实 验 报 告

实验题目：**保护模式与文件系统**

实验人： 李健恒

学号： 14348051

日期： 2016.05.26

院系：数据科学与计算机科学学院

专业：计算机科学

联系方式：1412441716@qq.com

指导老师：凌应标

一. 实验目的

1. 认识硬盘与相关的 IO 操作
2. 认识文件相关的系统调用

二. 实验要求

1. 实现四个系统调用 open，read，write，close

三. 实验方案

1. 本实验增加了两个重要的系统进程，文件系统（fs）与硬盘驱动（hd）. 文件系统负责处理上述四个系统调用，控制文件逻辑. 硬盘驱动负责核心的数据读写.

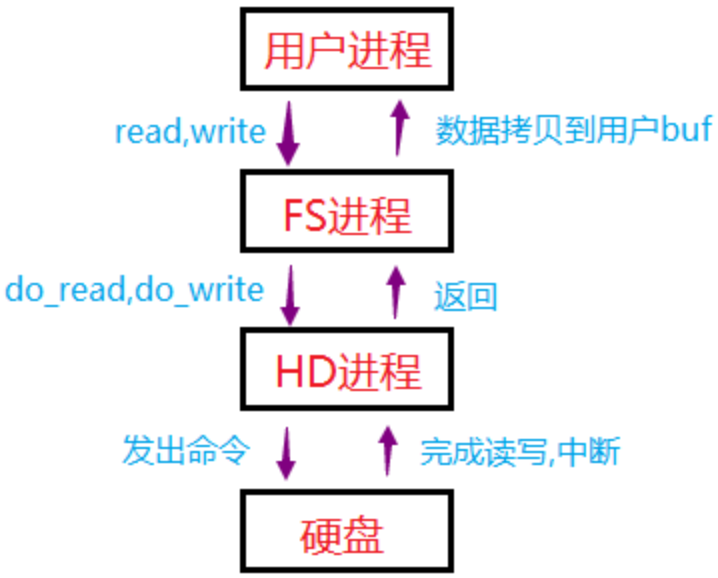

图 1 读写原理图

2. hd 采用异步读写的方式，即 hd 在向硬盘发出命令后会进入阻塞状态，硬盘完成工作后发出中断，解除 hd 阻塞. 具体的实现为实验八的中断消息系统.

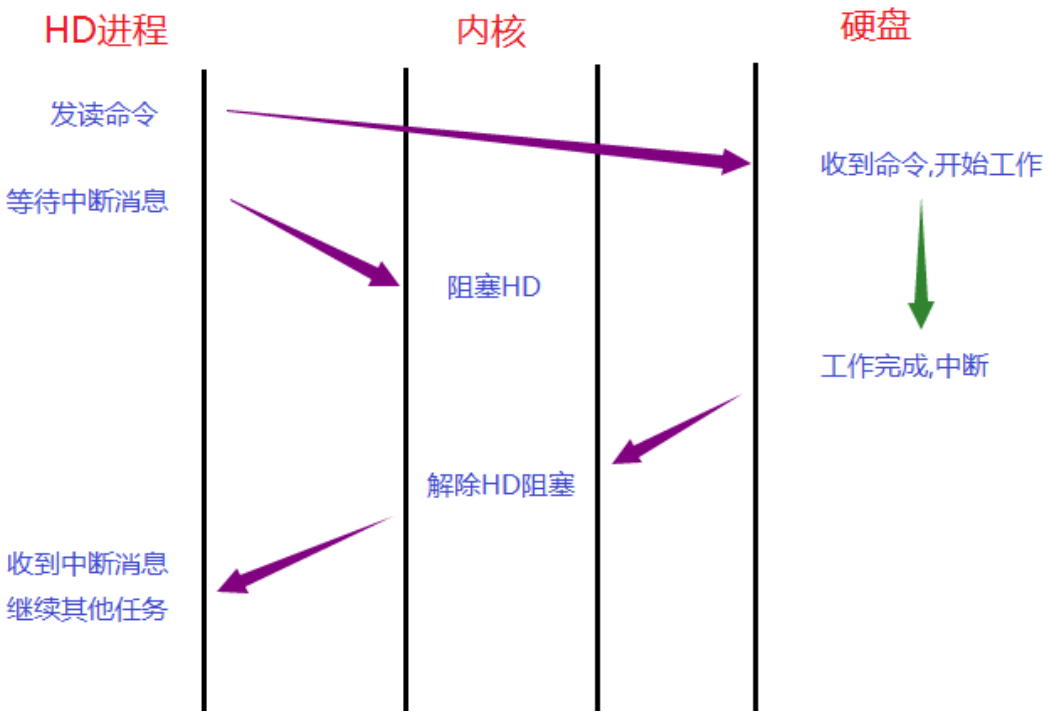

图 2 HD 工作原理图

3. fs 采用 FAT12 作为文件系统，簇号和根目录的读写使用即时更新的非缓存方法，文件方面则每个文件有一个扇区大小的缓冲区，满后才进行硬盘读写.

