基于 x86 架构的操作系统微内核设计与实现

吴 斯[1] 梁心雨[2]

（广西谷堆信息科技有限公司，广西 南宁 530012；桂林航天工业学院，广西 桂林 541004）

【摘 要】操作系统是计算机系统的关键组成部分，负责管理和协调计算机的所有硬件资源和软件资源，它使得人们能够能容易地使用计算机完成学习和工作。但是要理解操作系统本身的架构和运行机制是一件非常困难的事情，文章介绍了一种基于 x86 架构的操作系统设计方法，通过实现操作系统的内存管理、进程与线程管理以及中断和系统调用等基本功能，来帮助人们更深入地理解操作系统内部的实现机制。

【关键词】操作系统；进程；中断；内存管理

【中图分类号】TP316 【文献标识码】A 【文章编号】1008-1151(2020)10-0012-03

Design and Implementation of Operating System Microkernel Based on x86 Architecture

Abstract: Operating system is the key component of the computer system, which is responsible for the management and coordination of all the hardware and software resources of the computer. It makes it easy for people to use the computer to finish study and work. However, it is very difficult to understand the architecture and operating mechanism of the operating system itself. This paper introduces method to design operating system based on x86 architecture. By implementing the memory management, process and thread management, interrupt and system call and other basic functions of the operating system, people can understand the internal implementation mechanism of the operating system more deeply.

Key words: operating system; thread; interrupt; memory management

引言

从 1946 年第一台计算机诞生到 20 世纪 50 年代中期，一直都是手工操作计算机，之后才陆续出现了批处理系统、多道程序系统、分时系统以及最后的通用操作系统[1-3]。它集成了前面的多种系统功能为一体，并不断地提供新的功能，简化用户门槛，使得计算机越来越普及，即使是没有相关专业知识的人也可以非常方便轻松的使用计算机完成日常中的工作[4]。发展到如今，计算机更是成了人们生活中不可或缺的一部分，密切嵌入到日常生活中的方方面面，而这一部分也归功于操作系统的发展，带来的低门槛使用与高效协调硬件资源与软件程序，成功将计算机带入了千家万户[5]。

在操作系统带来越来越多方便的同时，客观上也带来了一些问题，高度的抽象与低门槛的使用，使得人们想要了解计算机的工作原理也变得越发困难。计算机历史的发展过程，某种程度上可以概括为在用户与计算机之间不断地增加抽象层，这些抽象层越多，用户越方便，但想要触及到原理性的知识，也意味着要不断地去挖掘、穿透这些抽象层，特别是鉴于操作系统这种工程性的产物[6]。在长达几十年的发展中，错误的探索或者无意义的历史包袱也会附加在其上，使其变得愈发臃肿，想要了解原理性的知识也变得愈发困难，如何能用一种趣味加有效的方法帮助人们更好地了解计算机原理知识就变得很有研究意义[7]。本文利用基于 x86 架构，设计操作系统的基本框架，实现内存管理、进程与线程管理以及中断和系统调用等模块的基本功能，通过这些功能的实现，便于人们更容易深入到操作系统内部，了解和掌握其核心架构。

1 设计思路

利用 32 位 Linux 系统作为编译机器，在其上运行着 gcc、nasm、make 三个软件，内核代码经过编译成功后，得到可执行的内核文件，然后使用真实的物理机运行该内核文件，使用 bochs 虚拟机作为内核镜像运行的载体，因为 bochs 与 VirtualBox 都支持跨平台，因此系统可以运行在任意平台上。

从层次结构上来划分，系统内核最底层是 MBR，主引导记录，其负责读取内核加载器并将控制权转交给内核加载器。内核加载器负责将整个内核加载到内存中，然后再将控制权交给内核。第二层是内核的基础模块，主要涵盖内存管理模块、进程管理模块、文件系统、中断与系统调用等 4 个最核心的方面。第三层是驱动层，实现硬盘驱动、键盘驱动等，以实现人机交互和文件管理。最上层则是用户相关的，包括

