Device Tree（一）：背景介绍

作者：linuxer 发布于：2014-5-22 16:46 分类：统一设备模型

一、前言

作为一个多年耕耘在linux 2.6.23内核的开发者，各个不同项目中各种不同周边外设驱动的开发以及各种琐碎的、扯皮的俗务占据了大部分的时间。当有机会下载3.14的内核并准备学习的时候，突然发现linux kernel对于我似乎变得非常的陌生了，各种新的机制，各种framework、各种新的概念让我感到阅读内核代码变得举步维艰。 还好，剖析内核的热情还在，剩下的就交给时间的。首先进入视线的是Device Tree机制，这是和porting内核非常相关的机制，如果想让将我们的硬件平台迁移到高版本的内核上，Device Tree是一个必须要扫清的障碍。

我想从下面三个方面来了解Device Tree：

1、为何要引入Device Tree，这个机制是用来解决什么问题的？（这是本文的主题）

2、Device Tree的基础概念（请参考DT基础概念）

3、ARM linux中和Device Tree相关的代码分析（请参考DT代码分析）

阅读linux内核代码就像欣赏冰山，有看得到的美景（各种内核机制及其代码），也有埋在水面之下看不到的基础（机制背后的源由和目的）。沉醉于各种内核机制的代码固然有无限乐趣，但更重要的是注入更多的思考，思考其背后的机理，真正理解软件抽象。这样才能举一反三，并应用在具体的工作和生活中。

本文主要从下面几个方面阐述为何ARM linux会引入Device Tree：

1、没有Device Tree的ARM linux是如何运转的？

2、混乱的ARM architecture代码和存在的问题

3、新内核的解决之道

二、没有Device Tree的ARM linux是如何运转的？

我曾经porting内核到两个ARM-based的平台上。一个是小的芯片公司的应用处理器，公司自己购买了CPU core，该CPU core使用ARM兼容的指令集（但不是ARM）加上各种公司自行设计的多媒体外设整合成公司的产品进行销售。而我的任务就是porting 2.4.18内核到该平台上。在黑白屏幕的手机时代，那颗AP（application process）支持了彩屏、camera、JPEG硬件加速、2D/3D加速、MMC/SD卡、各种音频加速（内置DSP）等等特性，功能强大到无法直视。另外一次移植经历是让2.6.23内核跑在一个大公司的冷门BP（baseband processor）上。具体porting的方法是很简单的：

1、自己撰写一个bootloader并传递适当的参数给kernel。除了传统的command line以及tag list之类的，最重要的是申请一个machine type，当拿到属于自己项目的machine type ID的时候，当时心情雀跃，似乎自己已经是开源社区的一份子了（其实当时是有意愿，或者说有目标是想将大家的代码并入到linux kernel main line的）。

2、在内核的arch/arm目录下建立mach-xxx目录，这个目录下，放入该SOC的相关代码，例如中断controller的代码，时间相关的代码，内存映射，睡眠相关的代码等等。此外，最重要的是建立一个board specific文件，定义一个machine的宏：

```
MACHINE_START(project name, "xxx公司的xxx硬件平台")
        .phys_io        = 0x40000000,
        .boot_params        = 0xa0000100,
        .io_pg_offst        = (io_p2v(0x40000000) >> 18) & 0xfffc,
        .map_io                 = xxx_map_io,
        .init_irq        = xxx_init_irq,
        .timer                  = &xxx_timer,
        .init_machine        = xxx_init,
MACHINE_END
```

在xxx_init函数中，一般会加入很多的platform device。因此，伴随这个board specific文件中是大量的静态table，描述了各种硬件设备信息。

3、调通了system level的driver（timer，中断处理，clock等）以及串口terminal之后，linux kernel基本是可以起来了，后续各种driver不断的添加，直到系统软件支持所有的硬件。

综上所述，在linux kernel中支持一个SOC平台其实是非常简单的，让linux kernel在一个特定的平台上“跑”起来也是非常简单的，问题的重点是如何优雅的”跑”。

