

实验九 在 FAT12 盘结构中引导操作系统 实验报告

数据科学与计算机学院 计算机科学与技术 2016 级

王凯祺 16337233

2018 年 6 月 5 日

1 实验目的

- 了解 FAT12 磁盘结构
- 会创建 FAT12 格式的软盘
- 会从 FAT12 格式的磁盘中寻找文件、复制文件到内存

2 实验要求

- 用 C 语言 + 汇编语言编写一个程序，编译产生一个 COM 格式的可执行程序，用于显示一个 FAT12 的 BPB 和 EBPB 内容。这个程序作为一个用户程序，保存在你的映像文件中的 FAT12 文件系统中。
- 用 C 语言 + 汇编语言编写一个程序，编译产生一个 COM 格式的可执行程序，用于列出一个 FAT12 的根目录中所有文件信息，如文件名、文件大小、文件创建日期等等。这个程序作为一个用户程序，保存在你的映像文件中的 FAT12 文件系统中。
- 修改你的操作系统原型 3，将内核执行体用一个文件形式存放在映像盘中，同时按 FAT12 盘格式的要求，修改引导程序，从 FAT12 根目录中加载内核。
- 修改内核程序，实现用命令行从 FAT12 中加载任意一个用户程序。

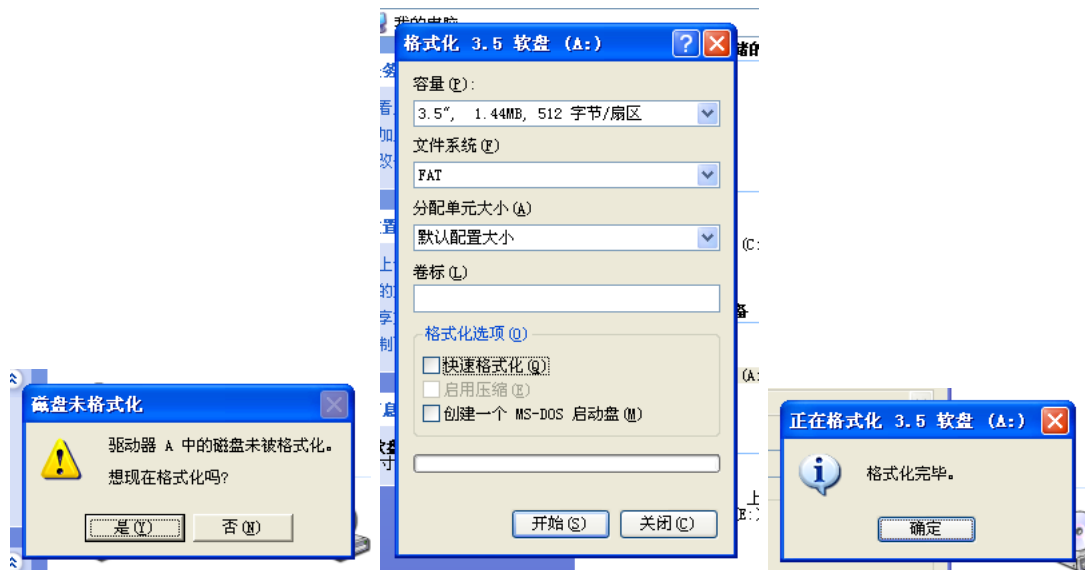
3 实验步骤

3.1 了解 FAT12 磁盘结构

看了老师提供的资料，我是一脸懵的。我需要弄懂几个问题：

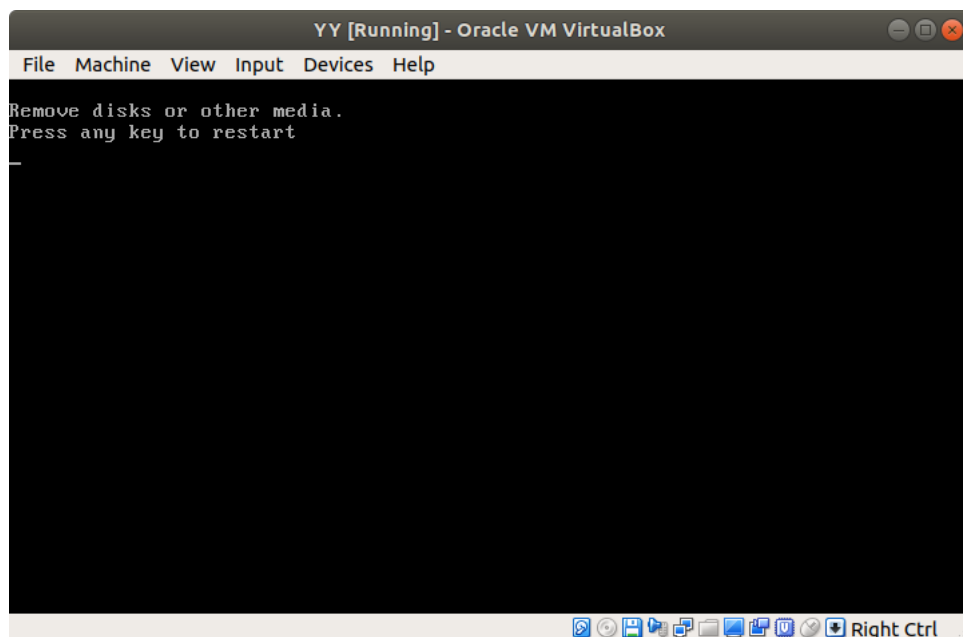
- FAT1 和 FAT2 是什么？它有什么用？
- 簇是什么？软盘中的簇有多大？
- 文件是如何存储的？

