



图书基本信息



内容概要

广泛的硬件支持、高效稳定的内核、开源共享的软件、优秀的开发工具、完善的网络通信和文件管理机制等特点，使嵌入式Linux获得了广泛应用，已成为嵌入式开发的主流平台。

本书是嵌入式Linux领域名著。

全面深入而又简明地阐述了构建嵌入式Linux系统的精髓。

书中不仅剖析了嵌入式Linux系统，而且讲述了处理器、内核、引导装入程序、设备驱动程序、文件系统等关键组件，介绍了嵌入式Linux系统的开发工具、调试技术。

作者多年积累总结的嵌入式Linux开发技巧和提示，无论对初学者还是有经验的开发人员，都弥足珍贵。

这一版不仅对原有章节进行了全面的修订、更新和改进，还新增了udev、USB和开源构建系统等内容。

。



作者简介

Christopher Hallinan 著名嵌入式Linux技术专家。
现任Mentor
Graphics公司技术市场工程师，曾任Monta
Vista软件公司现场应用工程师，3Com公司工程总监，Crosscomm公司工程总监。
他有25年以上网络和通信产品的软硬件开发经验。
曾担任
Linux咨询师，提供定制Linux主板接口、设备驱动程序和引导装入程序等方面的解决方案。



书籍目录

Chapter1 Introduction

- 1.1 Why Linux?
- 1.2 Embedded Linux Today
- 1.3 Open Source and the GPL
 - 1.3.1 Free Versus Freedom
- 1.4 Standards and Relevant Bodies
 - 1.4.1 Linux Standard Base
 - 1.4.2 Linux Foundation
 - 1.4.3 Carrier-Grade Linux
 - 1.4.4 Mobile Linux Initiative: Moblin
 - 1.4.5 Service Availability Forum
- 1.5 Summary
 - 1.5.1 Suggestions for Additional Reading

Chapter2 The Big Picture

- 2.1 Embedded or Not?
 - 2.1.1 BIOS Versus Bootloader
- 2.2 Anatomy of an Embedded System
 - 2.2.1 Typical Embedded Linux Setup
 - 2.2.2 Starting the Target Board
 - 2.2.3 Booting the Kernel
 - 2.2.4 Kernel Initialization: Overview
 - 2.2.5 First User Space Process: init
- 2.3 Storage Considerations
 - 2.3.1 Flash Memory
 - 2.3.2 NAND Flash
- 2.1.1 BIOS Versus Bootloader
- 2.2 Anatomy of an Embedded System
 - 2.2.1 Typical Embedded Linux Setup
 - 2.2.2 Starting the Target Board
 - 2.2.3 Booting the Kernel
 - 2.2.4 Kernel Initialization: Overview
 - 2.2.5 First User Space Process: init
- 2.3 Storage Considerations
 - 2.3.1 Flash Memory
 - 2.3.2 NAND Flash
 - 2.3.3 Flash Usage
 - 2.3.4 Flash File Systems
 - 2.3.5 Memory Space
 - 2.3.6 Execution Contexts
 - 2.3.7 Process Virtual Memory
 - 2.3.8 Cross-Development Environment
- 2.4 Embedded Linux Distributions
 - 2.4.1 Commercial Linux Distributions
 - 2.4.2 Do-It-Yourself Linux Distributions



