北极光

苏嵌教育

嵌入式技术与应用丛书

Linux驱动入门

主　编　魏　清
副主编　梁　庚　徐志国

◎ 按照初学者的思路来讲述Linux驱动
◎ 结合Linux内核源码的分析来进行Linux驱动的开发
◎ 立足实践，体现苏嵌教育多年的经验

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

嵌入式技术与应用丛书

Linux 驱动入门

主编 魏 清

副主编 梁 庚 徐志国

電子工業出版社
Publishing House of Electronics Industry
北京 · BEIJING

内 容 简 介

本书主要是从Linux内核、Linux总线和Linux子系统三个角度对Linux驱动进行介绍的，然后对字符设备、块设备和网络设备也分别做了大致介绍。从内容上来讲，本书包括4个部分：Linux内核部分（第1～6章），主要包括Linux进程调度与进程管理、中断机制、定时机制、并发与同步机制及内存管理，通过对本篇内容的学习，读者可以掌握Linux内核的基本概念；Linux总线部分（第7～13章），主要包括Platfrom总线、单总线、I2C总线、串口总线、PCI总线、SPI总线和USB总线，通过对本篇内容的学习，读者可以掌握设备是如何挂载到Linux内核总线上的；Linux子系统部分（第14～19章），主要包括Keyboard子系统，LED子系统、RTC子系统、Input子系统、Backlight子系统、Hwmon子系统，通过对本篇内容的学习，读者可以掌握如何使用内核中现有的子系统，给设备编写驱动；Linux驱动部分（第20～24章），主要包括看门狗驱动、LCD驱动、触摸屏驱动、块设备驱动和网络设备驱动，通过对本篇内容的学习，读者可以掌握具体的设备驱动程序的设计方法。

本书适合嵌入式驱动开发的初学者学习使用，也可作为相关专业的教材。

图书在版编目（CIP）数据

Linux驱动入门 / 魏清主编. — 北京：电子工业出版社，2014.3
（嵌入式技术与应用丛书）
ISBN 978-7-121-22461-4

I. ①L… II. ①魏… III. ①Linux操作系统 IV. ①TP316.89

中国版本图书馆CIP数据核字（2014）第025904号

责任编辑：田宏峰　　特约编辑：牛雪峰
印　　刷：涿州市京南印刷厂
装　　订：涿州市京南印刷厂
出版发行：电子工业出版社
　　北京市海淀区万寿路173信箱　邮编　100036
开　　本：787×1 092　1/16　印张：20　字数：510千字
印　　次：2014年3月第1次印刷
印　　数：4 000册　　定价：49.00元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至zlts@phei.com.cn，盗版侵权举报请发邮件至dbqq@phei.com.cn。

服务热线：（010）88258888。

前　言

目前，国内图书市场上驱动开发书籍虽多，但大多都脱离了 Linux 的内核资源，单一地介绍设备驱动。而作为一个合格的驱动工程师，应该能站在全局的角度，充分利用 Linux 内核中现有的关于总线和子系统的驱动代码，来实现具体设备的驱动。

本书从嵌入式底层驱动开发角度出发，结合 Linux 内核源码分析，将隐藏在嵌入式驱动开发背后的，关于 Linux 内核中的进程管理、内存管理，以及各类驱动等方面的机制和知识娓娓道来。不仅让读者知其然，更要让读者知其所以然，并让这些知识再反作用于实践，从而帮助读者掌握相应设备的驱动。具体说来，全书主要讨论了包括 Linux 内核的基础知识、Linux 内核中进程、内存、中断管理的实现，各类驱动机制的框架实现，字符设备、块设备、网络设备驱动的实现等多个方面的话题。为了读者真正理解这些理论，本书结合每一个总线和子系统相关的驱动代码，使用友善之臂的 Mini2440 开发板，对所讲的总线和子系统代码进行了实际应用，使读者能够真正掌握和使用 Linux 内核中现有的驱动资源。

