HZ BOOKS
华章IT

深度探索嵌入式操作系统

从零开始设计、架构和开发

彭东 著

Inside Embedded Operation System

Design, Structure and Develop from Scratch

- 作者是计算机领域“鬼才”，独立编写出基于x86_64平台的系统内核和基于ARM平台的嵌入式系统内核，可运行于真实物理机
- 从硬件和软件两个维度系统、深度阐述嵌入式操作系统的构成、原理和实现方法，真正教读者从零开始设计、架构和开发一个完整的、可工作的嵌入式操作系统

机械工业出版社
China Machine Press

深度探索嵌入式操作系统

从零开始设计、架构和开发

Inside Embedded Operation System

Design, Structure and Develop from Scratch

彭东 著

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

深度探索嵌入式操作系统：从零开始设计、架构和开发 / 彭东著 . —北京：机械工业出版社，2015.9
（Linux/Unix 技术丛书）

ISBN 978-7-111-51487-9

I. 深… II. 彭… III. 实时操作系统 IV. TP316.2

中国版本图书馆 CIP 数据核字（2015）第 216003 号

深度探索嵌入式操作系统

从零开始设计、架构和开发

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号　邮政编码：100037）

责任编辑：高婧雅　　责任校对：殷　虹

印　　刷：北京市荣盛彩色印刷有限公司　　版　　次：2015 年 10 月第 1 版第 1 次印刷

开　　本：186mm × 240mm　1/16　　印　　张：33.75

书　　号：ISBN 978-7-111-51487-9　　定　　价：99.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

客服热线：（010）88379426　88361066　　投稿热线：（010）88379604

购书热线：（010）68326294　88379649　68995259　　读者信箱：hzit@hzbook.com

Preface 前　言

为什么写这本书

随着移动互联网技术的成熟，物联网也开始崭露头角，由此产生了各种小型、低功耗的智能硬件，这些智能硬件被嵌入到移动电话、手表、冰箱、空调、电视机、洗衣机等常用电子产品中，使这些常用电子产品功能更加强大、更加智能，而且它们可以连接到网络，便于用户远程操控，从而大大改善了人们的生活。

因此，嵌入式行业才变得如此火热，那些被嵌入到电子产品中的智能硬件，也需要一些小巧、特殊的操作系统软件才能正常工作，这类小巧、特殊的操作系统软件，称为嵌入式系统。2013 年下半年，我开始学习嵌入式系统，并编写了一个嵌入式操作系统——LMOSEM。在互联网上也认识了不少研究嵌入式的朋友，在他们的要求和规劝下，我终于有勇气把我研究出来的东西归纳、整理成册，也算是我学习的笔记，于是就有了这本书。虽然有很多的顾虑，怕贻笑方家、怕误导同道……但是我的每行代码、每个点子，都在实机上测试过并证明了其正确性，所以也就心下一片坦然了。如果这本书能够被后来者借鉴一二，或者解决他们的一些疑惑，我自然欣慰万分。

关于 LMOSEM

关于 LMOSEM，这得从 LMOS 开始说起。2010 年下半年，我开始准备要写个操作系统内核，没有其他目的，只是出于学习，出于兴趣。由于是自己独立从零开始设计、编写的，我觉得自己这种行为有点疯狂，索性用 LMOS（liberty madness operating system）命名了我的操作系统。LMOS 经过这几年的独立开发，现在已经发布了 6 个测试版本。先后从 32 位单 CPU 架构发展到 64 位多 CPU 架构，现在的 LMOS 已经是多进程、多线程、多 CPU、支持虚拟内存的 x86_64 体系下的操作系统内核。LMOS 的这些特性，非常适合通用计算机领域，如 PC、工作站、小型服务器。这些特性导致 LMOS 代码量庞大，一些组件不够小巧，削剪起来非常复杂，很难保证削剪后的组件是否健壮，因此 LMOS 不适合于嵌入式领域，所以笔

者才重新开发了 LMOSEM——适合嵌入式领域的操作系统。

LMOSEM 依然删除了很多代码，因为写书要做到简单，便于理解。即便如此，LMOSEM 依然包含了现代操作系统的大部分重要组件，如内存管理、进程管理、驱动模型、文件系统等。这些组件的实现过程在本书中都会有详细的介绍。LMOSEM 不支持实时性功能，嵌入式操作系统也不一定要是实时性的操作系统，何况我们是出于学习的目的。为了代码的清晰、简单，我们暂不考虑安全性和性能方面的问题。等到明白了操作系统原理，我们再去不断修正、优化，使之功能变得更多，性能变得更强。笔者开发的 LMOSEM 操作系统项目，是在 Linux 操作系统下开发的，用到了 Linux 操作系统的很多工具。笔者不会和读者讨论为什么不用常用的 Windows 系统，也不会说谁好、谁不好。如果读者非常喜欢 Windows 系统，那么也可以尝试着把这个项目迁移到 Windows 系统下。但是笔者书中演示的环境还是 Linux 系统。关于如何搭建开发环境，本书后面的章节有详细的介绍。在那里读者会发现用 Linux 系统开发 LMOSEM 内核有很多方便之处，如会用到的 MAKE、GCC、LD 等，这些工具在 Linux 系统下都很容易得到，在 Windows 系统下虽然也能做到，但相对麻烦一点。何况今天的 Linux 系统已经很好用了。

