# 操作系统实验日志

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 学号 | 201808010718 | 姓名 | 肖鹏 | 专业年级班级 | 智能1802 |
| 实验日期 | 2019.10.10 | 实验项目 | 第三天：进入32位模式并导入C语言 | | |

## 一、实验主要内容

1、扇区读取

**内容：**

读取扇区数据到内存指定地址;

通过汇编代码实现读取一定数目的扇区，每次完成一个扇区的读取时，通过JNC指令调整段寄存器[ES:BX] 为下一个扇区准备空间。同时通过SI寄存器计数统计读取失败次数，当达到既定次数时停止读取并抛出错误。

**重点总结：**

a）缓冲区地址的理解：缓冲区地址是内存地址，也就是说在软盘上的程序不能直接在软盘上运行，而是要装载到内存中运行，而具体放在内存中的那个起始地址，就是由段寄存器的值决定。为了充分利用CPU的处理容量，使用两个寄存器[ES:BX]组合而成的段寄存器来表示内存起始地址。段寄存器并不是简单地把两个8-bits寄存器合成16-bits寄存器，表示的内存起始地址的计算是ES\*16（10）+BX，这样二者带入0xFFFF时，能够取到最大的内存范围约为1M。

b）对于ES寄存器，不能够直接对ES进行ADD操作，需要通过AX作为中间寄存器来实现扇区的切换。

c）注意，程序中读取的规则是：对于一个柱面i，读取完所有扇区上的柱面i才算对读取完成一个柱面。

d）频繁涉及到的寄存器：

AL——每次处理的扇区数目

DL——驱动器号（软盘序号）

DH——磁头号（0正面、1反面）

CL——扇区号（从1起始）

CH——柱面号

ES:BX——缓冲地址

**关键代码及注释：**



2、FAT12文件系统

**内容：**

学习FAT12文件系统，理解启动区的512-bytes中的格式，以及包括的固定指令是哪些；之后，理解为什么向一个空软盘保存文件时，为什么文件名会写在0x002600以后、文件内容会写在0x004200以后的空间。首先，说明一下1.4M的软盘一共有2880个扇区1474560-bytes。2880个扇区分为5部分：MBR引导记录、FAT1表、FAT2表、根目录、数据区。

**重点总结：**

**a） 各部分占用扇区情况：**

*启动区起始地址：0x0000*

*FAT表1起始地址：0x0200*

*FAT表2起始地址：0x1400*

*根目录区起始地址：0x2600*

*用户数据起始地址：0x4200*



当储存一个文件时，除了启动区，其余的区域都会被修改——文件名会存放在根目录区（0x2600后）、文件内容会存放在用户数据区（0x4200后）；

**b） FAT表1、FAT表2的联系与内容**

这两个表完全相同，FAT表2存在的意义就是在FAT1表出现错误时，用来修复FAT表1。一个FAT表占据9个扇区（一个扇区512-bytes），12-bits组成一个簇，所以FAT一共3072个簇。

簇中的12-bits表示这个簇指向的另一个簇的编号，这12-bits是组合而来的，如现有0xF0 0x04 0x5F三个bytes，那么第一个组成的簇是由第一个byte的8-bits作为簇的低8-bits、第二个byte的高4-bits作为簇的高4-bits（在Intel中采用大端法，即高有效位在高地址，所以0x04中4为高字节），即0x04F0为第一个簇；同理，第二个byte的低4-bits作为第二个簇的低4-bits、第三个字节的8-bits作为第二个簇的高8-bits。

在FAT文件表中，0号簇和1号簇是用来做坏簇标记0xFF0和结尾标志0xFFF。

**c）** 启动区的有着固定的格式，存储着一个固定的数据结构，大小为512-bytes。

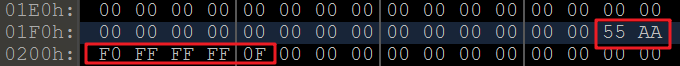
启动区的最后两个字节若为55 AA，则表示BIOS 会在启动时将这个扇区读取到0:7C00-0:7DFF处，然后跳转到0:7C00h处执行，操作系统即用此达到引导系统的目的。