三、混乱的ARM architecture代码和存在的问题

每次正式的linux kernel release之后都会有两周的merge window，在这个窗口期间，kernel各个部分的维护者都会提交各自的patch，将自己测试稳定的

代码请求并入kernel main line。每到这个时候，Linus就会比较繁忙，他需要从各个内核维护者的分支上取得最新代码并merge到自己的kernel source tree中。Tony Lindgren，内核OMAP development tree的维护者，发送了一个邮件给Linus，请求提交OMAP平台代码修改，并给出了一些细节描述：

1、简单介绍本次改动

2、关于如何解决merge conficts。有些git mergetool就可以处理，不能处理的，给出了详细介绍和解决方案

一切都很平常，也给出了足够的信息，然而，正是这个pull request引发了一场针对ARM linux的内核代码的争论。我相信Linus一定是对ARM相关的代码早就不爽了，ARM的merge工作量较大倒在其次，主要是他认为ARM很多的代码都是垃圾，代码里面有若干愚蠢的table，而多个人在维护这个table，从而导致了冲突。因此，在处理完OMAP的pull request之后（Linus并非针对OMAP平台，只是Tony Lindgren撞在枪口上了），他发出了怒吼：

> Gaah. Guys, this whole ARM thing is a f*cking pain in the ass.

负责ARM linux开发的Russell King脸上挂不住，进行了反驳：事情没有那么严重，这次的merge conficts就是OMAP和IMX/MXC之间一点协调的问题，不能抹杀整个ARM linux团队的努力。其他的各个ARM平台维护者也加入讨论：ARM平台如何复杂，如何庞大，对于arm linux code我们已经有一些思考，正在进行中……一时间，讨论的气氛有些尖锐，但总体是坦诚和友好的。

对于一件事情，不同层次的人有不同层次的思考。这次争论涉及的人包括：

1、内核维护者（CPU体系结构无关的代码）

2、维护ARM系统结构代码的人

3、维护ARM sub architecture的人（来自各个ARM SOC vendor）

维护ARM sub architecture的人并没有强烈的使命感，作为公司的一员，他们最大的目标是以最快的速度支持自己公司的SOC，尽快的占领市场。这些人的软件功力未必强，对linux kernel的理解未必深入（有些人可能很强，但是人在江湖身不由己）。在这样的情况下，很多SOC specific的代码都是通过copy and paste，然后稍加修改代码就提交了。此外，各个ARM vendor的SOC family是一长串的CPU list，每个CPU多多少少有些不同，这时候＃ifdef就充斥了各个源代码中，让ARM mach-和plat-目录下的代码有些不忍直视。

作为维护ARM体系结构的人，其能力不容置疑。以Russell King为首的team很好的维护了ARM体系结构的代码。基本上，除了mach-和plat-目录，其他的目录中的代码和目录组织是很好的。作为ARM linux的维护者，维护一个不断有新的SOC加入的CPU architecture code的确是一个挑战。在Intel X86的架构一统天下的时候，任何想正面攻击Intel的对手都败下阵来。想要击倒巨人（或者说想要和巨人并存）必须另辟蹊径。ARM的策略有两个，一个是focus在嵌入式应用上，也就意味着要求低功耗，同时也避免了和Intel的正面对抗。另外一个就是博采众家之长，采用license IP的方式，让更多的厂商加入ARM建立的生态系统。毫无疑问，ARM公司是成功的，但是这种模式也给ARM linux的维护者带来了噩梦。越来越多的芯片厂商加入ARM阵营，越来越多的ARM platform相关的代码被加入到内核，不同厂商的周边HW block设计又各不相同……