嵌入式驱动学习一直是所有初学者的高门槛，找到一本合适的参考教材往往非常困难。我在苏嵌的学习期间就已经在网络上分享和交流驱动学习经验，正是在 CSDN 论坛上与无数初学者的沟通和交流，才萌生了给驱动初学者创作本书的念头。在即将走上工作岗位之际，结合自己学成后的项目实践经验，最终成稿。本书的出版也得到了苏嵌教育驱动开发组老师的大力支持，苏嵌教育经过六年的沉淀，嵌入式硬件培训一直在行业内独树一帜，在驱动领域更是总结了很多培训成果，这些成果出于知识产权保护的原因，并未向大众开放。参与本书的作者大多数人都是苏嵌学生和老师中的杰出代表，他们全面总结自身嵌入式驱动学习经验，也第一次面向大众解密了苏嵌驱动开发教学中的许多特色细节。

在多年的项目开发过程中，我发现，要想成为一个嵌入式驱动开发工程师，需要“内外兼修”。内功就是熟悉 Linux 内核源码，对内核中的管理机制一定要很熟悉，并且要有很好的硬件基础。这里的基础不仅是所谓的模电和数电，而且要对 CPU 及其外围设备的时序及工作原理很精通，这是决定你是否能成为合格的驱动开发工程师的根本因素。而外功就是精通 C 语言、数据结构，至少 5 万行代码量，掌握 Linux 上层调用机制，深入理解 Linux 操作系统。对于在校的大学生，在校学习的课程相对独立，缺乏系统性，即使学习 Linux 操作系统，在大学四年都无法接触到 Linux 内核，只是具备少量的“外功”而已，更不要提毕业之后从事 Linux 驱动开发了。

大学生应如何修炼自己“内外功”呢？我们的建议是先从“外功”开始修炼。在大学低年级时先熟悉 C 语言、数据结构以及 Linux 上层的调用机制，熟悉 Linux 操作系统，日常开发在 Linux 平台下进行，并尝试的去看 Linux 内核源码，看不懂没关系，但至少能够对内核中所用到的一些概念及专业名词有所了解，为以后打下基础。大学三四年级就可以专心修炼“内功”了，结合大学所开设的专业课，弥补自己体系知识上的不足。最重要的就是要实践，动手写驱动，借助资料，一步一步地剖析 Linux 内核源码。这样度过大学四年的话，Linux 驱动工程师的梦想就不再是遥不可及了。最后，还要提醒大家在学习的过程中，一定要对自

己学习的东西多做总结，写学习和开发心得，这样能加快您前进的步伐。在苏嵌培训过的不少学员都进入了不错的研发企业，普通二本、三本的学生拿到了年薪 10 万元待遇的人很多。

感谢我的父母、老师和朋友一直以来对我的关心和帮助。参与本书编写的还有张成、李赛、闫坤等。本书汇集了作者的学习心得，对于想学习驱动的同学，无疑是一本高效的入门指南，期待更多如此的佳作问世！

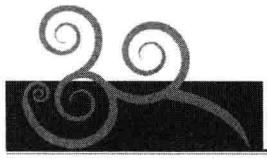
鉴于时间仓促，作者水平有限，书中难免有错误和不足之处，希望广大读者批评指正。联系 E-mail: training@jestc.com，并已开通 QQ 技术讨论群：1780328730。

作　者

2014 年 1 月于金陵

反侵权盗版声明

目　录



第 1 章

内 核 基 础

1.1 Linux 内核组成

Linux 内核主要是由进程调度、内存管理、虚拟文件系统、网络接口和进程通信五个子系统组成的。

（1）进程调度控制系统中的多个进程对 CPU 的访问，使得多个进程能在 CPU 中“微观串行，宏观并行”地执行。

（2）内存管理的主要作用是控制多个进程安全地共享主内存区域，当 CPU 提供内存管理单元时，Linux 内存管理完成为每个进程进行虚拟内存到物理内存的转换。一般而言，Linux 的每一个进程享有 4 GB 的内存空间，0～3 GB 为用户空间，3～4 GB 为内核空间，这 1 GB 的内核空间又被划分为物理内存映射区、虚拟内存分配区、高端页面映射区和系统保留映射区。物理内存映射区最大长度为 896 MB，系统物理内存 0～896 MB 就映射到这个物理内存映射区，系统物理内存中大于 896 MB 的数据属于高端内存，会被映射到高端页面映射区。由此可见，在 3～4 GB 的内核空间中，从低地址到高地址依次为：物理内存映射区、隔离带、虚拟内存分配区、隔离带、高端内存映射区、专用页面映射区和保留区。

