操作系统原理实验

实验四

在 FAT12盘结构中引导操作系统

姓名： 吴侃

学号： 14348134

班别： 2014级计算机系一班

日期： 2016.04.01 – 2016.04.07

目录

零、特色先览

本次实验的亮点包括:

1. 另建一个段, 专门存放 PCB, 分时系统管理的进程数可达16个!
   理论上, 最大进程数为: 0xFFFF / PCBSize 向下取整.
   同一程序可以有多个进程.
2. 自行创建 int 20h 中断, 模拟 DOS 从用户程序返回 Shell
3. 使用内嵌汇编 + C++结构比较完美读取 BPB, EBPB 信息
4. 使用 C++结合内嵌汇编加载用户程序
5. 模拟 Linux Shell, 如输入 ls 命令后, 在当前的命令行窗口输出对应信息
6. 同时支持分时运行与批处理运行
7. 良好的 C++ IO 接口, 可以10进制, 16进制输出
8. 基本解决了 gcc + nasm 交叉编译的问题, 能够很容易编写 C++程序.
9. 从磁盘加载文件时，可以指定段地址和偏移量

一、实验目的

1. 了解 FAT12磁盘格式, 在 FAT12盘中读取系统内核, 加载用户程序;
2. 用 C++/C 读取 FAT12磁盘的 BPB, 扩展 BPB 信息;
3. 显示 FAT12磁盘中各个文件的大小,日期,所占簇编号

二、实验要求

1. 从 FAT12盘引导系统内核, 加载用户程序;
2. 用 C++/C 读取 FAT12磁盘的 BPB, 扩展 BPB 信息;

3. 显示 FAT12磁盘中各个文件的大小,日期,所占簇编号

三、实验环境与工具

实验环境:

物理机操作系统: Arch Linux 4.4.5-1

调试使用虚拟机: qemu-system-i386, bochs

虚拟机软件: VMware Workstation 12 Pro

虚拟机配置: CPU: i7-4702MQ @ 2.20GHz, 使用单核单线程

内存:4 MB

硬盘:32 MB

实验工具:

编辑器: Vim 7.4

汇编工具: NASM 2.11.08

C++编译器: g++ 5.3.0

链接工具: GNU ld 2.26.0.20160302

构建工具: GNU Make 4.1

调试工具: Bochs x86 Emulator 2.6.8

虚拟机: qemu-system-i386

dosbox 0.74

VMWare Workstation 12 Pro

合并文件: dd

四、实验方案

1. 从 FAT12磁盘读取一个文件

(这里的 loadknl.asm 采用了凌老师的代码, 我把这段代码读了很多遍,
完全弄懂该实现, 进行了一点修改. 我自己也尝试去实现, 但由于时间关系,
最终采用凌老师的代码.)
要更好地实现 FAT12磁盘的加载, 还需从 BPB 中读取每扇区的字节数,
每簇有多少扇区等数据.

以下简述读取一个文件的过程:

2. 查找文件
磁盘中的文件名保存在19~32号扇区,
首先要从这些扇区中搜索出所需文件的文件目录项(Entry, 32字节)
从偏移00H 处检查长度为11的字符串, 若全部比对上则比对成功.
文件名共11字节, 前8个字节为文件名, 后3个字节为扩展名, 左对齐,
空白用0补全

.

3. 根据簇号加载文件

然后根据文件目录项, 在14H 偏移处获得文件的开始簇号.
根据簇号, 可以在文件数据区(33号扇区及其以上的扇区)找到文件占用的扇区。

由于0号簇和1号簇(共占3字节)为介质描述符和结束簇标记。第2号簇对应文件数据区第一个扇区, 3号簇对应文件数据区第二个扇区,依次类推.

读取文件数据区的对应扇区后, 在 FAT1表(1~9号扇区)中查找文件的下一个簇,偏移地址为1.5 bytes * 簇号, 由于是1.5字节, 需要根据簇号的奇偶作对应位运算, 若得到的数字小于 FF8, 则为文件的新簇号, 否则表示文件结束。

4. 从 FAT12磁盘加载内核

根据上述方法, 找到 KERNEL.BIN, 将其加载到0000h:7e00h, 并跳转到该位置, 完成内核加载.