读者对象

- 如果读者是一位纯粹的操作系统爱好者，对其有着浓厚的兴趣，那么本书将非常适合。
- 如果读者是嵌入式领域的从业者或者学生，也可以从本书中获得很多帮助。
- 如果读者是一位普通的应用软件开发者，业余时间也可以翻翻此书，书中的一些设计方法和编程手段，或许可以借鉴一二。
- 如果你只是想了解一些计算机硬件系统和软件系统的常识，那么本书同样会让你获益。

如何阅读本书

为了能更轻松地阅读这本书，笔者建议先了解 C 语言这门编程语言，对数据结构有所了解就更好了。除这些外，笔者假定读者没有其他任何技能。除了需要的上述技能，读者还需要对操作系统有强大的兴趣和求知欲，要有坚强的意志、永远不放弃的精神。开发操作系统内核本身就不是件容易的事，必然会有很多问题在等着我们，但是遇到问题不要害怕，静下心从容面对，只要我们不放弃，问题最终会解决。

本书很简单，没有拐弯抹角，没有反复修饰，但是必要的细节从不漏掉。宁可在细节上啰嗦一点，也不在不相关的地方多写一句。

本书的最终目的是构建一个用于学习的嵌入式操作系统内核，并工作在真正的物理机上。为了达到这一目的，本书大体上分为三部分：综述、硬件部分和软件部分。

综述部分（第 1 章）。

第 1 章，先说明操作系统的概念、功能和演进历史，最后得出现代操作系统的模型，使我们可以了解操作系统的轮廓。

硬件部分（第 2 ～ 3 章）。

第 2 章，从选择硬件平台开始，首先概述硬件平台的整体情况，接着了解编写操作系统内核必需的一些平台上的组件，如实时时钟、定时器、串口、中断控制器、内存芯片、Flash 芯片、CPU、MMU 等。让读者有初步的印象，在写代码用到某个组件时再详述其内部编程细节。

第 3 章，详细介绍处理器，重点介绍处理器的结构和特性、处理器的地址空间、处理器的状态和工作模式、处理器的寄存器和指令集。最后介绍处理器中的 MMU 和 Cache，对于 MMU，主要介绍 MMU 的作用和它对操作系统内核开发的影响、如何对 MMU 编程、MMU 的几种地址映射方式。而对于 Cache，重点介绍 Cache 的作用、Cache 的类型、Cache 的使用。

软件部分（第 4 ～ 12 章）。

第 4 章，介绍操作系统内核设计、操作系统内核的开发环境、开发操作系统内核的工具：GCC、LD、MAKE，以及它们的使用方法，最后介绍硬件平台的安装与测试。

第 5 章，首先介绍 C 语言使用寄存器的约定，以及它是如何处理参数、返回值的。接着介绍 C 语言基本数据类型的位宽及占用内存的大小，并用它们构建后面将要用到的一些基本的数据结构，如 list_h_t、spinlock_t、kwlst_t、sem_t。然后介绍 C 语言的数据结构在内存中存在的形式、对齐方式。最后介绍 GCC 独有的嵌入汇编代码的方式。

第 6 章，开始介绍 LMOSEM 的初始化，从第一行汇编代码开始、初始化 MMU 和中断向量、初始化串口设备、初始化内存管理数据结构和中断相应的数据结构，最后对一些数据结构进行测试。

第 7 章，开始介绍 LMOSEM 的内存管理组件。LMOSEM 的内存管理组件分为三层：块级内存管理、页级内存管理、字级内存管理，这三个层分别应对不同的内存分配请求。本章将结合实际代码介绍如何一步步实现这三大内存管理层。

第 8 章，介绍 LMOSEM 的中断管理组件，内容包括中断控制器的细节、中断管理组件的结构、如何处理中断、安装中断处理的回调函数等。

第 9 章，介绍 LMOSEM 的设备驱动模型，其中介绍 LMOSEM 如何管理众多的硬件设备、LMOSEM 支持的设备类型、驱动模型的数据结构和基础性代码，最后用两个驱动程序实例介绍如何在 LMOSEM 的驱动模型下编写规范的设备驱动程序。