为了了解内存管理单元 MMU，我们还需要知道 TLB（块表）和 TTW（转换表漫游，又叫做慢表，当 TLB 中没有缓冲对应的地址转换关系时，需要通过多级页表的访问来获得虚拟地址和物理地址的对应关系，TTW 成功后，结果写入 TLB）。

（3）虚拟文件系统隐藏各种硬件的具体细节，为所有的设备提供统一的接口。

（4）网络接口提供了对各种网络标准的存取和各种网络硬件的支持。

（5）进程通信支持提供进程之间的通信，包括信号量、共享内存、管道等。

Linux 内核五个部分的关系如下所述。

- 进程调度和内存管理之间的关系：相互依赖，程序要运行必须为其创建进程，而创建进程的第一件事就是将程序和数据装入内核。
- 进程通信与内存管理的关系：进程间通信需要依靠内存管理支持共享内存机制。
- 虚拟文件系统和网络接口的关系：虚拟文件系统利用网络接口支持网络文件系统，也利用内存管理支持 RAMDISK 设备。
- 虚拟文件系统与内存管理的关系：内存管理利用虚拟文件系统支持交换，当一个进程存储的内存映射被换出时，内存管理向文件系统发出请求，同时挂起当前正在运行的进程。

1.2 Linux 内核的引导

1. Boot Loader 简介

在 CPU 上电启动时，一般连内存控制器都没有初始化过，根本无法在主存中运行程序，更不可能处在 Linux 内核启动环境中。为了初始化 CPU 及其他外设，使得 Linux 内核可以在系统主存中运行，并让系统符合 Linux 内核启动的必备条件，必须要有一个先于内核运行的程序，即所谓的引导加载程序 Boot Loader，Boot Loader 是在操作系统内核启动之前运行的一段小程序。通过这段程序，系统可以初始化硬件设备，从而将系统的软/硬件环境带到一个合适的状态，以便为最终调用操作系统内核准备好正确的环境，最后从别处（Flash、以太网、UART）载入内核、映像到主存并跳到入口地址。U-Boot 是 Das U-Boot 的简称，其含义是 Universal Boot Loader，是遵循 GPL 条款的开放源码项目。Boot Loader 的操作方式包括启动加载模式和下载模式，我们开发的时候使用下载模式。

2. Boot Loader 的主要任务

依赖于 CPU 体系结构的代码，如设备初始化代码等，通常都放在 stage1 中，而且通常都用汇编语言来实现，以达到短小精悍的目的。而 stage2 则通常用 C 语言来实现，这样可以实现复杂的功能，而且代码会具有更好的可读性和可移植性。

3. Boot Loader 的 stage1 通常包括以下步骤（按执行的先后顺序）

（1）硬件设备初始化。

（2）为加载 Boot Loader 的 stage2 准备 RAM 空间。

（3）拷贝 Boot Loader 的 stage2 到 RAM 空间中。

（4）设置好堆栈。

（5）跳转到 stage2 的 C 入口点。

4. Boot Loader 的 stage2 通常包括以下步骤（按执行的先后顺序）

（1）初始化本阶段要使用到的硬件设备。

（2）检测系统内存映射。

（3）将 Kernel 映像和根文件系统映像从 Flash 上读到 RAM 空间中。

（4）为内核设置启动参数。

（5）调用内核。

嵌入式系统在复位后就直接运行 Boot Loader，当 Boot Loader 的控制权被释放后，内核阶段就开始了，内核在进行一些初始化操作之后，就调用“/init/main.c”中的 start_kernel 函数，该函数会调用一系列初始化函数来设置中断，执行进一步的内存配置。之后，“/arch/i386/kernel/process.c”中 kernel_thread 被调用以启动第一个核心线程，该线程执行 init

函数，作为核心线程的 init 函数完成外设及其驱动程序的加载和初始化，挂接根文件系统，搜索 init 程序的顺序是“/sbin/init”、“/etc/init”、“/bin/init”和“/bin/sh”。

