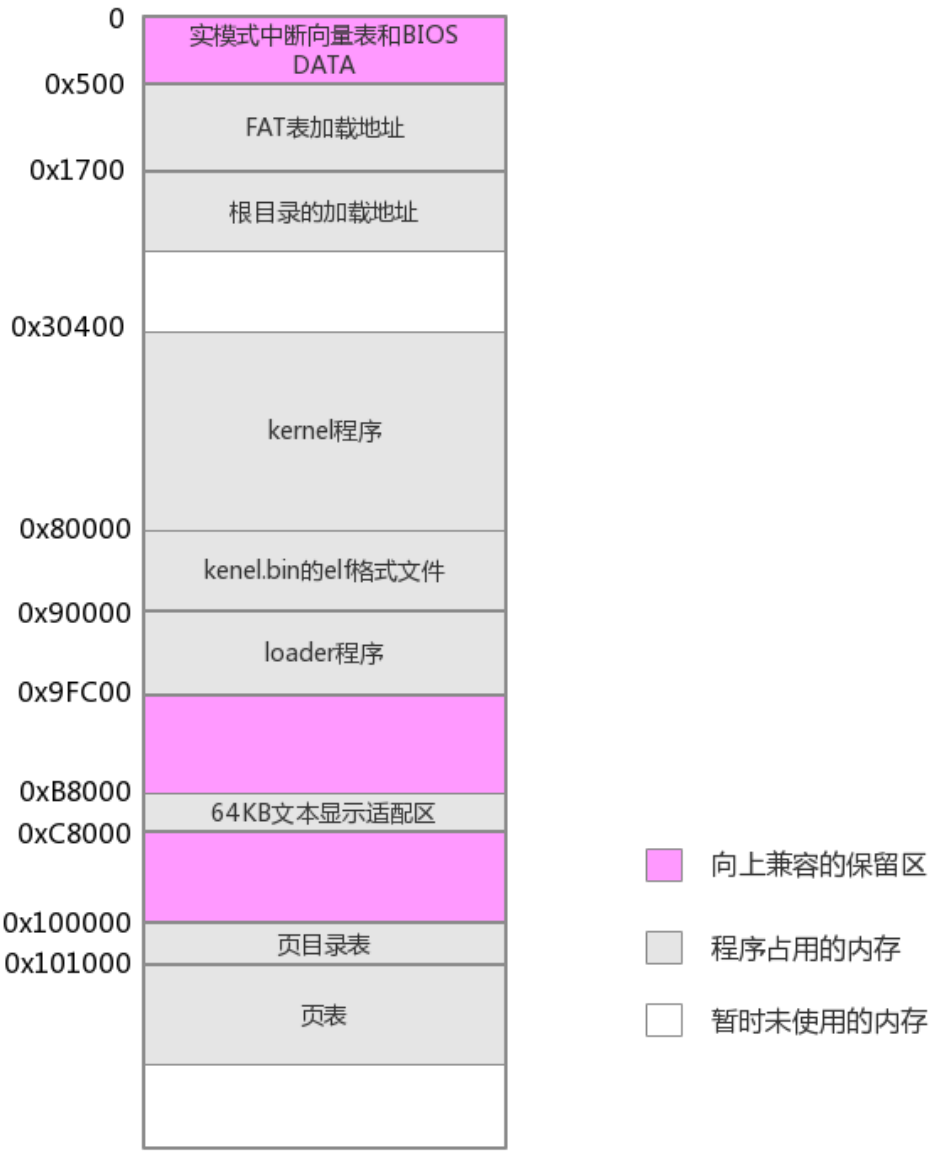
本系统采用微内核结构设计，将内核放在最高特权级，其中包括进程调度，进程间通信(IPC)，中断处理和其它一些时钟发生器等小程序；次之为驱动，包括硬件驱动和内存管理；再次之为服务，如文件系统等程序；最后未用户程序，由于主要关注与操纵系统内核的设计，只需要编写简单的小程序测试即可。

4.1.2 运行流程设计

操作系统内核是操作系统核心功能的高内聚集合，它并不能在计算机启动就能运行，由于历史兼容性问题和设计的高内聚，需要其他程序将内核加载进内存，并为内核的运行准备好运行条件。从启动到内核运行主要经历如下的过程：



普通的程序是由操作系统分配内存的，现在自己设计操作系统，需要了解内存的信息，然后自己规划程序的加载位置和占用的空间。大致的划分如下：



- 0x500～0x16FF 存储 FAT 表
- 0x1700~0x32FF 存储文件目录
- 0x30000~0x8FFFF 存储 kernel.bin，现在还没有写
- 0x90000~0x9FBFF 存储 loader
- 1M 以上作为跳入保护模式后的内存分配

之所以这样分配，是因为内存有些位置是有特殊用途的：

- 0~0x3FF 的 1 M 内存是中断向量表的位置，虽然 BIOS 并没有使用那么多，还是暂时不使用了
- 0x400~0x4FF 的内幕才能存放了许多 BIOS 的参数，有些操作系统提供 BIOS 调用功

能，所以这写参数最好保留。

- 0x9FC00~0xFFFFF 的内存有特殊的用途。不可以随意使用。

即便 0~0x9FBFF 可以被使用，仍然应该吧 BIOS 参数区保护取来以备后用。0x90000 开始的 63K 内存留给了 loader，0x80000 开始的 64K 留给了 kernel，0x30000 开始的 320K 留给调整后的内核（接下来要做的事情）。页目录和页表被放到 1MB 以上的内存。

loader 分配 63K 的空间，差一点不到 64K，一方面因为它本质上是个无格式的纯二进制文件，63KB 应该足够了。

kernel 分配了 64K 的空间，暂时够用了，如果超过了这个限制，可以根据需要再调整。

4.2 启动加载程序

4.2.1 开机启动程序

计算机在加电自检完成之后，就会到 BIOS 中设置好启动顺序的磁盘中寻找要允许的程序。首先会读取磁盘的第一个扇区，查看该扇区是否是启动扇区，如果是启动扇区则将其加载到内存 0x7c00 的地方，然后调到该出执行。为了能够让计算机执行自己写的程序，需要使用磁盘写入工具将软件写入第一个分区。

一个扇区只有 512 字节，甚至要除去两字节的启动分区标识和 64 字节的磁盘分区记录，所以只剩下 446 字节的空间可以写程序。在如此之小的空间内写一个内核架子程序是不可能的的，因为在加载内核之前，需要先读取文件系统，找到内核文件加载进内存，然后将 CPU 的运行模式转换到保护模式，保证在内核运行时，计算机已经处于保护模式下。

所以需要先加载一个过度程序，一般称为内核加载程序。为了加载内核加载程序进内存，需要先在磁盘中找到它。本设计在开发过程中一直使用 Bochs 作为运行环境，Bochs 可以创建一个虚拟软盘。为了简单，本设计使用 FAT12 来格式化软盘。引导扇区的程序需要能够识别 FAT12 格式，找到内核加载程序 loader.bin。然后调到其加载地址执行

4.2.2 内核加载程序

内核加载程序同样需要识别 FAT12 文件格式，先将内核加载到规划好的内存的位置。为了在内核编写时能够使用 C 语言，内核并不是跟 loader.bin 程序一样是顺序执行的程序格式，而是包含了段和节信息的 FLF 文件格式。所以 loader.bin 还需要识别 ELF 文件，根据 ELF 文件中的信息，将程序段放置到内存相应的位置，这样程序执行时，指令中的地址才能和程序实际所在的位置一致。

