Project1 Bootloader设计文档

中国科学院大学

段宏键

2017.09.26

# Bootblock设计流程

1. Bootblock主要完成的功能

Bootblock 本身存于硬盘的初始位置（第一个扇区），其会将内核从硬盘加载到内存中，是内核的启动程序。Bootblock 调用读盘函数在内存指定位置读入硬盘中的内核，接下来会调到内核的 main 函数位置开始执行内核。要注意的是，bootblock 需要有关于内核大小的信息，这在creatimage中完成。

1. Bootblock被载入内存后的执行流程

Bootblock被载入内存前，其内有一个data段的量用于存入要读 kernel 的扇区数， 载入后，会设置读盘函数的三个参数，分别为读取的目的地址（到$4）、SD 卡内部的偏移量(到$5)、要读取的字节数(到$6)，然后通过跳转调到读盘函数地址，调用 PMON 的读盘函数，最后跳转到内核的 main 函数位置开始执行内核。

1. Bootblock如何调用SD卡读取函数

首先向$4、$5、$6 三个寄存器中设置函数传入的三个参数：读取的目的地址、SD 卡内部的偏移量、要读取的字节数，然后用 jal 指令跳转到 0x8007b1a8 执行读盘函数。值得注意的是，要读的字节数并不是人为设定不变的，而是根据kernel所占扇区大小而定的。

1. Bootblock如何跳转至kernel入口

我的代码中，Bootblock 会把 Kernel 读取在 0xa0800200 处，通过 objdump 打印kernel内容会得到 Kernel 的 main 函数偏移地址为 0x6c。接下来用 jal 指令跳转到 0xa080026c 即可。

1. 任何在设计、开发和调试bootblock时遇到的问题和解决方法

一开始经常见到TLB错误，后来才知道是最后运行完代码之后又继续运行到了没有意义的乱码部分。在 Bootblock 调用内核之后添加一个死循环可以解决这个问题。

其次，如何向bootblock传递kernel所占扇区的这个参数呢？我开始试图直接把这个参数写进 $8 或其他寄存器里，发现无法实现。后来我试了很多种方法，后来发现需要在bootblock中设置一个.data中的变量，用objdump找到这个变量，把参数地址传进去，再把这个参数写入 $8 寄存器。

# Createimage设计流程

（1） Bootblock编译后的二进制文件、Kernel编译后的二进制文件，以及SD卡image文件这三者之间的关系

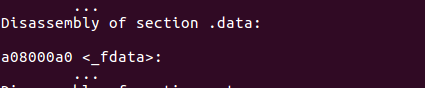
我们通过 createimage 过程创建image文件，其中image文件的内容来源就是Bootblock编译后的二进制文件中可执行代码、Kernel编译后的二进制文件中可执行代码，做完这些才把image写入SD卡。

image 前 512 字节的位置存放 Bootblock 编译后的二进制文件中可执行代码，从512字节之后开始存放 Kernel 编译后的二进制文件中可执行代码，所需空间与kernel所占扇区（512字节）多少有关。

（2） 如何获得Bootblock和Kernel二进制文件中可执行代码的位置和大小

用 Elf 的相关知识。 由于Bootblock和Kernel二进制文件均是 elf 文件格式，所以可以通过 elf 头文件的ehdr->e\_phoff 获得 program header phdr，从program header phdr 的 phdr->p\_offset 和 phdr->p\_filesz 可获得二进制文件中可执行代码的位置和大小。

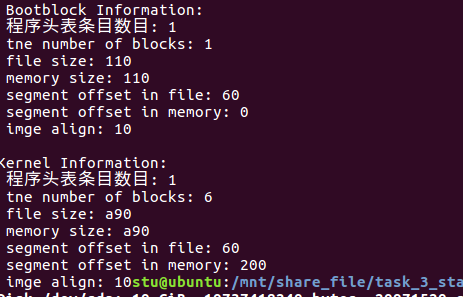
（3） 如何让Bootblock获取到Kernel二进制文件的大小，以便进行读取

这个我在本实验报告上面“Bootblock设计流程”的最后一个问题中已经进行了初步阐述。在 createimage 中计算得到 kernel 文件的大小（本处为所占扇区个数），在 bootblock 中设置 .data 段中一个参数，通过 objdump 得到 bootblock 文件中这个参数的位置，在 createimage 中传入储存kernel 文件的大小（本处为所占扇区个数）的变量的地址到 bootblock 所设的参数中，然后通过地址读取这个参数到寄存器中。