第 10 章，介绍 LMOSEM 的进程管理组件，包括进程的由来、进程相关的数据结构、系统空转进程的建立与运行、进程调度、新建进程、进程的睡眠与唤醒、进程测试等相关内容。

第 11 章，介绍 LMOSEM 的文件系统组件，包括文件系统的设计、文件系统的建立、文件系统的基础操作、文件本身的操作，如文件的打开、新建、读写、删除等。最后对文件系

统组件进行严格的测试。

第 12 章，介绍 LMOSEM 的接口，包括许多 LMOSEM 的 API 和库函数的实现细节，主要包括时间、进程、内存、文件与设备、标准输入 / 输出等方面的 API 和库函数。

勘误和支持

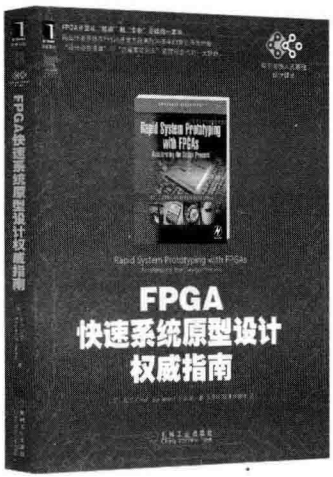
由于笔者水平有限，加之编写时间仓促，书中难免会出现一些不准确的地方，恳请读者批评指正，在技术之路上共勉。我的 CU 博客地址是：http://blog.chinaunix.net/uid/28032128.html。本书源代码已制作成光盘镜像文件，并上传到华章网站（www.hzbook.com），需要的读者可自行下载。

致谢

我，自幼患病，读书不多，计算机成了唯一的兴趣爱好，没有父母的长期支持，连生活都尚且不能自理，更别说完成此书了，他们对我的帮助和关爱，纵使千万言语也难表一二。由于经常在物理机上测试内核，要拆装一些设备和器件，这多亏了我的小弟，因为他一有时间就帮我做这部分工作。当然还有帮助过我的朋友，有一些是身边的，有些是网络中的。对父母、所有的亲人、朋友，我也只有常怀感恩之心，说声谢谢，谢谢他们一直的支持、帮助，谢谢他们一直对我那满满的关爱！

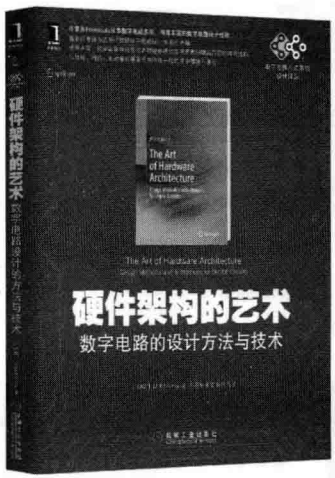
让笔者和你一起带着未知，带着好奇，带着兴奋，踏上操作系统的旅程吧！

推荐阅读



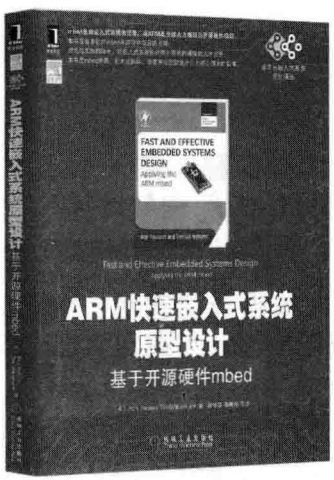
FPGA快速系统原型设计权威指南

作者：R.C. Cofer 等 ISBN：978-7-111-44851-8 定价：69.00元



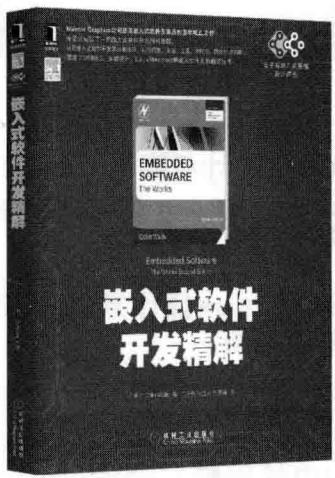
硬件架构的艺术：数字电路的设计方法与技术

作者：Mohit Arora ISBN：978-7-111-44939-3 定价：59.00元



ARM快速嵌入式系统原型设计：基于开源硬件mbed

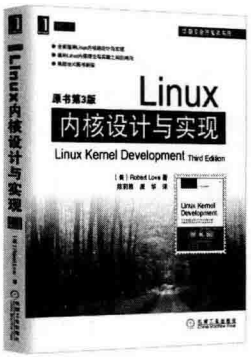
作者：Rob Toulson 等 ISBN：978-7-111-46019-0 定价：69.00元

