拨云见日

基于Android的内核与系统架构源码分析

王森◎著

清华大学出版社

拨云见日

基于Android的内核与系统架构源码分析

王森◎著

清华大学出版社
北京

内 容 简 介

本书包括上下两篇内容。上篇在保证完整 Linux 内核架构分析的前提下，着重分析 Android 系统中强烈依赖的 Linux 内核机制，如多核 ARM 架构的支持，而略去 Android 系统产品化没有用到内核机制，如 SWAP 机制。下篇主要分析 Android 系统层主要架构机制，尤其注重分析这些用户态机制与内核机制的接驳与交互。本书整理自作者多年积累的笔记，形式以源代码分析为主。

本书适合相关领域工程师作为实际项目的参考，以及有志于通过研读源码掌握 Android 系统与 Linux 内核精髓的读者。

图书在版编目（CIP）数据

拨云见日——基于 Android 的内核与系统架构源码分析 / 王森　著. —北京：清华大学出版社，2015
ISBN 978-7-302-38199-0

Ⅰ. ①拨…　Ⅱ. ①王…　Ⅲ. ①移动终端-应用程序-程序设计　Ⅳ. ①TN929.53

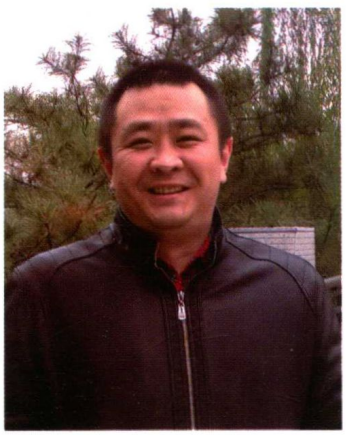
中国版本图书馆 CIP 数据核字（2014）第 230612 号

责任编辑：夏兆彦
封面设计：胡文航
责任校对：徐俊伟
责任印制：杨　艳

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　邮　　编：100084
社 总 机：010-62770175　　邮　　购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：清华大学印刷厂
装 订 者：三河市新茂装订有限公司
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张**：27　　**字　　数**：668 千字
版　　次：2015 年 1 月第 1 版　　**印　　次**：2015 年 1 月第1 次印刷
印　　数：1～3500
定　　价：69.00 元

产品编号：056459-01

作者简介



王森，1978年出生在安徽北部小城，现工作生活于北京。醉爱内核与系统，也以此谋生。十余年纵情于源码之间，初能心游于编码之上。生活中，虽为宅男一枚，但是性格外向、广交朋友。

本书推荐语

对于像 Android、Linux 这样复杂的大型软件，泛泛了解一下工作原理并不难，看看文档，或者随便找几本书看看，再实际用一下体验体验，也就差不多了。

但是若想要有比较深入透彻的理解，那就非得要阅读分析其源代码不可，这就比较难了。这时候，如果有人先行一步，下了苦功先把它弄懂，再把所获的知识和心得分享出来，那就是很有价值了。我觉得王森这本书就是这样，里面不光有代码的分析，还有他的见解，特别是还有他的一些经验之谈，相信读者会和我一样看了后觉得受益匪浅。

——《Linux 内核源代码情景分析》作者 毛德操

Android 系统几年来高速发展，不过国内工程师使用者居多，深入了解者少。本书深入浅出，将作者多年实践经验结合系统深层探索呈现给大家，可谓雪中送炭、广大 Android 工程师的福音。感谢本书作者的分享，相信本书会成为工程师们进阶修炼的极大助力。

——北京时代飞腾科技有限公司 技术总监 刘天宇

本书内容风趣幽默且比喻贴切，“Android 与 Linux 内核的关系，就好比一辆整车和底盘发动机的关系”；该书内容详实而又丰富，既有 MTK 平台的知识，又有三星平台的细节，既有 Linux 基于 ARM 的实现细节，又有 Android 的原理和机制。作者苦心钻研，颇有心得，深入浅出又提纲挈领地将 ARM 和 Android 的核心机密全盘托出，相信此书定会给无论是初学者还是资深开发者带来帮助和益处。