5. U-boot 烧写过程

使用 U-boot 将映像文件烧写到板上的 Flash，一般步骤是：

（1）通过网络、串口、U 盘、SD 卡等方式将文件传输到 SDRAM。

（2）使用 Nand Flash 或 Nor Flash 相关的读写命令将 SDRAM 中的数据烧入 Flash。

注意：如果使用 SD 卡和 U 盘形式更新 U-boot，那么首先 SD 卡和 U 盘中必须有 FAT32 文件系统，并在里面存放了 u-boot.bin 文件。

举例：通过 NFS 服务烧入 Nand Flash。

```
nfs 3000800 192.168.1.100:/home/tekkaman/development/share/u-boot.bin
nand erase 0 0x40000
nand write 0x30008000 0 0x40000
```

6. 内核的引导过程

内核的引导步骤如下。

（1）用 U-boot 的 mkimage 工具处理内核映像 zImage。

（2）通过网络、串口、U 盘、SD 卡等方式将处理过的内核映像传输到 SDRAM 的一定位置（一般使用 0x30008000）。

（3）使用“bootm”等内核引导命令来启动内核。

通过 Nand Flash 引导内核。首先要将处理过的内核映像文件烧入 Nand Flash 的一定位置（由内核分区表决定）。以后每次启动时用 Nand Flash 的读取命令先将这个内核映像文件读到内存的一定位置（由制作内核映像时的“-a”参数决定），再使用 bootm 命令引导内核。

举例：通过 NFS 服务引导内核。

```
nfs 30008000 192.168.1.100:/home/tekkaman/development/share/zImage.img
bootm 30008000
```

1.3 处　理　器

中央处理器体系架构可以分为两类：冯·诺依曼结构和哈佛结构。冯·诺依曼结构把程序和操作数都放在同一个存储器中，这个存储器通过总线与处理器相连，而哈佛结构是把程序放在程序存储器内，通过程序总线与处理器相连，然后把操作数放在操作数存储器中，通过操作数总线与处理器相连。

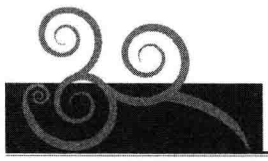
中央处理器从指令的角度也可以分为 RISC（精简指令集计算机）和 CISC（复杂指令集计算机）。中央处理器按应用领域可以分为通用处理器（GPP）、数字信号处理器（DSP）和专用处理器及 ASIC。其中 GPP 包括 MCU（微控制器，又叫做单片机）和 MPU（微处理器）；DSP 包括定点 DSP 和浮点 DSP。

1.4 存 储 器

存储器分为非易失性存储器（NVM）和 RAM，其中 NVM 包括 ROM、Flash 和光磁介质存储器；RAM 包括 SRAM、DRAM 和特定应用的 RAM。

NOR Flash：Intel 公司，程序可以直接在 NOR 内运行。

NAND Flash：东芝公司，NAND 以块方式进行访问，不支持芯片内执行，容量大、价格低，擦写次数和速度，以及编程速度远超 NOR。但是会出错，应用 ECC 检查错误。



第2章 进程管理

2.1 进程调度

多任务系统可以分为非抢占式和抢占式两种。Linux 提供抢占式多任务模式，进程在被抢占之前能够运行的时间叫进程的时间片，Linux 独一无二的公平调度程序本身并没有采用时间片来达到公平调度。

Linux 之前采用 $O(1)$调度器，它对大服务器的工作负载很理想，但是对响应时间敏感的程序却有不足。在 Linux 2.6.23 内核版本中用完全公平调度算法（CFS）代替了 $O(1)$调度算法。

进程可以被分为 I/O 消耗型和处理器消耗型。I/O 消耗型指进程的大部分时间用来提交 I/O 请求或是等待 I/O 请求；处理器消耗型是指进程把事件大多数用在执行代码上。调度策略通常在两个矛盾中寻找平衡：进程响应迅速和最大系统利用率。Linux 更倾向于优先调度 I/O 消耗型进程。

调度程序总是选择时间片未用尽而且优先级最高的进程运行。

Linux 采用两种不同的优先级范围：第一种是 nice 值，越大的 nice 值意味着更低的优先级；第二种是实时优先级，其值可以配置，越高的实时优先级数值意味着进程优先级越高。任何实时进程的优先级都高于普通进程。

