操作系统研讨课 实验报告

代瀚堃 2019K8009929051

一、实验中遇到的问题

1. 跳转指令中的相对偏移量

在完成 A-core 时，我原来使用了如下指令：

```
j kernel
```

按原来的想法，boot 把剩余指令拷贝到内核的后面，然后执行这条 J（伪）指令，跳转到内核入口，但实际上 kernel 这个标号会在链接时被转化为相对当前 PC 的偏移量，存储在指令中，但这一偏移量不一定是拷贝后的 J 指令所需要的，导致跳到内核后面以后，就没有再跳回来。

这个 bug 我花了很长时间才找到，因为之前一直在使用 Project1 自带的 run_qemu.sh 脚本，连接超时，而我通过 objdump，并在内核中显示之前 bootblock 放置的指令，发现 bootblock 拷贝的指令是正确的（显然，因为是原封不动搬过去的，甚至连偏移量都是一样的）。后来在助教的帮助下使用 gdb 调试，找到了问题。

另外，这个 bug 比较有趣的地方在于，我一开始用这种写法，不知道为什么，竟然跳转成功了，后来改动了一些地方，就再也没跳回去过，大概花了一天才解决。而且我之前的那一份代码没有提交，后来还修改了…所以还是应该谨记，一定要及时提交到 git!!!

2. 内核中节的补齐

这个 bug 也比较迷惑，我一开始的内核大小只有 1 个扇区，所以自己写的 createimage 还没有暴露出问题，后来在解决第一个 bug 的时候，为了排除自己 createimage 潜在的 bug，就使用了提供的 createimage，但后来跳转成功后，换回自己的 createimage，并将内核扩大到两个扇区，就出现了问题，原因是某些节（section）需要对齐，这个 bug 的现象比较明显，通过比对生成的镜像文件就能看出（当然，这很费眼睛），但它同样花费了我很长时间，因为我在修改的时候一直是以 mem_sz 作为循环条件的，但循环体的最后一个更新 read_byte 的语句增加的是节的大小，这个大小是在文件中的大小，即 file_sz，所以没有按时退出循环，导致段错误（buffer 越界），或者把多余的信息写入了镜像文件中，但我并没有意识到这个问题，所以无论怎么改也没办法把 0 补上，而 CLion 用起来也不熟练，只能用 gdb 或者 printf

调试。后来误打误撞把循环条件改作和 segment_byte，即 file_sz 作比较，才得到了预期的镜像文件。

```
    for(;read_byte<segment_byte;){
        fseek(fp,*section_header_start,SEEK_SET);
        fread(&shdr,ehdr.e_shentsize,1,fp);
        //get a section header
        *section_header_start+=ehdr.e_shentsize;

        int padding=shdr.sh_addr-ehdr.e_entry-read_byte;
        if(read_byte>0&&padding>0){
            read_byte+=padding;
        }

        fseek(fp,shdr.sh_offset,SEEK_SET);
        fread(buffer+read_byte,shdr.sh_size,1,fp);
        read_byte+=shdr.sh_size;
    }

    fwrite(buffer,1,mem_sz,img);
    *nbytes+=mem_sz;
    free(buffer);
```

3. 读完 SD 卡后没有用 FENCE.I 指令同步 L1 Cache

这个问题其实在 FAQ 里写到过，我在读完任务书时也看过，但我看到 bootblock 开头已经加了一条 FENCE，而且在 QEMU 上运行的时候也一直没出问题，所以后来也就没有在意过。到上板的时候就出问题了，boot 能正常启动，但是在选完内核，将要拷贝的时候就会打印出奇奇怪怪的字符串，验收时在助教的帮助下才找到了这个 bug。

二、还有待解决的问题

1. 制作镜像文件时分配的缓冲区

我用 malloc 分配的缓冲区大小根据 mem_sz 动态变化，想法是无论 mem_sz 是大还是小，都不会有浪费。但验收时助教指出，很多大内核的 mem_sz 很有可能会超过 malloc 可分配的空间，所以更合理的方法是每次分配一个固定的大小（比如几个 MB），分批将内核中的各个 section 搬运到镜像中。