启动区的第一条指令是一条跳转指令（约定为3字节大小），用来跳转到操作系统引导代码的部分。此外，其余的字节皆有固定的格式，由于本次实验涉及不大便不赘述，如图：



**关键代码及注释：**

以*day03\_harib00f中的haribote.img、haribote.lst、haribote .sys*为例，观察印象文件中不同区域数据被修改的情况：

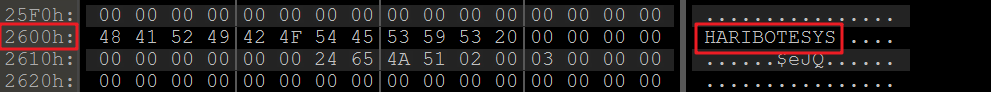
a）

*（启动区结束部分、FAT表1开始部分）*

*（FAT表2开始部分）*

1.启动区（0000h-01FFh）最后两字节为55 AA；

2.FAT表1（0200h-13FFh）开始的5个字节分别表示：存储介质、（第2、3字节表示）FAT文件分配表标识符、（第4、5字节组成簇0x0FFF）表示该簇为为文件的最后一簇。与FAT表2的内容相同。

b）根目录区写入了文件名（大写）、数据区域写入了程序中的指令。

*（根目录开始部分）*

*（数据区开始部分）*

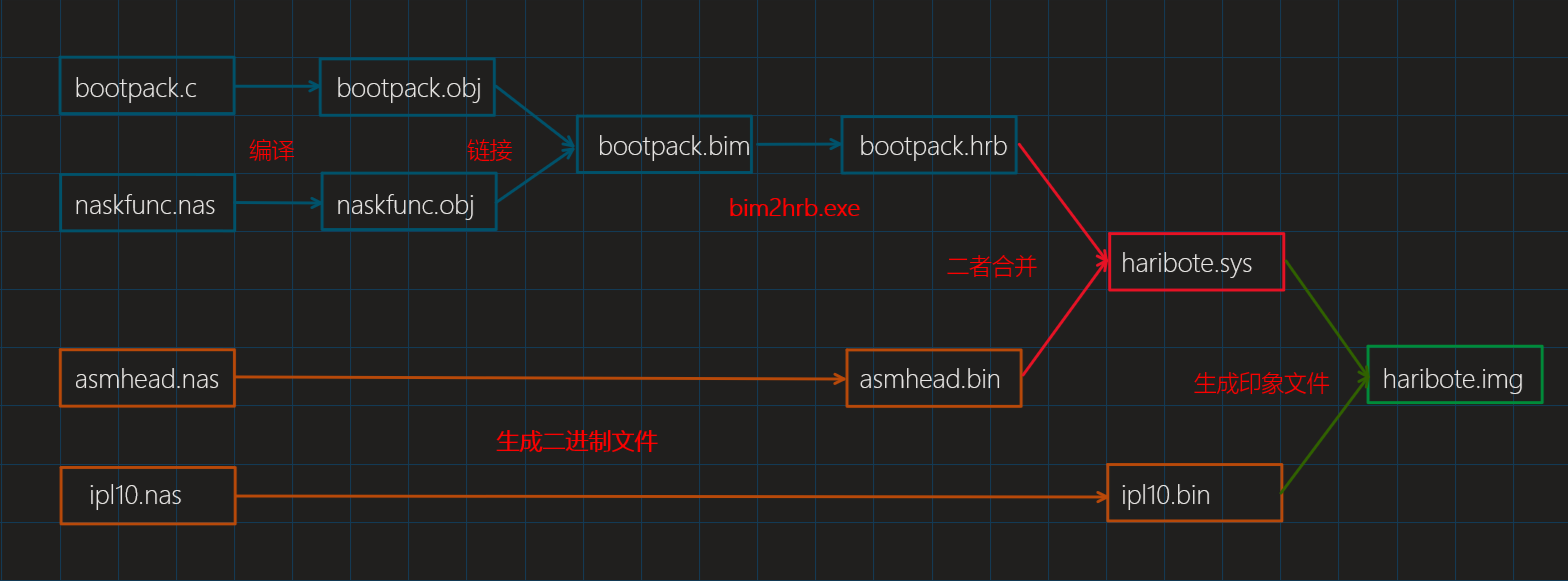
*（haribote.lst机器码-汇编指令对照）*

3、导入C语言程序

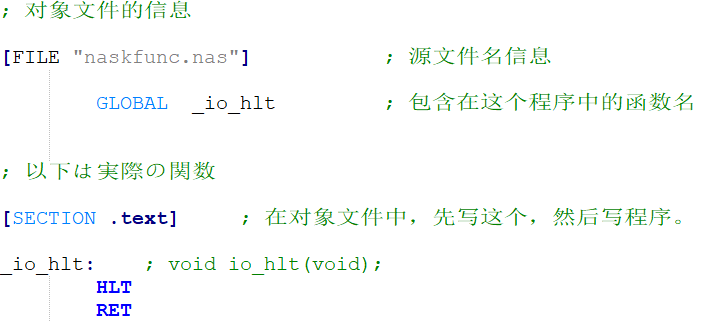
