Project1 Bootloader设计文档

中国科学院大学

袁峥

[学号] 2015K8009929008

2017.9.25

# Bootblock设计流程

（1）Bootblock主要完成的功能

bootblock是在系统启动时首先执行的程序。

在任务二中，bootblock的功能是直接输出一个字符串“Welcome to OS”。根据任务中介绍的，PMON 下的字符输出为串口输出，地址为 0xbfe48000, 输出字符串可以通过不断向该地址写字符实现。因此只需要写一个循环不断将字符存储到相应内存地址就可以完成输出。

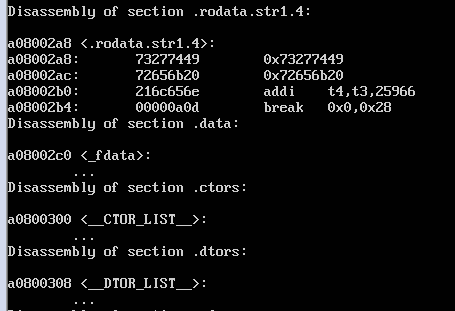
在任务三中，bootblock要完成的功能是将kernel的程序从硬盘加载到内存。kernel程序首先由kernel.c经过编译生成kernel的elf文件，然后在由createimage程序将bootblock和kernel一起制作成镜像，并拷到sd卡上。系统启动时，会自动将sd卡上的第一个扇区的内容（即bootblock）复制到内存0xa0800000处，并从0xa0800030处开始执行第一条指令。但sd卡上第二个扇区的内容（即kernel）不会被自动复制到内存，也就没有办法执行。因此任务三中将kernel拷到内存的任务将有bootblock来完成。

（2）Bootblock被载入内存后的执行流程

bootblock被载入内存后，将自动从0xa0800030开始执行第一条指令。

（3）Bootblock如何调用SD卡读取函数

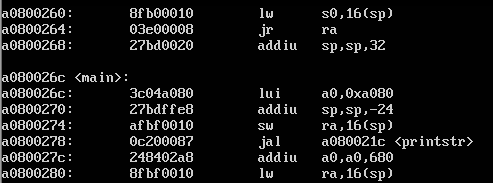
根据任务中介绍，PMON 中的读盘函数的地址为 0x8007b1a8，因为只需在bootblock中跳转到该地址就可以。由于MIPS是通过寄存器传参，根据MIPS指令规范，传参寄存器为$4,$5,$6和$7。读盘函数需要传递三个参数，第一个是读取的目的地址，即读取的数据在内存存放的位置，由于kernel将要被放在sd中第二个扇区，一个扇区大小为512，相应的对应到内存中也就是0xa0800200，而且在Makefile文件中也可以找到该地址。第二个参数为SD卡内部的偏移量，从该处开始读取。同样的，由于kernel被createimage 文件制作在了image的第二个扇区，因此偏移量为0x200，也就是512。第三个参数为要读取的字节数，这个数据可以通过反汇编生成的kernel文件来获取，通过mipsel-linux-objdump -D kernel命令，我们可以查到反汇编代码。



红框处是该输出的最后一行，加上该语句本身，我们算出kernel的大小为0x10b，因此读取的字节数必须大于该大小，当然，取个更大的值也可以。

（4）Bootblock如何跳转至kernel入口

先要找到kernel中的main函数地址，同样通过反汇编来查看。



因此kernel的入口地址即为0xa080026c，只需在bootblock完成将kernel载入内存后再跳转到该地址，便可以接着运行kernel程序。

（5）任何在设计、开发和调试bootblock时遇到的问题和解决方法

在完成该部分时，首先遇到的问题是上学期汇编语言课程学习的是X86的汇编，对MIPS指令集的了解较少，因此在编写bootblock.s先查看了MIPS指令手册，大致了解了有哪些指令可以用来使用。

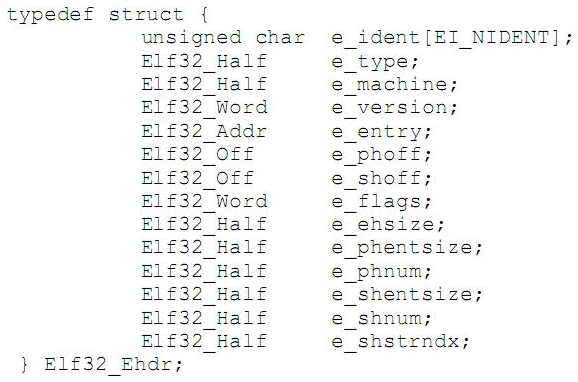
其次，在调用函数时，一开始使用的是j指令，在反复尝试发现不成功后，我查看了相关资料，发现调用函数时应该使用jal指令，该指令会自动将PC+8的地址放置到ra（$31）寄存器，然后在子程序中使用jr $31便可以调回原程序。