——三星半导体杭州研究所资深软件工程师 张秀文

本书在着力分析 Android 系统最常用到的内核机制之后，继续向上剖析 Android 用户层核心机制是如何接驳 Linux 内核的。而且书中分析涉及到 ARM 体系为较新 Cortex A9 SMP 架构，对于读者开发、研究工作有着实际的借鉴作用。

——红狼软件创始人；《深入剖析 Android 系统》作者 杨长刚

记得当年从研一开始，本书作者就埋头于 Linux 源代码的学习、分析，梦想着了解 Linux 的每一个角落。虽然他常常不修边幅，但只要谈起编程或 Linux 源代码他就两眼放光，滔滔不绝地大谈特谈心得体会。那时他的执着就让我十分钦佩。一晃十多年过去了，本书作者仍旧是那个严肃的程序员，依旧在执着地追求着他的梦想。

如果你也是一位严肃的程序员，有兴趣，有毅力去了解 Android 的实现；或者你是一个 Android 系统开发人员，推荐把本书作为手边的参考书。源代码注释的写作方式看起来有些简陋，但是这样对阅读 Android 源代码非常有帮助，根据书中的代码段可以很容易地搜索到你关注的 Android 源代码，从而大大缩短学习时间。

——本书作者同学；CA Technologies 研发经理 王晋强

前　言

1996 年，面对闪烁的 DOS 提示符，笔者心中产生了一个愿望——能够理解计算机每一条指令是如何工作的。

2000 年，笔者明白一个道理，内核才是机器的灵魂，只有阅读源码才能与机器的灵魂对话。于是笔者开始了漫长的源码阅读之旅。

2008 年，在经历了近十年内核以及驱动项目开发后，笔者发现尽管玄机重重，但是内核并非不能驾驭。在浩如烟海的 Linux 代码里有一个清晰的内核骨架，Linux 各种文件、驱动子系统围绕该骨架接驳起来，把握住这个骨架就驾驭了 Linux。

然而痛苦随之而来，以前笔者以为内核就是一切，驾驭了内核就理解了机器的一切。但是实际上，内核不是全部，要理解机器的机理还需要操作系统领域的探索，在经历了 Glibc、X、GTK 等痛苦且失败的探索之后，笔者发现了 Android。

2011 年，笔者多年的心愿总算有个了结。在经历了 2 年多 Android 源码研究之后，终于理解了 Android 如何将内核的强大能量释放给应用程序，如何对传统内核加以巧妙改造以满足当今以面向对象机制为基础的软件体系。这一年又是新的开始，一次难得的机会，笔者能够深入硬件开发。

2013 年，这两年可谓笔者的硬件生涯。无论是原理图、PCB 设计还是选择制板工厂、实施备料、贴片试产跟线，笔者都能够有幸深入一线体验硬件工程的复杂与艰辛。笔者尝试了软硬件协同设计，尽管在板级设计这一层面作用有限，但是通盘软考虑依然可以避免不少工程误区。

2014 年，笔者将这些年的笔记进行整理，攒成此书，也算是个人生的小结吧。

本书其实是实现本人年少时梦想的一部分——理解计算机每一条指令是如何工作的。

然而，这只是一个梦想，操作系统如同浩如烟海的原始森林，每一条指令只是其中的一片树叶，人们不可能读完每一行代码，也没有必要这样做。任何一个优秀的操作系统都有一个精悍的架构，Android 系统也不例外，本书通过分析这个架构来阐述 Android 机理。

笔者认为，尽管 Android 用户层面的代码量远大于内核部分，但是其关键架构却与内核息息相关，且其功能部分架构已经有很多优秀书籍和资料阐述。所以本书更多着墨于 Android 内核表现以及内核机制的 Android 运用。

