Project1 Bootloader设计文档

中国科学院大学

张旭

2017/9/26

# Bootblock设计流程

1.Bootblock 主要完成的功能

Bootblock负责调用地址为0x8007b1a8的读盘函数, 该函数将kernal可执行文件读入0xa0800200处。然后跳转到kernal文件的入口地址0xa080026c。kernal执行完成后返回PMON。

2. Bootblock被载入内存后的执行流程

Bootblock被载入内存后，PC自动跳转到0xa0800030处开始执行。向$a1,$a2,$a3寄存器写入数据进行传参，再通过jal命令跳转到0x8007b1a8处。函数返回后，再通过jal命令跳转到0xa080026c处。Kernal函数返回后，通过jr命令返回PMON。

3.Bootblock如何调用SD卡读取函数

Bootblock通过jal命令跳转到SD卡读取函数起始地址0x8007b1a8处开始执行。

4. Bootblock如何跳转至kernel入口

Bootblock通过jal命令跳转到kernel入口0xa080026c处开始执行。

5. 在设计、开发和调试bootblock时遇到的问题和解决方法

在指令执行完后，不加jr $31指令的话，就会发生例外：TLB miss，因为Bootblock运行在kseg1内（无缓存，无映射），按道理讲不会出现TLB表缺失的例外。可能是因为CPU一直在运转，导致PC执行到内存的其他区域，导致出现该例外。

# 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

Image文件第一个扇区保存Bootblock二进制文件，第二个扇区保存Kernel的二进制文件。Bootblock文件和Kernel文件的关系已在上面声明。

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

在Bootblock和Kernel二进制文件中的ELF表头信息保存有e\_phoff信息（程序头表的偏移量），我们查询程序头表，可执行文件的p\_type值为PT\_LOAD，然后我们读取p\_offset信息即可得到可执行代码在二进制文件的偏移量，读取p\_filesz可得到代码节的大小。

1. 如何让Bootblock获取到Kernel二进制文件的大小，以便进行读取

在Bootblock中增加一个变量OS\_SIZE，并初始化。在Createimage文件将Kernel装载进image中时，统计kernel大小，然后修改image第一扇区中Bootblock的二进制代码，将OS\_SIZE的值改为kernel大小。这样，Bootblock获取到Kernel二进制文件的大小。