**内容：**

导入C程序；想要运行自己写的C程序，相比之前实验中仅涉及到汇编是需要用到更多文件：（以本次实验为例）首先编写一个bootpack.c，在其中写好C函数，为了实现一些BIOS函数，还需要通过汇编编写一个“函数定义”的naskfunc.nas文件，这样二者链接才能生成能够执行的二进制文件。之后的步骤就是如何合并到之前实验的印象文件中了。

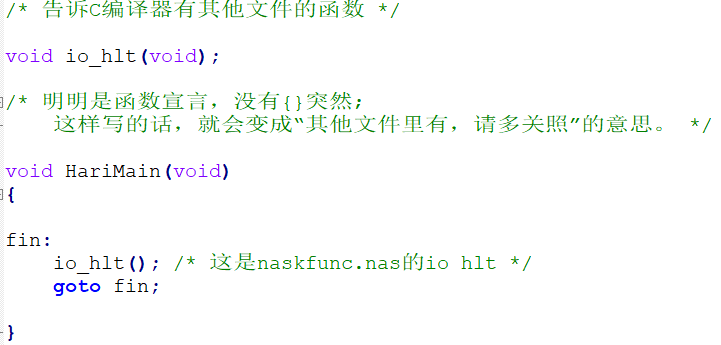
**重点总结：**

由于是导入C语言的初期准备，实验中涉及的代码不多，而且教材中跳过了大部分没有进行讲解，所以这一次的重点是理解导入C的流程以及每个文件的在流程中的作用，以下面的流程图作为重点总结：

**关键代码及注释：**



*（naskfunc.nas）*



*（bootpack.c）*

## 二、遇到的问题及解决方法

1、内存装载与ORG指令

**问题描述：**

修改代码的时候，尝试把程序装在到内存的地址进行修改，用户程序不变。一开始认为只要修改跳转语句，指定跳转到程序装载到内存的地址，程序就能够正常运行，而对于该地址可以随意修改；运行时出现错误。

**出现原因：**

**a）**首先对于“内存装载”的理解有问题；在JMP指令跳转到用户代码之前，（本实验中）扇区读入已经完成，也就是说这次读入的10个柱面（10\*18\*2=360个扇区）已经读入进连续的一片内存（可能物理上不是连续的，虚存连续），而数据区位于第34个扇区肯定已经被加载进内存。所以有：