5. 列出磁盘的所有文件

以此读取19~32号扇区(根目录), 过程为:

将一个扇区的内容写到一个长度为512的 char 数组

对于这个数组, 依次读取32个字节的文件目录项,

检查其文件名项,这里判断是否为文件的方法为检查其文件名, 若无扩展名, 认为该文件不存在.

显示大小, 创建日期等信息, 只需在 Entry 中读取对应数据并作相应计算.

6. 列出文件所占簇号

根据上述"从 FAT12磁盘读取一个文件"中获取簇号的方法得到簇号并输出.

打印 BPB, EBPB 信息:

将0号扇区的内容写入一个512字节的结构中, 依次输出对应项, 这里的不足为:

16位系统下, 无法正常显示32位信息, 需要特殊实现.

7. int 20h 中断

由于系统内核为分时系统, 需要有一个方法管理所有进程的创建与结束.

这里自行创建了一个20h 中断, 当运行这个中断时, 杀死除了 Shell 的进程, 并且返回 Shell.

写入中断的方法为, 计算20h 号中断在中断向量表中7的偏移地址,

写入中断处理代码的段和偏移量.

这里借用了0000h:7c00h 位为信号位, 当其被设为1时,

内核会杀死除了 Shell 外的进程. 当创建进程时, 该位被设置为0.

调用20h 号中断, 即在该位设置1.

由于用户程序不会自动清屏, ls 等用户程序的输出,
可以不删除原信息下添加新信息.

8. PCB 表管理
修改分时系统保存恢复寄存器部分的代码, 将 PCB 保存在额外的段.

9. G++ 与 NASM
C++用户程序的编写非常简单, 只需引用 io 等头文件, 就可以愉快的编程了。
其原理为:
有一个 header.asm 文件做用户程序头, 在 Makefile 中,将 C++编译的目标文件与 header.asm 编译的目标文件链接即可生成出 com 文件。
生成的 com 文件还可以在 dosbox 运行!
值得注意的是, 全局变量的存储位置在代码段(数据段与代码段值相同),
而局部变量(包括函数中的变量)存在栈中。
而栈大小为100h - 4, 局部变量超过该大小后会出错!
这个问题将在之后解决, 解决方法为调大栈段。.

五、实验操作

可用指令：

指令名称	功能
r	回到用户进程界面
ls	列出所有用户程序信息(大小时间占用簇)
bpb	显示 BPB
bs	显示 EBPB
cls	清屏
top	显示用户程序状态
killall	杀死所有用户进程
uname	显示操作系统信息

特殊按键：

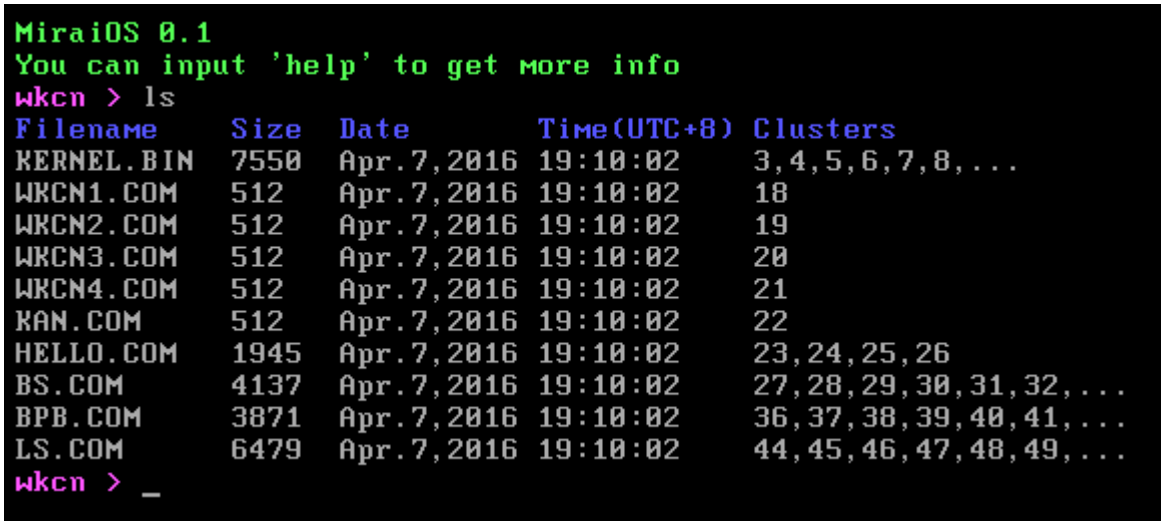
ESC: 返回 Shell 但是不杀死进程
Ctrl+C: 在非 Shell 模式进行清屏
Ctrl+Z: 返回 Shell 并且杀死所有进程

