Project1 Bootloader 设计文档

中国科学院大学
张旭
2017/9/26

1. 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。

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

   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。

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

   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。

3. 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。

4. 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。

3. 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：

```
if(ph[i].p_type == PT_LOAD) {
    filesz = ph[i].p_filesz;
    memsize = ph[i].p_memsz - filesz;
    addr = ph[i].p_offset;
    while(filesz >= 512){
        for(j = 0;j < 512; j++){
            buf[j] = 0;
        }
        fseek(fp,addr,SEEK_SET);
        fread(buf,512,1,fp);
        filesz -= 512;
        addr += 512;
        fwrite(buf, 512,1,fp2);
        sector_mem += 1;
    }
    for(j = 0;j < 512; j++){
        buf[j] = 0;
    }
    fseek(fp,addr,SEEK_SET);
    fread(buf,filesz,1,fp);
    while(memsize >= (512-filesz)){
        fwrite(buf, 512,1,fp2);
        memsize -= (512-filesz);
        for(j = 0;j < 512; j++){
            buf[j] = 0;
        }
        sector_mem += 1;
        filesz = 0;
    }
    buf[511] = 0x55;
    buf[510] = 0x55;
    buf[509] = 0xaa;
    buf[508] = 0xaa;
    fwrite(buf, 512,1,fp2);
```

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）。

```
fread(buf2,512,1,fp);
elf = (void *)buf2;
```

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

```
struct stat statbuf;
stat(file,&statbuf);
printf("%s imformation:\n",file);
printf("Total length of %s is %d\n",file,(int)statbuf.st_size);
```

参考文献

无

■