四. 实验过程和结果

过程：

文件系统写了非常多，单就 fs 就 8k 代码了．我看到内核一下子大了这么多，才发现已经写了 400 行 C 了．老师您说这只要 200 来行就搞定，我可做不到．而且我这个文件系统已经足够粗暴了，功能足够简单.

文件本身就是一个很复杂的数据结构，虽然它没有 PCB 那么复杂，并且有并发问题，但是它的很多数据本身在硬盘上，那很多事情都要考虑了，很多数据都是要相对定位了（幸好内存是平坦模式，不用考虑段的问题）.

让我不想做文件系统的一个重要原因是，保护模式下我不能使用 BIOS 中断，需要直接操作硬盘，虽然硬盘控制器由硬盘自己实现了，不像软盘那样什么都要自己弄（这也是为什么不使用软盘），但跟硬件打交道就离不开汇编，那就容易搞砸.

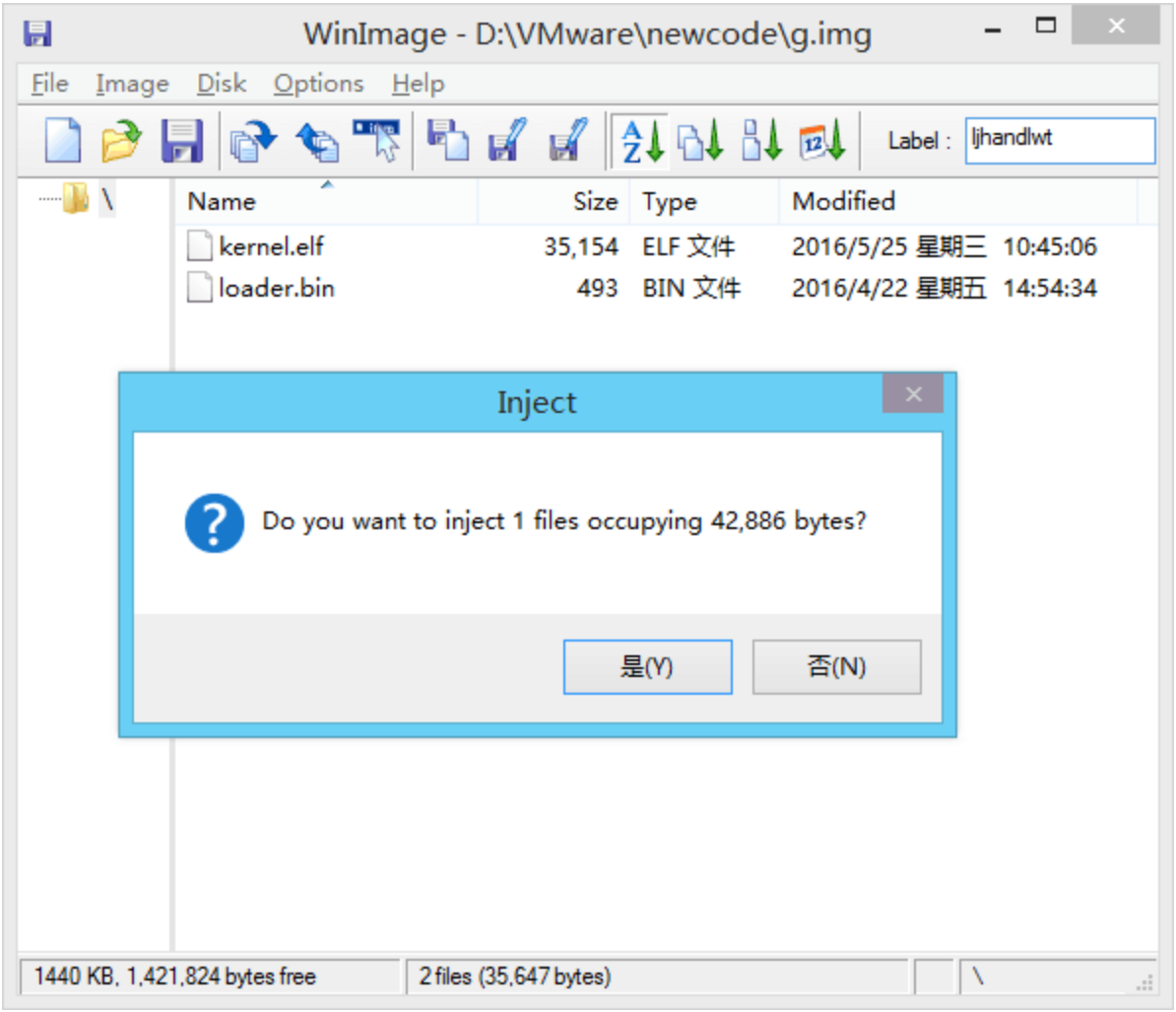

图 3 加上 fs 进程后，内核迅速增大