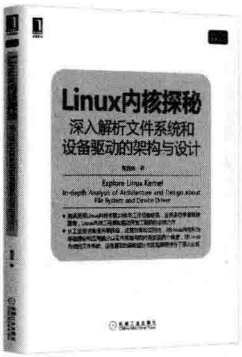


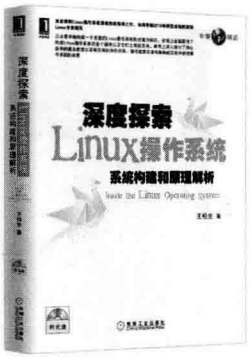
嵌入式软件开发精解

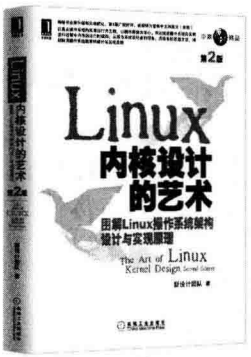
作者：Colin Walls ISBN：978-7-111-44952-2 定价：79.00元

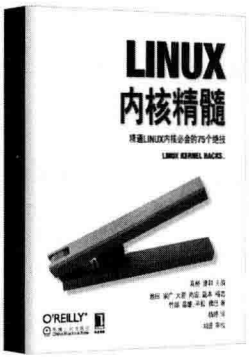
推荐阅读

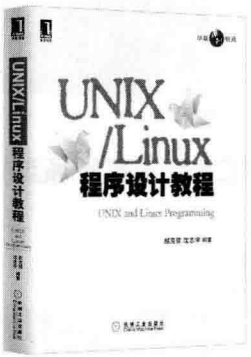












Linux内核设计与实现（原书第3版）

Linux内核开发人员Robert Love的力作，畅销多年的经典著作

Linux内核探秘：深入解析文件系统和设备驱动的架构与设计

腾讯顶级Linux系统专家和存储系统专家10年经验结晶

深度探索Linux操作系统：系统构建和原理解析

百度核心系统部门资深专家力作，Linux操作系统领域的里程碑作品

Linux内核设计的艺术：图解Linux操作系统架构设计与实现原理（第2版）

中国首部将版权输出到美国的计算机图书，中美两国取得骄人成绩

Linux内核精髓：精通Linux内核必会的75个绝技

日本多位一线内核技术专家的经验和智慧结晶

UNIX/Linux程序设计教程

UNIX/Linux权威著作，多所高校选定为教材

Contents 目　录

前言

第 1 章　操作系统的功能及为什么需要它 ····· 1

1.1　从 hello world 开始 ····· 1
1.2　操作系统功能演进 ····· 3
1.3　小结 ····· 7

第 2 章　硬件平台 ····· 8

2.1　选择平台 ····· 8
2.1.1　mini2440 ····· 8
2.1.2　mini2440 平台的信息 ····· 9
2.2　必须要关注的硬件 ····· 13
2.2.1　原因 ····· 14
2.2.2　RTC ····· 15
2.2.3　定时器 ····· 16
2.2.4　串口 ····· 17
2.2.5　中断控制器 ····· 18
2.2.6　SDRAM ····· 19
2.2.7　Norflash ····· 23
2.2.8　Nandflash ····· 24
2.3　小结 ····· 26

第 3 章　处理器 ····· 28

3.1　ARM 公司与其处理器 ····· 28
3.2　ARM920T 的结构与特性 ····· 29
3.2.1　ARM920T CPU 结构 ····· 29
3.2.2　ARM920T CPU 特性 ····· 32
3.3　ARM920T 存储体系 ····· 33
3.3.1　ARM920T 地址空间 ····· 33
3.3.2　ARM920T 存储器格式 ····· 34
3.3.3　ARM920T 存储地址对齐 ····· 35
3.4　ARM920T 状态 ····· 35
3.4.1　ARM 状态 ····· 36
3.4.2　Thumb 状态 ····· 36
3.5　ARM920T 处理器的 7 种工作模式 ····· 37
3.6　寄存器 ····· 38
3.7　异常和中断 ····· 43
3.7.1　什么是异常和中断 ····· 43
3.7.2　异常中断向量 ····· 46
3.8　ARM920T 指令集 ····· 47
3.8.1　指令及其编码格式 ····· 48
3.8.2　分支跳转指令 ····· 50
3.8.3　数据处理指令 ····· 53
3.8.4　装载和存储指令 ····· 63
3.8.5　程序状态寄存器操作指令 ····· 73
3.8.6　协处理器操作指令 ····· 76
3.8.7　异常中断产生指令 ····· 79
3.9　MMU ····· 80
3.9.1　MMU 概述 ····· 80