上图即为 objdump 得到的 bootblock 的 data 段参数的位置。

（4） extended\_opt 的输出信息。

用的后来上传的 kernel.c 代码



（5）任何在设计、开发和调试createimage时遇到的问题和解决方法

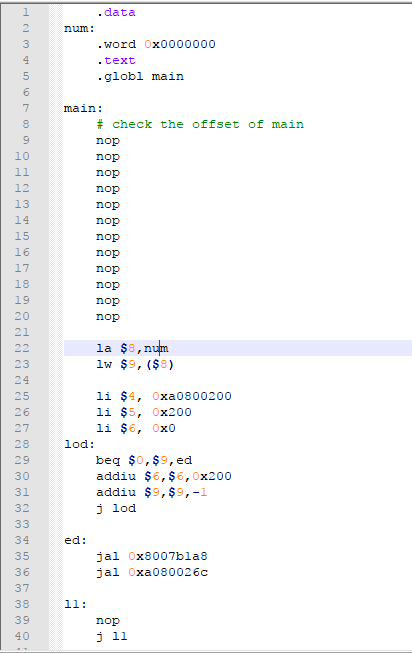
在写 creatimage.c 时，由于某些函数入口有一些双\*（表示指针）的定义，但我在写的时候并没有注意到，所以开始一直导致相应的 ehdr 与 phdr 读不出值来，后来我把读elf文件头函数声明中用到双\*（表示指针）的地方改为了单\*（表示指针），就能读出数据了。

其次，createimage.c 中我开始写的写 kernel 到 image 的函数 kernel 大小是固定的，之后听到老师说要换更大的 kernel 进行测试时我才改过来，bootblock也是这个时候才改，我在本实验报告上面“Bootblock设计流程”的最后一个问题中已经进行了阐述。

最后，初步写好 createimage 后 make 操作时会出现不知名的“段错误”，我在 creatimage.c 中更改为声明一个指向elf文件头与程序头的指针后马上给这个指针分配空间，段错误就消失了，我目前还不知道具体原因。我认为可能是不分配空间而连续声明的指针会在之后分配空间后有指向的重合，即可能是第二个指针指向了第一个结构体的中间部分。

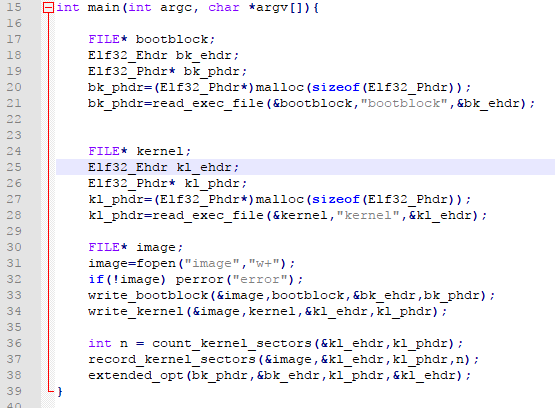
# 关键函数功能

Bootblock.s

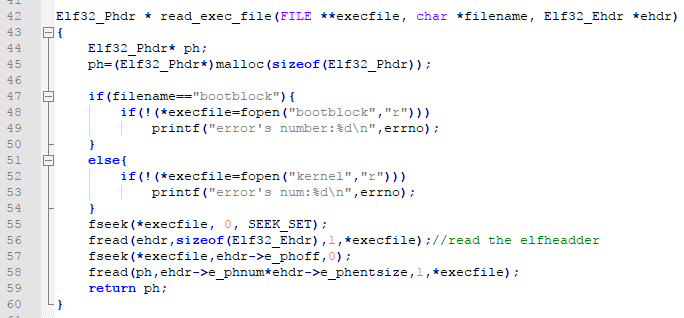


Createimage.c

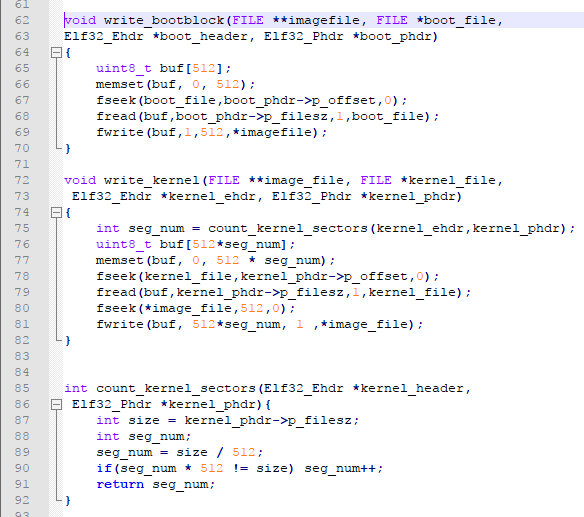
Main:



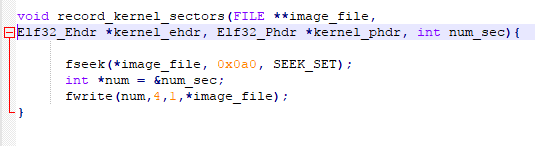
Read\_elf



Write image & count sectors



To bootblock



参考文献

[1] MIPS32 手册

[2] Project1-Bootloader 任务书

▄