prj1设计文档

设计思路：通过make命令将文件bootblock、kernel0、kernel1 编译之后，会分别形成对应的可执行文件。接着由createimage.c将三个可执行文件合并生成image（make image），只留下可执行文件的段的信息，即可以直接开始执行的指令信息。image的每一段会按照扇区补齐、并被拷贝到SD卡内。开始运行的时候，BIOS（BBL）会把SD卡第一个扇区的内容（bootblock）拷贝出来，放在内存指定地址中，然后开始执行bootblock。

debug时遇到的主要问题：

1.在增加了双内核功能后，没有重定位write_segment函数中写img的起始地址。

单内核的时候，write_os_size函数中会重定位img的指针到0x000001fc，但是是在kernel的write_segment执行完了之后才是它的write_os_size，所以对于向image中写kernel段使的img起始地址没有影响。增加了双内核功能后，第一个文件执行了write_os_size后，指针到了0x000001fa，后面还有第二个文件的读写，故需要在写segment之前重定位一下img指针。

2.重定位了bootblock后，跳转到0x50200000执行kernel时，未刷新cache。

将kernel移到0x50200000，并跳转到0x50200000开始执行指令时，我没有刷新cache。原本是bootblock的指令在0x50200000那一个扇区，将bootblock移到高地址0x50400000后，cache中0x50200000那一个扇区对应的块的内容仍然不变。后续跳到kernel执行时，cache就认为之前的bootblock指令就是现在需要的指令(地址相同）。因此，在上板的时候，会出现多次进行拷贝的情况，即本应该进入kernel的时候，却再次执行了bootblock中的指令。在qemu上表现正常，可能是qemu实现时没有用到cache。

3.扇区数量计算的错误。

在拷贝了2048的大核之后，出现了data error的报错，调试许久没有找到原因。经过检查读入image的kernel1的扇区数和调用函数SBI_SD_READ写入内存时的扇区数相同，各参数也没有问题。后来经过询问同学得知，大核的扇区数大小是40，而我这里算的是42。经过核查发现，createimage.c中写入image时写入扇区数量的计算有问题。write_segment时，最后一次会读入余量（phdr.p_filesz%512个字节）到数组f（之前已经初始化为全0）中，然后将f写入img。这里如果段的大小是正好是512的倍数，即整扇区的话，确实没有数据读入到f中，但是还是写了一个全0的512个字节的数组f到了img中。自己的kernel大小不一定是整扇区数，并且小kernel的第二段中没有内容，所以多写了全0也没有问题。这里因为大kenel的两段segment都多写了一个全0扇区，故在data_check时，第二段segment起始数据的值变成了0。

实际上，就是少了一个条件判断，即下图的if语句。

```
if((phdr.p_filesz%512)!=0){
  fread(f,1,phdr.p_filesz%512,fp);
  fwrite(f,1,512,img);
  (*nbytes)+=512;
}
```