Project1 Bootloader 设计文档

中国科学院大学
段宏键
2017.09.26

1. 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。

（2）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。

（3）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。

（4）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。

（5）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。

2. 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。

```
...
Disassembly of section .data:

a08000a0 <_fdata>:
...
```

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

```
Bootblock Information:
程序头表条目数目: 1
tne number of blocks: 1
file size: 110
memory size: 110
segment offset in file: 60
segment offset in memory: 0
imge align: 10

Kernel Information:
程序头表条目数目: 1
tne number of blocks: 6
file size: a90
memory size: a90
segment offset in file: 60
segment offset in memory: 200
imge align: 10stu@ubuntu:/mnt/share_file/task_3_sta
```

（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。

3. 关键函数功能

Bootblock.s

```
    .data
num:
    .word 0x0000000
    .text
    .globl main

main:
    # check the offset of main
    nop
    nop
    nop
    nop
    nop
    nop
    nop
    nop
    nop
    nop
    nop

    la $8,num
    lw $9,($8)

    li $4, 0xa0800200
    li $5, 0x200
    li $6, 0x0
lod:
    beq $0,$9,ed
    addiu $6,$6,0x200
    addiu $9,$9,-1
    j lod

ed:
    jal 0x8007b1a8
    jal 0xa080026c

ll:
    nop
    j ll
```

Createimage.c

Main:

```
15  int main(int argc, char *argv[]){
16
17      FILE* bootblock;
18      Elf32_Ehdr bk_ehdr;
19      Elf32_Phdr* bk_phdr;
20      bk_phdr=(Elf32_Phdr*)malloc(sizeof(Elf32_Phdr));
21      bk_phdr=read_exec_file(&bootblock,"bootblock",&bk_ehdr);
22
23
24      FILE* kernel;
25      Elf32_Ehdr kl_ehdr;
26      Elf32_Phdr* kl_phdr;
27      kl_phdr=(Elf32_Phdr*)malloc(sizeof(Elf32_Phdr));
28      kl_phdr=read_exec_file(&kernel,"kernel",&kl_ehdr);
29
30      FILE* image;
31      image=fopen("image","w+");
32      if(!image) perror("error");
33      write_bootblock(&image,bootblock,&bk_ehdr,bk_phdr);
34      write_kernel(&image,kernel,&kl_ehdr,kl_phdr);
35
36      int n = count_kernel_sectors(&kl_ehdr,kl_phdr);
37      record_kernel_sectors(&image,&kl_ehdr,kl_phdr,n);
38      extended_opt(bk_phdr,&bk_ehdr,kl_phdr,&kl_ehdr);
39  }
```

Read_elf

```
42  Elf32_Phdr * read_exec_file(FILE **execfile, char *filename, Elf32_Ehdr *ehdr)
43  {
44      Elf32_Phdr* ph;
45      ph=(Elf32_Phdr*)malloc(sizeof(Elf32_Phdr));
46
47      if(filename=="bootblock"){
48          if(!(*execfile=fopen("bootblock","r")))
49              printf("error's number:%d\n",errno);
50      }
51      else{
52          if(!(*execfile=fopen("kernel","r")))
53              printf("error's num:%d\n",errno);
54      }
55      fseek(*execfile, 0, SEEK_SET);
56      fread(ehdr,sizeof(Elf32_Ehdr),1,*execfile);//read the elfheadder
57      fseek(*execfile,ehdr->e_phoff,0);
58      fread(ph,ehdr->e_phnum*ehdr->e_phentsize,1,*execfile);
59      return ph;
60  }
```

Write image & count sectors

```
61
62  void write_bootblock(FILE **imagefile, FILE *boot_file,
63  Elf32_Ehdr *boot_header, Elf32_Phdr *boot_phdr)
64  {
65      uint8_t buf[512];
66      memset(buf, 0, 512);
67      fseek(boot_file,boot_phdr->p_offset,0);
68      fread(buf,boot_phdr->p_filesz,1,boot_file);
69      fwrite(buf,1,512,*imagefile);
70  }
71
72  void write_kernel(FILE **image_file, FILE *kernel_file,
73   Elf32_Ehdr *kernel_ehdr, Elf32_Phdr *kernel_phdr)
74  {
75      int seg_num = count_kernel_sectors(kernel_ehdr,kernel_phdr);
76      uint8_t buf[512*seg_num];
77      memset(buf, 0, 512 * seg_num);
78      fseek(kernel_file,kernel_phdr->p_offset,0);
79      fread(buf,kernel_phdr->p_filesz,1,kernel_file);
80      fseek(*image_file,512,0);
81      fwrite(buf, 512*seg_num, 1 ,*image_file);
82  }
83
84
85  int count_kernel_sectors(Elf32_Ehdr *kernel_header,
86   Elf32_Phdr *kernel_phdr){
87      int size = kernel_phdr->p_filesz;
88      int seg_num;
89      seg_num = size / 512;
90      if(seg_num * 512 != size) seg_num++;
91      return seg_num;
92  }
```

To bootblock

```
void record_kernel_sectors(FILE **image_file,
Elf32_Ehdr *kernel_ehdr, Elf32_Phdr *kernel_phdr, int num_sec){

    fseek(*image_file, 0x0a0, SEEK_SET);
    int *num = &num_sec;
    fwrite(num,4,1,*image_file);
}
```

参考文献

[1] MIPS32 手册

[2] Project1-Bootloader 任务书

■