一步一脚印

越努力，越幸运。

详解 **ARM Linux**启动过程分析

ARM Linux启动过程分析是本人要介绍的内容，嵌入式 **Linux** 的可移植性使得我们可以在各种电子产品上看到它的身影。对于不同体系结构的处理器来说**Linux**的启动过程也有所不同。本文以S3C2410 **ARM**处理器为例，详细分析了系统上电后bootloader的执行流程及 **ARM Linux**的启动过程。

1、引 言

Linux 最初是由瑞典赫尔辛基大学的学生 Linus Torvalds在1991 年开发出来的，之后在 GNU的支持下，Linux 获得了巨大的发展。虽然 Linux 在桌面 PC 机上的普及程度远不及微软的 Windows 操作系统，但它的发展速度之快、用户数量的日益增多，也是微软所不能轻视的。而近些年来 Linux 在嵌入式领域的迅猛发展，更是给 Linux 注入了新的活力。

一个嵌入式 Linux 系统从软件角度看可以分为四个部分[1]：引导加载程序（bootloader）， Linux 内核，文件系统，应用程序。

其中 bootloader是系统启动或复位以后执行的第一段代码，它主要用来初始化处理器及外设，然后调用 Linux 内核。Linux 内核在完成系统的初始化之后需要挂载某个文件系统做为根文件系统（Root Filesystem）。根文件系统是 Linux 系统的核心组成部分，它可以做为Linux 系统中文件和数据的存储区域，通常它还包括系统配置文件和运行应用软件所需要的库。

应用程序可以说是嵌入式系统的“灵魂”，它所实现的功能通常就是设计该嵌入式系统所要达到的目标。如果没有应用程序的支持，任何硬件上设计精良的嵌入式系统都没有实用意义。

从以上分析我们可以看出 bootloader 和 Linux 内核在嵌入式系统中的关系和作用。Bootloader在运行过程中虽然具有初始化系统和执行用户输入的命令等作用，但它最根本的功能就是为了启动 Linux 内核。在嵌入式系统开发的过程中，很大一部分精力都是花在bootloader 和 Linux 内核的开发或移植上。如果能清楚的了解 bootloader 执行流程和 Linux的启动过程，将有助于明确开发过程中所需的工作，从而加速嵌入式系统的开发过程。而这正是本文的所要研究的内容。

2、**Bootloader**

（1）Bootloader的概念和作用

Bootloader是嵌入式系统的引导加载程序，它是系统上电后运行的第一段程序，其作用类似于 PC 机上的 BIOS。在完成对系统的初始化任务之后，它会将非易失性存储器（通常是 Flash或 DOC 等）中的Linux 内核拷贝到 RAM 中去，然后跳转到内核的第一条指令处继续执行，从而启动 Linux 内核。由此可见，bootloader 和 Linux 内核有着密不可分的联系，要想清楚的了解 Linux内核的启动过程，我们必须先得认识 bootloader的执行过程，这样才能对嵌入式系统的整个启过程有清晰的掌握。

（2）Bootloader的执行过程

不同的处理器上电或复位后执行的第一条指令地址并不相同，对于 ARM 处理器来说，该地址为 0x00000000。对于一般的嵌入式系统，通常把 Flash 等非易失性存储器映射到这个地址处，而 bootloader就位于该存储器的最前端，所以系统上电或复位后执行的第一段程序便是 bootloader。而因为存储 bootloader的存储器不同，bootloader的执行过程也并不相同，下面将具体分析。

嵌入式系统中广泛采用的非易失性存储器通常是 Flash，而 Flash 又分为 Nor Flash 和Nand Flash 两种。 它们之间的不同在于： Nor Flash 支持芯片内执行（XIP， eXecute In Place），这样代码可以在Flash上直接执行而不必拷贝到RAM中去执行。而Nand Flash并不支持XIP，所以要想执行 Nand Flash 上的代码，必须先将其拷贝到 RAM中去，然后跳到 RAM 中去执行。

实际应用中的 bootloader根据所需功能的不同可以设计得很复杂，除完成基本的初始化系统和调用 Linux 内核等基本任务外，还可以执行很多用户输入的命令，比如设置 Linux 启动参数，给 Flash 分区等；也可以设计得很简单，只完成最基本的功能。但为了能达到启动Linux 内核的目的，所有的 bootloader都必须具备以下功能[2] ：