硬盘 IO 有个比较坑的地方是，读写的操作和事件顺序不一样的，读是发命令后先等中断再读端口，写却是发命令后先写端口，再等中断．我一开始在这里实现写操作的时候卡了挺久，也没有人告诉我这个细节，后来翻书才发现．之所以这样做，我猜是硬盘有一个高速缓存，硬盘驱动对数据的读写都是在高速缓存下做的，而实际硬盘的读写则是高速缓存和硬盘之间的操作，由硬盘控制器控制．

还有一个细节是，每次硬盘读写是否有中断也是要控制的，端口是 0x3d6．错过了这个，可能就一直纠缠在中断初始化那些代码里了．

另外，由于 vm 的虚拟硬盘是 vm 特殊处理的，因此一开始很多操作我都在 bochs 下进行，这样我就能用 winhex 或者 winimage 看看硬盘到底有没有成功写入．

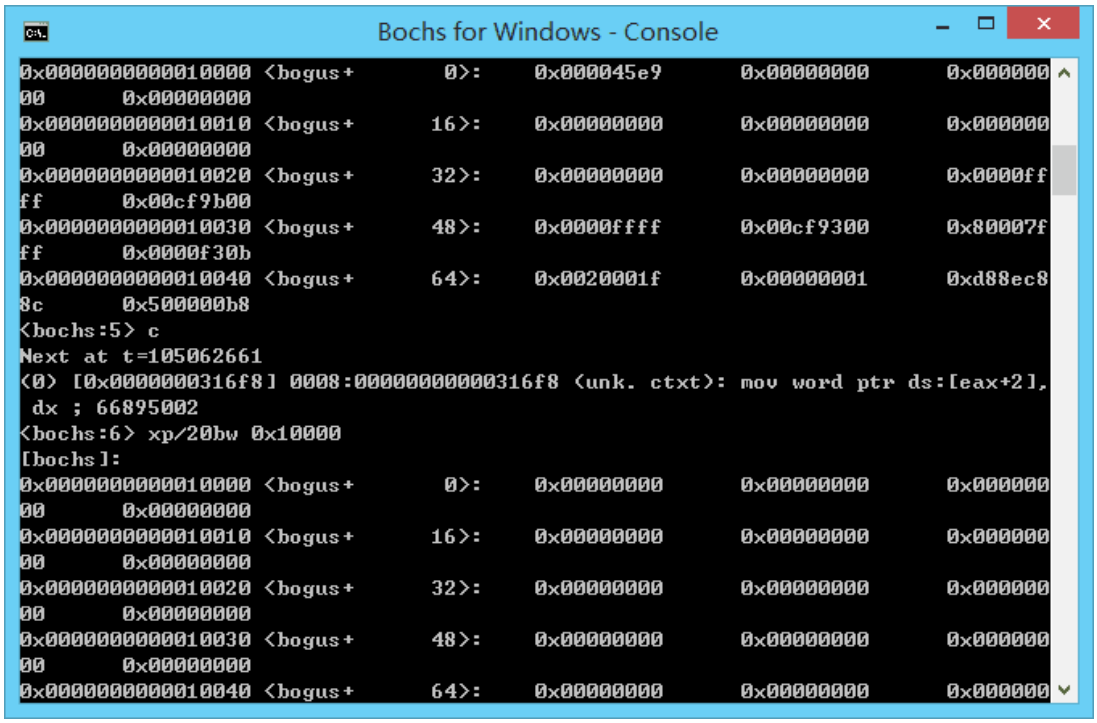

图 4 检查内存

上图是第一次进行硬盘读操作，把硬盘全 0 空扇区读到 loader 的 0x10000 处，从上图可以看出原本有代码的内存区域被全 0 覆盖，这说明读取成功.