4.3 中断程序

作为分时操作系统最重要的概念，进程是实现时基础也是尽早应该引入的功能。进程就是一段代码，跟其它程序代码没有什么本质区别。进程之间存在这切换，这种切换不是由程序本身发起的，它不是程序之间的跳转。这种跳转在进程中的程序感知不到，似乎根本不存在,所以就有每个进程就像单独占有系统资源的说法。然而从系统层面上，进程的控制和切换必须依照一定的设置，有序发生。而触发的源就是中断。CPU 只有一个，要么操作系统在实行，要么进程在执行。这就需要一种切换的机制，即使程序正在执行，也能在进程不感知的情况下切换到操作系统执行。实现这种功能的就是中断机制。

中断可异常有许多相似之处，因为他们处理触发条件不一样，使用的处理机制是一样的，甚至处理中使用表格都是同一个。实际上他们都是在执行过程时发生程序转移，跳转到相应的处理程序运行。（我是感觉它们是先后的关系，异常发生后需要终止程序，进行处理，这也是一中中断。而其他的由外部条件触发的处理，叫中断。）中断通常在程序执行时由外部硬件触发二随机触发。比如鼠标和键盘的输入。软件通过 int 指令也能够产生中断，除了触发条件，处理的机制和跳转过程完全相同。通常，异常是在处理器执行指令过程中处理器无法处理的情况或错误，比如除零错误。

为了使中断或异常触发与处理程序对应起来，X86 的工程师们设计了一种机制，给每种中断（异常）一个编号，成为中断向量号。通过这个编号，与一个表格结合，将何种类型的中断（异常）与相应的处理程序对应起来。

为了能够在发生中断或异常时，CPU 调到预期的程序执行，需要将需要执行的程序的位置写入到相应的中断向量中。

4.4 任务调度

进程作为资源分配的单位，在操作系统中占据着非常重要的作用。进程的切换及调度等内容是和保护模式的相关内容紧密相连的。由于其太过于底层，在不同类型的机器上实现起来完全不同。

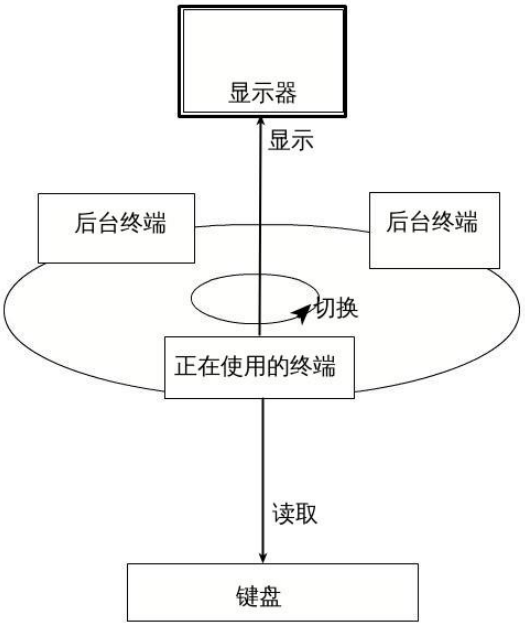
进程的切换是随机的，要想在进程恢复时能够继续原先的中断点处继续执行，就需要和进程切换时保存进程的运行和状态信息。这个用于保存进程信息和运行状态的数据结构就是进程控制块(PCB)。

4.5 输入输出系统

输入输出系统在操作系统中占有绝对重要的地位，没有那个操作系统是封闭不与外界通信的。分时操作系统的出现，能够支持多任务处理，其中最重要的目的就是实现交互功能，人们希望在操作系统工作的同时能够接收用户的操作，并作出相应的相应。现代计算机最重要的输入接口就是鼠标和键盘了，由于这里没有实现图形化界面，所以，只实现键盘的处理；而输出最基本也是最常见的就是屏幕。本设计着重原理的学习与演示，所以只实现屏幕的输出功能。

由于保护模式对系统内核提供了保护机制，也就是对不同功能的任务进行了分级，不同级别的功能具有不同的权限，低特权级的功能不能够直接访问高特权级功能。只提供可供低特权级访问的系统调用，这样就实现了系统的保护。最核心且与系统的运行安全相关的放到了最高的特权级，经常使用有事系统必不可少的功能，被当做服务划分成中间特权级。显然，输入输出系统应该作为服务，即为用户进程提供访问调用的接口，有要隔离在核心功能之外，降低因为程序运行错误对内核的破坏。

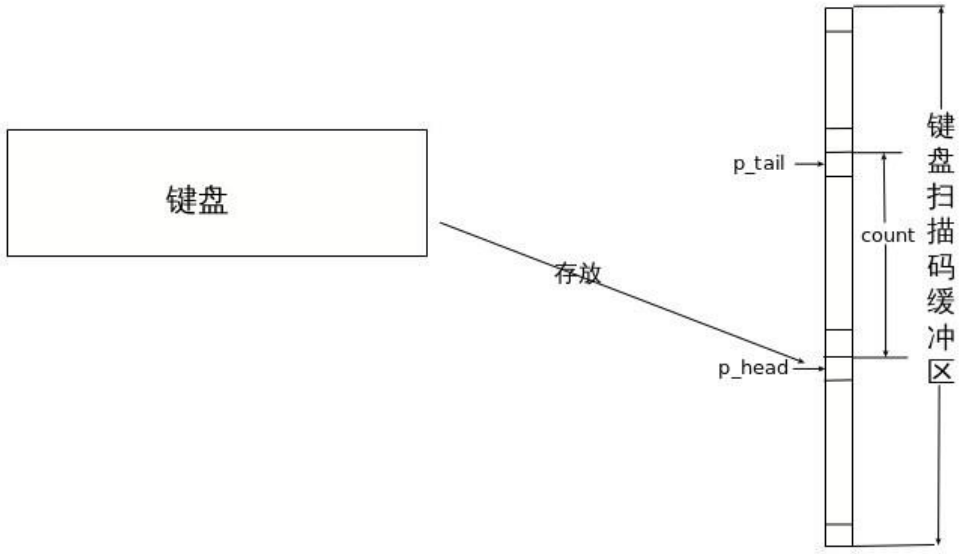
本设计为了向 UNIX 致敬，将实现多个终端，在按下 Alt+Fn 时，能够切换到不同的终端。然而在一台计算机上只有一个键盘和一个显示屏，多个中断就要复用输入输出设备，这也体现了操作系统对资源的管理与分配。



