精簡講述linux內核啟動過程

1. Linux內核啟動過程概述

一個嵌入式 Linux 系統從軟件角度看可以分為四個部分：引導加載程序（Bootloader），Linux 內核，文件系統，應用程序。其中 Bootloader是系統啟動或reset以後執行的第一段代碼，它主要用來初始化處理器及外設，然後調用 Linux 內核。Linux 內核在完成系統的初始化之後需要掛載某個文件系統做為根文件系統（Root Filesystem）。根文件系統是 Linux 系統的核心組成部分，它可以做為Linux 系統中文件和數據的存儲區域，通常它還包括系統配置文件和運行應用軟件所需要的庫。應用程序可以說是嵌入式系統的「靈魂」，它所實現的功能通常就是設計該嵌入式系統所要達到的目標。如果沒有應用程序的支持，任何硬件上設計精良的嵌入式系統都沒有實用意義。

2. Bootloader啟動過程

Bootloader在運行過程中雖然具有初始化系統和執行用戶輸入的命令等作用，但它最根本的功能就是為了啟動 Linux 內核。

2.1 Bootloader的概念和作用

Bootloader是嵌入式系統的引導加載程序，它是系統上電後運行的第一段程序，其作用類似於 PC 機上的 BIOS。在完成對系統的初始化任務之後，它會將非易失性存儲器（通常是Flash或DOC等）中的Linux 內核拷貝到 RAM 中去，然後跳轉到內核的第一條指令處繼續執行，從而啟動 Linux 內核。由此可見，Bootloader 和 Linux 內核有著密不可分的聯繫，要想清楚的瞭解 Linux內核的啟動過程，我們必須先得認識 Bootloader的執行過程，這樣才能對嵌入式系統的整個啟動過程有清晰的掌握

2.2 Bootloader的執行過程

不同的處理器上電或reset後執行的第一條指令地址並不相同，對於 ARM 處理器來說，該地址為 0x00000000。對於一般的嵌入式系統，通常把 Flash 等非易失性存儲器映射到這個地址處，而 Bootloader就位於該存儲器的最前端，所以系統上電或reset 後執行的第一段程序便是Bootloader。而因為存儲 Bootloader的存儲器不同，Bootloader的執行過程也並不相同，下面將具體分

嵌入式系統中廣泛採用的非易失性存儲器通常是 Flash，而 Flash 又分為 Nor Flash 和Nand Flash 兩種。 它們之間的不同在於：Nor Flash 支持芯片內執行（XIP， eXecute In Place），這樣代碼可以在Flash上直接執行而不必拷貝到RAM中去執行。而Nand Flash並不支持XIP，所以要想執行 Nand Flash 上的代碼，必須先將其拷貝到 RAM中去，然後跳到 RAM 中去執行

2.3 Bootloader的功能

(1)初始化 RAM

因為 Linux 內核一般都會在 RAM 中運行，所以在調用 Linux 內核之前 bootloader 必須設置和初始化 RAM，為調用 Linux內核做好準備。初始化 RAM 的任務包括設置CPU 的控制寄存器參數，以便能正常使用 RAM 以及檢測RAM 大小等

(2)初始化串口

串口在 Linux 的啟動過程中有著非常重要的作用，它是 Linux內核和用戶交互的方式之一。Linux 在啟動過程中可以將信息通過串口輸出，這樣便可清楚的瞭解 Linux 的啟動過程。雖然它並不是 Bootloader 必須要完成的工作，但是通過串口輸出信息是調試Bootloader和Linux 內核的強有力的工具，所以一般的 Bootloader 都會在執行過程中初始化一個串口做為調試端口

(3)檢測處理器類型

Bootloader在調用 Linux內核前必須檢測系統的處理器類型，並將其保存到某個常量中提供給 Linux 內核。Linux 內核在啟動過程中會根據該處理器類型調用相應的初始化程序