为此，我特意用 Windows XP 加载一张空的软盘镜像，并格式化。



我们可以看到，在第一个扇区，Windows 为这个软盘写了 BPB 和 EBPB 信息，并写了一个操作系统进去（功能是显示两句话~）！

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	EB	3C	90	4D	53	44	4F	53	35	2E	30	00	02	01	01	00	?恣SDOS5.0....A
00000010	02	E0	00	40	0B	F0	09	00	12	00	02	00	00	00	00	00	.?@.?......NA
00000020	00	00	00	00	00	00	29	8D	9A	0E	7E	4E	4F	20	4E	41)岬.~NO NA
00000030	4D	45	20	20	20	20	46	41	54	31	32	20	20	20	33	C9	ME FAT12 3
00000040	8E	D1	BC	F0	7B	8E	D9	B8	00	20	8E	C0	FC	BD	00	7C	幛捺{欠? 帮 . :
00000050	38	4E	24	7D	24	8B	C1	99	E8	3C	01	72	1C	83	EB	3A	8N\$}\$娶集<.r.泮:
00000060	66	A1	1C	7C	26	66	3B	07	26	8A	57	FC	75	06	80	CA	f? &f;. &奥黠.e .
00000070	02	88	56	02	80	C3	10	73	EB	33	C9	8A	46	10	98	F7	.均.e?s?藕F.樟v.
00000080	66	16	03	46	1C	13	56	1E	03	46	0E	13	D1	8B	76	11	f..F..V..F..禪v.
00000090	60	89	46	FC	89	56	FE	B8	20	00	F7	E6	8B	5E	0B	03	`圾鼓V .塵輝..
000000A0	C3	48	F7	F3	01	46	FC	11	4E	FE	61	BF	00	00	E8	E6	尙黠.F?N??.枕舒
000000B0	00	72	39	26	38	2D	74	17	60	B1	0B	BE	A1	7D	F3	A6	.r9&8-t.`?尽)螃乎
000000C0	61	74	32	4E	74	09	83	C7	20	3B	FB	72	E6	EB	DC	A0	at2Nt.亾;鸛驢軒
000000D0	FB	7D	B4	7D	8B	F0	AC	98	40	74	0C	48	74	13	B4	0E	鹽硠嬌瑯@t.Ht.?
000000E0	BB	07	00	CD	10	EB	EF	A0	FD	7D	EB	E6	A0	FC	7D	EB	?.?脬胆}脬城}
000000F0	E1	CD	16	CD	19	26	8B	55	1A	52	B0	01	BB	00	00	E8	鬼.??嬲.R???. 牟
00000100	3B	00	72	E8	5B	8A	56	24	BE	0B	7C	8B	FC	C7	46	F0	.:r娶葵\$? 嫩葬
00000110	3D	7D	C7	46	F4	29	7D	8C	D9	89	4E	F2	89	4E	F6	C6	=}葬?)屬塏驂N塏
00000120	06	96	7D	CB	EA	03	00	00	20	0F	B6	C8	66	8B	46	F8	.枕岁...度f婢
00000130	66	03	46	1C	66	8B	D0	66	C1	EA	10	EB	5E	0F	B6	C8	f.F.f嫩f陵.臙.度
00000140	4A	4A	8A	46	0D	32	E4	F7	E2	03	46	FC	13	56	FE	EB	JJ奄.2澈?F?V 43
00000150	4A	52	50	06	53	6A	01	6A	10	91	8B	46	18	96	92	33	JRP.Sj.j.悽F.料3
00000160	D2	F7	F6	91	F7	F6	42	87	CA	F7	76	1A	8A	F2	8A	E8	吟鯨踏B轉鱗.姦嫻
00000170	C0	CC	02	0A	CC	B8	01	02	80	7E	02	0E	75	04	B4	42	捞..谈..e~..u.簪
00000180	8B	F4	8A	56	24	CD	13	61	61	72	0B	40	75	01	42	03	嫻葵\$?aar.@u.B.es
00000190	5E	0B	49	75	06	F8	C3	41	BB	00	00	60	66	6A	00	EB	.Iu. A?.`fj.es
000001A0	B0	4E	54	4C	44	52	20	20	20	20	20	20	20	0D	0A	52	瘡TLDR ..Res
000001B0	6D	6F	76	65	20	64	69	73	6B	73	20	6F	72	20	6F	74	move disks or ots
000001C0	68	65	72	20	6D	65	64	69	61	2E	FF	0D	0A	44	69	73	her media. ..Dis
000001D0	6B	20	65	72	72	6F	72	FF	0D	0A	50	72	65	73	73	20	k error ..Press
000001E0	61	6E	79	20	6B	65	79	20	74	6F	20	72	65	73	74	61	any key to resta



然后我往里面写了三个文件，分析 FAT 表结构。

我发现 FAT1 和 FAT2 是一样的，一个位于第 2 扇区（偏移地址 0x0200），一个位于第 10 扇区（偏移地址 0x1400），我使用 Google 搜索后，知道他们是主要 FAT 表和备份 FAT 表的关系。FAT 表存储的是下一个簇放在什么地方。每个文件可以看成是一条链表，一直指到 0xFFFF 表示结束。

c. txt	txt	34 bytes	2018-06-04 19:06:24	2018-06-04 19:06:24	2018-06-04 19:06:24	2018-06-04 19:06:24	A	33
FAT 1		4.5 KB						1
FAT 2		4.5 KB						10
空闲空间								
启动扇区		0.5 KB						0