3.9.2 为什么要有 MMU ····· 82
3.9.3 ARM920T CP15 协处理器 ····· 85
3.9.4 MMU 页表 ····· 98
3.9.5 MMU 页面访问权限的控制 ····· 113
3.9.6 MMU 的快表 TLB ····· 113
3.9.7 MMU 的编程接口 ····· 114
3.10 Cache ····· 115
3.10.1 ARM920T 的 Cache ····· 115
3.10.2 Cache 的原理 ····· 116
3.10.3 Cache 的类型及要注意的问题 ····· 117
3.10.4 ARM920T Cache 的编程接口 ····· 119
3.11 小结 ····· 120

第 4 章 操作系统内核的设计与构建 ····· 122
4.1 操作系统内核的设计 ····· 122
4.1.1 内核要完成的功能 ····· 123
4.1.2 内核的架构 ····· 124
4.1.3 分离硬件的相关性 ····· 126
4.1.4 我们的选择 ····· 127
4.2 开发环境及相关工具 ····· 129
4.2.1 Linux 环境 ····· 129
4.2.2 文本编辑器 ····· 132
4.2.3 GCC ····· 134
4.2.4 LD ····· 136
4.2.5 make ····· 139
4.3 LMOSEM 的构建系统 ····· 142
4.3.1 LMOSEM 的 makefile ····· 142
4.3.2 LMOSEM 的链接脚本 ····· 147
4.4 开发板的安装 ····· 150
4.5 小结 ····· 154

第 5 章 语言间调用约定与基本数据结构 ····· 156
5.1 寄存器使用约定 ····· 156
5.1.1 寄存器别名 ····· 157
5.1.2 参数传递与返回值 ····· 157
5.2 基本数据结构 ····· 159
5.2.1 C 语言的基本数据结构 ····· 160
5.2.2 list_h_t 数据结构 ····· 161
5.2.3 spinlock_t 数据结构 ····· 164
5.2.4 kwlst_t 数据结构 ····· 165
5.2.5 sem_t 数据结构 ····· 166
5.3 数据结构存在于内存中的形式 ····· 168
5.4 C 与汇编的混用 ····· 170
5.5 小结 ····· 174

第 6 章 内核初始化 ····· 175
6.1 开始 ····· 175
6.1.1 第一行汇编代码 ····· 175
6.1.2 第一个 C 函数 ····· 178
6.2 MMU 和中断向量的初始化 ····· 181
6.2.1 初始化 MMU ····· 181
6.2.2 复制中断向量 ····· 186
6.3 串口初始化 ····· 190
6.3.1 串口硬件 ····· 190
6.3.2 内核的 printf ····· 196
6.4 机器数据结构 ····· 201
6.4.1 设计数据结构 ····· 201
6.4.2 确定一些重要数据结构与内核的地址 ····· 203
6.5 初级内存管理初始化 ····· 205
6.5.1 设计一些数据结构 ····· 205
6.5.2 初始化 mmapdsc_t 结构数组 ····· 209
6.5.3 建立起内存分配数据结构 ····· 212
6.6 中断初始化 ····· 215
6.6.1 设计一些数据结构 ····· 215
6.6.2 初始中断源描述符 ····· 220
6.7 初始化测试 ····· 222
6.8 小结 ····· 225

第 7 章　内存管理……226
7.1　内核功能层入口 ……226
7.2　内存管理组件的设计……228
7.3　块级内存管理……229
7.3.1　块级内存管理数据结构视图 ……229
7.3.2　块级内存管理接口 ……230
7.3.3　主分配函数 ……232
7.3.4　分配时查找 alcfrelst_t ……234
7.3.5　分配时查找和操作 mmapdsc_t……236
7.3.6　分配代码写得对吗 ……239
7.3.7　主释放函数 ……240
7.3.8　释放时查找 alcfrelst_t ……241
7.3.9　释放时查找和操作 mmapdsc_t……242
7.3.10　测试块级内存管理层 ……246
7.4　页级内存管理……248
7.4.1　页级内存管理接口及调用流程……248
7.4.2　相关的数据结构 ……251
7.4.3　页级内存管理初始化 ……254
7.4.4　分配主函数 ……256
7.4.5　分配时查找 mplhead_t……257
7.4.6　分配时新建页级内存池 ……258
7.4.7　分配时操作 mplhead_t……262
7.4.8　分配代码写得对吗 ……263
7.4.9　释放主函数 ……265
7.4.10　释放时查找 mplhead_t……266
7.4.11　释放时操作 mplhead_t……268
7.4.12　释放时删除页级内存池 ……269
7.4.13　测试页级内存管理层 ……271
7.5　字级内存管理……273
7.5.1　字级内存接口及调用流程 ……274
7.5.2　相关的数据结构 ……275
7.5.3　分配主函数 ……276
7.5.4　分配时查找 mplhead_t……278
7.5.5　分配时新建字级内存池 ……279
7.5.6　分配时操作 mplhead_t……282
7.5.7　分配代码写得对吗 ……283
7.5.8　释放主函数 ……285
7.5.9　释放时查找 mplhead_t……285
7.5.10　释放时操作 mplhead_t……287
7.5.11　释放时删除字级内存池……288
7.5.12　测试字级内存管理层 ……289
7.6　小结……292

