

<<嵌入式Linux基础教程（第2版）>>

图书基本信息

书名：<<嵌入式Linux基础教程（第2版）>>

13位ISBN编号：9787115278272

10位ISBN编号：711527827X

出版时间：2012-5

出版单位：人民邮电出版社

作者：[美] Christopher Hallinan

页数：454

字数：697000

译者：周鹏

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>

## <<嵌入式Linux基础教程（第2版）>>

### 内容概要

《嵌入式Linux基础教程(第2版)》是嵌入式Linux的经典教程，介绍了引导加载程序、系统初始化、文件系统、闪存和内核、应用程序调试技巧等，还讲述了构建Linux系统的工作原理，用于驱动不同架构的配置，Linux内核源码树的特性，如何根据需求配制内核运行时的行为，如何扩展系统功能，用于构建完整嵌入式Linux发行版的常用构建系统，USB子系统和系统配置工具udev等内容。更重要的是，《嵌入式Linux基础教程(第2版)》阐述了如何修改系统使之满足读者自身的需求，确保读者能够从中学习一些嵌入式工程中非常有用的提示和技巧。

《嵌入式Linux基础教程(第2版)》适合Linux程序员阅读，也可作为高等院校相关专业师生的参考读物

作者简介

作者:(美)Hallinan

书籍目录

第1章 入门

- 1.1 为什么选择Linux
- 1.2 嵌入式Linux现状
- 1.3 开源和GPL
- 1.4 标准及相关组织
  - 1.4.1 Linux标准基础
  - 1.4.2 Linux基金会
  - 1.4.3 电信级Linux
  - 1.4.4 移动Linux计划：Moblin
  - 1.4.5 服务可用性论坛
- 1.5 小结

第2章 综述

- 2.1 嵌入与非嵌入
- 2.2 剖析嵌入式系统
  - 2.2.1 典型的嵌入式Linux开发环境
  - 2.2.2 启动目标板
  - 2.2.3 引导内核
  - 2.2.4 内核初始化：概述
  - 2.2.5 第一个用户空间进程：init
- 2.3 存储
  - 2.3.1 闪存
  - 2.3.2 NAND型闪存
  - 2.3.3 闪存的用途
  - 2.3.4 闪存文件系统
  - 2.3.5 内存空间
  - 2.3.6 执行上下文
  - 2.3.7 进程虚拟内存
  - 2.3.8 交叉开发环境
- 2.4 嵌入式Linux发行版
  - 2.4.1 商业Linux发行版
  - 2.4.2 打造自己的Linux发行版
- 2.5 小结

第3章 处理器基础

- 3.1 独立处理器
  - 3.1.1 IBM 970FX
  - 3.1.2 英特尔奔腾M
  - 3.1.3 英特尔凌动TM
  - 3.1.4 飞思卡尔MPC7448
  - 3.1.5 配套芯片组
- 3.2 集成处理器：片上系统
  - 3.2.1 Power架构
  - 3.2.2 飞思卡尔Power架构
  - 3.2.3 飞思卡尔PowerQUICC I
  - 3.2.4 飞思卡尔PowerQUICC II
  - 3.2.5 PowerQUICC II Pro