[a. img]	Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
文件系统: FAT12	000001D0	6B	20	65	72	72	6F	72	FF	0D	0A	50	72	65	73	73	20	k error
默认的编辑模式	000001E0	61	6E	79	20	6B	65	79	20	74	6F	20	72	65	73	74	61	any key t
状态: 原始	000001F0	72	74	0D	0A	00	00	00	00	00	00	00	AC	CB	D8	55	AA	rt.....
撤销级别: 0	00000200	00	FF	FF	FF	FF	FF	05	60	00	07	80	00	FF	0F	00	00	?
撤销相反: n/a	00000210	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
分配可见的驱动器空间。	00000220	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

根目录的文件摘要信息存储在第 20 - 33 扇区（偏移地址 0x2600 - 0x41FF），每个文件的摘要占 32 字节，格式 PPT 已经说得很清楚啦！

a. txt	txt	49 bytes	2018-06-04 19:06:24	2018-06-04 19:06:24	2018-06-04 19:06:24	2018-06-04 19:06:24	A	34
b. txt	txt	2.4 KB	2018-06-04 19:06:40	2018-06-04 19:06:40	2018-06-04 19:06:40	2018-06-04 19:06:40	A	35
c. txt	txt	34 bytes	2018-06-04 19:06:52	2018-06-04 19:06:52	2018-06-04 19:06:52	2018-06-04 19:06:52	A	33
FAT 1		4.5 KB						1
FAT 2		4.5 KB						10
空闲空间								
启动扇区		0.5 KB						0

[a. img]	Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
文件系统: FAT12	000025F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
默认的编辑模式	00002600	43	20	20	20	20	20	20	20	54	58	54	20	18	0A	E1	98	C
状态: 原始	00002610	C4	4C	C4	4C	00	00	DA	98	C4	4C	02	00	22	00	00	00	陳陳..趾陳.."
撤销级别: 0	00002620	41	20	20	20	20	20	20	20	54	58	54	20	18	0B	E1	98	A
撤销相反: n/a	00002630	C4	4C	C4	4C	00	00	CC	98	C4	4C	03	00	31	00	00	00	陳陳..虞陳..1...
分配可见的驱动器空间。	00002640	42	20	20	20	20	20	20	20	54	58	54	20	18	0C	E1	98	B
簇编号: n/a	00002650	C4	4C	C4	4C	00	00	D4	98	C4	4C	04	00	8A	09	00	00	陳陳..詞陳..?.十
根目录	00002660	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	00002670	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

通过对大文件的分析，我知道一簇是 512 字节。

了解好这些之后，可以开始写代码啦！

3.2 编写引导扇区程序

这一部分是本实验中最最最最难的！为什么这样说？我们先来了解一下需要干什么。

- 复制 FAT 表到内存
- 在根目录寻找 LOADER.COM
- 找到 LOADER.COM 的起始簇号
- 根据簇号计算磁头号、柱面号、扇区号
- 复制扇区
- 根据该簇号在 FAT 表寻找下一簇号

而这一切，都必须在 448 字节以内（512 字节扣去”55AA”的 2 字节、BPB 和 EBPB 的 62 字节）完成，多一个都不行！

而且，更要命的是，这不同于操作系统内核可以用 C 语言实现，这里的所有的代码都必须用汇编写。由此可见，这次项目的工作量非常非常大。

BPB 和扩展 BPB 这部分代码照抄老师的，把 OEMName 和序列号等个性化信息改成了我的。

```
1  org 7c00h          ; BIOS将把引导扇区加载到0:7C00h处，并开始执行
2
3      jmp short Start          ; Jump, 2 Bytes
4      nop                    ; 1 Bytes
5      BS_OEMName DB 'AM-OS9.0'      ; OEM String, 8 Bytes
6      BPB_BytsPerSec DW 512
7      BPB_SecPerClus DB 1
8      BPB_RsvdSecCnt DW 1
9      BPB_NumFATs DB 2
10     BPB_RootEntCnt DW 224
11     BPB_TotSec16 DW 2880
12     BPB_Media DB 0xF0
13     BPB_FATSz16 DW 9
14     BPB_SecPerTrk DW 18
15     BPB_NumHeads DW 2
16     BPB_HiddSec DD 0
17     BPB_TotSec32 DD 0
18     BS_DrvNum DB 0
19     BS_Reserved1 DB 0
20     BS_BootSig DB 29h
21     BS_VolID DD 16337233h
22     BS_VolLab DB 'AM-OS_9.0_ _'      ; 11 Bytes
23     BS_FileSysType DB 'FAT12_ _ _'    ; 8 Bytes
```

定义常量 为了避免硬编码，我定义了以下常量。

```
1  FATMem equ 2000h          ; FAT 分区表保存的段地址
2  KernelCS equ 0a00h        ; 加载内核的段地址
3  KernelIP equ 0100h        ; 加载内核的偏移地址
4                               ; 内核加载在 0xA100
```