4.5.1 基本的输入-键盘

键盘是操作系统最重要的输入来源。早期计算机没有图形界面，程序的运行和文档的编写，都要使用键盘作为输入。

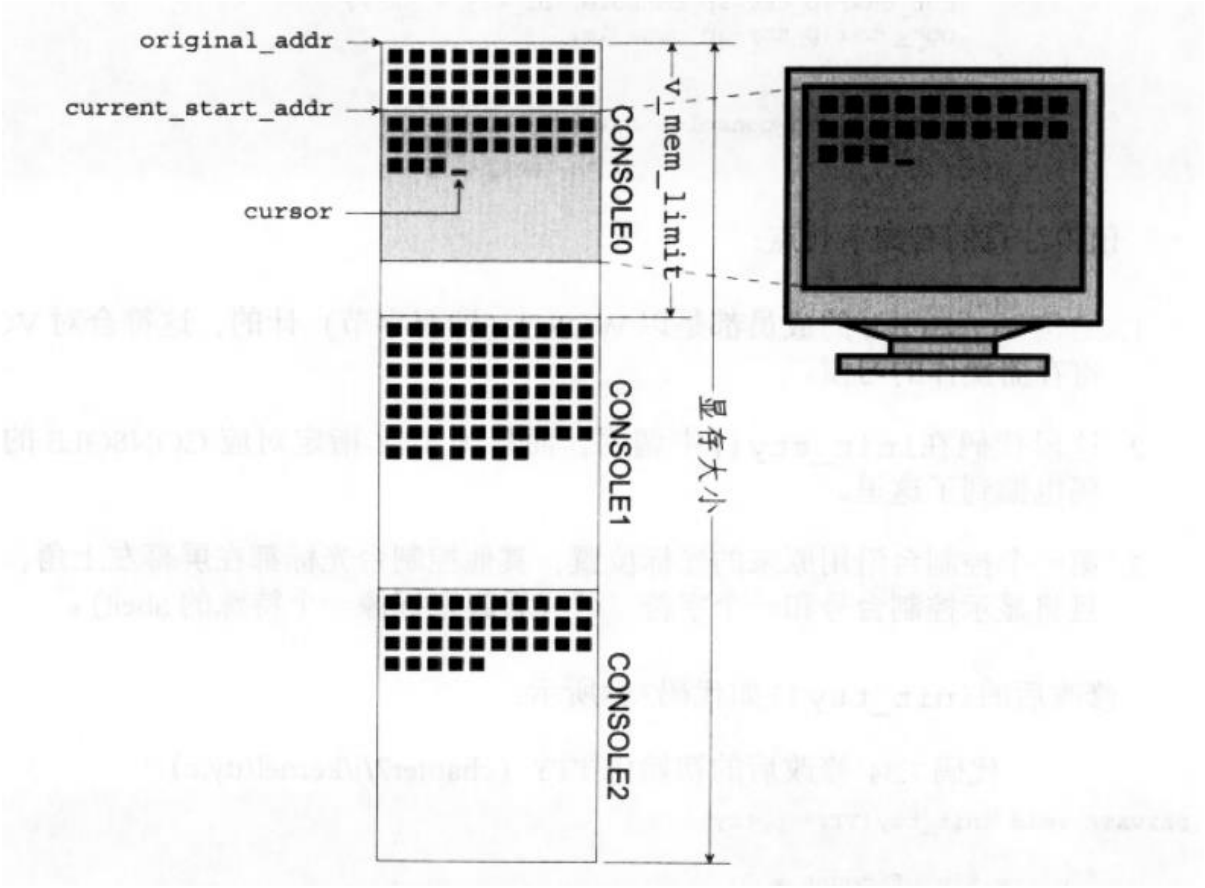
由于键盘是字符设备，输入的速度较慢，并且有时候多个按键的组合才构成一个完成的操作，所以要为键盘建立一个缓冲区，每次键盘中断发生时都将键盘扫描码存入缓冲区，然后结束中断。



4.5.2 基本的输出-屏幕

文本界面的屏幕输出是最基本的功能了，没有输出，人们无法得知计算机的运行状态和系统内的资源。

在 80×25 文本模式下，显存 32KB，其中一屏幕需要使用 80×25×2＝4KB 的显存，如果每个终端占用 4KB，则最多可以模拟出 8 个终端，但是为了能够具备屏幕滚动的效果，需要为每个终端多分配一些内存。本设计将模拟三个终端的运行。

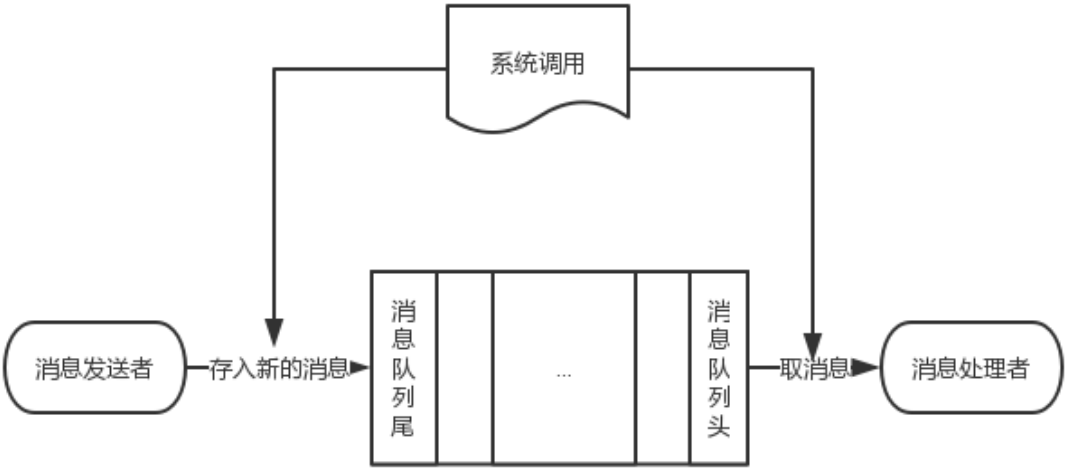


通过将显存分成三部分，分别分给三个终端程序，当某个终端被选中作为当前使用的终端时，将其对应的显存设置为正在显示的内容。

4.6 进程间通信(IPC)

进程间的通信机制是使用操作系统系统调用实现的，不同的进程间通信机制区分出微内核和宏内核。本设计将使用消息队列机制，实现微内核的进程间通信，以体验进程微内核设计的巧妙和高内聚。

微内核的系统调用分为发送消息和接受消息，为了能够相互通信，建立了一个消息队列。发送者将消息放到消息队列中，而接受者从消息队列中获取消息进行处理。发行者和接受者都和消息队列进行调用，而相互之间没有任何关联。这使得微内核结构非常清晰和小巧。