最后需要提醒读者的是，无论多丰富的语言在源码面前也是苍白的，本书选择通过源码注释的方式来描述 Android 架构，建议读者结合源码来阅读本书。

编者

2014 年 4 月

目　录

上篇　内核

上篇　内核

Android 与 Linux 内核的关系，就好比一辆整车和底盘发动机的关系。作为底盘和发动机的 Linux 内核，尽管不能直接被用户和应用开发者感知到，但是却决定了 Android 系统运行时最核心的机制。

第 1 章　ARM 多核处理器

计算性能是处理器演进的第一动力。然而，尽管各种架构的高性能处理器层出不穷，但真正大规模普及开来的似乎只有 Intel 和 ARM 体系。观察其中的现象不难发现如下规律。

虽然处理器性能得到大幅改善，但如果无法得到现有主流操作系统的支持，就无法大规模应用。进一步来讲，即使某种处理器得到主流操作系统的支持，但是由于其指令集的不兼容性，导致大量的应用无法运行，这种处理器也是难以普及的，安腾的失败是个很好的例子。

“兼容性”是处理器演进的第一法则。现有的软件体系是计算机世界的主宰，所有不服从现在软件体系的指令集都只能被边缘化或者被淘汰。所以看到 Intel 和 ARM 不断扩展寄存器长度、增加发射单元、支持乱序、扩展总线单元等一系列手段来改进处理器架构，就是不敢废弃一条既有指令。

降低计算功耗是处理器演进的第二动力。对于服务器来说功耗就是运营成本，对于移动计算来说功耗就是生命。似乎台式机功耗要求不大，但是台式机与服务器、移动设备不仅在处理器的生产及设计上共享基础设施外，软件体系也有着相同的基础，导致整个软件体系为了照顾移动设备和服务器而收敛了扩张的步伐。这就是人们常说的“计算过剩”，其实计算永远不会过剩，只是庞大的软件体系有所收敛罢了。

频率提升是功耗的大敌，为了提高能耗比，设计频率更低而核心更多的处理器是好的方法，由此，近年来处理器的扩张由单核高频转变为多核体系。

Linux 支持的多核体系的基本特点为：所有处理器拥有完全相同的指令集，所有处理器共享同一内存。如果不符合以上任何一点，操作系统内核都需要做架构及修改才能支持。若满足这两个前提，其他方面的问题，都可以通过内核和系统小规模修改来支持。如 Big.little 架构下互相搭配的处理器尽管核心内部实现不同，但各核心有着相同的指令集，Linux 内核可以将它们当作同样的处理器来分配线程。内核在完全不加改动的情况下，也许会出现一个重量级线程被分配到了一个 little 的核心上，但是内核本身可以通过处理器负载均衡将其平衡到别的处理器上。当 little 负载较高时，big 必定会上线运行，但是系统有可能会出现重负载线程独自霸占 little 的情形的极限情况，这时需要将调度算法稍加修改，首先记录每个处理器能力，然后监测 little 上单线程高负载发生的时长，若超出一定阀值则将高负载线程时间片减为零，从 little 摘下来，再将其挂到 big 处理器运行队列即可。

进一步讲，多核心之间只要基本的读写、跳转之类指令相同，其余指令集不同，Linux 也可以支持的。内核只需用到基本指令即可，各核心用户空间指令可以不同。内核需做如下修改：每颗核心的调度队列记录下其核心用户态指令集特性，每个线程创建之初申明自身需要的指令集特性，内核在给线程分配核心及做负载均衡时比较两者是否匹配。用户层动态加载器也要做修改，针对当前处理器指令集特性动态加载、跳转到不同版本的动态库中，且在动态库执行过程中标记自身线程不可切换处理器。当然这种复杂情况超出了当前