复制 FAT 分区表到内存指定位置 此部分代码将软盘的前 33 个扇区（包括引导扇区、FAT1、FAT2 和根目录文件摘要）拷贝到 FATMem (= 0x2000) 处。

```

1 Start:
2
3 LoadFAT:
4     ;读软盘或硬盘上的若干物理扇区到内存的ES:BX处:
5     mov word bx, FATMem
6     mov es, bx
7     xor bx, bx
8     mov ah,2             ; 功能号
9     mov al,33            ; 扇区数
10    mov dl,0             ; 驱动器号 ; 软盘为0, 硬盘和U盘为80H
11    mov dh,0             ; 磁头号 ; 起始编号为0
12    mov ch,0             ; 柱面号 ; 起始编号为0
13    mov cl,1             ; 起始扇区号 ; 起始编号为1
14    int 13H ;             调用读磁盘BIOS的13h功能
15    ; FAT 根目录已加载到指定内存区域 0x2000:0x0000 中

```

在根目录查找内核并返回首簇地址 这部分代码就比较复杂了。这一共两层循环，外层循环是 0 - 223，表示遍历到根目录的第几个文件，内层循环则是做字符串匹配，若某个文件的文件名匹配“LOADER.COM”能成功，则将该摘要中的首簇号提取出来，放在 dx 寄存器中。

```

1 getFAT:
2     ; Output: if LOADER.COM found in [file_num], return dx = [start sector], bx = [
3     ;           size]
4     ;           otherwise, dx = 0
5     xor dx, dx           ; dx = 0
6     mov ax, 2600h
7     xor cx, cx           ; cx = 0
8
9 FATfindLoop:
10    xor bx, bx
11    mov ds, bx
12    mov [axbak], ax
13    push ax
14    push cx
15    mov ax, KernelStr     ; ax = addr("KERNEL BIN")
16    xor cx, cx           ; cx = 0
17
18 KernelStrMatch:
19    push ax
20    xor ax, ax
21    mov ds, ax
22    mov bx, [axbak]
23    mov ax, FATMem
24    mov ds, ax
25    pop ax
26    add bx, cx            ; the cx-th bit
27    mov bl, [bx]
28    push ax
29    xor ax, ax
30    mov ds, ax

```

```

31     mov byte [char1], bl
32     pop ax
33
34     mov bx, ax
35     mov bl, [bx]
36     cmp bl, [char1]
37     jnz KernelNotMatch
38
39     inc ax
40     inc cx
41     cmp cx, 11
42     jb KernelStrMatch
43     xor bx, bx
44     mov ds, bx
45     mov dx, [axbak]           ; matched, dx = kernel addr
46 KernelNotMatch:
47
48     pop cx
49     pop ax
50     add ax, 20h
51     inc cx
52     cmp dx, 0
53     jnz KernelOK
54     cmp cx, 224
55     jb FATfindLoop
56     call dispStr
57     jmp $
58
59 KernelOK:
60     ; dx = kernel addr
61     call dispStr
62     add dx, 1ah
63     mov bx, dx
64     mov ax, FATMem
65     mov ds, ax
66     mov dx, [bx]             ; dx = first cluster
67
68     mov ax, KernelIP

```

复制扇区 这个难点就在于磁头号、柱面号、扇区号的计算了。查阅了《软盘结构 (磁头号 and 起始扇区的计算方法)》(<https://blog.csdn.net/littlehedgehog/article/details/2147361>) 后, 就能计算出正确的磁头号、柱面号、扇区号, 正确地加载指定扇区到指定内存啦!

```

1  copy_sector:
2  ; Input: dx = cluster number
3  ;         ax = memory offset
4  ;         (KernelCS is DS)
5      push ax
6      push dx
7      xor bx, bx
8      mov ds, bx
9      mov word [axbak], ax
10     add dx, 31

```

```

11     mov ax, dx
12     xor dx, dx
13     mov cx, 18
14     div cx
15     inc dx
16     mov byte [qishishanqu], dl
17     xor dx, dx
18     mov cx, 2
19     div cx
20     mov byte [citou], dl
21     mov byte [zhumian], al
22     mov ax, KernelCS
23     mov es, ax
24     mov bx, [axbak]
25     mov cl, [qishishanqu]
26     mov ah, 2
27     mov al, 1
28     mov dl, 0
29     mov dh, [citou]
30     mov ch, [zhumian]
31     int 13h
32
33     pop dx
34     pop ax
35     ret

```

寻找下一簇 在查 FAT 表时，首先要分簇号的奇偶两种情况来讨论。

记当前簇号为 x ， $t_a = \lfloor 3x/2 \rfloor$ ， $t_b = t_a + 1$ ， d_a 为第 t_a 字节的数据， d_b 为第 t_b 字节的数据。

我对着 FAT 表观察了很久，总结出规律：对于偶簇，选 d_a 的全部 8 位作为低位和 d_b 的低 4 位作为高位组成 12 位整数；对于奇簇，选 d_a 的高 4 位作为低位和 d_b 的全部 8 位作为高位组成 12 位整数。那么这个 12 位整数就是下一簇。

由于涉及分类讨论和位运算操作，这个地方特别难写，也特别难调试。我使用 bochs 一步步跟踪，确认寄存器的值。最后发现是因为奇簇算错了，导致扇区复制错了……

```

1 find_next_sector:
2 ; Input: dx = cluster number
3 ; Output: dx = next cluster number
4     push ax
5     mov bx, FATMem
6     mov ds, bx
7     mov ax, dx
8     mov bx, dx
9     xor dx, dx
10    mov cx, 3
11    mul cx
12    mov cx, 2
13    div cx
14    add ax, 200h
15    xor dx, dx
16    test bx, 1h
17    jnz odd_num
18    ; even_number