(4)設置 Linux啟動參數

Bootloader在執行過程中必須設置和初始化 Linux 的內核啟動參數。目前傳遞啟動參數主要採用兩種方式：即通過 struct param_struct 和struct tag（標記列表，tagged list）兩種結構傳遞。struct param_struct 是一種比較老的參數傳遞方式，在 2.4 版本以前的內核中使用較多。從 2.4 版本以後 Linux 內核基本上採用標記列表的方式。但為了保持和以前版本的兼容性，它仍支持 struct param_struct 參數傳遞方式，只不過在內核啟動過程中它將被轉換成標記列表方式。標記列表方式是種比較新的參數傳遞方式，它必須以 ATAG_CORE 開始，並以ATAG_NONE 結尾。中間可以根據需要加入其他列表。Linux內核在啟動過程中會根據該啟動參數進行相應的初始化工作

(5)調用 Linux內核映像

Bootloader完成的最後一項工作便是調用 Linux內核。如果 Linux 內核存放在 Flash 中，並且可直接在上面運行（這裡的 Flash 指 Nor Flash），那麼可直接跳轉到內核中去執行。但由於在 Flash 中執行代碼會有種種限制，而且速度也遠不及 RAM 快，所以一般的嵌入式系統都是將 Linux內核拷貝到 RAM 中，然後跳轉到 RAM 中去執行

不論哪種情況，在跳到 Linux 內核執行之前 CPU的寄存器必須滿足以下條件：r0 = 0，r1 = 處理器類型，r2 = 標記列表在 RAM中的地址

3.Linux啟動過程

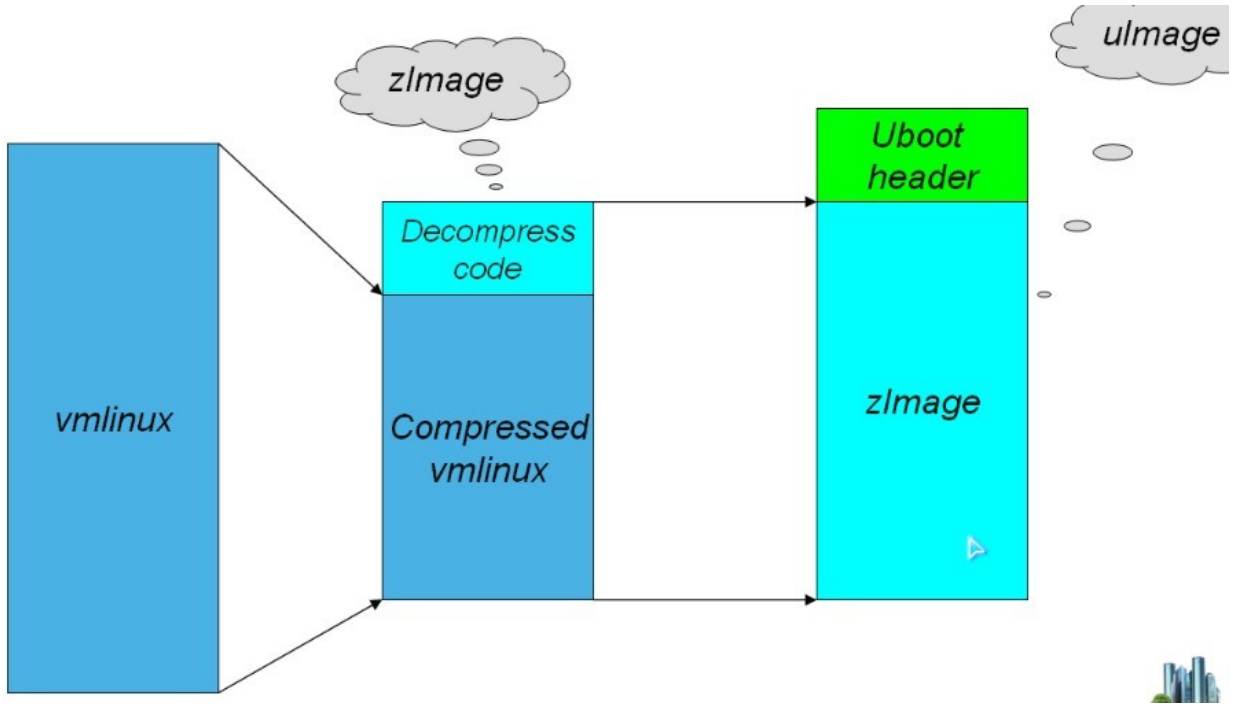
在Bootloader將 Linux 內核映像拷貝到 RAM 以後，可以通過下例代碼啟動 Linux 內核：

