Project1 Bootloader 设计文档

中国科学院大学
张磊
2019 年 9 月 13 日星期五

1. Bootblock 设计

（1）Bootblock 主要完成的功能是将内核从硬盘读取到内存的指定位置；
（2）Bootblock 通过将 read_SD_card 所需要的参数按照 mips 参数调用规范放到指定的寄存器中，然后通过 jal 指令，跳转到函数入口地址，达到调用函数的目的；
（3）Bootblock 在完成将 kernel 加载到内存制定位置后，再次用 jal 指令即可跳转到 kernel 在内存中的位置；
（4）一开始不太清楚怎么使用函数入口地址调用函数，对多个参数的函数的参数存放顺序也不确定，后来查阅资料后才弄明白；

2. Createimage 设计

（1） 写入 SD 卡的 image 文件是由 Bootblock 和 Kernel 编译后的二进制文件中的有效数据组合而成的；
（2） 通过读取 Bootblock 和 Kernel 的二进制文件的文件头 ELF，找到其中对应的程序头表，程序头表中记录了这个 ELF 文件的程序的个数，进入程序头，可以得到当前程序有效数据里文件头的偏移 offset，以及 filesz，memsz，实际开发中从 Kernel 的 ELF 文件中拷贝了 1 个 segment；
（3） 在 creatimage.c 中可以获取到 kernel 的大小并写到 image 中 bootblock 部分的末尾；
（4） 由于初始代码中没有注释，很多变量所代表的意义都不清楚，导致实验时进度较慢；

3. Bonus 设计（可选）

（1）可以将 bootloader 加载到内存的其他区域，只要不被 Kernel 覆盖即可这样就能在加载完 kernel 用 jal 指令后跳转到新的 kernel 的内存地址；
（2）能，仅需在读取完 kernel 后在返回 PC+1 的位置即可；

4. 关键函数功能

（1） Bootblock.s 文件

```
main:
    # 1) task1 call BIOS print string "It's bootblock!"

    la  $a0, msg
    lw  $t0, printstr
    jal $t0

    # 2) task2 call BIOS read kernel in SD card and jump to kernel start

    lw  $a0, kernel #addr
    li  $a1, 0x0200 #offset
    li  $a2, 0x0200 #size

    lw  $t0, read_sd_card
    jal $t0

    lw  $t0, kernel_main
    jal $t0
```

（2） Kernel.c 文件

```
    // Call PMON BIOS printstr to print message "Hello OS!"
    char *s = "Hello OS! ";
    char *v = "Version:";
    char strnumb[20];

    void (*print)(char *);

    print = PRINTSTR;
    (*print)(s);
    (*print)(v);


    int power;
    int j = value;

    for(power = 1; j > 10; j /= 10)
        power  *= 10;

    int i;
    for(i = 0;power > 0;power /= 10)
    {
        strnumb[i] = '0' + value/power;
        value %= power;
        i++;
    }

    strnumb[i] = '\0';

    (*print)(strnumb);
```

（3） createimage.c 文件

（返回程序头）

```
Elf32_Phdr *read_exec_file(FILE *opfile)
{
    Elf32_Phdr *ph;
    Elf32_Ehdr *eh;

    ph = (Elf32_Phdr *)malloc(sizeof(Elf32_Phdr));
    eh = (Elf32_Ehdr *)malloc(sizeof(Elf32_Ehdr));

    fread(eh,sizeof(Elf32_Ehdr),1,opfile);

    fseek(opfile,(long)(eh->e_phoff),SEEK_SET);

    fread(ph,sizeof(Elf32_Phdr),1,opfile);

    return ph;
}
```

（写 bootblock）

```
void write_bootblock(FILE *image, FILE *file, Elf32_Phdr *phdr)
{
    char buf[520];
    Elf32_Off p_offset;
    Elf32_Word p_filesz;

    p_offset = phdr->p_offset;
    p_filesz = phdr->p_filesz;

    fseek(file,(long)(p_offset),SEEK_SET);
    fread(buf,(int)(p_filesz),1,file);
    fwrite(buf,(int)(p_filesz),1,image);

    return;
//  fseek(image,512L,SEEK_SET);
}
```

（写 kernel）

```
void write_kernel(FILE *image, FILE *knfile, Elf32_Phdr *Phdr, int kernelsz)
{
    char buf[520];
    Elf32_Off p_offset;
    Elf32_Word p_filesz;

    p_offset = Phdr->p_offset;
    p_filesz = Phdr->p_filesz;

    fseek(knfile,(long)(p_offset),SEEK_SET);

    int i = 0;
    for(i = 0; i < kernelsz;i++)
    {
        fread(buf,512,1,knfile);
        fwrite(buf,512,1,image);
    }

    return;
//  fseek(image,1024L,SEEK_SET);
}
```

参考文献——无