```

```

19     mov bx, ax
20     mov dl, [bx]
21     inc bx
22     mov dh, [bx]
23     and dh, 0fh
24     jmp even_odd_all_ok
25 odd_num:
26     ; odd number
27     mov bx, ax
28     mov dl, [bx]
29     inc bx
30     mov dh, [bx]
31     shr dl, 4
32     mov cl, dh
33     and cl, 0fh
34     shl cl, 4
35     add dl, cl
36     shr dh, 4
37 even_odd_all_ok:
38     pop ax
39     ret

```

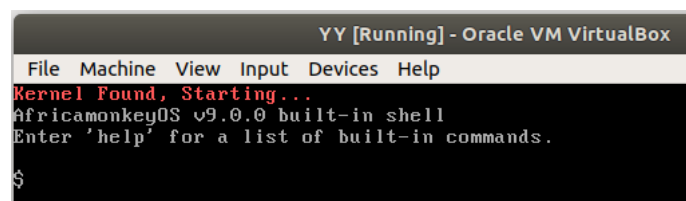
跳转内核 写个循环搬内核到内存，读到 FF8 - FFF 文件结束信号就可以收工，跳转至内核啦！

```

1 go_and_fetch:
2     call copy_sector
3     call find_next_sector
4     add ax, 200h
5     cmp dx, 0ff8h
6     jb go_and_fetch
7
8     mov ax, KernelCS
9     mov bx, KernelIP
10    shl ax, 4
11    add bx, ax
12    jmp bx

```

编写好引导扇区后，把未经修改的 LOADER.COM 通过 Windows XP 拖进软盘就能引导啦！



3.3 修改内核

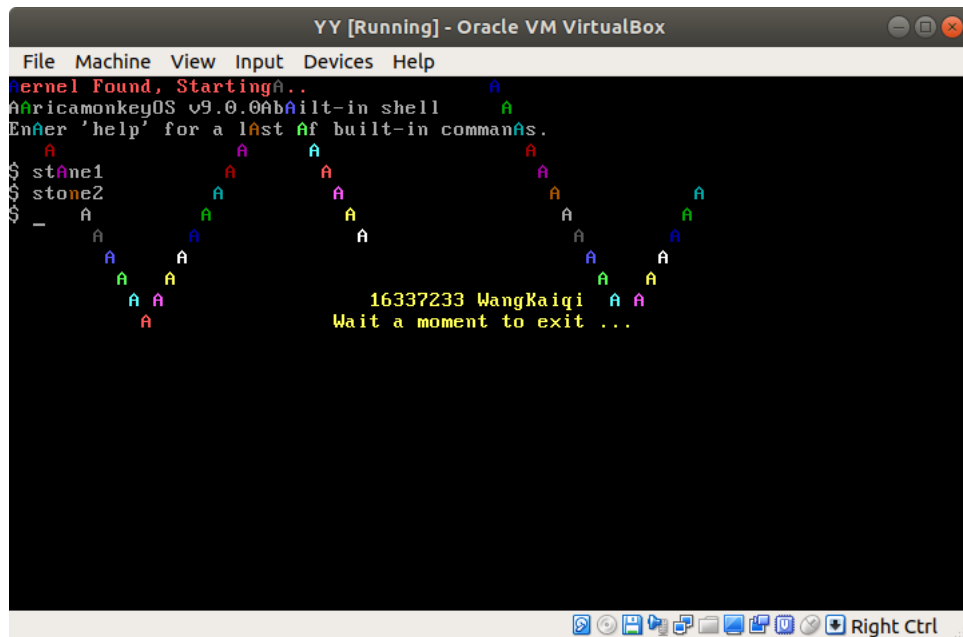
由于之前用户程序是按照别的方式从软盘读到内存，现在更换为 FAT12 文件系统，内核在读取用户程序时也需要作出相应的修改。

具体修改的地方：

- 修改寻找用户程序的函数（从根目录区域搜索），返回值为首簇
- 新增查找下一簇的函数
- 修改复制用户程序的函数（由复制连续的好几个扇区改为每次只复制一扇区并查找下一扇区）