第五章 实现

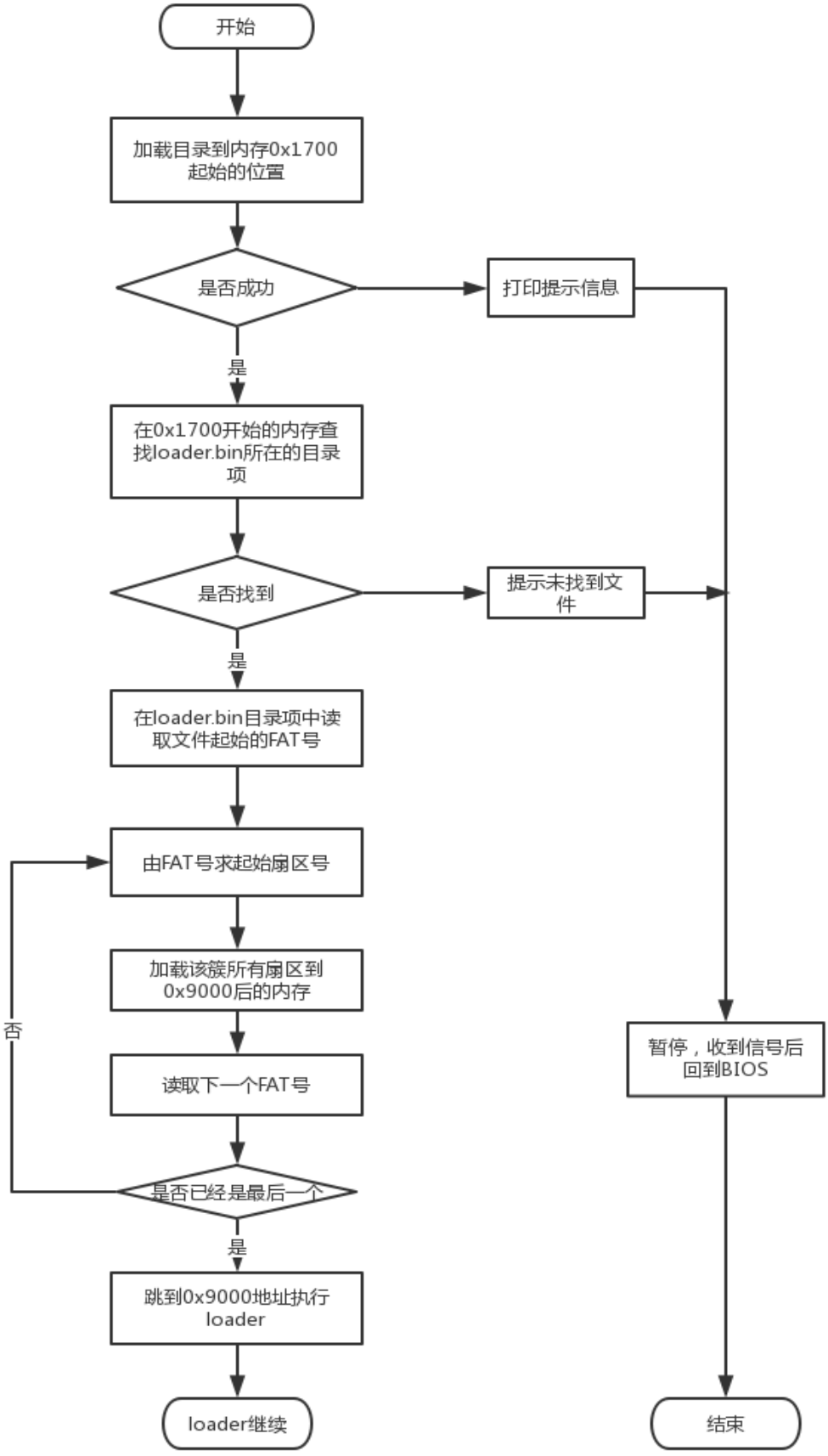
本章将介绍模拟操作系统内核的实现过程，除了内核之外，还包括了启动加载程序和内存管理程序、文件系统。因为只有内核是无法运行的，只有在这些功能的配合下才能很好的运行，也方便与我们看到执行效果和调试。

5.1 启动加载程序

启动加载程序由两部分组成，引导扇区的 boot 程序和负责内核加载和准备的 loader 程序。

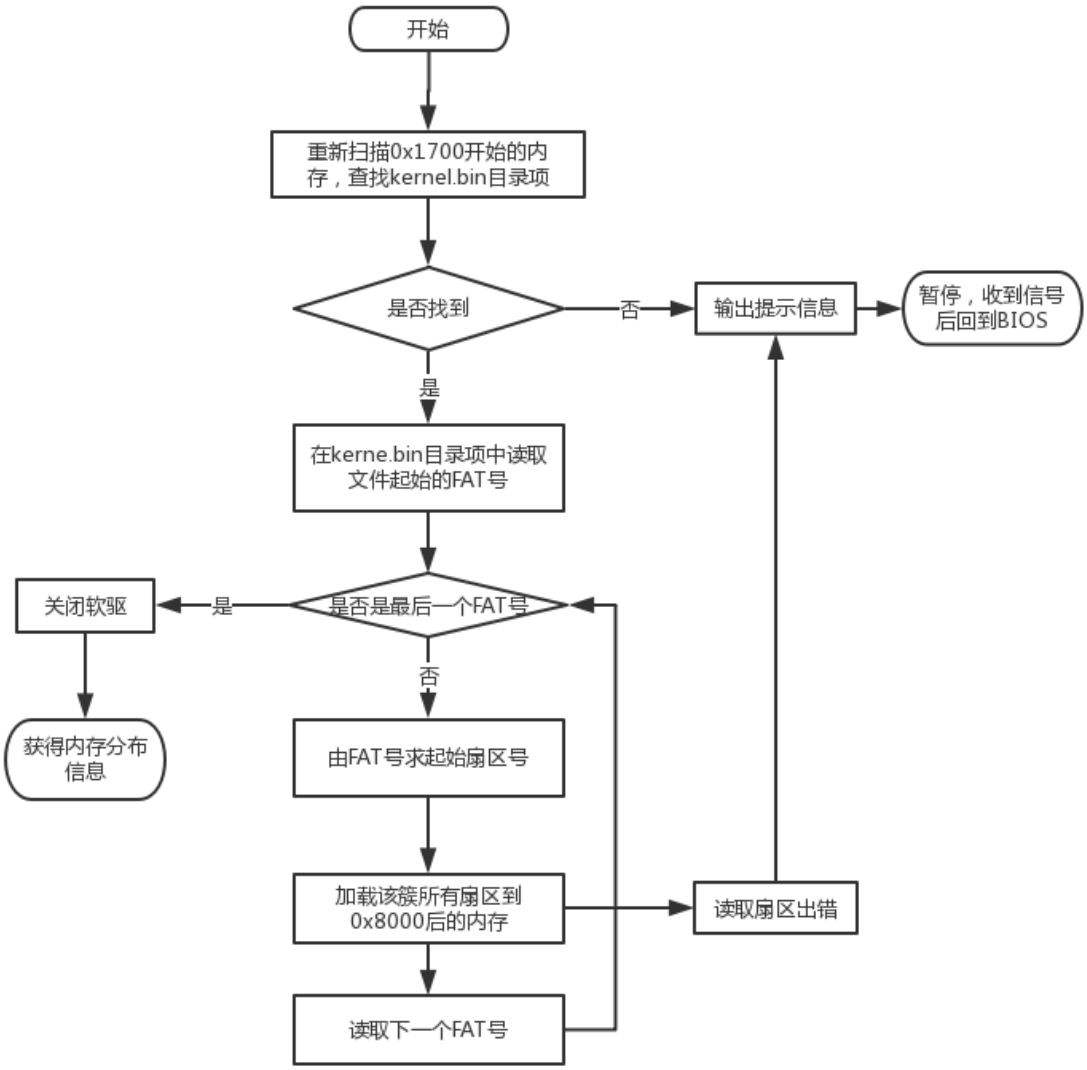
5.1.1 开机启动程序

电脑刚启动时，运行在实模式下。无法使用 C 语言变成，需要使用 16 位汇编指令。由于启动扇区的空间太小，不能完成内核运行环境的所有准备工作，所以先加载一个 loader 程序。为了简单，loader 和内核坐在的软盘使用 FAT12 文件格式。下面就是 boot 程序所做的工作：