CFS 做法：允许每个进程运行一段时间，循环轮转，选择运行最少的进程作为下一个运行进程，而不再采用分配给每一个进程时间片的做法，CFS 在所有可运行进程总数基础上计算出一个进程应该运行多久，而不是依靠 nice 值来计算时间片。CFS 引入每个进程获得的时间片底线（最小粒度）默认情况为 1 ms，绝对的 nice 值不再影响调度决策，任何进程所获得的处理器时间由它自己和其他所有可运行进程 nice 值的相对差决定。

2.2 Linux 调度的实现

（1）时间记账。调度器实体结构是 struct sched_entity，它嵌入在进程描述符 struct task_struct 中。虚拟时间以 ns 为单位，存在 vruntime 变量中，它和定时器节拍不再相关，vruntime 变量可以准确测定进程的运行时间，而且可以知道谁应该是下一个被运行的进程。

（2）进程选择。当 CFS 需要选择下一个运行进程时，它会挑一个具有最小 vruntime 的进程，这就是 CFS 调度算法的核心：选择具有最小的 vruntime 的任务。这里采用红黑树实现的。

（3）调度器入口。进程调度的主要入口点是 schedule 函数，该函数会以优先级为序，从高到低，依次检查每一个调度器，并且从最高优先级的调度器中，选择最高优先级的进程。

（4）睡眠和唤醒。对于睡眠，内核的操作是：进程把自己标记为休眠状态，从可执行红黑树中移出，放入等待队列，然后调用 schedule 函数选择和执行一个其他进程。对于唤醒，正好相反，进程被设置为可执行状态，然后从等待队列中移到可执行红黑树中。

休眠有两种进程状态：TASK_INTERRUPTIBLE 和 TASK_UNINTERRUPTIBLE。它们唯一的区别在于处在 TASK_UNINTERRUPTIBLE 的进程会忽略信号，处于 TASK_INTERRUPTIBLE 状态的进程如果接收到一个信号，会被提前唤醒并响应该信号。两种状态的进程位于同一个等待队列上，等待某些事情，不能够运行。

关于休眠还需要注意一点：存在虚假的唤醒，所以有时候需要一个循环处理来保证进程被唤醒的条件真正大达成。等待队列的使用：

```
static DECLARE_WAIT_QUEUE_HEAD(button_waitq);          //定义等待队列头
wait_event_interruptible(button_waitq, ev_press);      //休眠
wake_up_interruptible(&button_waitq);                  //唤醒
```

2.3 抢占和上下文切换

上下文切换就是从一个可执行进程切换到另一个可执行进程。Schedule 函数主要完成两个工作：其一，把虚拟内存从上一个进程映射切换到新进程中；其二，从上一个进程的处理器状态切换到新进程的处理器状态。

内核必须知道在什么时候调用 schedule 函数，如何仅靠用户程序代码显式地调用 schedule，它们可能就会被永远执行下去，所以内核中提供了一个 need_resched 标志来表明是否需要重新执行一次调度，该标志对于内核来讲是一个信息，它表示有其他进程应当被运行了，要尽快调用调度程序。

用户抢占：内核即将返回用户空间的时候，如果 need_resched 标志被设置，会导致 schedule 函数被调用，此时就会发生用户抢占。用户抢占在如下情况时产生：其一，从系统调用返回用户空间时；其二，从中断处理程序返回用户空间时。

内核抢占：在 Linux 2.6 版内核中，内核引入抢占能力，只要重新调度是安全的，即只要没有持有锁，内核就可以在任何时候抢占正在执行的任务。为了支持内核抢占，在每个进程的 thread_info 引入 preempt_count 计数器，当做一把锁来使用。内核抢占在如下情况时产生。

- 中断处理程序正在执行，且返回用户空间之前。
- 内核代码再一次具有可抢占性的时候。
- 如果内核中的任务显式地调用 schedule 函数。
- 如果内核中的任务阻塞（这样会调用 schedule 函数）。

2.4 进 程 概 念

进程是正在执行的程序代码的实时结果，是处于执行期的程序以及相关的资源的总称。线程是在进程中活动的对象，内核调度的对象是线程，而不是进程。对于 Linux 系统而言，