用户程序：

ls,bpb,bs,hello,wkcn1,wkcn2,wkcn3,wkcn4,kan

输入数字1234也可运行程序
当连续输入数字时使用批处理
单独输入，为分时模式

ls 命令：列出了磁盘中的所有文件大小，创建时间以及所占簇



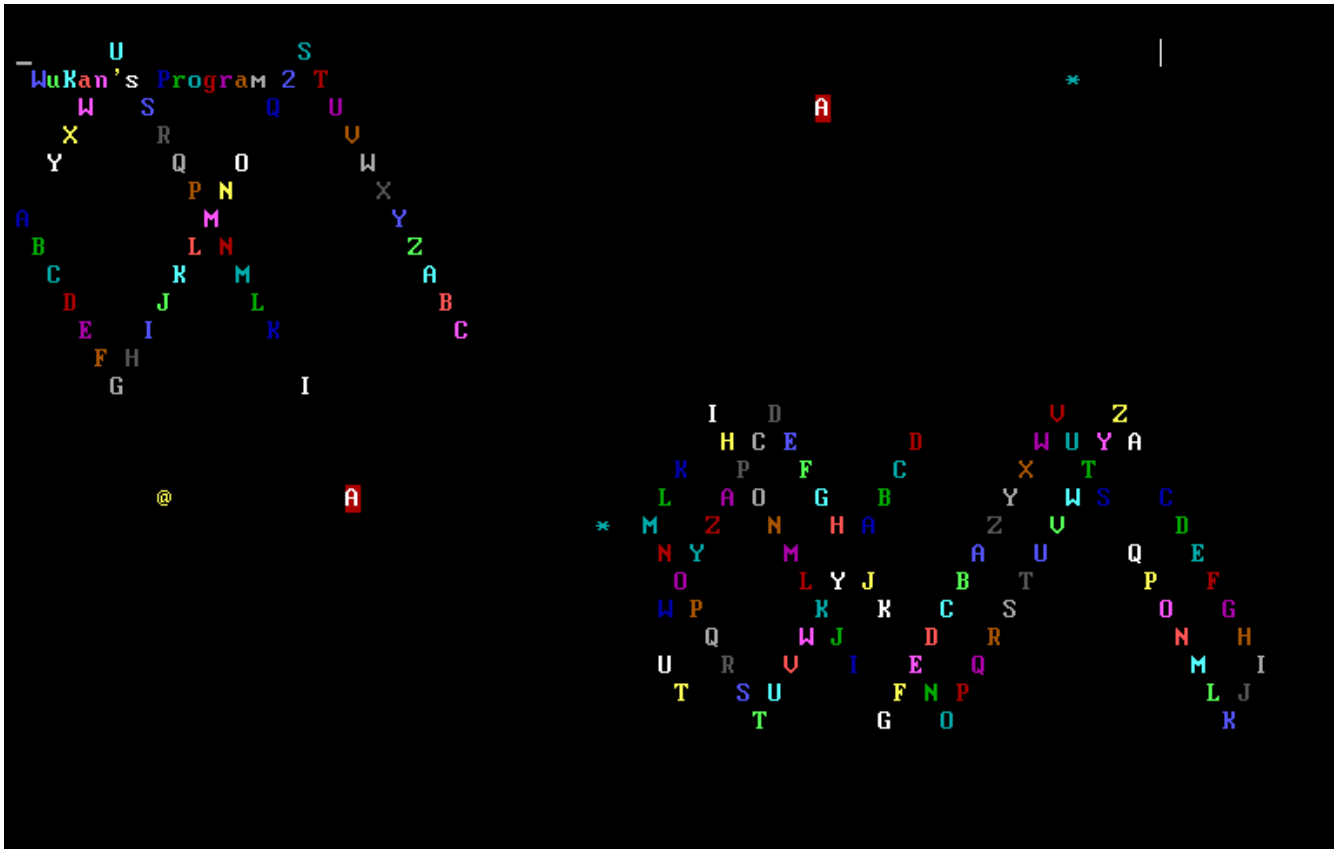
bpb 命令：列出 BPB

```
HELLO.COM   1945  Apr.7,2016
BS.COM      4137  Apr.7,2016
BPB.COM     3871  Apr.7,2016
LS.COM      6479  Apr.7,2016
wkcn > bpb
jmpShort dw 0x3CEB
nop
OEMName db "MiraiOS "
BytesPerSec dw 512
SecPerClus db 1
ResvdSecCnt dw 1
NumFATs db 2
RootEntCnt dw 224
TotSec16 dw 2880
Media db 0xF0
FatSz16 dw 9
SecPerTrk dw 18
NumHeads dw 2
HiddSec dd 0x0
TotSec32 dd 0x0
wkcn > _
```