1. 在设计、开发和调试createimage时遇到的问题和解决方法

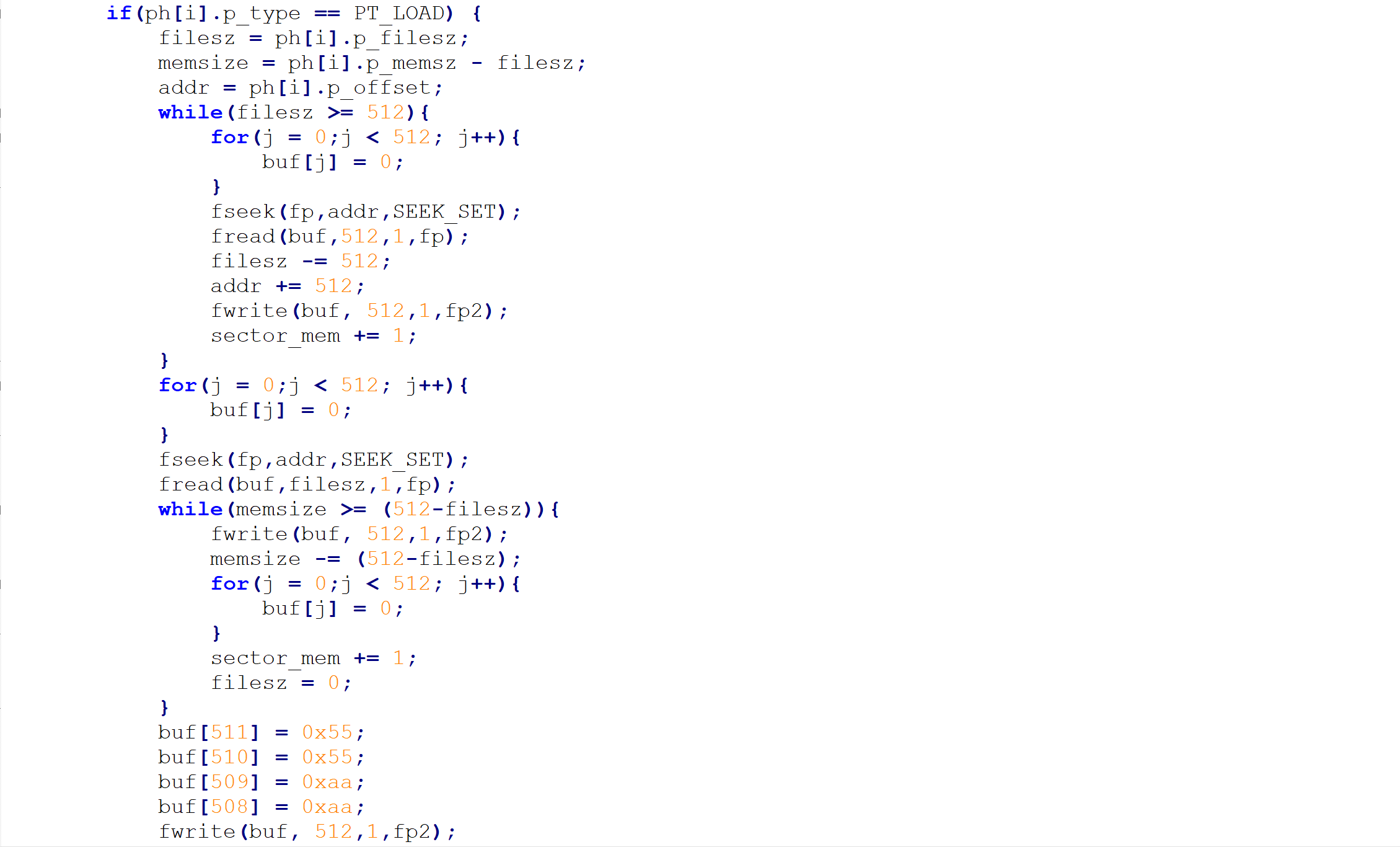
我发现原image文件在每个扇区结束处有aa55两个字节，应该是结束符。我尝试不加aa55，结果不能正常打印结果。

装载kernel时，我考虑了有多个p\_type值为PT\_LOAD的段，装载进镜像时还要根据两个段的p\_vaddr算出相应的装载地址。我用了for循环处理这个问题，不过kernel只有一个需要装载的段。

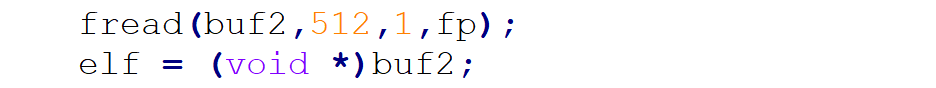
此外，我考虑了p\_memsz大于p\_filesz和p\_memsz大于512字节的情况。

# 关键函数功能

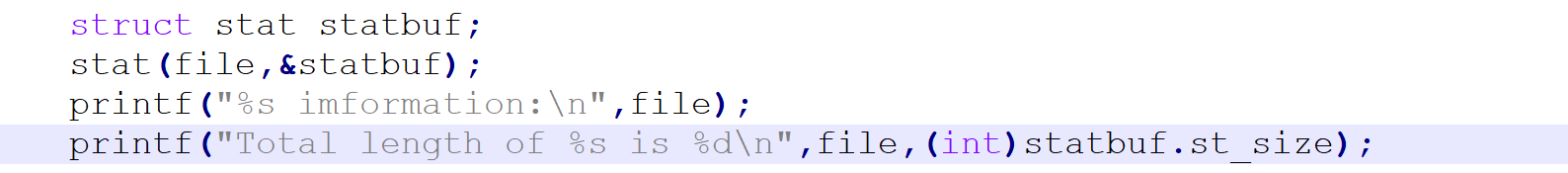
在load函数中，处理p\_memsz大于p\_filesz和p\_memsz大于512字节情况的代码如下：



我将ELF表头和程序表头读到buf2数组中，将可加载段单独读到buf数组中。读取ELF表头和程序表头时，应该将buf2大小设为一个页表的大小（4K），但考虑到kernel比较小，就设为一个扇区大小（512B）。



我使用stat.h库中的stat函数读取Bootblock和Kernel二进制文件的大小。



参考文献

无

▄