由于上述的功能在前面引导扇区用汇编写过了，我用 C 语言写当然是毫不费劲啦！
这样，内核就可以加载程序啦！

测试时间片轮转功能 下面测试的是实验六实现的时间片轮转功能，功能正常。

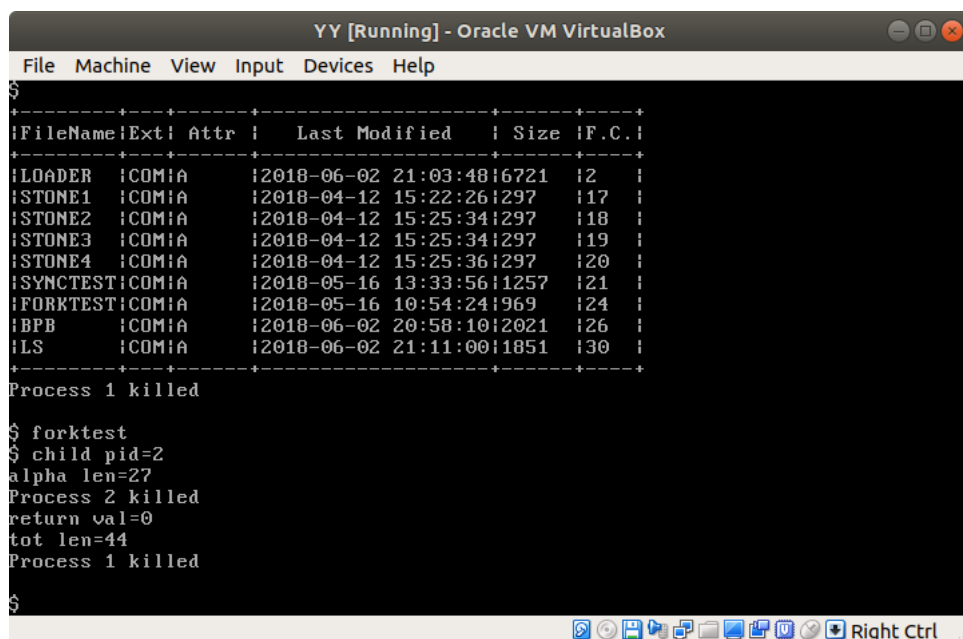


```

YY [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Kernel Found, Starting...
AracamonkeyOS v9.0.0Abilt-in shell
Enter 'help' for a list of built-in commands.
$ stane1
$ stone2
$ -
16337233 WangKaigi
Wait a moment to exit ...

```

测试 fork 功能 下面测试的是实验七实现的 fork 功能，功能正常。

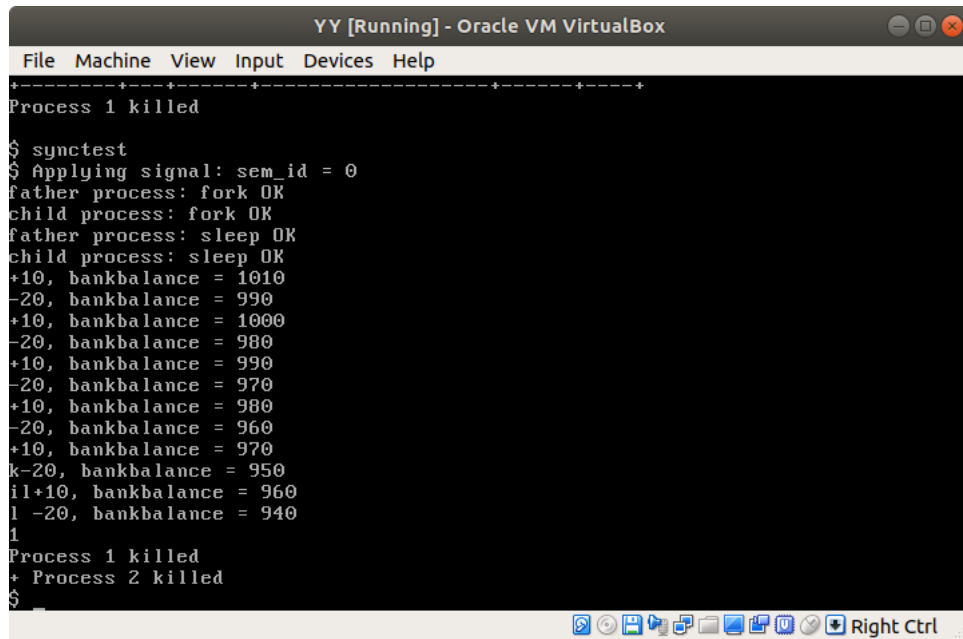


```

YY [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
$
+-----+-----+-----+-----+-----+-----+
|FileName|Ext| Attr |   Last Modified   | Size |F.C.|
+-----+-----+-----+-----+-----+-----+
|LOADER  |COM|A   |2018-06-02 21:03:48|6721  |12  |
|STONE1  |COM|A   |2018-04-12 15:22:26|1297  |17  |
|STONE2  |COM|A   |2018-04-12 15:25:34|1297  |18  |
|STONE3  |COM|A   |2018-04-12 15:25:34|1297  |19  |
|STONE4  |COM|A   |2018-04-12 15:25:36|1297  |20  |
|SYNCTEST|COM|A   |2018-05-16 13:33:56|1257  |21  |
|FORKTEST|COM|A   |2018-05-16 10:54:24|1969  |24  |
|BPB     |COM|A   |2018-06-02 20:58:10|2021  |26  |
|LS      |COM|A   |2018-06-02 21:11:00|1851  |30  |
+-----+-----+-----+-----+-----+-----+
Process 1 killed
$ forktest
$ child pid=2
alpha len=27
Process 2 killed
return val=0
tot len=44
Process 1 killed
$