*Addr\_disk(启动区)-Addr\_disk(用户程序) = Addr\_memory(启动区)-Addr\_memory(用户程序)*

所以，既然地址偏差确定，那么就不能随便更改ORG指令将程序“装载”到其他地址。

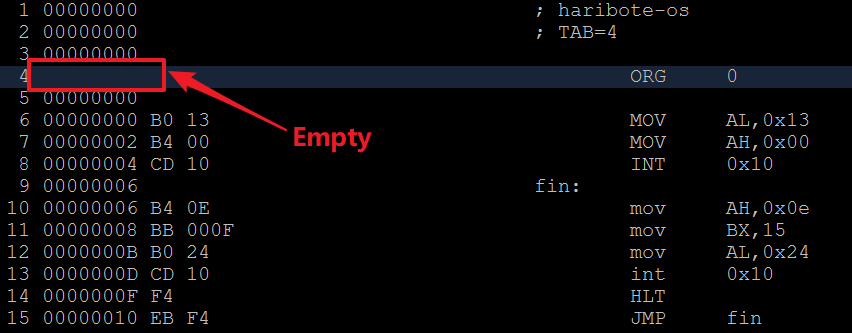
b）对于ORG指令的理解也有一定偏差，教材中对于ORG指令的解释是“把程序装载到内存指定地址”。在之前的实验中能够理解，但是现在有点歧义了——装载步骤已经在读扇区的时候就已经完成了，这里的“装载”的意义何在?

注释掉haribote.nas中ORG指令后，简易操作系统能够运行，但是无法执行用户程序（也就是黑屏程序），所以不能去掉ORG。

网上看了一些解释后有了如下新理解：汇编源程序中第一条指令用伪指令

*ORG 0xnn*

来告诉“编译器”，该程序中首字节放在内存地址为0xnn的位置，之后的字节顺序放置（除非遇到另一条ORG指令）。从haribote.lst也可以看出，ORG并不是一条实际的CPU指令。对于纯汇编语言，从汇编代码到可执行文件只有汇编和链接两个步骤，伪指令ORG在链接时使用，作用是告诉NASM编译器这个程序“要”被装载到哪个内存位置，并不能“真正决定”程序要被装载到哪里，换句话就是ORG就是个通风报信的，没有实际“装载”动作的假把式！



**解决方法：**

所以,在扇区已经被“按照某种顺序”读入时，那么用户程序在内存中的起始地址就是确定的（根据读扇区的顺序和启动区的内存位置），不能随意改变。讲到这里的话又有一个猜测——那我先不读入扇区，用户程序就没有被固定，然后用ORG随意指定内存中一个起始地址再JMP过去，是不是也能执行呢？回到之前讲的，ORG没有实际的“装载内存”操作，如果扇区没有被提前读入，那么链接时，ORG所给的地址对于NASM编译器没有任何意义，因为没有任何用户程序被读入，自然也就没有指令供执行。

## 三、程序设计创新点

1、内存装载地址变动

**创新点：**

根据之前对于装载程序到内存问题的思考，现在随意改动一下程序中的扇区读入地址和ORG、JMP指令地址，来观察是否能够照常运行。主要修改的点就是，把扇区将要被读入的内存地址、ORG告诉NASM编译器的内存地址、柱面读取完后JMP到的内存地址同步修改（在本实验中，“同步”表示地址偏差固定为0x4200，也就是软盘中用户区的地址）。对以上三处修改后，预计能够正常执行用户程序。