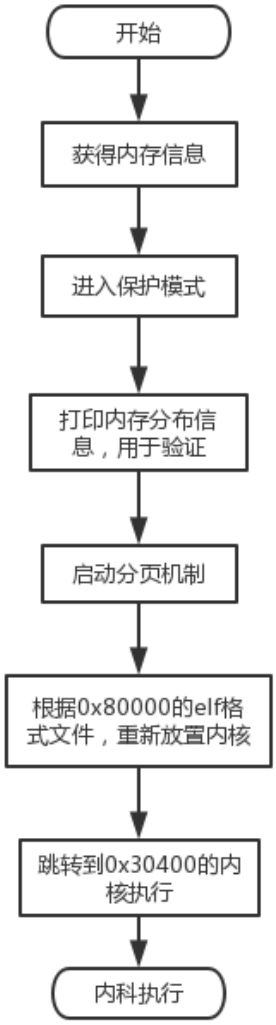

开始
加载目录到内存0x1700起始的位置
是否成功
打印提示信息
是
在0x1700开始的内存查找loader.bin所在的目录项
是否找到
提示未找到文件
是
在loader.bin目录项中读取文件起始的FAT号
由FAT号求起始扇区号
加载该簇所有扇区到0x9000后的内存
读取下一个FAT号
否
是否已经是最后一个
是
跳到0x9000地址执行loader
loader继续
暂停，收到信号后回到BIOS
结束

5.2.2 内核加载程序

内核加载 loader 程序主要负责将内核加载到内存，然后根据 elf 文件格式的信息，重新放置内核程序和数据到相应的位置。为了能够在内核中使用保护模式和 C 程序，需要调到保护模式。由于保护模式使用段页式的内存管理方式。需要获取内存信息。然后开启段页式内存管理。最后将系统控制权交给内核执行。

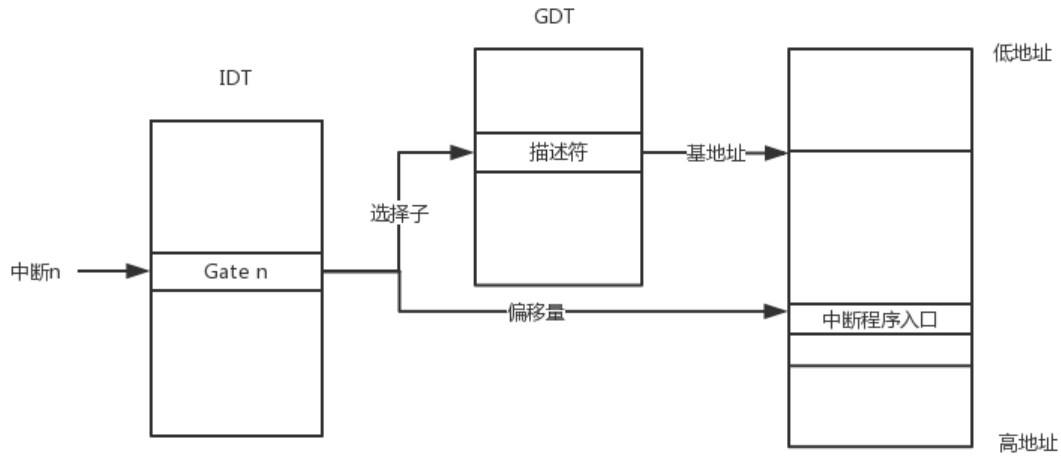


将内核加载进内存之后，就需要为进入保护模式做准备，然后进入保护模式。进入保护模式后最大的好处就是内存地址访问空间的增大，为了方便的使用内存，在输出了内存的信息后，打开了段页式内存管理。最后根据内核的地址信息，从新放置内核。然后跳到内核执行。



5.3 中断程序

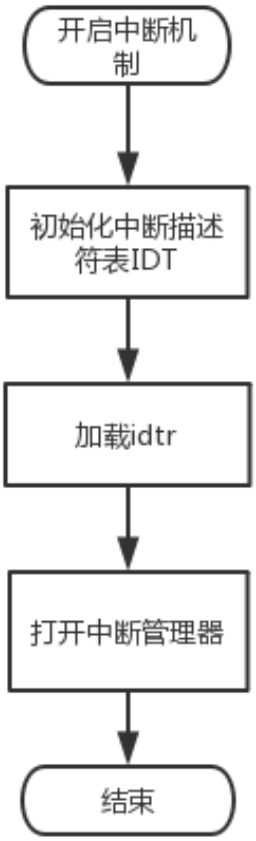
中断处理程序在操作系统中占有很大的作用，它是许多功能实现所依赖的机制。例如进程，作为分时操作系统最重要的概念，进程是实现时基础也是尽早应该引入的功能。进程就是一段代码，跟其它程序代码没有什么本质区别。进程之间存在这切换，这种切换不是由程序本身发起的，它不是程序之间的跳转。这种跳转在进程中的程序感知不到，似乎根本不存在,所以就有每个进程就像单独占有系统资源的说法。然而从系统层面上，进程的控制和切换必须依照一定的设置，有序发生。而触发的源就是中断。CPU 只有一个，要么操作系统在实行，要么进程在执行。这就需要一种切换的机制，即使程序正在执行,也能在进程不感知的情况下切换到操作系统执行。实现这种功能的就是中断机制。

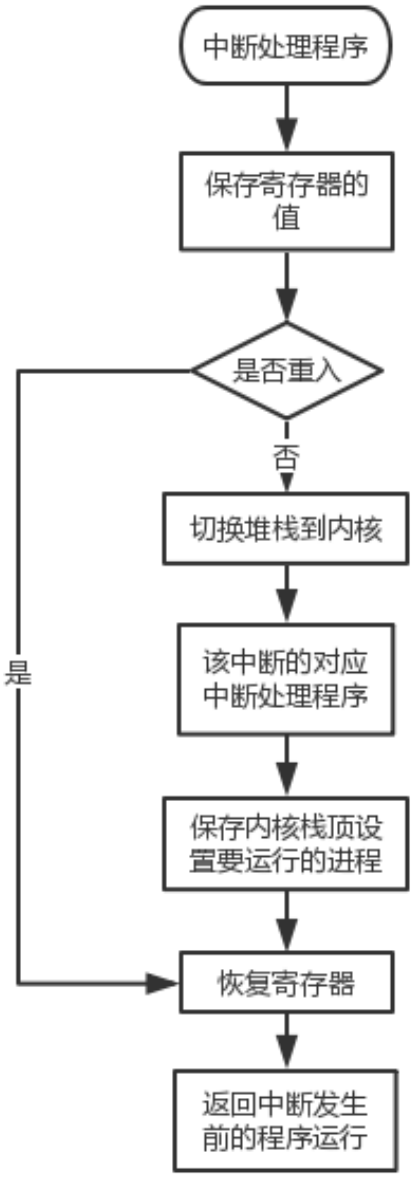


保护模式下的中断处理延续了实模式的方式，同时添加了许多保护机制，保护模式下的中断向量不再是简单的地址值，而是包含选择子、偏移量、属性和特权级的综合中断门描述符。

中断描述符结构体定义：

```
/* 定义门描述符，包括中断门，陷阱门，任务门和调用门 */
typedef struct s_gate{
    u16 offset_low;      // 偏移的低 1 6 位
    u16 selector;        // 选择子
    u8  param_count;     /* 该字段只有在调用门时有效。如果在利用调用门调用子程序
时引起特权级的转移和堆栈的改变
                         * 需要将外层堆栈中的参数复制到内层堆栈，该计数字段就是
用于说明这种情况发生时需要复制的
                         * 双字个数
                         */
    u8  attr;            // attribute 属性
    u16 offset_hight;    // 偏移的高 1 6 位
}GATE;
```





