Project1 Bootloader 设计文档

中国科学院大学

崔鹏来 孔静

2016.9.20

1. Bootblock 设计流程

Bootblock 是 BIOS 中的特定区域，其中包含用于引导的最小指令集。

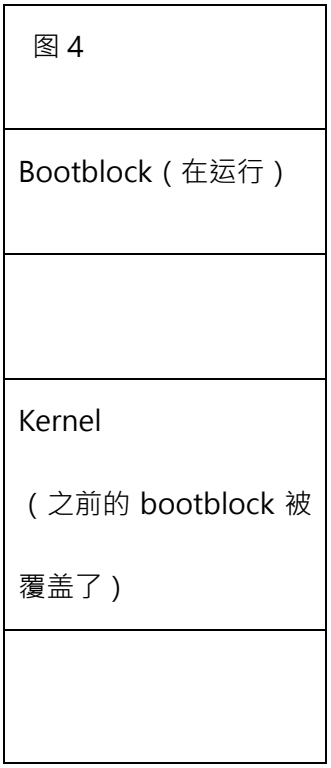
在本次试验中，我们设计的 Bootblock 主要是起到加载 kernel 的作用。

在试验 4 中，我们的 BootBlock 需要移动一个大小为 9 个扇区的内核，这个比较容易，因为我们有一个神奇而强大的指令，int0x13，这个 int13 函数就像一个搬运工一样可以把我们想要移动的对象移动到准确的位置。所以在实验 4 中，只要我们对 int13 下好正确的指令，搞准出发地点和终点就能顺利地完成任务。

但是在 bonus 中，情况发生了变化。我们要移动的物体--kernel 变大了将近十倍，这个时候如果我们还按照原来的方案移动 kernel，就会导致 kernel 将 bootloader 覆盖掉，产生严重的结果。为了不让 kernel 的搬用将 bootloader 覆盖，我们需要在移动 kernel 之前将 bootloader 转移到一个安全的地方。那么哪里是安全的地方呢？我们做一个简单的计算就可以得到。放置 kernel 的起始位置是 0x1000，而 kernel 的大小是 84 个扇区，每个扇区 512B。所以 kernel 的总大小是 42KB，在 16 进制下表示就是 0xa800，也就是说只要把 bootloader 放在 0xb800

以外的地方，就可以保证我这个 bootloader 不被 kernel 影响到。

内存的具体变化请看下图：



本来，想通过 os_size 的大小，然后计算将 bootloader 向后的距离，但是具体实施的时候，不知道出了啥 bug，调了一段时间，没有改好，最终废弃，决定直接定数据。

较好的设计想法是：先读取 os_size，然后判断是否需要移动 bootloader，若要移动，计算移动到哪里合适。然后读取 INT13 08H 功能，读出柱面数磁头数扇区数，套几层循环，读完一次判断是否还有扇区要读，若要读，判断 dh 是否要加 1，还是 dh 已经到最大值需要清零并对 ch 加 1，然后加载 kernel，设置内存栈，跳转 kernel。

上述只是我们菜鸟的美好的想法，具体实行的时候，先是一下次按照上述设计流程，写完，然后发现各种不知道哪里出 bug，最后打算从 task4 基础版本开始修改，再加上时间有限，就只写成了，移动 bootloader 到 0xf000，然后再移动 kernel，而且因为 INT13 08H 功能不太会用，用了就跑不了，赋值了 18 个最大扇区数，2 个磁头，进行了循环读取。可以兼容小内核，也能兼容 112 个扇区以内的中内核。缺点是，内核扇区不足 36 时，也要先移动 bootloader，浪费了时间，以及当内核大于 112 扇区的时候，仍然会覆盖 bootloader。如果时间充足，会继续进行修改。(给自己的弱小找借口 orz，写代码习惯不好，太容易出 bug)

2. Createimage 设计流程

我们都知道启动系统在启动的时候，需要从磁盘中读取必要的信息。在这里起到至关重要作用的磁盘就是我们这次试验要做出来的 image 文件。那么这个 image 文件究竟是什么呢？我们这个 image 文件主要由两个部分组成一个是 bootblock，一个是 kernel。在系统开始运行的时候，启动系统会自动读取这个 image 文件，然后把存在 image 里的 bootblock 加载到 0x7c00 处，接着在 bootblock 的指示下读取内核。

综上所述，我们通过 createimage 实现的，就是一个写有 bootblock 和 kernel 的 image。具体的实现方法也比较简单。首先我们需要使用 fopen 打开编译好的 bootblock

和 kernel 文件，并获得相应的文件指针。然后还是使用 fopen 创建一个可以写的 image 文件，然后，我们把 bootblock 的内容通过 fwrite 写入数组，再将数组中的内容 fread 到 image 中，这样我们就完成了对 bootblock 的写入。之后，我们通过 fseek 移动指向 image 的指针到距离文件头 512 个字节的位置，写入 55aa。然后再用类似于写入 bootblock 的流程写入 kernel。

写完代码之后，其实还是有缺陷，因为题给的 elf 文件 program header table 的条目只有一个，所以没有写读取条目数量，进行循环的操作，一切从简。还有文件没有打开成功之类的检错操作，均省略了。嘛，给自己找个借口，如果时间够多，会增加的。

3. 关键函数功能

本次实验要实现 6 个函数，其中后三个函数主要是对 os_size 的修改和打印一些信息，不是很重要，最主要实现功能的是前三个函数。

read_exec_file(); 这个函数的主要作用是读取一个 elf 文件，把 elf 文件中程序头这部分写入到一个指针里，然后我们返回这个指针。

write_bootblock();我们通过这个函数将 bootblock 写入磁盘的第一个扇区。实现的过程需要 p_phoff 确定有效段的位置，然后通过 p_filesz 确定写入段的大小。

write_kernel(); 我们通过这个函数将 kernel 写入 image 第二个扇区后的位置。

这次我们主要是使用了 4 个系统定义的函数，分别是 fopen，fseek，fread，fwrite。这四个函数的作用依次是打开文件，定位置，读文件和写文件。

参考文献

[1] http://www.cnblogs.com/whiteyun/archive/2009/08/08/1541822.html

[2] http://baike.baidu.com/link?url=F5aOnSwcYmfReX9AlZnhHbAfnwK8cMgeM09D5rpVROT_s5fZoDjxNevUrtc5HGEyfYi_qYoS8VR_O9HSGNISJxn1p9R9B3NiTh1Piiy-KYK

■