【收稿日期】2020-08-08

【作者简介】吴斯（1980—），女，广西玉林人，广西谷堆信息科技有限公司工程师，研究方向为计算机应用技术。

终端与一部分常用命令，例如目录切换、增删查改文件、文件夹等操作。

2 模块的实现

本系统主要实现了内存管理、进程与线程管理以及中断和系统调用等 4 个操作系统基本模块的功能，下面针对每个模块的设计思路进行介绍。

2.1 内存管理的实现

内存管理模块是操作系统微内核的主要模块，实现内存的分配与释放。内存管理机制建立在二级页表机制之上，由于虚拟内存机制，对每个进程来说都可独享 4 G 地址空间，因此每个用户进程都具有自己的二级页表，内核也有自己的独立二级页表。关于内存的操作都是修改各自的二级页表来实现，然后二级页表都通过修改内核中唯一的内存池来建立与真实物理内存的映射关系。

系统用位图管理真实的物理内存，用内存池将整个内存划分为内核池与用户池，并用位图来管理。位图是使用位（bit）来映射某种资源的数据结构，每一位用来代表内存中的 4K 空间，和分页机制的粒度保持一致。为了映射整个 4G 空间，总共需要 1,048,576（1024*1024）位来表示整个 4G 空间，合计 128K 的空间。位图只有内核持有并且可以操作。

针对位图数据结构，定义一些常用操作：（1）给定一个索引，判断位图中其值是否为 1。（2）给定一个索引与目标值，将位图中其值设置为目标值。（3）给定一个索引，长度和目标值，将从索引开始的长度个位均设置为目标值。

内存池是使用这种数据结构来管理内存的模块，其作用是管理内存，对系统而言，需要管理的有理物理内存和虚拟内存。基于内存池，需要提供分配一个页面与释放一个页面这两个操作，下面是页分配和页释放的算法流程。

页分配：

（1）申请方查找自己的虚拟内存池，找出空闲的 1 位，修改位图并将其所表示的 4 K 内存的起始虚拟地址 A1 给申请方。

（2）内核检查用户内存池，查找空闲的 1 位，修改位图并获取其代表的 4 K 内存的起始物理地址 P1。

（3）内核获取申请方的二级页表，利用地址 A1 的高 10 位索引到对应的目录项，若此目录项对应的 1024 个页表项不存在，也创建这 1024 个页表项。

（4）其次利用地址 A1 的中 10 位在 1024 个页表项中索引对应的页表项。

（5）将地址 P1 写入索引到的页表项，针对其他标志位写入正确的值，则页分配成功。

（6）此时，申请方申请成功，可以开始对地址 A1 进行操作。

页释放：

（1）申请方请求内核，并传入待释放的虚拟地址 A1。

（2）内核利用 A1 的高 10 位索引目录项。

（3）利用 A1 的中 10 位索引页表项，将其标志位 P 置为 0。

因为判断一个虚拟地址到物理地址关键的点便是页表项的标志位 P，意为 present，若为 0 表示该映射不在内存中。

2.2 进程和线程管理的实现

进程是操作系统中运行任务的实体，系统实现了内核级别的线程支持和用户级别进程。从 CPU 的角度看，线程是一道执行流，需要被调度到 CPU 上运行，是进程调度的最小单位，而进程是在系统中具有唯一性的系统资源，同时可以有多道执行流。因此在实现内核线程基础上，只需要再找个位置保存系统资源，让属于同一个进程的线程共享这些系统资源，便实现了用户进程。

