Project1 Bootloader 设计文档

中国科学院大学
吴俊亮
2020 年 9 月 22 日

1. Bootblock 设计

Bootblock 主要完成的功能包括打印字符串“It's bootblock!”、将 kernel 代码拷入内存中和跳转至 kernel 入口。为调用 SBI 提供的 SD 卡读取函数，需要将内存地址（0x50201000）存入 a0 寄存器，将 kernel 占用的扇区数量从指定位置（0x502001fc）读出并存入 a1 寄存器，将起始扇区号（1 号）存入 a2 寄存器，最后使用 SBI_CALL SBI_SD_READ 调用 SD 卡读取函数。链接器脚本已经将内核的入口函数地址置于 0x50201000，所以直接使用跳转指令跳转到该地址即可。我在开始设计 bootblock.S 时不理解部分汇编指令，不清楚 li、la 之间的区别，导致汇编时报错，但在网上读了部分 RISC-V 汇编的资料[1]后顺利完成了这部分实验。

kernel 的入口函数是 head.S 中的_start，这段代码主要完成清空 bss 段所在内存、设置栈指针和跳转到 kernel.c 的 main 函数三个功能。我在开始设计时不清楚如何确定 bss 段所在的内存地址范围，于是使用 objdump 指令查看 bss 段的起始地址和大小并将其添加到代码中。后来发现如果改动 kernel.c，bss 段的起始地址和大小也会随之变化，直接使用绝对地址不具有通用性。在网上看到了 Linux 源码中清空 bss 段的代码[2]后了解到 bss 段起始和结束地址可能有标签指示，随后我发现链接器脚本文件 riscv.lds 中定义了 bss 段开始和结束的两个标签__bss_start 和__BSS_END__，使用标签即可解决修改 kernel.c 造成的 bss 段地址和大小变化影响代码的问题。

2. Createimage 设计

createimage 的主要工作是将 bootblock 编译后的二进制文件和 kernel 编译后的二进制文件中各个程序头指示的段读出，并写在 image 文件中正确的位置。对于一个二进制文件，首先读取位于文件开头的文件头，文件头中的 e_phnum 域和 e_phoff 域指示了该二进制文件的程序头个数和程序头在文件中的位置。依次读取文件的程序头，程序头中的 p_filesz 域和 p_memsz 域指示该段在 ELF 文件中的大小以及 ELF 程序被加载器加载后在内存中所占大小，p_offset 域指示可执行代码在文件中的位置。本实验 kernel 的可执行代码中有 2 个 segment，

但其中一个大小为 0，所以可以视为只有一个 segment。

为了让 bootblock 获取到 Kernel 的大小，以便进行读取，按照讲义要求将 kernel 的大小置于第一个扇区的倒数第四个字节处，即 image 文件的第 508 个字节处。将 kernel 的可执行代码扩展到 5-8 个扇区大小，使用 qemu 和 gdb 调试发现 bootblock 可以正确读取 kernel 的大小。

3. A-Core/C-Core 设计（可选）

如图 1 左侧所示，bootblock 在执行到 0x50200032 处的 ecall 后即进入 SBI 的 SD 卡读取函数，内存的内容将被覆盖。执行完 ecall 后 pc 的值为 0x50200036，内存中已经是内核的代码（head.S 和 kernel.c 产生的代码），如果直接执行肯定会出错，可以想到只要将拷贝后内存中 0x50200036 处的指令设置为我们想要执行的 bootloader 在加载 kernel 后的其他工作的指令，并在完成后跳转至 head.S 的开头处，即可以完成重定位的过程。

如图 1 右侧所示，在 head.S 中的 j main 指令（跳转到 kernel 的 main 函数）后加入特定数量的 nop 以及我们想要执行的刷新缓存指令和跳转指令，并使 0x50200036 处的指令恰好为需要执行的第一条指令即可。如果 bootloader 在加载 kernel 后还有其他工作要完成，在 fence.i 和 j 0x50200000 两条指令之间添加新增的工作所需的指令即可。