bs 命令：列出 EBPB

```
wkcn > bs
DrvNum db 0
Reserved1 db 0
BootSig db 0x29
VolID db 0
VolLab db "MiraiOS    "
FileSysType db "FAT12   "
wkcn >  _
```

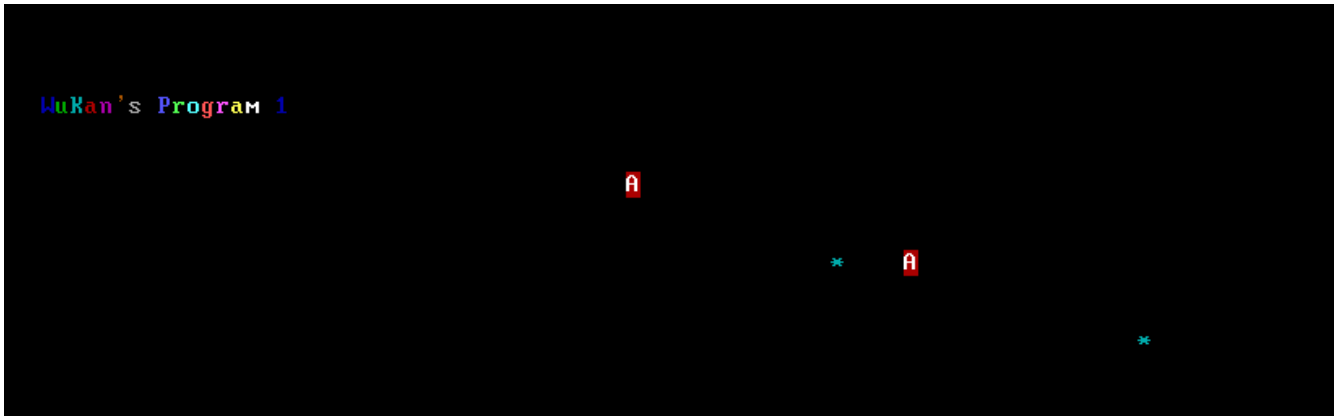
4个进程：



显示进程数：

```
wkcn > top
12 User Progresses are running :-)
wkcn > _
```

同一程序多个进程：



六、总结

这次实验, 与实验三相比, 轻松了一点, 但还是花了我很多时间.

我觉得造成这些困难的原因, 主要是我选择了16位实模式下 GCC+NASM 交叉编译,

真的特别多坑! 原因是 GCC 原生是不支持32位的, 之所以能在16位运行,

是因为 Common Misconception(在 osdev 网站上有详细描述)

大概意思是, 32位的寄存器如 eax, 它在16位系统上也能使用,

只是32位中的高16位被截断了.

现在总结下这些坑:

1. 带参数的函数声明时, 需要在函数前声明用多少个寄存器,

避免 gcc 编译出的32位程序在16位模式下压栈的问题.

2. 链接文件时, 需要使用 ld 文件链接, 光用参数-Ttext 是不行的, ds 值会出错!

3. 指针为0x00007e00时, 也不代表它对应了0000h:7e00h 这个位置,

还是要看数据段.

4. 单单编译 cpp 文件会缺少_start 标签, 解决方法有: 用汇编文件做_start 头,

然后链接; 或者用 ld 文件声明入口.

5. 注意位运算, 当少字节数据转多字节数据时, 会进行符号扩展

6. 16位模式下的 int 是16位的!

比较高兴的是, 现在 PCB 另存一段, 能开16个以上的进程了。

并且同一程序可以有多个进程, 未来我将实现 fork()函数。

可以做到不清屏列出一个程序的信息了。

计划做彩色图片显示。

G++和 NASM 交叉编译的问题基本解决了。

分时系统非常稳定。

我觉得操作系统实验花了很多时间, 但也能收获很多。