第 8 章　中断管理……293
8.1　中断与中断控制器……293
8.1.1　什么是中断 ……293
8.1.2　S3C2440A 中断控制器……294
8.2　中断管理的架构与相关数据结构 ……298
8.2.1　中断管理的架构 ……298
8.2.2　设计数据结构 intfltdsc_t 和 intserdsc_t……298
8.3　中断处理……301
8.3.1　中断辅助例程 ……301
8.3.2　从中断向量开始 ……305
8.3.3　保存 CPU 上下文……306
8.3.4　中断主分派例程 ……310
8.3.5　确定中断源 ……314
8.3.6　调用中断处理例程 ……317
8.4　安装中断回调例程……319
8.5　小结……322

第 9 章　驱动模型……323
9.1　操作系统内核如何管理设备 ……323
9.1.1　分权而治 ……323
9.1.2　设备类型 ……325
9.1.3　驱动程序 ……327

9.2 相关数据结构……328
9.2.1 驱动……329
9.2.2 派发例程类型……329
9.2.3 设备 ID……330
9.2.4 设备……331
9.2.5 IO 包……332
9.2.6 设备表……333
9.3 驱动模型的基础设施……335
9.3.1 驱动程序从哪里执行……335
9.3.2 新建与注册设备……340
9.3.3 注册回调函数……344
9.3.4 发送 IO 包……345
9.3.5 调用驱动程序函数……346
9.3.6 等待服务……347
9.3.7 完成服务……350
9.3.8 驱动模型辅助函数……352
9.4 systick 驱动程序实例……356
9.4.1 systick 硬件……356
9.4.2 systick 驱动程序框架……360
9.4.3 systick 驱动程序实现……362
9.4.4 测试 systick 驱动程序……368
9.5 RTC 驱动程序实例……370
9.5.1 RTC 硬件……370
9.5.2 RTC 驱动程序实现……375
9.6 小结……385

第 10 章 进程……386

10.1 应用程序的运行……386
10.1.1 程序运行需要什么资源……387
10.1.2 任何时刻资源都可用吗……388
10.1.3 提出多道程序模型……389
10.2 相关的数据结构……390
10.2.1 设计进程的数据结构……391
10.2.2 调度进程表……392
10.3 LMOSEM 内核的第一个进程……394
10.3.1 进程管理组件的初始化……395
10.3.2 建立空转进程……396
10.3.3 空转进程运行……399
10.4 新建进程……404
10.4.1 分配进程描述符……404
10.4.2 分配内存空间……406
10.4.3 加入进程调度表……408
10.5 进程调度……410
10.5.1 调度算法……410
10.5.2 处理进程时间片……411
10.5.3 检查调度状态……414
10.5.4 选择进程……415
10.5.5 进程切换……418
10.5.6 进程等待与唤醒……421
10.5.7 进程测试……425
10.6 小结……428

第 11 章 文件系统……430

11.1 文件系统设计……430
11.1.1 文件系统只是一个设备……430
11.1.2 数据格式与存储块……432
11.1.3 如何组织文件……433
11.1.4 关于我们文件系统的限制……434
11.2 相关的数据结构……434
11.2.1 超级块……435
11.2.2 位图……435
11.2.3 目录……437
11.2.4 文件管理头……438
11.3 文件系统格式化……440
11.3.1 建立超级块……440
11.3.2 建立位图……445
11.3.3 建立根目录……448
11.4 文件系统基础操作……452
11.4.1 获取与释放根目录文件……453
11.4.2 字符串操作……455

11.4.3 分解路径名……457
11.4.4 检查文件是否存在……459
11.5 文件操作……460
11.5.1 新建文件……461
11.5.2 删除文件……463
11.5.3 打开文件……466
11.5.4 读写文件……469
11.5.5 关闭文件……472
11.5.6 驱动整合……473
11.6 文件系统测试……475
11.6.1 格式化测试……475
11.6.2 文件操作测试……479
11.7 小结……482

第 12 章 系统调用与应用程序库……483

12.1 系统调用机制……483
12.1.1 软中断指令……484
12.1.2 传递系统调用参数……486
12.1.3 系统调用分发器……488
12.2 时间管理系统调用……489
12.3 进程管理系统调用……492
12.3.1 进程的运行与退出……492
12.3.2 获取进程的 ID……494
12.4 内存管理系统调用……496
12.5 设备与文件系统调用……498
12.5.1 设备与文件的打开……498
12.5.2 设备与文件的关闭……506
12.5.3 设备与文件的读写……508
12.5.4 设备与文件的控制……512
12.6 应用程序库……514
12.7 测试……520
12.8 小结……526