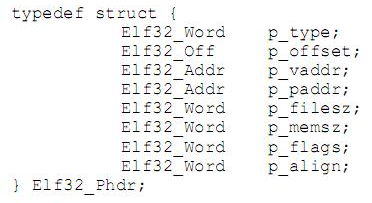
# Createimage设计流程

（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

bootblock和kernel编译后的二进制文件是elf文件，其中包含了elf文件头、程序头表及各程序段，其中各程序段中有一些为可装载段，而这些段也就是需要通过createimage程序生成到image文件中的内容。

（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

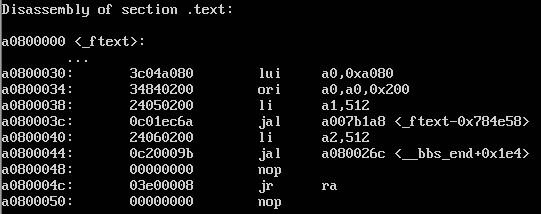


上图为elf文件头的结构，首先在elf文件头中我们可以通过e\_phoff得知程序头表在文件中的偏移，从e\_phentsize得知每一个程序头的大小，从e\_phnum得知程序头的数量，因此我们可以先找到程序头表的开始位置，然后从第一个程序段开始检查。

上图为程序头的结构，我们首先要判断p\_type，如果该数据为PT\_LOAD，那么就是我们真正要写入image文件的内容，也就是可执行代码。该程序段的位置可由p\_offset得知，表示该程序段在文件中的偏移，通过p\_filesz可以得出该程序段的大小。通过遍历所有程序头可以完成所有程序段的搜索。

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

我们在将kernel二进制文件中的程序段加载进image时，可以记录下加载的程序段的总长度。



通过反汇编bootblock文件，我们得到如上信息，可以看到，在a0800040的地址写入了需要读取的字节数，也就是我们之前计算的程序段的总长度。从地址可以看出，字节数被写在了bootblock文件开始后的0x40偏移，也就是image文件中的0x40位置，由于是小尾端，因此只需要将iamge中0x40处开始两个字节的大小修改为程序段总长度即可。

（4）任何在设计、开发和调试createimage时遇到的问题和解决方法

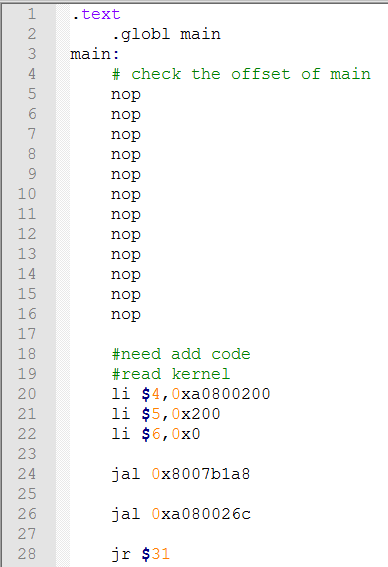
在此次实验之前，对ELF文件格式不太了解，因此在完成此次实验时首先先完整了解了ELF文件的格式。并学习了如果通过各种头文件来找到所需要的数据，并定位到ELF文件中的相应段。

在写createimage.c的过程中，指针操作较多，因此在编写时十分容易出错。而且在调试的过程中由于只能通过上板查看结果来判断是否成功，因此调试的环节也不太方便，前前后后插拔读卡器和实验板有近二三十次。