2.5 Summary

2.5.1 Suggestions for Additional Reading

Chapter3 Processor Basics

3.1 Stand-Alone Processors

3.1.1 IBM FX

3.1.2 Intel Pentium M

3.1.3 Intel Atom?

3.1.4 Freescale MPC7448

3.1.5 Companion Chipsets

3.2 Integrated Processors: Systems on Chip

3.2.1 Power Architecture

3.2.2 Freescale Power Architecture

3.2.3 Freescale PowerQUICC I

3.2.4 Freescale PowerQUICC II

3.2.5 PowerQUICC II Pro

3.2.6 Freescale PowerQUICC III

3.2.7 Freescale QorIQ?

3.1.4 Freescale MPC7448

3.1.5 Companion Chipsets

3.2 Integrated Processors: Systems on Chip

3.2.1 Power Architecture

3.2.2 Freescale Power Architecture

3.2.3 Freescale PowerQUICC I

3.2.4 Freescale PowerQUICC II

3.2.5 PowerQUICC II Pro

3.2.6 Freescale PowerQUICC III

3.2.7 Freescale QorIQ?

3.2.8 AMCC Power Architecture

3.2.9 MIPS

3.2.10 Broadcom MIPS

3.2.11 Other MIPS

3.2.12 ARM

3.2.13 TI ARM

3.2.14 Freescale ARM

3.2.15 Other ARM Processors

3.3 Other Architectures

3.4 Hardware Platforms

3.4.1 CompactPCI

3.4.2 ATCA

3.5 Summary

3.5.1 Suggestions for Additional Reading

Chapter4 The Linux Kernel: A Different Perspective

4.1 Background

4.1.1 Kernel Versions

4.1.2 Kernel Source Repositories



- 4.1.3 Using git to Download a Kernel
- 4.2 Linux Kernel Construction
 - 4.2.1 Top-Level Source Directory
 - 4.2.2 Compiling the Kernel
 - 4.2.3 The Kernel Proper: vmlinux
 - 4.2.4 Kernel Image Components
 - 4.2.5 Subdirectory Layout
- 4.3 Kernel Build System
 - 4.1.1 Kernel Versions
 - 4.1.2 Kernel Source Repositories
 - 4.1.3 Using git to Download a Kernel
 - 4.2 Linux Kernel Construction
 - 4.2.1 Top-Level Source Directory
 - 4.2.2 Compiling the Kernel
 - 4.2.3 The Kernel Proper: vmlinux
 - 4.2.4 Kernel Image Components
 - 4.2.5 Subdirectory Layout
 - 4.3 Kernel Build System
 - 4.3.1 The Dot-Config
 - 4.3.2 Configuration Editor(s)
 - 4.3.3 Makefile Targets
 - 4.4 Kernel Configuration
 - 4.4.1 Custom Configuration Options
 - 4.4.2 Kernel Makefiles
 - 4.5 Kernel Documentation
 - 4.6 Obtaining a Custom Linux Kernel
 - 4.6.1 What Else Do I Need?
 - 4.7 Summary
 - 4.7.1 Suggestions for Additional Reading

Chapter5 Kernel Initialization

- 5.1 Composite Kernel Image: Piggy and Friends
 - 5.1.1 The Image Object
 - 5.1.2 Architecture Objects
 - 5.1.3 Bootstrap Loader
 - 5.1.4 Boot Messages
- 5.2 Initialization Flow of Control
 - 5.2.1 Kernel Entry Point: head.o
 - 5.2.2 Kernel Startup: main.c
 - 5.2.3 Architecture Setup
- 5.3 Kernel Command-Line Processing
 - 5.3.1 The __setup Macro
- 5.4 Subsystem Initialization
 - 5.4.1 The *__initcall Macros
- 5.5 The init Thread
- 5.2 Initialization Flow of Control
 - 5.2.1 Kernel Entry Point: head.o



- 5.2.2 Kernel Startup: main.c
- 5.2.3 Architecture Setup
- 5.3 Kernel Command-Line Processing
 - 5.3.1 The __setup Macro
- 5.4 Subsystem Initialization
 - 5.4.1 The *__initcall Macros
- 5.5 The init Thread
 - 5.5.1 Initialization Via initcalls
 - 5.5.2 initcall_debug
 - 5.5.3 Final Boot Steps
- 5.6 Summary
 - 5.6.1 Suggestions for Additional Reading

Chapter6 User Space Initialization

- 6.1 Root File System
 - 6.1.1 FHS: File System Hierarchy Standard
 - 6.1.2 File System Layout
 - 6.1.3 Minimal File System
 - 6.1.4 The Embedded Root FS Challenge
 - 6.1.5 Trial-and-Error Method
 - 6.1.6 Automated File System Build Tools
- 6.2 Kernel 's Last Boot Steps
 - 6.2.1 First User Space Program
 - 6.2.2 Resolving Dependencies
 - 6.2.3 Customized Initial Process
- 6.3 The init Process
 - 6.3.1 inittab
 - 6.3.2 Sample Web Server Startup Script
- 6.4 Initial RAM Disk
 - 6.4.1 Booting with initrd
 - 6.4.2 Bootloader Support for initrd
 - 6.4.3 initrd Magic: linuxrc
 - 6.4.4 The initrd Plumbing
 - 6.4.5 Building an initrd Image
- 6.5 Using initramfs
 - 6.5.1 Customizing
- 6.3.1 inittab
- 6.3.2 Sample Web Server Startup Script
- 6.4 Initial RAM Disk
 - 6.4.1 Booting with initrd
 - 6.4.2 Bootloader Support for initrd
 - 6.4.3 initrd Magic: linuxrc
 - 6.4.4 The initrd Plumbing
 - 6.4.5 Building an initrd Image
- 6.5 Using initramfs
 - 6.5.1 Customizing initramfs
- 6.6 Shutdown