```

测试信号量功能 下面测试的是实验八实现的信号量功能，功能正常。



3.4 编写显示 BPB 和 EBPB 内容的程序

由于引导扇区程序已经将软盘的前 33 个扇区都加载到 0x2000 段中，这个用户程序就不用再做同样的操作了，直接开始解释 BPB！

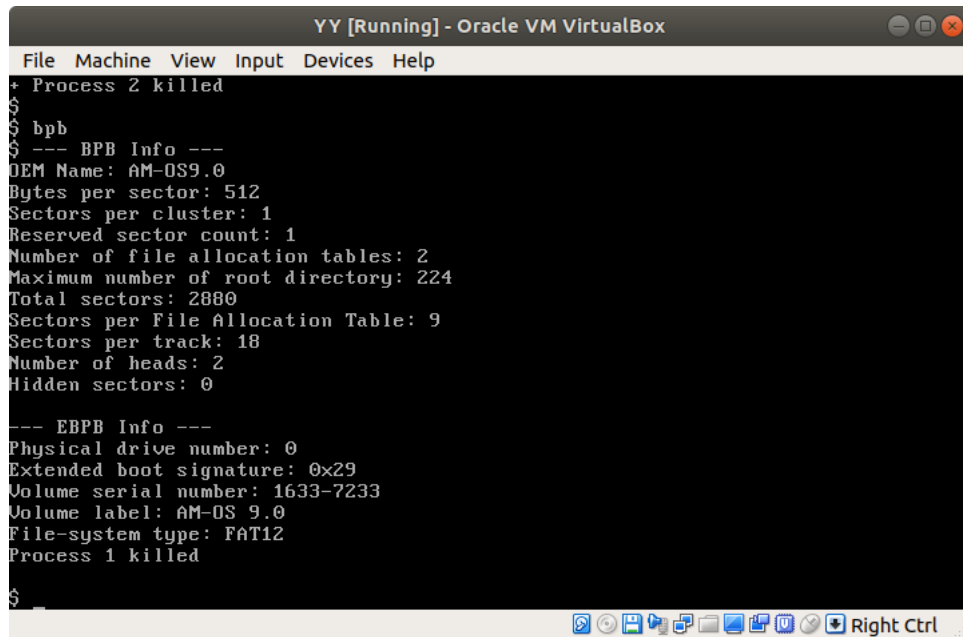
```
1 #include "stdlib.h"
2 #include "stdio.h"
3
4 void bpbmain() {
5     unsigned char st[0x40], tmp;
6     int i;
7     for (i = 0; i < 0x40; ++i) {
8         asm mov bx, i
9         asm push ds
10        asm mov ax, 2000h          /* FAT table */
11        asm mov ds, ax
12        asm mov al, [bx]
13        asm pop ds
14        asm mov tmp, al
15        st[i] = tmp;
16    }
17    puts("---_BPB_Info_---");
18    puts_no_new_line("OEM_Name:_");
19    for (i = 0; i < 8; ++i) putchar(st[i + 0x03]);
20    puts("");
21    puts_no_new_line("Bytes_per_sector:_");
22    i = (int)st[0x0b] + (((int)st[0x0c]) << 8);
23    printint(i);
24    puts_no_new_line("Sectors_per_cluster:_");
25    printint((int)st[0x0d]);
26    puts_no_new_line("Reserved_sector_count:_");
27    i = (int)st[0x0e] + (((int)st[0x0f]) << 8);
```

```

28     printint(i);
29     puts_no_new_line("Number_of_file_allocation_tables:_");
30     printint((int)st[0x10]);
31     puts_no_new_line("Maximum_number_of_root_directory:_");
32     i = (int)st[0x11] + (((int)st[0x12]) << 8);
33     printint(i);
34     puts_no_new_line("Total_sectors:_");
35     i = (int)st[0x13] + (((int)st[0x14]) << 8);
36     printint(i);
37     puts_no_new_line("Sectors_per_File_Allocation_Table:_");
38     i = (int)st[0x16] + (((int)st[0x17]) << 8);
39     printint(i);
40     puts_no_new_line("Sectors_per_track:_");
41     i = (int)st[0x18] + (((int)st[0x19]) << 8);
42     printint(i);
43     puts_no_new_line("Number_of_heads:_");
44     i = (int)st[0x1a] + (((int)st[0x1b]) << 8);
45     printint(i);
46     puts_no_new_line("Hidden_sectors:_");
47     i = (int)st[0x1c] + (((int)st[0x1d]) << 8);
48     printint(i);
49     puts("");
50     puts("---_EBPB_Info_---");
51     puts_no_new_line("Physical_drive_number:_");
52     printint((int)st[0x24]);
53     puts_no_new_line("Extended_boot_signature:_");
54     printhex((int)st[0x26]);
55     puts_no_new_line("Volume_serial_number:_");
56     printhex_no_new_line_no_0x(st[0x2a] >> 4);
57     printhex_no_new_line_no_0x(st[0x2a] & 15);
58     printhex_no_new_line_no_0x(st[0x29] >> 4);
59     printhex_no_new_line_no_0x(st[0x29] & 15);
60     puts_no_new_line("-");
61     printhex_no_new_line_no_0x(st[0x28] >> 4);
62     printhex_no_new_line_no_0x(st[0x28] & 15);
63     printhex_no_new_line_no_0x(st[0x27] >> 4);
64     printhex_no_new_line_no_0x(st[0x27] & 15);
65     puts("");
66     puts_no_new_line("Volume_label:_");
67     for (i = 0; i < 11; ++i) putchar(st[i + 0x2B]);
68     puts("");
69     puts_no_new_line("File-system_type:_");
70     for (i = 0; i < 8; ++i) putchar(st[i + 0x36]);
71     puts("");
72 }