后记……528

第 1 章 Chapter 1

操作系统的功能及为什么需要它

你或许已经卷起了衣袖，或许在摩拳擦掌准备大干一场，打一场硬仗。年轻人嘛，行事总是风风火火的。但不是笔者扫你的兴，泼你冷水，在我们写代码之前还有很长一段路要走，要静下心来。如果写操作系统是一次旅行，那么千万不要错过沿途的风景……

1.1 从 hello world 开始

操作系统也是软件，也是由一大堆程序组成的，所以不要觉得它多么神秘。既然是程序，我们就要知道它是干什么的以及为什么需要，笔者想用大多数人写的第一个程序——“hello, world!!”来描述这个原因。当然这个程序是用C语言写的。大致过程如图 1-1 所示。

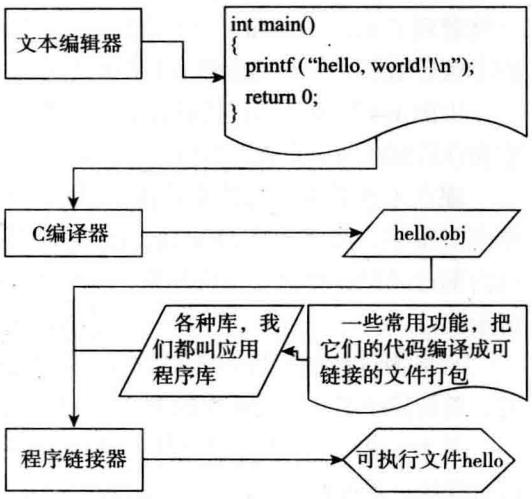


图 1-1 hello，world!! 程序的生成过程

这个程序正常运行后，会在屏幕上输出 hello, world!! 这个字符串。显然这个程序中最重要的是 printf 函数，然而我们并没有实现它。如果你能发现这一点，并迫切地想知道它究竟在背后干了些什么，那说明你和笔者一样有着对问题追根溯源的性格。这个也是我们研究问题本相的原动力。

那么 printf 在哪儿呢？都干了些什么呢？

它就在应用程序库中，代码虽然没有实现它，但库中实现了，程序链接器最终把它和代码链接装配在一起，这样程序也就能正确工作了，如图 1-2 所示。

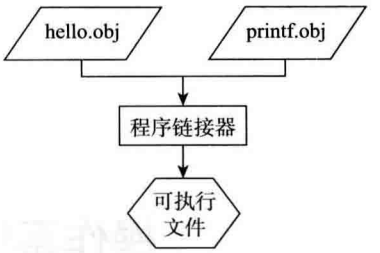


图 1-2 hello, world!! 程序的链接过程

实际中可能有差异，可能 printf 不是在单独的库中，也可能不只是链接这一个库就行了。但是道理都一样。

printf 函数把程序传给它的“ hello, world!!”字符串数据复制到一个内存缓冲区中，然后调用操作系统提供的 API（应用程序编程接口）函数。可能不同的实现之间有差异，但是大致动作如图 1-3 所示。

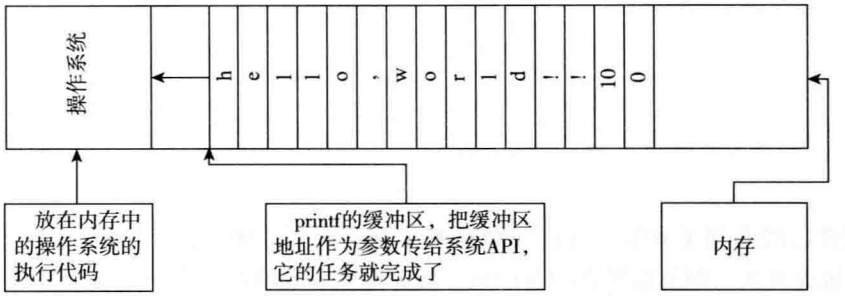


图 1-3 printf 函数的逻辑工作机理

一旦调用操作系统 API，代码的控制权就交给了操作系统，执行的也是操作系统的代码。至于实现这一机制的进程，后面的章节会详细讨论。操作系统会根据不同的 API 函数以及应用程序传递进来的参数完成不同的动作和功能。比如，程序中的 printf、调用的 API 和传递的参数，是要告诉操作系统把它缓冲区中的数据写入标准输出设备上。于是我们在屏幕上就看到了 hello, world!!，然后程序就逐层返回到 main 函数，最后 main 函数退出，这个小程序也就结束了。下面用图 1-4 来描述这一过程。