3.2.6 飞思卡尔PowerQUICC III

3.2.7 飞思卡尔QorIQTM

3.2.8 AMCC Power架构

3.2.9 MIPS

3.2.10 Broadcom MIPS

3.2.11 其他MIPS

3.2.12 ARM

3.2.13 德州仪器ARM

3.2.14 飞思卡尔ARM

3.2.15 其他ARM处理器

3.3 其他架构

3.4 硬件平台

3.4.1 CompactPCI

3.4.2 ATCA

3.5 小结

第4章 Linux内核：不同的视角

4.1 背景知识

4.1.1 内核版本

4.1.2 内核源码库

4.1.3 使用git下载内核代码

4.2 Linux内核的构造

4.2.1 顶层源码目录

4.2.2 编译内核

4.2.3 内核主体：vmlinux

4.2.4 内核镜像的组成部分

4.2.5 子目录的布局

4.3 内核构建系统

4.3.1 .config文件

4.3.2 配置编辑器

4.3.3 Makefile目标

4.4 内核配置

4.4.1 定制配置选项

4.4.2 内核Makefile

4.5 内核文档

4.6 获得定制的Linux内核

4.7 小结

第5章 内核初始化

5.1 合成内核镜像：Piggy及其他

5.1.1 Image对象

5.1.2 与具体架构相关的对象

5.1.3 启动加载程序

5.1.4 引导消息

5.2 初始化时的控制流

5.2.1 内核入口：head.o

5.2.2 内核启动：main.c

5.2.3 架构设置

5.3 内核命令行的处理

## 5.4 子系统初始化

### 5.5 init线程

#### 5.5.1 通过initcalls进行初始化

#### 5.5.2 initcall\_debug

#### 5.5.3 最后的引导步骤

### 5.6 小结

## 第6章 用户空间初始化

### 6.1 根文件系统

#### 6.1.1 FHS：文件系统层次结构标准

#### 6.1.2 文件系统布局

#### 6.1.3 最小化的文件系统

#### 6.1.4 嵌入式根文件系统带来的挑战

#### 6.1.5 试错法

#### 6.1.6 自动化文件系统构建工具

### 6.2 内核的最后一些引导步骤

#### 6.2.1 第一个用户空间程序

#### 6.2.2 解决依赖关系

#### 6.2.3 定制的初始进程

### 6.3 init进程

#### 6.3.1 inittab

#### 6.3.2 Web服务器启动脚本示例

### 6.4 初始RAM磁盘

#### 6.4.1 使用initrd进行引导

#### 6.4.2 引导加载程序对initrd的支持

#### 6.4.3 initrd的奥秘所在：linuxrc

#### 6.4.4 initrd探究

#### 6.4.5 构造initrd镜像

### 6.5 使用initramfs

### 6.6 关机

### 6.7 小结

## 第7章 引导加载程序

### 7.1 引导加载程序的作用

### 7.2 引导加载程序带来的挑战

#### 7.2.1 DRAM控制器

#### 7.2.2 闪存与RAM

#### 7.2.3 镜像的复杂性

#### 7.2.4 执行环境

### 7.3 通用引导加载程序：Das U-Boot

#### 7.3.1 获取U-Boot

#### 7.3.2 配置U-Boot

#### 7.3.3 U-Boot的监控命令

#### 7.3.4 网络操作

#### 7.3.5 存储子系统

#### 7.3.6 从磁盘引导

### 7.4 移植U-Boot

#### 7.4.1 EP405的U-Boot移植

#### 7.4.2 U-Boot Makefile中的配置目标

- 7.4.3 EP405的第一次构建
- 7.4.4 EP405 处理器初始化
- 7.4.5 与具体板卡相关的初始化
- 7.4.6 移植总结
- 7.4.7 U-Boot镜像格式
- 7.5 设备树对象 (扁平设备树)
- 7.5.1 设备树源码
- 7.5.2 设备树编译器
- 7.5.3 使用DTB的其他内核镜像
- 7.6 其他引导加载程序
- 7.6.1 Lilo
- 7.6.2 GRUB
- 7.6.3 其他更多的引导加载程序
- 7.7 小结
- 第8章 设备驱动程序基础
- 8.1 设备驱动程序的概念
- 8.1.1 可加载模块
- 8.1.2 设备驱动程序架构
- 8.1.3 最小设备驱动程序示例
- 8.1.4 模块构建的基础设施
- 8.1.5 安装设备驱动程序
- 8.1.6 加载模块
- 8.1.7 模块参数
- 8.2 模块工具
- 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 驱动程序方法
- 8.3.1 驱动程序中的文件系统操作
- 8.3.2 设备号的分配
- 8.3.3 设备节点和mknod
- 8.4 综合应用
- 8.5 在内核源码树外构建驱动
- 8.6 设备驱动程序和GPL
- 8.7 小结
- 第9章 文件系统
- 9.1 Linux文件系统概念
- 9.2 ext2
- 9.2.1 挂载文件系统
- 9.2.2 检查文件系统的完整性
- 9.3 ext3
- 9.4 ext4
- 9.5 ReiserFS
- 9.6 JFFS2

- 9.7 cramfs
- 9.8 网络文件系统
- 9.9 伪文件系统
  - 9.9.1 /proc文件系统
  - 9.9.2 sysfs
- 9.10 其他文件系统
- 9.11 创建简单的文件系统
- 9.12 小结
- 第10章 MTD子系统
  - 10.1 MTD概述
    - 10.1.1 开启MTD服务
    - 10.1.2 MTD基础
    - 10.1.3 在目标板上配置MTD
  - 10.2 MTD分区
    - 10.2.1 使用Redboot分区表进行分区
    - 10.2.2 使用内核命令行传递分区信息
    - 10.2.3 映射驱动
    - 10.2.4 闪存芯片驱动
    - 10.2.5 与具体板卡相关的初始化
  - 10.3 MTD工具
  - 10.4 UBI文件系统
    - 10.4.1 配置UBIFS
    - 10.4.2 构建UBIFS镜像
    - 10.4.3 使用UBIFS作为根文件系统
  - 10.5 小结
- 第11章 BusyBox
  - 11.1 BusyBox简介
  - 11.2 BusyBox的配置
  - 11.3 BusyBox的操作
    - 11.3.1 BusyBox的init
    - 11.3.2 rcS初始化脚本示例
    - 11.3.3 BusyBox在目标板上的安装
    - 11.3.4 BusyBox小应用
  - 11.4 小结
- 第12章 嵌入式开发环境
  - 12.1 交叉开发环境
  - 12.2 对主机系统的要求
  - 12.3 为目标板提供服务
    - 12.3.1 TFTP服务器
    - 12.3.2 BOOTP/DHCP 服务器
    - 12.3.3 NFS服务器
    - 12.3.4 目标板使用NFS挂载根文件系统
    - 12.3.5 U-Boot中使用NFS挂载根文件系统的例子
  - 12.4 小结
- 第13章 开发工具
  - 13.1 GNU调试器 (GDB)
    - 13.1.1 调试核心转储