call_linux(0, machine_type, kernel_params_base)

其中，machine_tpye 是Bootloader檢測出來的處理器類型， kernel_params_base 是啟動參數在 RAM 的地址。通過這種方式將 Linux 啟動需要的參數從 bootloader傳遞到內核

Linux 內核有兩種映像：一種是非壓縮內核，叫 Image，另一種是它的壓縮版本，叫 zImage。根據內核映像的不同，Linux 內核的啟動在開始階段也有所不同。zImage 是 Image 經過壓縮形成的，所以它的大小比 Image 小。但為了能使用 zImage，必須在它的開頭加上解壓縮的代碼，將 zImage 解壓縮之後才能執行，因此它的執行速度比 Image 要慢。但考慮到嵌入式系統的存儲空容量一般比較小，採用 zImage 可以佔用較少的存儲空間，因此犧牲一點性能上的代價也是值得的。所以一般的嵌入式系統均採用壓縮內核的方式

對於ARM 系列處理器來說，zImage 的入口程序即為 arch/arm/boot/compressed/head.S。它依次完成以下工作：開啟 MMU 和 Cache，調用 decompress_kernel()解壓內核，最後通過調用 call_kernel()進入非壓縮內核 Image 的啟動。下面將具體分析在此之後 Linux 內核的啟動過程



補充 壓縮(zImage)與非壓縮內核映像(Image)

非壓縮內核映像(Image)是真正的Linux內核代碼。

壓縮內核映像是把非壓縮內核映像作為數據進行

壓縮打包,並加上了解壓縮代碼。也就是說,它是一個自解壓的可執行映像。壓縮內核映像執行時,先解壓內部包含的數據塊(即非壓縮內核映像),再去執行非壓縮內核映像。

3.1 Linux內核入口

Linux 非壓縮內核的入口位於文件/arch/arm/kernel/z 中的stext 段。該段的基地址就是壓縮內核解壓後的跳轉地址。如果系統中加載的內核是非壓縮的 Image，那麼bootloader將內核從 Flash中拷貝到 RAM 後將直接跳到該地址處，從而啟動 Linux 內核。不同體系結構的 Linux 系統的入口文件是不同的，而且因為該文件與具體體系結構有關，所以一般均用彙編語言編寫。對基於 ARM 處理的 Linux 系統來說，該文件就是head-armv.S。該程序通過查找處理器內核類型和處理器類型調用相應的初始化函數，再建立頁表，最後跳轉到 start_kernel()函數開始內核的初始化工作。檢測處理器內核類型是在彙編子函數 __lookup_processor_type中完成的。通過以下代碼可實現對它的調用：

bl __lookup_processor_type

__lookup_processor_type調用結束返回原程序時，會將返回結果保存到寄存器中。其中r8 保存了頁表的標誌位，r9 保存了處理器的 ID 號，r10 保存了與處理器相關的 stru proc_info_list 結構地址。

檢測處理器類型是在彙編子函數 __lookup_architecture_type 中完成的。與 __lookup_processor_type類似，它通過代碼：「bl __lookup_processor_type」來實現對它的調用。該函數返回時，會將返回結構保存在 r5、r6 和 r7 三個寄存器中。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的頁表偏移地址。當檢測處理器內核和處理器類型結束後，將調用__create_page_tables 子函數來建立頁表，它所要做的工作就是將 RAM 基地址開始的 4M 空間的物理地址映射到 0xC0000000 開始的虛擬地址處。對筆者的 S3C2410 開發板而言，RAM 連接到物理地址 0x30000000 處，當調用 __create_page_tables 結束後 0x30000000 ~ 0x30400000 物理地址將映射到 0xC0000000 ~ 0xC0400000 虛擬地址處。當所有的初始化結束之後，使用如下代碼來跳到 C 程序的入口函數 start_kernel()處，開始之後的內核初始化工作：b SYMBOL_NAME(start_kernel)