系统选择使用轮询的方式去实现调度算法，任务切换方式采用 Linux 系统的实现，进程数据结构 PCB 使用双向链表维护。下述描述具体的实现方法。

进程实现线程，线程的实现有两种：

（1）内核线程：是指用内核层面上去实现一个调度管理线程的机制，用户程序通过系统提供的系统调用创建线程。

（2）用户线程：在内核层面没有线程机制的情况下，内核只看得到进程这种独立执行体，线程进而由用户进程自身实现，用户自己实现线程的创建与调度。

进程：在实现内核线程的基础上，进程管理还需要解决创建进程、执行进程和调度进程等问题。。

（1）创建进程：其本质是创建 PCB（Process Control Block）的过程，即程序控制块，它是进程的身份证，用来记录与此进程相关的信息，如进程状态、PID、优先级、上下文环境等。PCB 的具体实现没有统一的规则，本系统实现的 PCB 如图 1 所示，其中，寄存器映像用来保存进程在被调度下 CPU 时所有寄存器的值。当用户进程通过系统调用请求内核时，内核将使用 0 级特权栈与栈指针完成操作。PID 表示进程的 ID，唯一标示某个进程。进程状态分别有运行、就绪、阻塞、僵死等状态。优先级则会影响进程的可执行时间片，可执行时间片多的进程占用 CPU 的时间也会多。运行时间片指进程被调度上 CPU 的时间片数量，每发生时钟中断，处理程序都会检查当前进程的运行时间片，以判断是否要发起调度。进程的页表地址是进程重要的系统资源之一，是内存管理的基础，每个进程都会有自己的二级页表。打开的文件描述符是在涉及文件系统操作时需要。本系统提供 fork 系统调用，因此需要追踪父进程。

图 1 PCB

（2）运行进程：程序保存在硬盘上，当运行此程序时，操作系统便会将程序加载进内存中，构建为进程并开始执行，因此运行进程的本质就是读取硬盘文件，解析可执行文件并放置在内存中。这个过程与 bootloader 模块中加载并解析内核镜像的本质是一样的，在此便不再赘述。

（3）调度进程：调度进程涉及管理所有进程以及调度算法。系统使用的是根据时间片与优先级去轮询的策略。进程在产生时，根据优先级会被分配不同数量的时间片，在进程运行时，每次发生时钟中断都会减少当前 CPU 上进程的时间片，当时间片减少为 0 后，将启动调度机制，调度其他的进程上 CPU 执行。

有了 PCB 后，就可以完整地保存一个进程执行、切换的完整信息了。但是光有 PCB 保存一个进程的信息还不够，需要有一个数据结构能保存所有的进程信息。在本系统中就是使用双向链表来存储。

2.3 中断的实现

2.3.1 中断

是指当 CPU 正在执行某个程序时，暂时停止执行当前程序，转而执行另一段程序，执行完毕后再返回继续执行原先的程序。整个过程称为中断处理。系统包含外部中断和内部中断。外部中断来自 CPU 之外的某个硬件，例如网卡收到了网络包，硬盘准备好了数据，等等，此时通知 CPU 让其读取数据，这些是可屏蔽中断。除此之外，还有可能遇到灾难性错误，例如系统掉电、内存读写错误、总线校错失败等，这些是不可屏蔽中断，当发生这些中断时，本质上是计算机遇到严重问题即将宕机，因此通知 CPU。CPU 通过图 2 的形式接受外部中断。

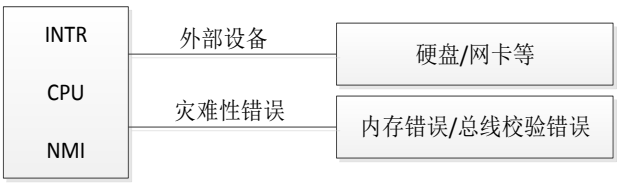

图 2 外部中断图

内部中断是 CPU 内部发起的中断，常见的内部中断如表 1 所示。

表 1 中断向量表

编号	描述	来源	类型
0	Divide Error	Div and IDIV instructions	Fault
1	Debug	Any code or data reference	Fault/Trap
2	NMI interrupt	Non-maskable external interrupt	Interrupt
3	Breakpoint	INT3 instructions	Trap
…	…	…	…