13.1.2 执行GDB

13.1.3 GDB中的调试会话

13.2 数据显示调试器

13.3 cbrowser/cscope

13.4 追踪和性能评测工具

13.4.1 strace

13.4.2 strace命令行选项

13.4.3 ltrace

13.4.4 ps

13.4.5 top

13.4.6 mtrace

13.4.7 dmalloc

13.4.8 内核oops

13.5 二进制工具

13.5.1 readelf

13.5.2 使用readelf查看调试信息

13.5.3 objdump

13.5.4 objcopy

13.6 其他二进制实用程序

13.6.1 strip

13.6.2 addr2line

13.6.3 strings

13.6.4 ldd

13.6.5 nm

13.6.6 prelink

13.7 小结

第14章 内核调试技术

14.1 内核调试带来的挑战

14.2 使用KGDB进行内核调试

14.2.1 KGDB的内核配置

14.2.2 在开启KGDB时引导目标板

14.2.3 一些有用的内核断点

14.2.4 KGDB与控制台共享一个串行端口

14.2.5 调试非常早期的内核代码

14.2.6 主线内核对KGDB的支持

14.3 内核调试技术

14.3.1 gdb远程串行协议

14.3.2 调试优化的内核代码

14.3.3 GDB的用户自定义命令

14.3.4 有用的内核GDB宏

14.3.5 调试可加载模块

14.3.6 printk调试

14.3.7 Magic SysReq key

14.4 硬件辅助调试

14.4.1 使用JTAG探测器对闪存进行编程

14.4.2 使用JTAG探测器进行调试

14.5 不能启动的情况

14.5.1 早期的串行端口调试输出

14.5.2 转储printk的日志缓冲区

14.5.3 使用KGDB调试内核异常

14.6 小结

第15章 调试嵌入式Linux应用程序

15.1 目标调试

15.2 远程(交叉)调试

15.3 调试共享程序库

15.4 调试多个任务

15.4.1 调试多个进程

15.4.2 调试多线程应用程序

15.4.3 调试引导加载程序/闪存代码

15.5 其他远程调试选项

15.5.1 使用串行端口进行调试

15.5.2 附着到运行的进程上

15.6 小结

第16章 开源构建系统

16.1 为什么使用构建系统

16.2 Scratchbox

16.2.1 安装Scratchbox

16.2.2 创建一个交叉编译目标

16.3 Buildroot

16.3.1 安装Buildroot

16.3.2 配置Buildroot

16.3.3 构建Buildroot

16.4 OpenEmbedded

16.4.1 OpenEmbedded的组成

16.4.2 BitBake元数据

16.4.3 配方基础

16.4.4 任务

16.4.5 类

16.4.6 配置元数据

16.4.7 构建镜像

16.5 小结

第17章 实时Linux

17.1 什么是实时

17.1.1 软实时

17.1.2 硬实时

17.1.3 Linux调度

17.1.4 延时

17.2 内核抢占

17.2.1 抢占的障碍

17.2.2 抢占模式

17.2.3 SMP内核

17.2.4 抢占延时的根源

17.3 实时内核补丁

17.3.1 实时补丁的特性

- 17.3.2 O(1)调度器
- 17.3.3 创建实时进程
- 17.4 实时内核的性能分析
  - 17.4.1 使用Ftrace追踪内核行为
  - 17.4.2 检测抢占被关闭的延时
  - 17.4.3 检测唤醒延时
  - 17.4.4 检测中断被关闭的延时
  - 17.4.5 检测Soft Lockup
- 17.5 小结
- 第18章 通用串行总线
  - 18.1 USB概述
    - 18.1.1 USB的物理拓扑结构
    - 18.1.2 USB的逻辑拓扑结构
    - 18.1.3 USB版本
    - 18.1.4 USB连接器
    - 18.1.5 USB线缆
    - 18.1.6 USB模式
  - 18.2 配置USB
  - 18.3 sysfs和USB设备命名
  - 18.4 实用的USB工具
    - 18.4.1 USB文件系统
    - 18.4.2 使用usbview