補充 __create_page_tables作用

bl __create_page_tables

設置初始頁表，這裡只設置最起碼的數量，只要能使內核運行即可，**r8 = machinfo，r9 = cpuid，r10 = procinfo，**在r4寄存器中返回物理頁表地址。

__create_page_tables例程在文件arch/arm/kernel/head.S中定義：

__create_page_tables:

pgtbl r4　　@ page table address

3.2 start_kernel函數

start_kernel是所有 Linux 平台進入系統內核初始化後的入口函數，它主要完成剩餘的與硬件平台相關的初始化工作，在進行一系列與內核相關的初始化後，調用第一個用戶進程－init 進程並等待用戶進程的執行，這樣整個 Linux 內核便啟動完畢。該函數所做的具體工作有：調用 setup_arch()函數進行與體系結構相關的第一個初始化工作；對不同的體系結構來說該函數有不同的定義。對於 ARM 平台而言，該函數定義在arch/arm/kernel/Setup.c。它首先通過檢測出來的處理器類型進行處理器內核的初始化，然後通過 bootmem_init()函數根據系統定義的 meminfo 結構進行內存結構的初始化，最後調用 paging_init() 開啟 MMU，創建內核頁表，映射所有的物理內存和 IO空間。創建異常向量表和初始化中斷處理函數；初始化系統核心進程調度器和時鐘中斷處理機制；初始化串口控制台（serial-console）；ARM-Linux 在初始化過程中一般都會初始化一個串口做為內核的控制台，這樣內核在啟動過程中就可以通過串口輸出信息以便開發者或用戶瞭解系統的啟動進程。創建和初始化系統 cache，為各種內存調用機制提供緩存，包括;動態內存分配，虛擬文件系統（VirtualFile System）及頁緩存。初始化內存管理，檢測內存大小及被內核佔用的內存情況；初始化系統的進程間通信機制（IPC）；當以上所有的初始化工作結束後，start_kernel()函數會調用 rest_init()函數來進行最後的初始化，包括創建系統的第一個進程－init 進程來結束內核的啟動。init 進程首先進行一系列的硬件初始化，然後通過命令行傳遞過來的參數掛載根文件系統。最後 init 進程會執行用戶傳遞過來的「init = 」啟動參數執行用戶指定的命令，或者執行以下幾個進程之一：

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

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

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

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

當所有的初始化工作結束後，cpu_idle()函數會被調用來使系統處於閒置（idle）狀態並等待用戶程序的執行。至此，整個 Linux 內核啟動完畢。

補充 在沒有啟動 MMU 之前創建臨時頁表(__create_page_tables) CPU運行於物理地址

/*創建臨時頁表*/

__create_page_tables:

pgtbl　　r4　　--r4=0x50004000 該值被靜態設置

```
        mov r0, r4
        mov r3, #0
        add  r6, r0, #0x4000
1:      str    r3, [r0], #4
        str    r3, [r0], #4
        str    r3, [r0], #4
        str    r3, [r0], #4
        teq   r0, r6
        bne  1b
```

上面是將0x50004000 到0x50008000這段16k的內存清零。也許大家會困惑那kernel image被覆制到哪兒呢，其實kernel image其實地址為0x50008000. 而 0x50004000 ~ 0x50008000這段16k內存用於創建臨時頁表，該臨時頁表將在paging_init中清除 。而0x50000000~0x0x50004000這16k的內存則是用於uboot與kernel之間的信息交互 。如在uboot中設置的bootcmd會通過這段內存傳替過來。