2.3.2 中断描述符

当 CPU 接受到某个中断信号时，CPU 的目的是针对这个中断去执行一段中断处理程序，而这段程序便是有中断描述符定义的，其格式如图 3 所示。

图 3 中断描述符

其中 Selector 是代码段描述符选择子，在 GDT 中定义，biases 是段内偏移量。利用中断描述符便可以定义一段程序来执行某种中断处理。

2.3.3 8259 可编程中断控制器

8259 是用来管理和控制可屏蔽中断的代理，可以进行中断优先级裁决、屏蔽某些中断等操作，最后得出目标中断向量号传递给 CPU。在本系统中使用级连的方式来组建 8529 芯片，如图 4 所示。

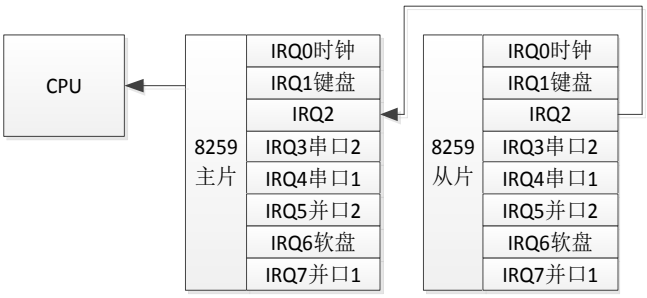

图 4 8259 可编程控制器

2.4 系统调用的实现

系统调用本质上是用户进程需要完成一段操作，由于进程本身没有权限，需要请求内核帮忙完成操作。而请求内核的操作涉及到 3 特权级向 0 特权级切换，也称陷入内核。既然涉及到底特权级向高特权级切换，必然需要使用 CPU 提供的几种机制，这里采用中断机制完成切换操作。在前文实现中断时，预留了 0x80 号中断向量为系统调用，因此，当用户请求系统调用时，首先用 int 0x80 触发中断，随后进入中断处理程序，在中断处理程序中，拿到用户传递进来的系统调用号与参数，即可进行相应操作，去帮助用户进程完成其没有权限的操作。

3 结论

本文介绍了实现一个内核最基本的框架和机制，通过利用基于 x86 架构，实现了操作系统的内存管理、进程与线程管理以及中断和系统调用等功能，通过这些功能的实现，使人们更容易深入了解操作系统内部的实现机制。

（下转第 34 页）

关系数 r 为 0.9998，吡唑醚菌酯的线性回归方程为 y = 2864966.79x + 2545.8，相关系数 r 为 0.9997，线性关系较好，可以满足样品定量分析要求。

表 1　苯醚甲环唑及吡唑醚菌酯的浓度与峰面积的关系

名称	苯醚甲环唑						吡唑醚菌酯					
重复次数		1	2	3	4	5		1	2	3	4	5
浓度/mg mL^{-1}	0.00	0.08	0.16	0.24	0.32	0.40	0.00	0.11	0.22	0.33	0.44	0.55
峰面积	0	237966	451786	694092	926728	1142663	0	132421	251758	386899	515586	640399

2.3　方法的精密度实验

按照 1.2 检测条件，对 40%苯醚甲环唑•吡唑醚菌酯悬浮剂连续进样 5 针，计算样品含量，得到苯醚甲环唑的标准偏差为 0.1383，变异系数为 0.96%，吡唑醚菌酯的标准偏差为 0.2371，变异系数为 0.97%，如表 2 所示。

表 2　方法的精密度试验测定结果

有效成分	测定值/%	平均值/%	标准偏差	变异系数/%
苯醚甲环唑	14.29	14.43	0.1383	0.96
	14.58			
	14.51			
	14.48			
	14.27			
吡唑醚菌酯	24.21	24.42	0.2371	0.97
	24.69			
	24.56			
	24.52			
	24.14			