**关键代码：**



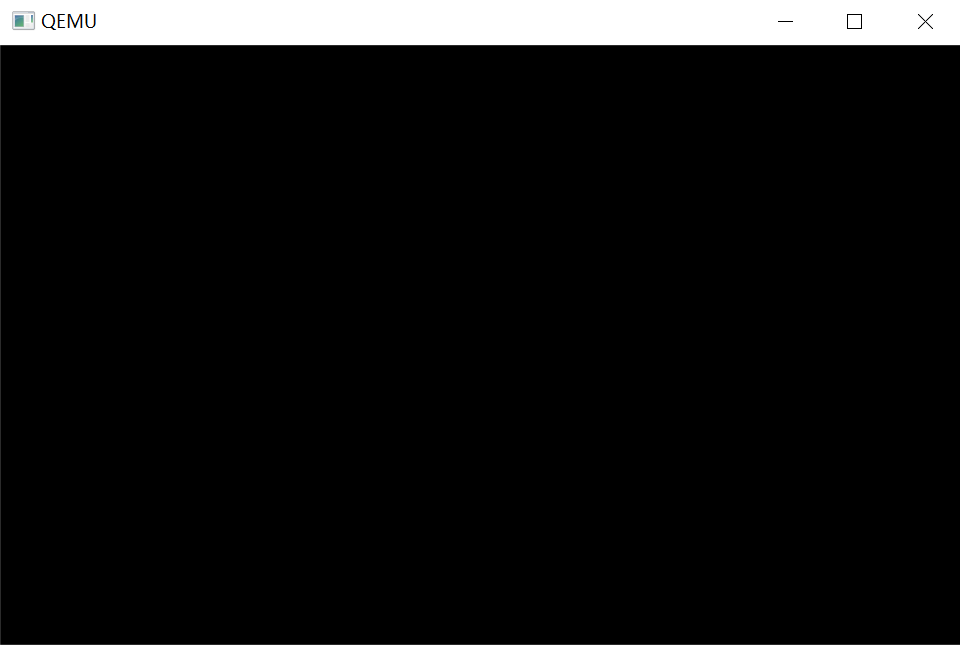


*（ipl10.nas）*



*（haribote.nas）*

**结果截图：（黑屏）**

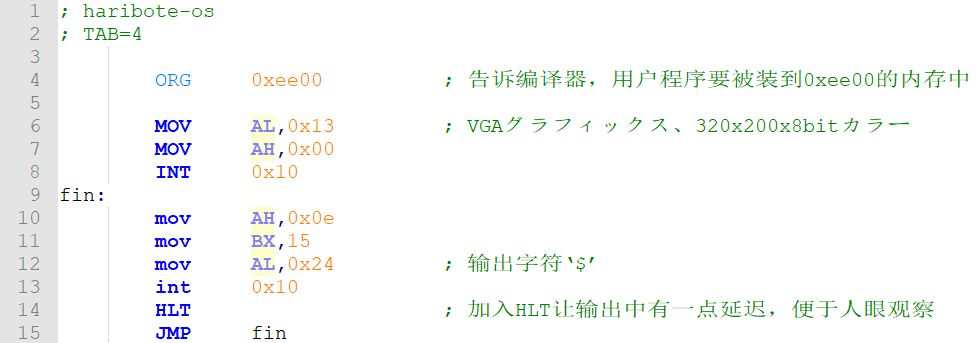


2、用户程序修改

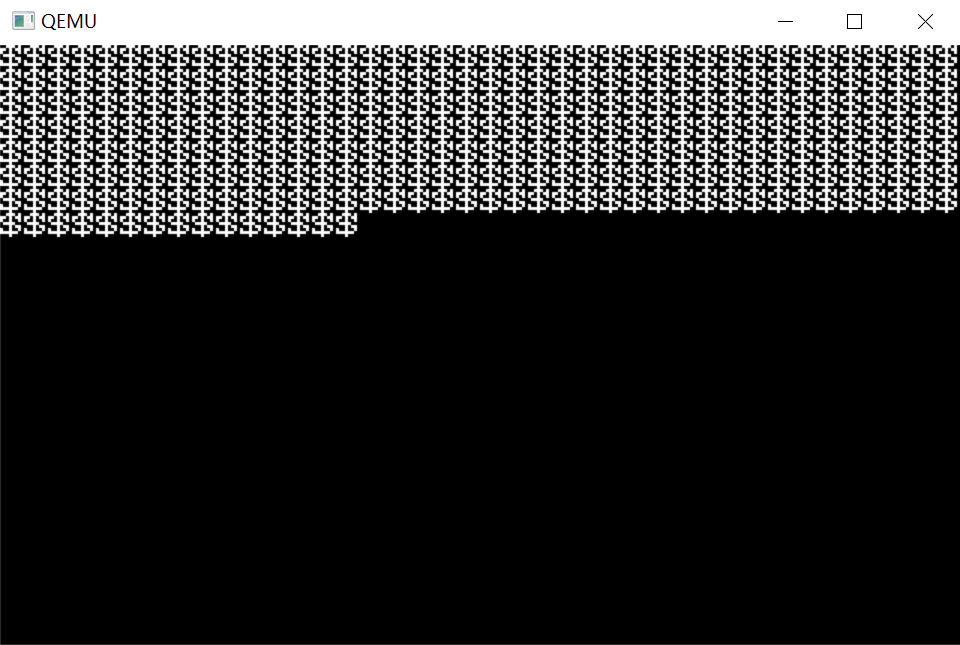
**创新点：**

修改用户程序代码，做一些黑屏以外的事情，比如我这里选的是通过BOIS中断打印字符，置于一个死循环中。由于之前一直在和内存地址打交道，所以在改用户程序文件haribote.nas的时候有一种莫名的害怕——担心这样改又会对内存地址有影响了，会不会待会不能正常执行了……其实操作系统没这么脆弱，只要我们写的haribote.nas不会超过这一次读取扇区总数的大小（10\*18\*2=360个扇区，约180KB，已经很大了），就能够被完整装入，所以这一项修改按理来讲不会出现很大bug。

**关键代码：**



**结果截图：**



## 四、实验心得体会

这次试验花了比较长的时间，因为确实需要学的东西有点多：主要是FAT12文件系统、ORG指令的重新理解以及导入C程序的流程，而这些恰巧是教材中“蜻蜓点水”的地方，有的甚至容引起误解——ORG指令（应该是作者为了让新手理解，所以用一种简单的方式解释）。

其中对于FAT12文件系统主要是通过理论知识上和实验涉及到的文件的验证去理解，其中个人认为比较重要的知识点都写进来了，尽管一部分在本次实验中没有涉及到，但是对于理解教材中提到的实验现象是十分有帮助的，所以一并写了进去，使得这次实验报告显得“略长”……

而教材中着墨较多的扇区读入部分，尽管循序渐进地分了很多步骤来讲解，但其实在代码上十分浅显易懂，所以在本次报告中也就稍稍带过。

实际在实验过程中花较多时间操作的还是对几处“地址”的修改，最初的尝试修改ORG、JMP指令，到之后修改段寄存器，多次的重复尝试和百度之后达到了预期的效果。看起来修改这些“地址”貌似和实验要我们做的事情“关系不大”，但却不是在做没有意义的事情，通过对这些地址的修改让笔者对软盘的结构、读扇区的动作、以及伪指令ORG这三个之前一直“模糊”的知识点有了更清楚的理解，所以将其作为这次试验的重点。