初始化 **RAM**

因为 Linux 内核一般都会在 RAM 中运行，所以在调用 Linux 内核之前 bootloader 必须设置和初始化 RAM，为调用 Linux 内核做好准备。初始化 RAM 的任务包括设置 CPU 的控制寄存器参数，以便能正常使用 RAM 以及检测RAM 大小等。

初始化串口

串口在 Linux 的启动过程中有着非常重要的作用，它是 Linux内核和用户交互的方式之一。Linux 在启动过程中可以将信息通过串口输出，这样便可清楚的了解 Linux 的启动过程。虽然它并不是 bootloader 必须要完成的工作，但是通过串口输出信息是调试 bootloader 和Linux 内核的强有力的工具，所以一般的 bootloader 都会在执行过程中初始化一个串口做为调试端口。

检测处理器类型

Bootloader在调用 Linux内核前必须检测系统的处理器类型，并将其保存到某个常量中提供给 Linux 内核。Linux 内核在启动过程中会根据该处理器类型调用相应的初始化程序。

设置 **Linux**启动参数

Bootloader在执行过程中必须设置和初始化 Linux 的内核启动参数。目前传递启动参数主要采用两种方式：即通过 struct param_struct 和struct tag（标记列表，tagged list）两种结构传递。struct param_struct 是一种比较老的参数传递方式，在 2.4 版本以前的内核中使用较多。从 2.4 版本以后 Linux 内核基本上采用标记列表的方式。但为了保持和以前版本的兼容性，它仍支持 struct param_struct 参数传递方式，只不过在内核启动过程中它将被转换成标记列表方式。标记列表方

我的随笔
我的评论
我的参与
最新评论
我的标签

随笔分类

android study(39)
C语言实例(30)
Hadoop study(4)
java study(4)
linux study(14)
Linux应用编程(1)
springmvc study
uC/OS-II源码分析(19)
web study(6)
成都的感悟(1)
创业知识(7)
单片机学习(14)
汇编学习(8)
裸奔-ARM(11)
嵌入式童年(8)
嵌入式学习(11)
生活感悟(7)
生活经历(5)
数据结构实例(39)

随笔档案

2018年1月 (2)
2017年11月 (1)
2017年2月 (1)
2017年1月 (13)
2016年12月 (122)
2016年11月 (22)
2016年10月 (15)
2016年9月 (9)

式是种比较新的参数传递方式，它必须以 ATAG_CORE 开始，并以ATAG_NONE 结尾。中间可以根据需要加入其他列表。Linux内核在启动过程中会根据该启动参数进行相应的初始化工作。

调用 **Linux**内核映像

Bootloader完成的最后一项工作便是调用 Linux内核。如果 Linux 内核存放在 Flash 中，并且可直接在上面运行（这里的 Flash 指 Nor Flash），那么可直接跳转到内核中去执行。但由于在 Flash 中执行代码会有种种限制，而且速度也远不及 RAM 快，所以一般的嵌入式系统都是将 Linux内核拷贝到 RAM 中，然后跳转到 RAM 中去执行。不论哪种情况，在跳到 Linux 内核执行之前 CUP的寄存器必须满足以下条件：r0＝0，r1＝处理器类型，r2＝标记列表在 RAM中的地址。

3、**Linux**内核的启动过程

在 bootloader将 Linux 内核映像拷贝到 RAM 以后，可以通过下例代码启动 Linux 内核：call_linux(0, machine_type, kernel_params_base)。

其中，machine_tpye 是 bootloader检测出来的处理器类型， kernel_params_base 是启动参数在 RAM 的地址。通过这种方式将 Linux 启动需要的参数从 bootloader传递到内核。Linux 内核有两种映像：一种是非压缩内核，叫 Image，另一种是它的压缩版本，叫zImage。根据内核映像的不同，Linux 内核的启动在开始阶段也有所不同。zImage 是 Image经过压缩形成的，所以它的大小比 Image 小。但为了能使用 zImage，必须在它的开头加上解压缩的代码，将 zImage 解压缩之后才能执行，因此它的执行速度比 Image 要慢。但考虑到嵌入式系统的存储空容量一般比较小，采用 zImage 可以占用较少的存储空间，因此牺牲一点性能上的代价也是值得的。所以一般的嵌入式系统均采用压缩内核的方式。