现状，尽管有用户态指令集不同的架构，但是运行指令集差集的线程不需要动态加载库的支持，且直接绑定到指定核心上。

1.1　SMP 相关基础数据结构

CPU 管理的特点是自我管理，除了在启动、休眠、调频受控于 CPU0 的工作以外，处理器相关的绝大部分工作都由处理器自我管理。处理器是内核的执行体，又被内核控制。内核中准备了表征处理器运行状态的相应数据结构，处理器在运行时将自己的状态记录在这些结构中，而处理器也能通过别的处理器的表征结构了解其他处理器状态或发起控制。

每颗 CPU 都对应一个通用 CPU 结构，将 ARM Core 与 device 联系起来。

```
struct cpu {
/*cpu 编号*/
    int node_id;
/*对于 arm smp 该值为真，表示每颗 cpu 都可以被关掉*/
    int hotpluggable;
/*正如每个外设都有一个 struct device 表示自身一样，cpu 也不例外*/
    struct device dev;
};
```

处理器在内核中也被作为系统设备存在，而其相关操作以驱动形式通过处理器系统设备类——struct sysdev_class cpu_sysdev_class 来识别管理处理器。在处理器拓扑初始化时，内核完成设备侧的注册。

```
//处理器拓扑初始化
static int __init topology_init(void)
{   …
    /*对于每个物理存在处理器的操作，位图 cpu_possible_bits 描述了系统中的处理器，无论处理器是否 online，都对应其中一位*/
    for_each_possible_cpu(cpu) {
        struct cpuinfo_arm *cpuinfo = &per_cpu(cpu_data, cpu);
        /*ARM SMP 的架构是每个 core 都可以被单独关闭的*/
        cpuinfo->cpu.hotpluggable = 1;
        /*将当前 cpu 的 struct device 注册到 struct sysdev_class cpu_sysdev_class*/
        register_cpu(&cpuinfo->cpu, cpu);
    }
    …
}

//注册一颗处理器的设备
int __cpuinit register_cpu(struct cpu *cpu, int num)
{
    …
```

```
    //对 CA9，非 NUMA，为 0
    cpu->node_id = cpu_to_node(num);
    cpu->sysdev.id = num;
    //处理器系统设备类
    cpu->sysdev.cls = &cpu_sysdev_class;
    /* 向处理器系统设备类注册该处理器，在每注册一个设备时，都会调用设备类驱动来初始化
    该驱动*/
    error = sysdev_register(&cpu->sysdev);
    ...
}
```

在对方驱动侧，通过向处理器系统设备类注册驱动来匹配初始化处理器，参见 1.3 节。

struct cpuinfo_arm 对应于每颗 Arm core 实体，记录其基本信息，在 ARM 初始化时会依次初始化并注册每个 struct cpuinfo_arm。

```
struct cpuinfo_arm {
    struct cpu  cpu;
#ifdef CONFIG_SMP
    //当前 cpu 的 idle 线程
    struct task_struct *idle;
    unsigned int    loops_per_jiffy;
#endif
};
```

CPU 设备集合，每个 CPU 的 struct device 和 struct sysdev_driver 都会被加进来。这是联系 CPU 设备集合类驱动的纽带，当 CPU 设备或驱动注册的时候，会依次扫描这里的驱动列表或 CPU 设备，并进行初始化，如 cpufreq_sysdev_driver。

```
struct sysdev_class cpu_sysdev_class = {
    .name = "cpu",
    .attrs = cpu_sysdev_class_attrs,
};
```

struct cpumask：CPU 状态的基本数据结构定义如下：

```
 typedef struct cpumask { DECLARE_BITMAP(bits, NR_CPUS); } cpumask_t;
#define DECLARE_BITMAP(name,bits) \
    unsigned long name[BITS_TO_LONGS(bits)]
```

展开如下：

```
typedef struct cpumask { unsigned long bits[BITS_TO_LONGS(NR_CPUS)] }
cpumask_t;
```

cpu_possible_mask 位图，用来表示系统中的 CPU，每颗处理器对应其中一位。

cpu_online_mask 位图，用来表示当前处于工作状态的 CPU，每颗处理器对应其中一位。

```
cpu_bit_bitmap:
```