由图 1-4 可知，真正操控计算机硬件完成显示 hello, world!! 的是操作系统这个软件。是它在背后默默地工作着。当然它的功能并不只是操控计算机硬件，我们后面会慢慢讨论。

现在来看看没有操作系统能完成这个 hello, world!! 的程序吗？有！只是我们要做的工作多了很多，我们得亲自实现 printf 函数、完成控制计算机硬件的程序，并且还要求计算机内部这时只运行这一个应用程序，因为有些硬件同一时间只能被一个程序使用，如图 1-5 所示。

写上面这样的程序可能对程序员的要求有点高，因为必须要知道计算机硬件的每个细节，否则程序不会出现正常的结果。好吧，我们还是应该承认这个世界存在这种编程高手。

关于 hello, world!! 这个程序，我们就说到这里。如果没有操作系统，无论开发程序还是运行程序，难度都会很高。

现在去看看先辈是如何一步一步创造出操作系统的。

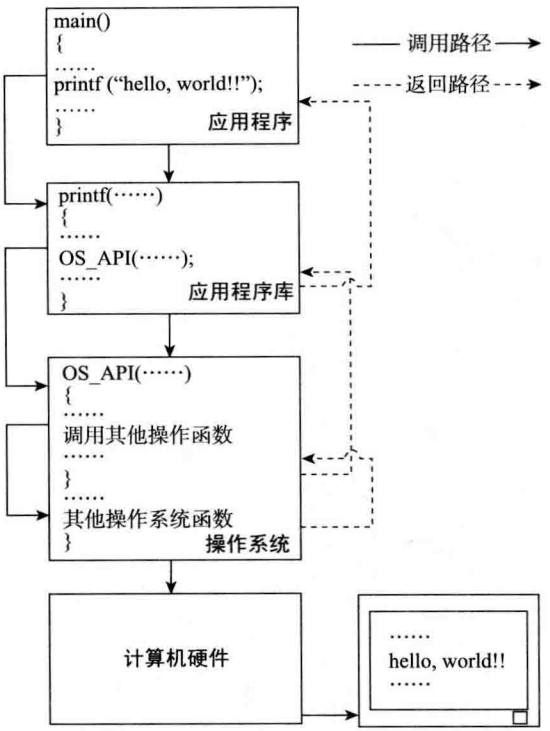


图 1-4　hello, world!! 程序调用操作系统 API 的过程

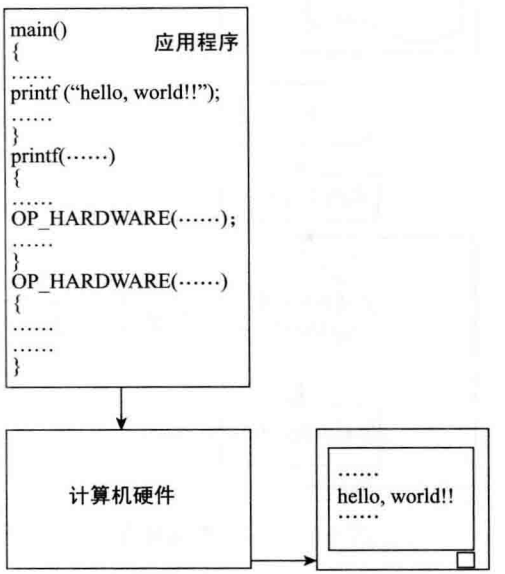


图 1-5　裸机 hello, world!! 程序

1.2　操作系统功能演进

在很久以前确实存在上述这种模式的程序，并且每次计算机中只存在一道这样的程序，运行完一道，手工装入第二道……你会发现这太慢了，也太麻烦了，对昂贵的计算机硬件的利用率也太低了，因为这样的程序是不可能让计算机内部所有的设备都工作起来的。

然而首先知道这个问题的不是我们，先辈早已看到这类问题，他们想了个方法，在计算机的内部放入一道监控程序，然后把要执行的所有程序放在一个固定的介质上，每次那个监控程序选择一道要运行的程序装入计算机内部去执行，这道程序执行完了，再装入下一道程序……直到运行完所有的程序。如图 1-6 所示。从全局看效率高了不少，但是从局部来看，它仍然没有让计算机的所有设备都工作起来。同时也还是没有降低程序员的难度，因为还是要操控程序要用到的硬件。

我们马上会想到，能不能把那个所谓的监控程序稍稍改进和扩展一下，把那些操作各种硬件的代码和实现重要功能的代码，放进监控程序里。然后我们对这些功能进行编号，比如，1 号功能是操作键盘的、2 号功能是打开硬盘上的文件……最后应用程序要完成什么功能，只要 call ×× 号功能，计算机的控制权就会交给监控程序，当监控程序中的代码完成这个功能号对应的服务时就返回到应用程序中。图 1-7 展示了这一过程。