对于 ARM 系列处理器来说，zImage 的入口程序即为 arch/arm/boot/compressed/head.S。它依次完成以下工作：开启 MMU 和 Cache，调用 decompress_kernel()解压内核，最后通过调用 call_kernel()进入非压缩内核 Image 的启动。下面将具体分析在此之后 Linux 内核的启动过程。

（1）Linux内核入口

Linux 非压缩内核的入口位于文件/arch/arm/kernel/head-armv.S 中的 stext 段。该段的基地址就是压缩内核解压后的跳转地址。如果系统中加载的内核是非压缩的 Image，那么bootloader将内核从 Flash中拷贝到 RAM 后将直接跳到该地址处，从而启动 Linux 内核。不同体系结构的 Linux 系统的入口文件是不同的，而且因为该文件与具体体系结构有关，所以一般均用汇编语言编写[3]。对基于 ARM 处理的 Linux 系统来说，该文件就是head-armv.S。该程序通过查找处理器内核类型和处理器类型调用相应的初始化函数，再建立页表，最后跳转到 start_kernel()函数开始内核的初始化工作。

检测处理器内核类型是在汇编子函数__lookup_processor_type中完成的。通过以下代码可实现对它的调用：bl __lookup_processor_type。__lookup_processor_type调用结束返回原程序时，会将返回结果保存到寄存器中。其中r8 保存了页表的标志位，r9 保存了处理器的 ID 号，r10 保存了与处理器相关的 struproc_info_list 结构地址。

检测处理器类型是在汇编子函数 __lookup_architecture_type 中完成的。与__lookup_processor_type类似，它通过代码：“bl __lookup_processor_type”来实现对它的调用。该函数返回时，会将返回结构保存在 r5、r6 和 r7 三个寄存器中。其中 r5 保存了 RAM 的起始基地址，r6 保存了 I/O基地址，r7 保存了 I/O的页表偏移地址。当检测处理器内核和处理器类型结束后，将调用__create_page_tables 子函数来建立页表，它所要做的工作就是将 RAM 基地址开始的 4M 空间的物理地址映射到 0xC0000000 开始的虚拟地址处。对笔者的 S3C2410 开发板而言，RAM 连接到物理地址 0x30000000 处，当调用 __create_page_tables 结束后 0x30000000 ～ 0x30400000 物理地址将映射到0xC0000000～0xC0400000 虚拟地址处。

当所有的初始化结束之后，使用如下代码来跳到 C 程序的入口函数 start_kernel()处，开始之后的内核初始化工作：

2016年8月 (39)
2016年7月 (3)

相册

谨色安年(5)

最新评论

1. Re:经验分享——嵌入式工程师必看书籍(转载)
总结的很好，谢谢分享！
--二小西

2. Re:工作与人生 (摘抄)
可能我也会去写小说，我觉得能够将自己的所见所闻，内心的感受写成小说是件很有趣的事情，但是这势必会分心，想问下楼主是怎么平衡这两点的。
--二小西

3. Re:工作与人生 (摘抄)
看楼主是学技术的，为什么要去写小说了呢？请问你现在的职业是什么呢？
--二小西

4. Re:嵌入式技术学习路线
嵌入式主要内容包括linux系统，C语言开发，数据库等，JAVA部分的安卓开发等，学成后可以开发应用软件的，内核开发，驱动开发等工作，做项目。目前国内零基础的入门性课程较多，如果自己是软件开发相关专业.....
--不知不觉1

5. Re:栈的的后缀计算实现
咱们还早嘞！
--王小波的博客

阅读排行榜

b SYMBOL_NAME(start_kernel)

（2）start_kernel函数

start_kernel是所有 Linux 平台进入系统内核初始化后的入口函数，它主要完成剩余的与硬件平台相关的初始化工作，在进行一系列与内核相关的初始化后，调用第一个用户进程－init 进程并等待用户进程的执行，这样整个 Linux 内核便启动完毕。该函数所做的具体工作有[4][5]：