表 3　方法的准确度试验测定结果

有效成分	样品量/g	添加标样量/g	理论值/g	实测值/g	回收率/%	平均回收率/%
苯醚甲环唑	0.0187	0.0021	0.0208	0.0207	99.41	98.69
	0.0175	0.0042	0.0217	0.0213	98.42	
	0.0184	0.0062	0.0246	0.0244	99.10	
	0.0184	0.0083	0.0267	0.0261	97.66	
	0.0199	0.0104	0.0303	0.0299	98.85	
吡唑醚菌酯	0.0317	0.0028	0.0344	0.0342	99.40	98.62
	0.0296	0.0055	0.0352	0.0346	98.37	
	0.0311	0.0083	0.0394	0.0391	99.09	
	0.0311	0.0111	0.0422	0.0412	97.77	
	0.0337	0.0139	0.0476	0.0470	98.48	

2.4　方法的准确度实验

准确称取 5 个平行样品，分别加入 1.3.1 中配制的 A 溶液 1 mL、2 mL、3 mL、4 mL、5 mL，在上述色谱条件下进行测定，测得苯醚甲环唑的平均回收率为 98.69%，吡唑醚菌酯的平均回收率为 98.62%，如表 3 所示。

3　结论

40%苯醚甲环唑•吡唑醚菌酯悬浮剂是使用越来越广泛的杀菌剂，为解决害虫对单一农药制剂的抗药性提供了新的选择。本文讨论的 40%苯醚甲环唑•吡唑醚菌酯悬浮剂的反相高效液相色谱检验方法，实验的精密度，准确度均满足要求，且线性的相关性良好，试验的方法简单，能够使实验人员避免使用毒性和挥发性较大的正己烷，能够在同一实验条件下同时对两个成分进行分析，实现了对苯醚甲环唑异构体、吡唑醚菌酯与样品中的杂质完全分离的效果，节省了实验时间，提高了效率，具有方便、快捷、准确的优点，且重现性、稳定性，分离效果均能满足产品的检验要求，是一种可行的分析方法。

【参考文献】

[1] 陈格新，鞠光秀，牛淑妍. 苯甲•吡唑酯微乳剂研制及田间防效[J]. 现代农药，2016，4(2): 32-35.

[2] 徐妍，马超，刘世禄，等. 浅谈农药悬浮剂的质量提升[J]. 现代农药，2010，9(2): 18-24.

[3] 董立峰，邵彦坡，刘希玲，等. 40%苯醚甲环唑•吡唑醚菌酯 SC 配方的研制[J]. 世界农药，2016，10(2): 47-50.

[4] 杨珂，李雪生，刘晓亮，等. 吡唑醚菌酯•苯醚甲环唑在黄瓜及土壤中的残留分析[J]. 农药，2016，55(11): 832-834.

（上接第 14 页）

【参考文献】

[1] 李忠，王晓波，余洁著. x86 汇编语言：从实模式到保护模式[M]. 北京：电子工业出版社，2013.

[2] (美)克尼汉(Kernighan，B. W.)，(美)里奇(Ritchie， D. M.)著. C 语言程序设计[M]. 北京：机械工业出版社，2004.

[3] 郑钢. 操作系统真象还原[M]. 北京：人民邮电出版社，2016.

[4] (荷)塔嫩鲍姆(Tanenbaum， A. S.). 现代操作系统[M]. 北京：机械工业出版社，2009.

[5] (美)Randal E. Bryant. 深入理解计算机系统[M]. 北京：机械工业出版社，2016.

[6] 任振强. MTX 操作系统内核与启动流程分析[J]. 微型机与应用，2019，38(2): 50-54.

[7] 欧阳湘臻，朱怡安，李联，等. 一种安全关键的嵌入式实时操作系统内核设计[J]. 计算机工程，2019，45(7): 78-85.