内核维护者是真正对操作系统内核软件有深入理解的人，他们往往能站在更高的层次上去观察问题，发现问题。Linus注意到每次merge window中，ARM的代码变化大约占整个ARCH目录的60％，他认为这是一个很明显的符号，意味着ARM linux的代码可能存在问题。其实，60％这个比率的确很夸张，因为unicore32是在2.6.39 merge window中第一次全新提交，它的代码是全新的，但是其代码变化大约占整个ARCH目录的9.6％（需要提及的是unicore32是一个中国芯）。有些维护ARM linux的人认为这是CPU市场占用率的体现，不是问题，直到内核维护者贴出实际的代码并指出问题所在。内核维护者当然想linux kernel支持更多的硬件平台，但是他们更愿意为linux kernel制定更长远的规划。例如：对于各种繁杂的ARM平台，用一个kernel image来支持。

经过争论，确定的问题如下：

1、ARM linux缺少platform（各个ARM sub architecture，或者说各个SOC）之间的协调，导致arm linux的代码有重复。值得一提的是在本次争论之前，ARM维护者已经进行了不少相关的工作（例如PM和clock tree）来抽象相同的功能模块。

2、ARM linux中大量的board specific的源代码应该踢出kernel，否则这些垃圾代码和table会影响linux kernel的长期目标。

3、各个sub architecture的维护者直接提交给Linux并入主线的机制缺乏层次。

四、新内核的解决之道

针对ARM linux的现状，最需要解决的是人员问题，也就是如何整合ARM sub architecture（各个ARM Vendor）的资源。因此，内核社区成立了一个ARM sub architecture的team，该team主要负责协调各个ARM厂商的代码（not ARM core part），Russell King继续负责ARM core part的代码。此外，建立一个ARM platform consolidation tree。ARM sub architecture team负责review各个sub architecture维护者提交的代码，并在ARM platform consolidation tree上维护。在下一个merge window到来的时候，将patch发送给Linus。

针对重复的代码问题，如果不同的SOC使用了相同的IP block（例如I2C controller），那么这个driver的code要从各个arch/arm/mach-xxx中独立出来，变成一个通用的模块供各个SOC specific的模块使用。移动到哪个目录呢？对于I2C或者USB OTG而言，这些HW block的驱动当然应该移动到kernel/drivers目录。因为，对于这些外设，可能是in-chip，也可能是off-chip的，但是对于软件而言，它们是没有差别的（或者说好的软件抽象应该掩盖底层硬件的不同）。对于那些system level的code呢？例如clock control、interrupt control。其实这些也不是ARM-specific，应该属于linux kernel的核心代码，应该放到linux/kernel目录下，属于core-Linux-kernel frameworks。当然对于ARM平台，也需要保存一些和framework交互的code，这些code叫做ARM SoC core architecture code。OK，总结一下：

1、ARM的核心代码仍然保存在arch/arm目录下

2、ARM SoC core architecture code保存在arch/arm目录下

3、ARM SOC的周边外设模块的驱动保存在drivers目录下

4、ARM SOC的特定代码在arch/arm/mach-xxx目录下

5、ARM SOC board specific的代码被移除，由Device Tree机制来负责传递硬件拓扑和硬件资源信息。

OK，终于来到了Device Tree了。本质上，Device Tree改变了原来用hardcode方式将HW 配置信息嵌入到内核代码的方法，改用bootloader传递一个DB的形式。对于基于ARM CPU的嵌入式系统，我们习惯于针对每一个platform进行内核的编译。但是随着ARM在消费类电子上的广泛应用（甚至桌面系统、服务器系统），我们期望ARM能够象X86那样用一个kernel image来支持多个platform。在这种情况下，如果我们认为kernel是一个black box，那么其输入参数应该包括：

1、识别platform的信息

2、runtime的配置参数

3、设备的拓扑结构以及特性

对于嵌入式系统，在系统启动阶段，bootloader会加载内核并将控制权转交给内核，此外，还需要把上述的三个参数信息传递给kernel，以便kernel可以有较大的灵活性。在linux kernel中，Device Tree的设计目标就是如此。

标签：Device tree