6.7 Summary

6.7.1 Suggestions for Additional Reading

Chapter7 Bootloaders

7.1 Role of a Bootloader

7.2 Bootloader Challenges

7.2.1 DRAM Controller

7.2.2 Flash Versus RAM

7.2.3 Image Complexity

7.2.4 Execution Context

7.3 A Universal Bootloader: Das U-Boot

7.3.1 Obtaining U-Boot

7.3.2 Configuring U-Boot

7.3.3 U-Boot Monitor Commands

7.3.4 Network Operations

7.3.5 Storage Subsystems

7.3.6 Booting from Disk

7.4 Porting U-Boot

7.4.1 EP405 U-Boot Port

7.4.2 U-Boot Makefile Configuration Target

7.4.3 EP405 First Build

7.4.4 EP405 Processor Initialization

7.4.5 Board-Specific Initialization

7.4.6 Porting Summary

7.4.7 U-Boot Image Format

7.5 Device Tree Blob (Flat Device Tree)

7.3.6 Booting from Disk

7.4 Porting U-Boot

7.4.1 EP405 U-Boot Port

7.4.2 U-Boot Makefile Configuration Target

7.4.3 EP405 First Build

7.4.4 EP405 Processor Initialization

7.4.5 Board-Specific Initialization

7.4.6 Porting Summary

7.4.7 U-Boot Image Format

7.5 Device Tree Blob (Flat Device Tree)

7.5.1 Device Tree Source

7.5.2 Device Tree Compiler

7.5.3 Alternative Kernel Images Using DTB

7.6 Other Bootloaders

7.6.1 Lilo

7.6.2 GRUB

7.6.3 Still More Bootloaders

7.7 Summary

7.7.1 Suggestions for Additional Reading

Chapter8 Device Driver Basics



- 8.1 Device Driver Concepts
 - 8.1.1 Loadable Modules
 - 8.1.2 Device Driver Architecture
 - 8.1.3 Minimal Device Driver Example
 - 8.1.4 Module Build Infrastructure
 - 8.1.5 Installing a Device Driver
 - 8.1.6 Loading a Module
 - 8.1.7 Module Parameters
- 8.2 Module Utilities
 - 8.2.1 insmod
 - 8.2.2 lsmod
 - 8.2.3 modprobe
 - 8.2.4 depmod
 - 8.2.5 rmmod
 - 8.2.6 modinfo
- 8.3 Driver Methods
 - 8.1.5 Installing a Device Driver
 - 8.1.6 Loading a Module
 - 8.1.7 Module Parameters
- 8.2 Module Utilities
 - 8.2.1 insmod
 - 8.2.2 lsmod
 - 8.2.3 modprobe
 - 8.2.4 depmod
 - 8.2.5 rmmod
 - 8.2.6 modinfo
- 8.3 Driver Methods
 - 8.3.1 Driver File System Operations
 - 8.3.2 Allocation of Device Numbers
 - 8.3.3 Device Nodes and mknod
- 8.4 Bringing It All Together
- 8.5 Building Out-of-Tree Drivers
- 8.6 Device Drivers and the GPL
- 8.7 Summary
 - 8.7.1 Suggestions for Additional Reading

Chapter9 File Systems

- 9.1 Linux File System Concepts
 - 9.1.1 Partitions
- 9.2 ext2
 - 9.2.1 Mounting a File System
 - 9.2.2 Checking File System Integrity
- 9.3 ext3
- 9.4 ext4
- 9.5 ReiserFS
- 9.6 JFFS2
 - 9.6.1 Building a JFFS2 Image



- 9.7 cramfs
- 9.8 Network File System
 - 9.8.1 Root File System on NFS
- 9.9 Pseudo File Systems
 - 9.9.1 /proc File System
 - 9.9.2 sysfs
- 9.4 ext4
- 9.5 ReiserFS
- 9.6 JFFS2



章节摘录

版权页：插图：Nearly everyone is familiar with Compact Flash and SD cards used in a wide variety of consumer devices, such as digital cameras and PDAs (both great examples of embedded systems) . These modules, based on Flash memory technology, can be thought of as solid-state hard drives, capable of storing many megabytes—and even gigabytes-of data in a tiny footprint. They contain no moving parts, are relatively rugged, and operate on a single common power supply voltage. Several manufacturers of Flash memory exist. Flash memory comes in a variety of electrical formats, physical packages, and capacities. It is not uncommon to see embedded systems with as little as 4MB or 8MB of nonvolatile storage. More typical storage requirements for embedded Linux systems range from 16MB to 256MB or more. An increasing number of embedded Linux systems have nonvolat.



媒体关注与评论

“这本书很令我振奋。

它为那些想在嵌入式系统中使用Linux的开发人员提供了极好的学习路线指导。

本书内容简洁、准确，组织合理。

Christopher的知识和见解贯穿全书，你不仅能得到很多信息和帮助，也能享受到阅读的乐趣。

”——Arnold Robbins，著名Linux专家“本书涵盖了嵌入式Linux开发的方方面面……强烈推荐每一位嵌入式Linux开发人员阅读。

”——LinuxQuestions.org



版权说明

本站所提供下载的PDF图书仅提供预览和简介，请支持正版图书。

更多资源请访问:<http://www.tushu007.com>