门描述符的初始化函数：

```
/*
初始化门描述符，本意能初始化所有类型的门描述符，然而调用门和其他的还没有接触，如果有不足的地方以后再修改
p_gate: 门描述符表首地址
vector: 要初始化的门描述符在表中的偏移量
funcPointer: 函数指针，即要调用的函数的地址
privilege: 特权级
*/
```

```
PRIVATE void initGateDesc( u8 vector, u8 desc_type, void (* inteHandler)(), u8 privilege){
    GATE * gateDesc                = &inte_desc[vector];
    u32 offset                     = (u32)inteHandler;
    gateDesc->offset_low       = offset & 0xFFFF;
    gateDesc->selector         = SELECTOR_KERNEL_CS;
    gateDesc->param_count = 0;
    gateDesc->attr                 = desc_type | (privilege << 5);
    gateDesc->offset_hight     = (offset >> 16) & 0xFFFF;
}
```

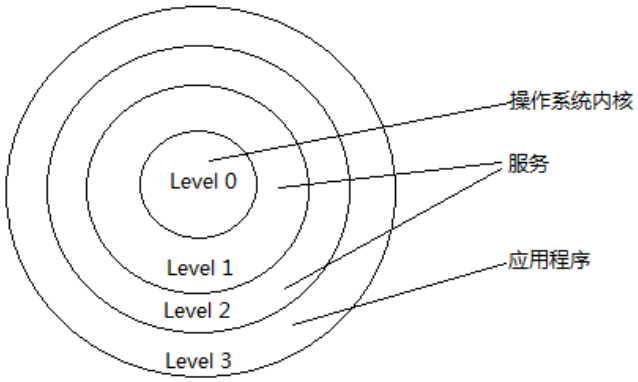
在附录中给出了本设计代码的 github 仓库地址，可以通过如下指令查看对应的代码：

git tag

git checkout interrupt

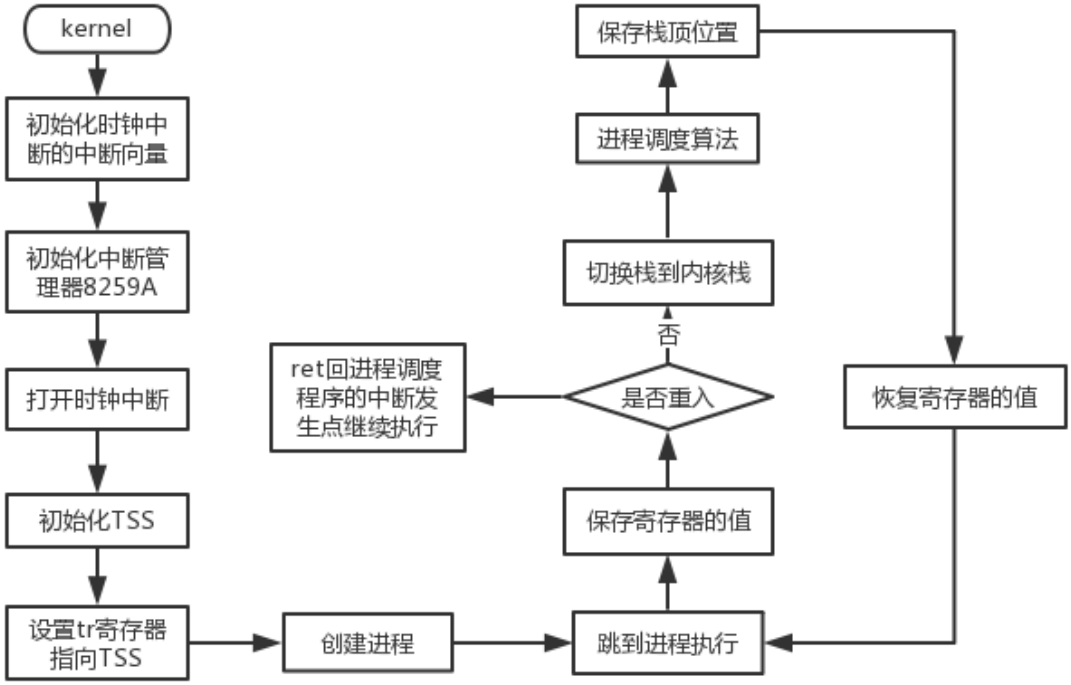
5.4 进程

保护模式下引入权限的同时赋予了不同程序以不同的特权级。进程切换牵涉到特权级的转换。在 IA32 的分段机制中，特权级用两位表示，所以最多有 4 个特权级可以使用，从高到底分别是 0 、 1 、 2 、 3 。由于数字越大特权级越小，所以为了避免混淆，也将高特权级称作内层，而把低特权级称作外层。



特权级

高特权级的代码可以直接调用低特权级的程序，但低特权级想要调用高特权级的程序则需要通过门描述符。由于中断门是特殊的门描述符，所以可以通过时钟发生器和中断门来完成进程调度。



PCB 的定义：

```
typedef struct s_proc {
    STACK_FRAME regs;              /* process registers saved in stack frame */
    u16 ldtSelect;                 /* gdt selector giving ldt base and limit */
    DESCRIPTOR ldts[LDT_SIZE];     /* local descriptors for code and data */
    int ticks;                     /* remained ticks */
    int priority;
    u32 pid;                       /* process id passed in from MM */
    char p_name[PROC_NAME_LEN];            /* name of the process */
} PCB;
```

STACK_FRAME 的定义：

```
typedef struct s_stackframe {
        u32 gs;              //'. 段寄存器应该是使用了双字对齐，才是定义成 32 位的。
        u32 fs;              // |  保存剩余的寄存器，进程运行的点的状态
        u32 es;              // |
        u32 ds;              // /
        u32 edi;             //'.
        u32 esi;             // |  由指令 popad 弹出，顺序固定
        u32 ebp;             // |
        u32 kernel_esp; // |<-popad 会忽略该寄存器。因此可以用此寄存器保存内核栈顶位置
        u32 ebx;             // |
        u32 edx;             // |
        u32 ecx;             // |
        u32 eax;             // /
        u32 eip;             //'.  这部分的顺序是固定的，因为它们是被调用门压栈
        u32 cs;              // |  自动放置的，而不是我们手动保存的。
        u32 eflags;          // |
        u32 esp;             // |
        u32 ss;              // /
} STACK_FRAME;
```

TSS 的定义：

```
typedef struct s_tss{
        u32 backlink;
        u32 esp0;      /*  内核栈  */
        u32 ss0;       /*  内核栈  */
        u32 esp1;
        u32 ss1;
        u32 esp2;
        u32 ss2;
        u32 gr3;
```

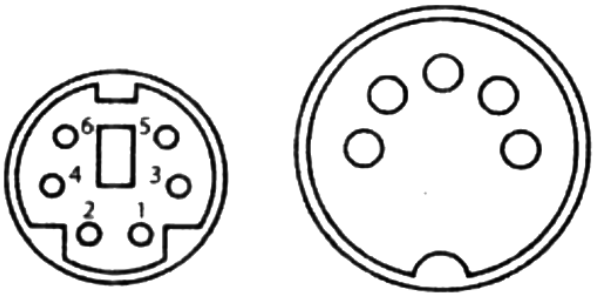
```
 u32 ds;
      u32 fs;
      u32 gs;
      u32 ldt;
      u16 trap;
      u16 iobase; /* I/O 位图基址大于或等于 TSS 段界限，就表示没有 I/O 许可位图 */
}TSS;
```