调用 setup_arch()函数进行与体系结构相关的第一个初始化工作；

对不同的体系结构来说该函数有不同的定义。对于 ARM 平台而言，该函数定义在arch/arm/kernel/Setup.c。它首先通过检测出来的处理器类型进行处理器内核的初始化，然后通过 bootmem_init()函数根据系统定义的 meminfo 结构进行内存结构的初始化，最后调用paging_init()开启 MMU，创建内核页表，映射所有的物理内存和 IO空间。

a、创建异常向量表和初始化中断处理函数；

b、初始化系统核心进程调度器和时钟中断处理机制；

c、初始化串口控制台（serial-console）；

d、ARM-Linux 在初始化过程中一般都会初始化一个串口做为内核的控制台，这样内核在启动过程中就可以通过串口输出信息以便开发者或用户了解系统的启动进程。

e、创建和初始化系统 cache，为各种内存调用机制提供缓存，包括;动态内存分配，虚拟文件系统（VirtualFile System）及页缓存。

f、初始化内存管理，检测内存大小及被内核占用的内存情况；

g、初始化系统的进程间通信机制（IPC）；

当以上所有的初始化工作结束后，start_kernel()函数会调用 rest_init()函数来进行最后的初始化，包括创建系统的第一个进程－init 进程来结束内核的启动。Init 进程首先进行一系列的硬件初始化，然后通过命令行传递过来的参数挂载根文件系统。最后 init 进程会执行用 户传递过来的"init＝"启动参数执行用户指定的命令，或者执行以下几个进程之一：

1 execve("/sbin/init",argv_init,envp_init);

2 execve("/etc/init",argv_init,envp_init);

3 execve("/bin/init",argv_init,envp_init);

4 execve("/bin/sh",argv_init,envp_init)。

当所有的初始化工作结束后，cpu_idle()函数会被调用来使系统处于闲置（idle）状态并等待用户程序的执行。至此，整个 Linux 内核启动完毕。

4. 结论

Linux 内核是一个非常庞大的工程，经过十多年的发展，它已从从最初的几百 KB 大小发展到现在的几百兆。清晰的了解它执行的每一个过程是件非常困难的事。但是在嵌入式开发过程中，我们并不需要十分清楚 linux 的内部工作机制，只要适当修改 linux 内核中那些与硬件相关的部分，就可以将 linux 移植到其它目标平台上。通过对 linux 的启动过程的分 析，我们可以看出哪些是和硬件相关的，哪些是 linux 内核内部已实现的功能，这样在移植linux 的过程中便有所针对。而 linux内核的分层设计将使 linux 的移植变得更加容易。

/********** Stay hungry, Stay foolish. @Rocky **********/

1. 如何使用MASM来编译、连接、调试汇编语言(7461)
2. ARM-汇编指令集（总结）(7045)
3. 详解 ARM Linux启动过程分析(4609)
4. 嵌入式技术学习路线(3819)
5. shell与变量的声明的操作(3274)

评论排行榜

1. 栈的的后缀计算实现(6)
2. 正是孤独让你出众(2)
3. 静态链表(2)
4. 工作与人生 (摘抄)(2)
5. 嵌入式技术学习路线(1)

推荐排行榜

1. ARM-汇编指令集（总结）(2)
2. 静态链表(1)
3. 线性表的链式存储结构(1)
4. 工作与人生 (摘抄)(1)
5. 数组与指针实例(1)

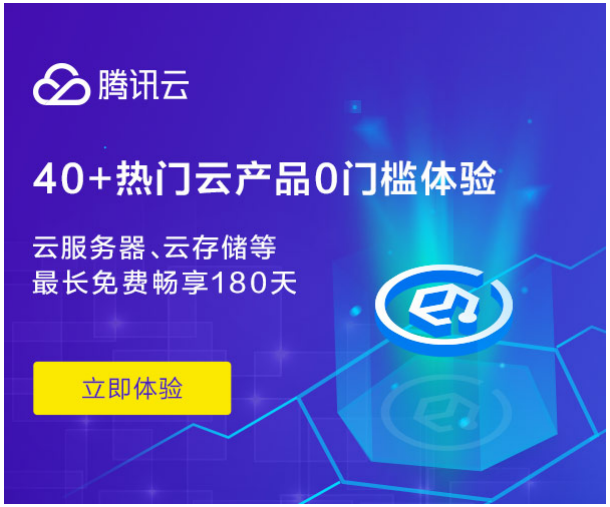
分类: 嵌入式童年

王小波的博客
关注 - 2
粉丝 - 16
+加关注

0 0

« 上一篇：一页纸商业计划书模板(转载)
» 下一篇：BSS段、数据段、代码段、堆与栈

posted @ 2017-01-09 19:11 王小波的博客 阅读(4608) 评论(0) 编辑 收藏



DON'T FORGET TO HAVE FUN