在编写的过程中，也参考了上学期计算机组成原理实验中有一个实验所提供的代码，其功能也是加载elf文件，因此从中得到了不少的借鉴。

# 关键函数功能

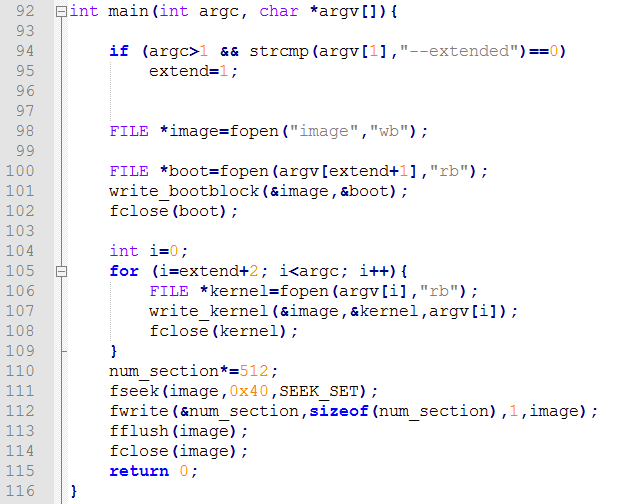
1、bootblock.s



这个程序整体思路比较清晰，显示准备好了读盘函数所需的三个参数，然后调用读盘函数，此时kernel已经被加载到内存，于是跳转到相应的首地址执行kernel程序。

其中第22行参数为0是因为该参数需要在createimage程序中将其进行修改。

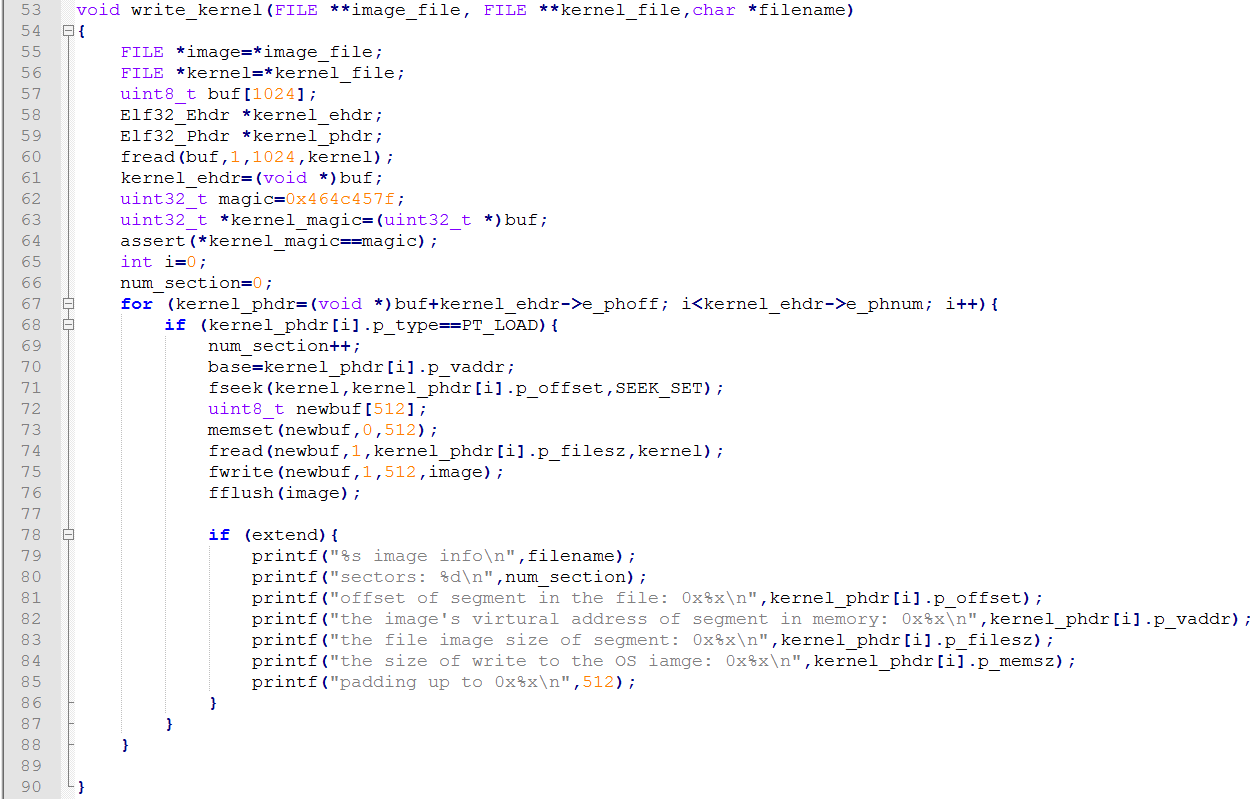
2、createimage.c



主程序主要是完成了—extended选项的判断，并且分别调用write\_bootblock()和write\_kernel()，并且在111行完成了对bootblock程序中需要读盘的字节数进行了修改。



write\_bootblock函数首先解析了elf头文件，判断幻数是否正确，然后找到程序段表，并逐段进行解析，若判断为可装载段，先找到该程序段在文件中的偏移并将文件指针指向该处，然后按照程序段大小读取相应长度的内容，并将其写入image文件内。且如果在调用时有—extended选项，则输入相关信息。



write\_kernel函数的内容与write\_bootblock几乎一致。