5.5 输入输出系统

要想进程进一步有实际的作用，需要和 I/O、内存管理等功能一起配合。不可能在一开机就创建所有进程，而是根据实际的需要，根据输入启动新的进程。同时为进程分配空间。

5.5.1 基本的输入-键盘

PS/2 键盘和 USB 键盘是现在流行的键盘，在笔记本上甚至只有内接和 USB 可以使用。还有一种老一点的 AT 键盘，它和 PS/2 都是圆形的，但要比 PS/2 大一些，并且少了一个孔。因此 AT 键盘称为"大口"键盘，而 PS/2 又称"小口"键盘。它们的接口分别叫做“5-pin DIN"和"6-pin Mini DIN"。



6-pin DIN　　5-pin DIN

键盘的敲击分为两个方面：动作和内容。

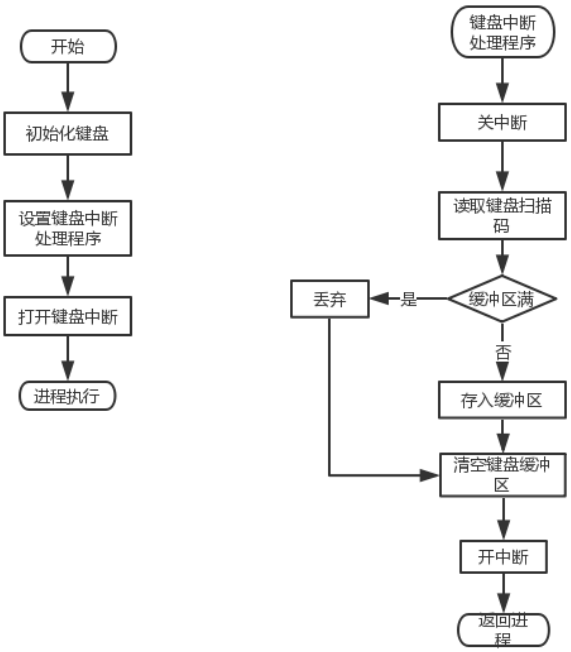
动作：按下，保持，松开

内容：哪个键

为了反应以上的内容，敲击键盘产生的扫描(Scan Code）码分成了两类：Make Code 和 Break Code。当一个键被按下或保持按住时，会产生 Make Code，当一个键松开时会产生 Break Code。处理 Pause 键之外，每一个键都产生一个 Make Code 一个 Break Code。

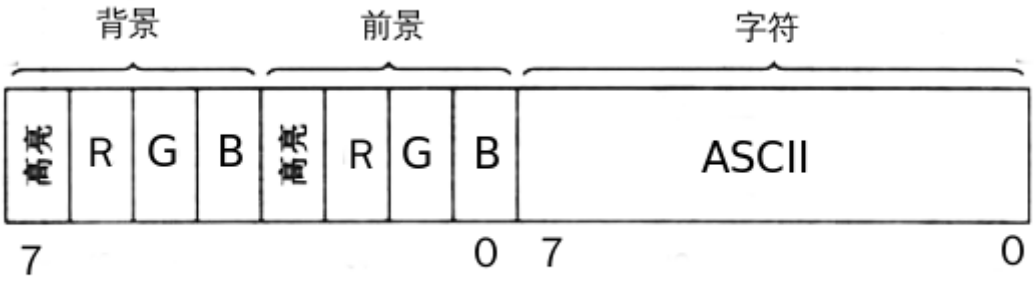
扫描码总共有三套，Scan Code set1,Scan Code set 2 和 Scan Code set3。Scan Code set1 是早期 XT 键盘使用的扫描码，现在的键盘默认都支持 Scan Code set 2，而 Scan Code set 3 则很少使用。

当按键产生动作后，8048 会根据键盘的动作产生扫描码，并把它发送给 8042，为了兼容老式机 8042 默认是将其转换成 Scan Code set 1 码，并将其放在输出缓冲区，然后向 8259A 发出中断请求。如果此时又有新的扫描码送来，8042 并不会接收，直到缓冲区被清空。



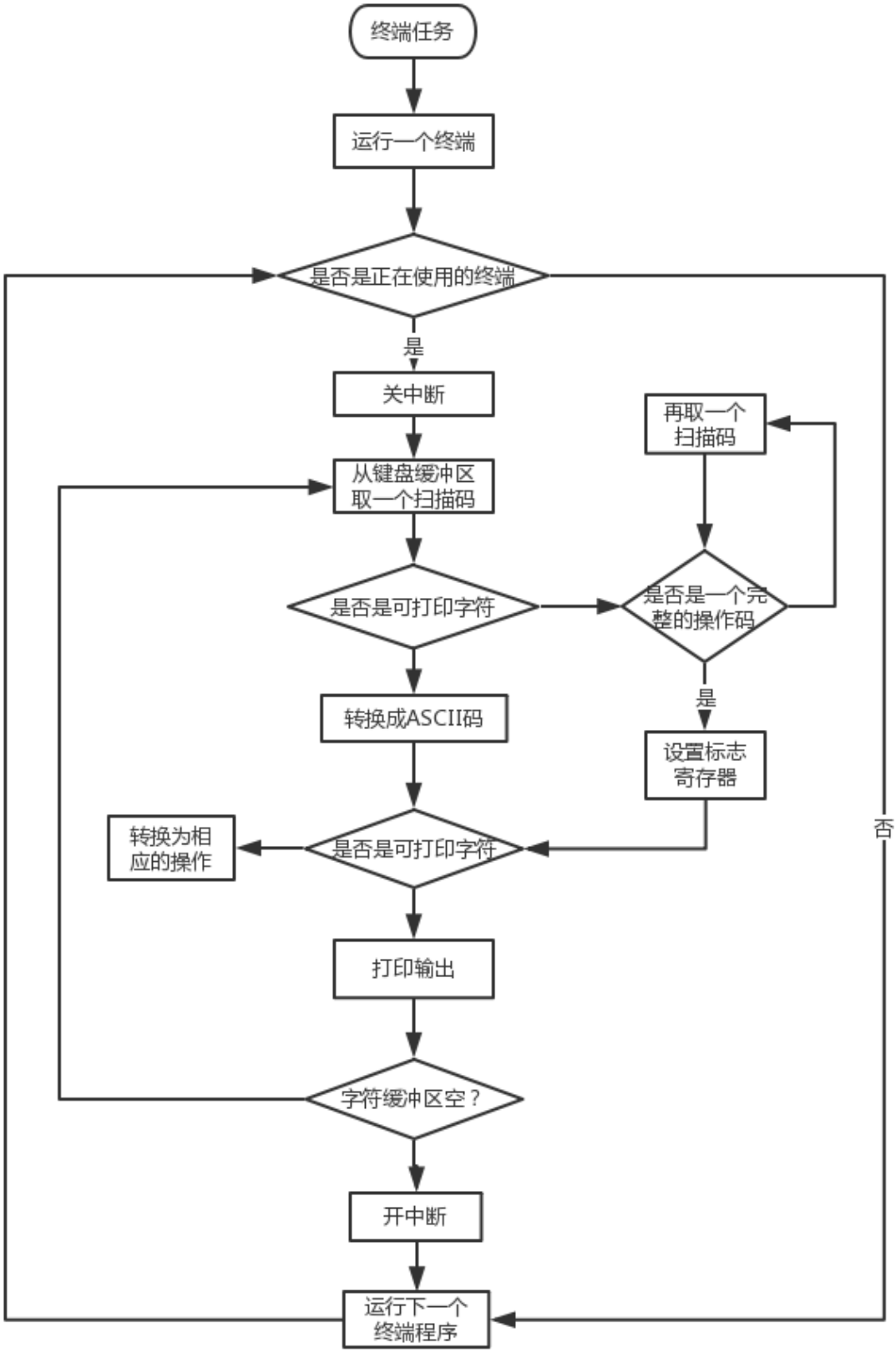
5.5.1 基本的输出-屏幕

计算机的显示是一个复杂的问题，显示适配器可以被设置成不同的模式，以显示更多的色彩，更有趣的图像的动画。不过这里仅设计文本模式。这种模式下，显存为 32K，能够显示 80×25 个字符。每两个字节显示一个字符，高字节表示属性，低字节存储字符。两个字节的具体内容如下：