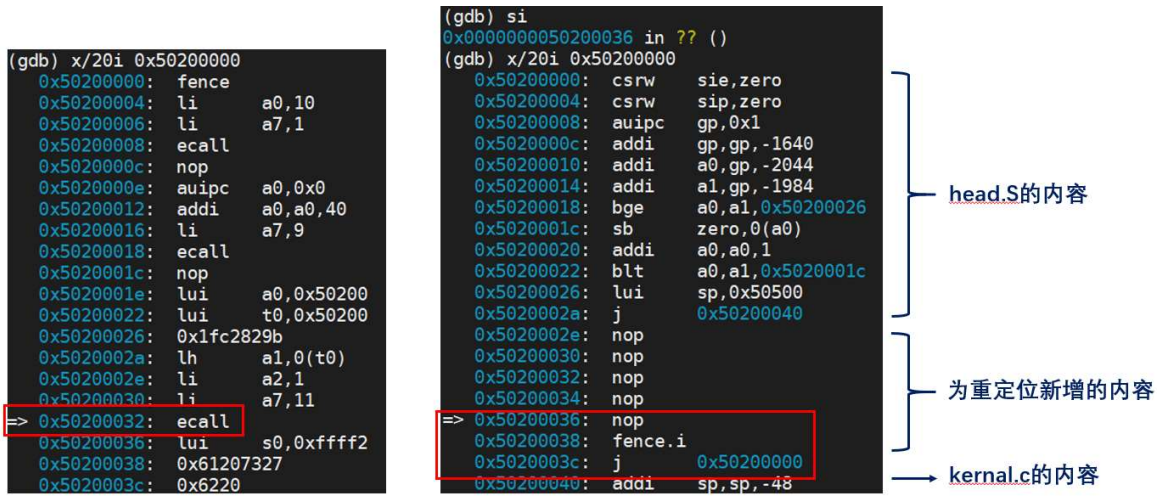

图 1 重定位的实现

4. 关键函数功能

以下为 head.S 中清空 bss 段的代码，首先将 bss 段的起始地址和结束地址存入 a0 和 a1 两个寄存器中。若结束地址大于等于起始地址，则 bss 段长度为 0，无需清空；否则逐字节

将 bss 段所在的内存空间置为 0。

```
  la a0, __bss_start
  la a1, __BSS_END__
  ble a1, a0, clear_bss_done
clear_bss:
  sb zero, 0(a0)
  addi a0, a0, 1
  blt a0, a1, clear_bss
clear_bss_done:
```

以下为 createimage.c 中 write_segment 函数的代码，该函数的功能是根据程序头 phdr 将文件 fp 的指定内容拷贝到文件 img 中，nbytes 所指变量用于指示 img 中已写入的字节数。首先将 phdr.p_memsz 向上对 512 取整，存到 bytes 变量中，表示此次需要向 img 中写入的字节数；然后初始化一个 bytes 大小的数组；最后从 fp 中读取 phdr.p_filesz 个字节的内容，存在数组的前 phdr.p_filesz 个元素中，并将数组写入 img 中，更新 nbytes 所指的数据。至此就完成了一个程序头所指数据的拷贝。

```
static void write_segment(Elf64_Phdr phdr, FILE * fp,
                          FILE * img, int *nbytes)
{
    /* 512 bytes align */
    int bytes;
    if(phdr.p_memsz % 512 == 0) {
        bytes = phdr.p_memsz;
    }
    else {
        bytes = phdr.p_memsz + (512 - phdr.p_memsz % 512);
    }

    /* initialize the data array */
    char segment_file[bytes];
    int i;
    for(i = 0; i < bytes; i++) {
        segment_file[i] = 0;
    }

    /* write data to img */
    fseek(fp, phdr.p_offset, SEEK_SET);
    fread(segment_file, phdr.p_filesz, 1, fp);
    fseek(img, (*nbytes), SEEK_SET);
    fwrite(segment_file, bytes, 1, img);
```

```
    (*nbytes) += bytes;

    return;
}
```

参考文献

[1] RISC-V 嵌入式开发入门篇 2：RISC-V 汇编语言程序设计（中），https://blog.csdn.net/zoomdy/article/details/83657042

[2] RISC-V Linux 源码分析 (一)：kernel 启动，https://zhuanlan.zhihu.com/p/78049406