```

结果如下：



3.5 编写显示文件摘要的程序

这个程序列举文件系统根目录所有文件的摘要，包括文件名、扩展名、属性、修改时间、大小、首簇号。

这程序用 C 写，也非常好写。至少比引导扇区的汇编好多了！

```
1 #include "stdlib.h"
2 #include "stdio.h"
3
4 void lsmain() {
5     int i, j, k, offset;
6     unsigned int t1, t2;
7     unsigned char st[32], tmp;
8     puts("");
9     puts("+-----+---+-----+-----+-----+");
10    puts("|FileName|Ext|_Attr_|_Last_Modified_|_Size_|F.C.|");
11    puts("+-----+---+-----+-----+-----+");
12    for (i = 0; i < 224; ++i) {
13        for (j = 0; j < 0x20; ++j) {
14            offset = 0x2600 + i * 0x20 + j;
15            asm mov bx, offset
16            asm push ds
17            asm mov ax, 2000h /* FAT table */
18            asm mov ds, ax
19            asm mov al, [bx]
20            asm pop ds
21            asm mov tmp, al
22            st[j] = tmp;
23        }
24        if (st[0] == 0x00) continue; /* Empty entry */
25        if (st[0] == 0xe5) continue; /* Erased entry */
26        putchar('|');
```

```

27     for (j = 0; j < 8; ++j) putchar(st[j]);
28     putchar('|');
29     for (j = 8; j < 11; ++j) putchar(st[j]);
30     putchar('|');
31     tmp = st[11];
32     k = 0;
33     if (tmp & 0x01) putchar('R'), ++k;
34     if (tmp & 0x02) putchar('H'), ++k;
35     if (tmp & 0x04) putchar('S'), ++k;
36     if (tmp & 0x08) putchar('V'), ++k;
37     if (tmp & 0x10) putchar('D'), ++k;
38     if (tmp & 0x20) putchar('A'), ++k;
39     for (j = k; j < 6; ++j) putchar('_');
40     putchar('|');
41     t1 = st[0x18];
42     t2 = st[0x19];
43     t1 = t2 << 8 | t1;
44     t2 = t1 & 511;
45     t1 >>= 9;
46     t1 += 1980;
47     printint_zero_format(t1, 4);
48     putchar('-');
49     t1 = t2 >> 5;
50     t2 &= 31;
51     printint_zero_format(t1, 2);
52     putchar('-');
53     printint_zero_format(t2, 2);
54     putchar('_');
55     t1 = st[0x16];
56     t2 = st[0x17];
57     t1 = t2 << 8 | t1;
58     t2 = t1 & 2047;
59     t1 >>= 11;
60     printint_zero_format(t1, 2);
61     putchar(':');
62     t1 = t2 >> 5;
63     t2 &= 31;
64     printint_zero_format(t1, 2);
65     putchar(':');
66     t2 *= 2;
67     printint_zero_format(t2, 2);
68     putchar('|');
69     t1 = st[0x1e];
70     t2 = st[0x1f];
71     if (t1 || t2) {
72         puts_no_new_line("65535+");
73     } else {
74         t1 = st[0x1c];
75         t2 = st[0x1d];
76         t1 = t2 << 8 | t1;
77         printint_format(t1, 6);
78     }
79     putchar('|');
80     t1 = st[0x1a];

```

```

81     t2 = st[0x1b];
82     t1 = t2 << 8 | t1;
83     printint_format(t1, 4);
84     putchar('|');
85     puts("");
86 }
87 puts("+-----+---+-----+-----+-----+-----+");
88 }

```

结果如下：

```

YY [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Volume serial number: 1633-7233
Volume label: AM-OS 9.0
File-system type: FAT12
Process 1 killed

$
$
$ ls
$
+-----+---+-----+-----+-----+-----+
|FileName|Ext| Attr |   Last Modified   | Size |F.C.|
+-----+---+-----+-----+-----+-----+
|LOADER  |COM|A   |2018-06-02 21:03:48|6721  |12  |
|STONE1  |COM|A   |2018-04-12 15:22:26|297   |17  |
|STONE2  |COM|A   |2018-04-12 15:25:34|297   |18  |
|STONE3  |COM|A   |2018-04-12 15:25:34|297   |19  |
|STONE4  |COM|A   |2018-04-12 15:25:36|297   |20  |
|SYNCTEST|COM|A   |2018-05-16 13:33:56|1257  |21  |
|FORKTEST|COM|A   |2018-05-16 10:54:24|969   |24  |
|BPB     |COM|A   |2018-06-02 20:58:10|2021  |26  |
|LS      |COM|A   |2018-06-02 21:11:00|1851  |30  |
+-----+---+-----+-----+-----+-----+
Process 1 killed

$

```

其中,LOADER.COM 为操作系统内核;STONE*.COM 为 4 个弹来弹去的用户程序;SYNCTEST.COM 为信号量测试程序;FORKTEST.COM 为 fork 测试程序;BPB.COM 为 BPB 显示程序;LS.COM 为显示根目录文件摘要的程序。

执行程序时，只需键入文件名（不含扩展名）即可。

4 实验总结

FAT12 分区本身并不复杂，但要弄清楚它，还是要花费一些时间的，最花时间的就是引导扇区程序了。由于很难在屏幕上显示调试信息，在写这个程序的时候，得不断地运行 bochs 来查看各个变量的值是否正确。

编写好引导扇区程序之后，我们只需要把操作系统内核、用户程序等等通过 Windows XP 复制到这个分区中即可，不再需要用 WinHex 来修改软盘啦！用 WinHex 修改软盘有多个弊端：有时删多了删少了导致软盘大小不等于 1.44 MB，无法启动；有时把程序放错了位置。有了文件系统后，复制更方便了。我觉得文件系统这个实验应该在第三个实验（加载用户程序）之后立刻就做，以后的实验用上这个文件系统来复制用户程序就很方便。