之后的C语言导入部分，本次实验涉及的为初期内容，没有进行过多的代码讲解，所以以一张流程图概况Makefile中以此执行的指令，包括新增文件的作用。

*学习参考资料：*

*中断表:* [*https://blog.csdn.net/piaopiaopiaopiaopiao/article/details/9735633*](https://blog.csdn.net/piaopiaopiaopiaopiao/article/details/9735633)

*FAT12文件：* [*https://zhuanlan.zhihu.com/p/121807427*](https://zhuanlan.zhihu.com/p/121807427)

[*https://blog.csdn.net/c234jc/article/details/70991050*](https://blog.csdn.net/c234jc/article/details/70991050)

[*http://blog.sina.com.cn/s/blog\_3edcf6b80100cr08.html*](http://blog.sina.com.cn/s/blog_3edcf6b80100cr08.html)

[*http://blog.sina.com.cn/s/blog\_3edcf6b80100crz1.html*](http://blog.sina.com.cn/s/blog_3edcf6b80100crz1.html)

*ORG指令：* [*https://blog.csdn.net/u011542994/article/details/46707815*](https://blog.csdn.net/u011542994/article/details/46707815)

[*https://blog.csdn.net/judyge/article/details/52298054*](https://blog.csdn.net/judyge/article/details/52298054)

[*https://zhidao.baidu.com/question/565770121760425644.html*](https://zhidao.baidu.com/question/565770121760425644.html)