Project1 Bootloader设计文档

中国科学院大学

刘杰

2018.9.26

# Bootblock设计流程

1. Bootblock主要完成的功能是将操作系统的代码搬运到内存。

（2）Bootblock被载入内存后通过调用BIOS函数read\_sd\_card读取SD卡上操作系统的内核，并放置到内存的制定位置0xa0800200，读取完内存后，Bootblock最后跳转到内核代码入口处开始执行。

（3）在bootblock.s汇编文件中加入read\_sd\_card函数调用相关指令实现读取SD卡上内核到内存指定位置，具体实现是先用三个LI指令把read\_sd\_card函数的三个参数顺序存入a0-a2这三个寄存器中，再使用JAL指令跳转到read\_sd\_card函数的入口地址。

（4）在bootblock.s汇编文件加入调用read\_sd\_card函数相关指令后，再使用JAL指令跳转到内核镜像的入口地址，是在内核文件的最前面，放到内存后地址为0xa0800200。

（5）在PMON界面，输入loadboot后重复打印HELLO OS，因为一开始我使用的是JR指令跳转到内核镜像的地址，此时31寄存器保存的是执行JR前一条JAL指令的返回地址，所以从内核镜像返回后又回到JR指令执行，陷入死循环。把JR指令改为JAL指令即可。

# 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都是ELF文件，在最终生成的image镜像中不包含这两个ELF文件中用于分析代码和数据位置的header，只保留segment部分。

（2）打开ELF文件后，先读入ELF header，再通过ehdr结构体中e\_phoff找到程序头位置，读入programs header，在phdr结构体中p\_offset找到可执行代码位置，p\_filesz得到可执行代码大小。

（3）通过调用函数计算kernel的扇区数，并调用函数写入bootblock的os\_size位置来通知bootblock内核的大小，方便读取kernel。

（4）写入bootblock时应该在image偏移量为0x1fe依次写入0x55，0xAA确保写入的bootblock块为512B大小。

# 关键函数功能

1. bootblock.s代码片段

main:

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

la $a0, msg //写入printstr参数

jal 0x8007b980 //调用printstr函数

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

li $a0, 0xa0800200

li $a1, 0x00000200

li $a2, 0x00000090 //依次写入read\_sd\_card三个参数

jal 0x8007b1cc //调用read\_sd\_card函数

jal 0xa0800200 //跳转至内核入口地址

# while(1) --> stop here

1. createimage代码片段

Elf32\_Phdr \*read\_exec\_file(FILE \*opfile)

{

uint8\_t numread, num\_phdr;

Elf32\_Ehdr ehdr;

Elf32\_Phdr \*phdr = NULL;

//read in elf header

fseek(opfile, 0, SEEK\_SET);

numread=fread(&ehdr, 1,sizeof(Elf32\_Ehdr), opfile);

assert(numread == sizeof(Elf32\_Ehdr));

num\_phdr = ehdr.e\_phnum;

//read in program headers

assert(ehdr.e\_phentsize == sizeof(Elf32\_Phdr));

fseek(opfile, ehdr.e\_phoff, SEEK\_SET); //set read position to phdr

phdr = calloc(sizeof(Elf32\_Phdr), num\_phdr); //allocate memmory to store phdr

numread = fread(phdr, sizeof(Elf32\_Phdr),num\_phdr, opfile);

assert(numread == num\_phdr);

return phdr;

}

void write\_bootblock(FILE \*image, FILE \*file, Elf32\_Phdr \*phdr)

{

size\_t numread;

char buffer[512];

/\* set file offsets \*/

fseek(file, (\*phdr).p\_offset, SEEK\_SET);

numread=fread(buffer, 1, (\*phdr).p\_filesz, file);

assert(numread == (\*phdr).p\_filesz);

// write bytes

fseek(image, 0, SEEK\_SET);

numread = fwrite(buffer, 1, (\*phdr).p\_filesz, image);

assert(numread == (\*phdr).p\_filesz);

//write terminator

fseek(image, 0x1fe, SEEK\_SET);

fputc(0x55, image);

fputc(0xAA, image);

}