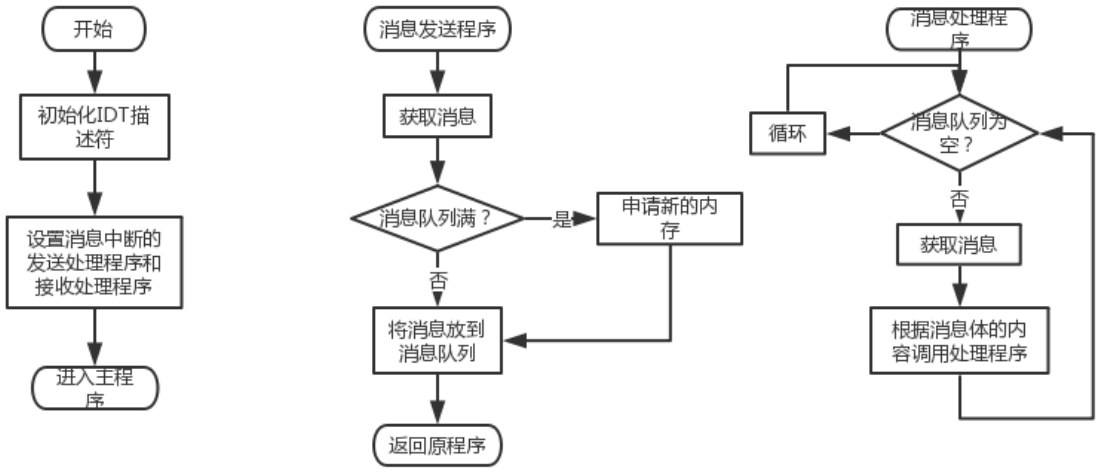
在同一屏幕上切换显示不同的内容，实际上是显示了显存的不同位置。在同一屏幕或者说显示器中显示不同的内容，其实是对显存或者显卡进行操作。在 VGA 寄存器中 Cursor Location High Register 和 Cursor Location Low Register 用两字节表示光标位置，注意是以两字节为单位的，也就是字符的偏移量。Start Address High Register 和 Start

Address Low Register 两字节为显示器开始显示的位置。通过这个这两组寄存器能够达到似乎是在不同屏幕切换的感觉。


终端任务
运行一个终端
是否是正在使用的终端
是
关中断
从键盘缓冲区
取一个扫描码
是否是可打印字符
转换成ASCII码
转换为相
应的操作
是否是可打印字符
打印输出
字符缓冲区空？
开中断
运行下一个
终端程序
再取一个
扫描码
是否是一个完
整的操作码
是
设置标志
寄存器
否

5.6 进程间通信(IPC)

基于消息机制的进程间通信，需要有系统调用来实现，一个系统调用专门用来发送消息，一个进程专门用来接收消息。消息体中封装了要完成的工作，和要完成该工作的程序。



结论

本设计完成了模拟操作系统微内核的设计与实现，将操作系统最核心的保护机制、任务调度、进程间通信、中断处理进行了最简的实现。同时，为了达到可以运行和测试

的效果，实现了内核的加载，文件系统。提供了用户交互界面。可以说麻雀虽小，五脏俱全。本设计的操作系统内核，完全可以满足操作系统理论的学习和教学需求。

同时为了更好的开发，本设计指出了一整套内核开发的解决方案。包括使用开源软件组织提供的开发工具链。以及将主流的版本控制工具 git 用于本设计，使用 github 为开发过程备份。这些开发方案和管理、备份方案，不仅适用于本设计，也适用于普遍的项目中。

参考文献

[1] Anderson, T.E., Bershad, B.D., Lazowska, E.D., and Levy, H.M.: Scheduler Activations: Effective Kernel Support for the User-Level Management of Parallelism [J]. ACM Trans.on Computer Systems, Feb.1992,vol.10:pp.53-79

[2] BACH, M,J.: The Design of UNIX Operating System, Upper Saddle River[M]. NJ: Prentice Hall, 1987.

[3] Bic,L.F.,and Shaw, A.S.: Operating System Principles, Upper Saddle River[M]. NJ: Prentice Hall, 2003.

[4] Levine, G.G.: Defining Deadlocks[J].Opearting Review ,Jan.2003a,vol.37:pp.54-64.

[5] LI,K.,and Hudak,P.: Memory Coherence in Shared Virtual Memory System[J]. ACM Trans. On Computer Systems,Nov.1989, vol. 7:pp.321-359

[6] JeffDuntemann.:Assembly Language Step By Step 2nd Edition[M].Robert Ipsen,1992

[7] Intel Inc.:Intel 64 and IA-32 Architectures Software Developer’s Manual:Basic Architecture(2006)

[8] Intel Inc.:Intel 64 and IA-32 Architectures Software Developer’s Manual:Instruction Set Reference(2006)

[10] Andrew S.Tananbaum. Modern Operating Systems(Second Edition),Prentice Hall,2001

[11] 汤小丹、梁红兵、哲凤屏、汤子赢.计算机操作系统 第四版[M]。西安电子科技大学出版社

[12] OSDev.org[BE/OL].http://wiki.osdev.org/Main_Page

附录

本设计中所有的代码都保存在 github。使用如下的命令可以从从库中克隆该设计完成的内核源代码：

git clone git@github.com:guobool/operating-system.git

也可以访问 github 的代码仓库：

https://github.com/guobool/operating-system

关于开发过程中的详细文档，可以在仓库的 Readme 中找到详细说明。

致谢

首先应该感谢我的父母，他们润物无声的爱给了我。他们没有文化，只是知道默默的付出，正是他们的付出让我有时间完成大学的学习和本设计。在这里由衷的祝愿他们身体健康。

同时，感谢开源运动的支持者和开源社区，本设计中用到的所有工具都是来自开源社区无数奉献者的辛勤劳动。也是开元运动的支持，才有如此多的开元软件和系统可以供我学习，也是开源运动让现在关于操作系统学习的书籍和资料如此的广泛和结合实际。虽然我知识窥到了其中一些边角，已经受益颇深，希望自己也能投身于开源的怀抱之中。

感谢于渊老师，他写的《Orange's 一个操纵系统设计与实现》是本设计参考的主要内容。由于经历有限，还无法对系统进行完全自己的设计，希望自己能够学习更多的操作系统知识，将系统完善的更有自己的特色。