    - 18.4.3 USB 实用程序(lsubst)
  - 18.5 通用USB子系统
    - 18.5.1 USB大容量存储类
    - 18.5.2 USB HID类
    - 18.5.3 USB CDC类驱动
    - 18.5.4 USB网络支持
  - 18.6 USB调试
    - 18.6.1 usbmon
    - 18.6.2 实用USB杂记
  - 18.7 小结
- 第19章 udev
  - 19.1 什么是udev
  - 19.2 设备发现
  - 19.3 udev的默认行为
  - 19.4 理解udev规则
    - 19.4.1 Modalias
    - 19.4.2 典型的udev规则配置
    - 19.4.3 udev的初始系统设置
  - 19.5 加载平台设备驱动程序
  - 19.6 定制udev的行为
  - 19.7 持久的设备命名
  - 19.8 udev和busybox配合使用
    - 19.8.1 busybox mdev
    - 19.8.2 配置mdev
  - 19.9 小结

附录A 可配置的U-Boot命令

附录B BusyBox命令

附录C SDRAM接口注意事项

附录D 开源资源

附录E 简单的BDI-2000配置文件

## 章节摘录

版权页：插图：在交叉开发环境中开发软件要求编译器运行于开发主机上，但生成的二进制可执行文件的格式与开发主机不兼容，不能在上面运行。

这类工具存在的主要原因是，在资源（一般指内存大小和CPU性能）受限的嵌入式系统上本地开发和编译代码常常是不现实或不可能的。

这种开发方式隐藏着很多陷阱，嵌入式开发的新手稍不留神就会中招。

当编译一个程序时，编译器一般都知道怎样找到所需要的头文件和正确编译代码必需的程序库。

为了说明这些概念，我们再看一下“Hello world”。

代码清单2—4qh的示例代码是使用下面的命令行进行编译的：`gcc -Wall -o hello hello.c` 在代码清单2.4中，我们看到这个程序代码包含了一个头文件`stdio.h`。

这个文件和我们在`gcc`命令行中指定的文件`hello.C`不在同一个目录中。

那么，编译器是如何找到它的呢？

另外，函数`printf()`也不是在文件`hello.C`中定义的。

因此，编译`hello.C`后，它会包含一个对此符号的未解析的引用（`unresolved reference`）。

链接器在链接时是怎样解析这个引用的呢？

编译器使用一些默认的路径来定位头文件。

在代码中引用某个头文件时，编译器在默认的几个搜索路径中查找这个文件。

类似地，链接器也是以这种方式来解析对外部符号`printf()`的引用。

链接器知道默认在C库（`libc.a`）中搜索未解析的引用，并且知道在系统中的哪些位置可以找到这些程序库。

再说明一下，这种默认行为是内置于工具链中的。

现在假设你为某个采用Power架构的嵌入式系统编写应用程序。

显然，你需要一个交叉编译器，用于生成兼容Power架构处理器的二进制可执行文件。

如果你使用交叉编译器，并采用类似的编译命令来编译前面的`hello.C`程序，在解析对外部符号`printf()`的引用时，链接器很可能会意外地将二进制可执行文件链接到一个x86版本的c库。

当然，由于生成的可执行程序混合了Power架构和x86二进制指令，如果运行这个错误的混合物，其结果是可以预见的，那就是系统崩溃！

摆脱这个困境的方法是指引交叉编译器在非标准路径中进行查找，以使用针对目标架构的头文件和程序库。

我们将在第12章中详细讨论这个主题。

这个例子旨在说明两种开发环境的区别，即本地开发环境和嵌入式系统所需的交叉编译开发环境。

这只是交叉开发环境复杂性的一个方面。

交叉调试中也会出现相同的问题和解决方案，从第14章开始，你会了解到这些内容。

正确地搭建交叉开发环境对于成功至关重要，你将在第12章中看到，这不仅仅涉及编译器，还包括其他很多内容。

2.4嵌入式Linux发行版 到底什么是Linux发行版？

Linux内核完成系统引导后，它会找到并挂载一个根文件系统。

一旦合适的根文件系统被成功挂载，启动脚本会启动很多系统需要的程序和实用工具。

编辑推荐

嵌入式Linux权威著作Amazon全五星评价全面剖析构建嵌入式Linux开发，揭示大量技术内幕

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>