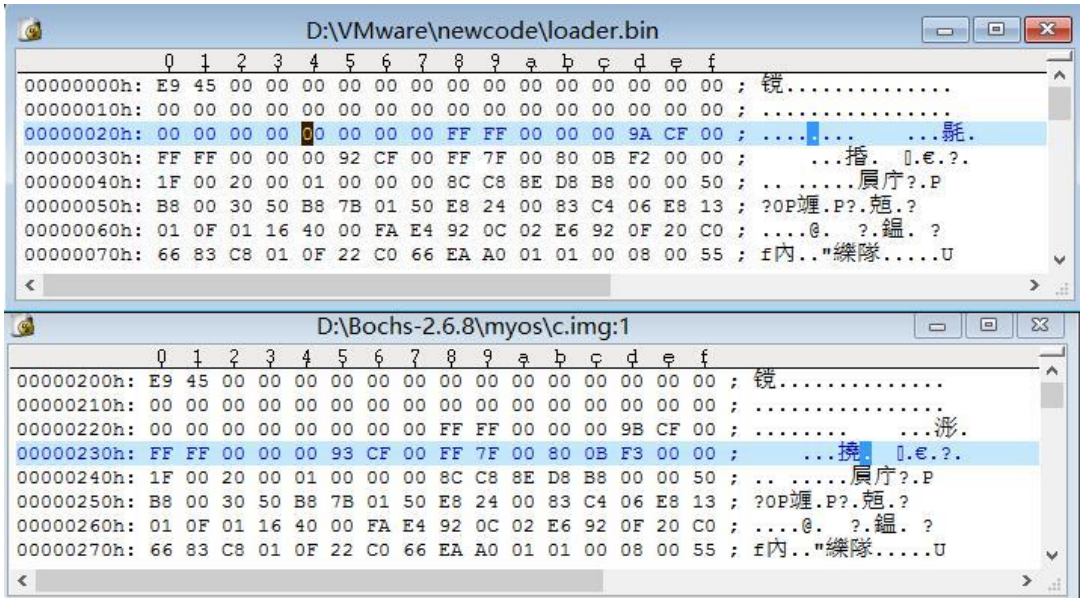

图 5 检查硬盘

上图是第一次进行硬盘写操作，把 0x10000 上的 loader 代码写到硬盘的第一个扇区．从上图可以看到，写入成功.

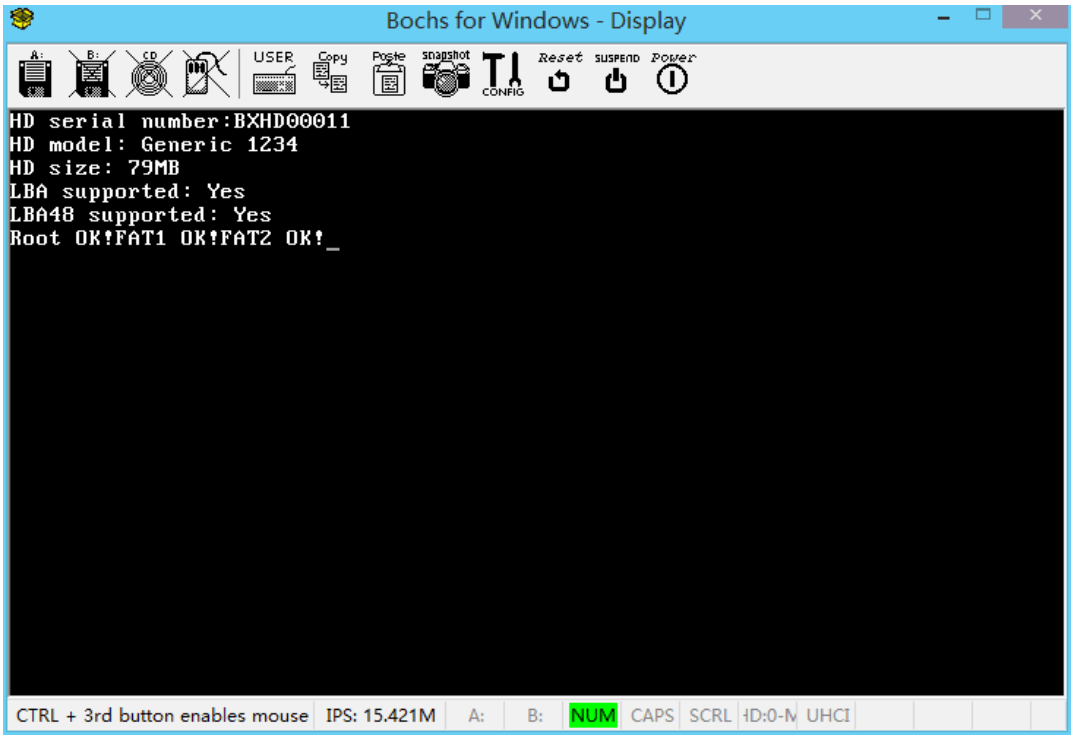

图 6 检查引导扇区

上图前面五行为输出硬盘信息.

而最后一行，为把 0x7c00 的带有 bpb 和 ebpb 的 boot 写到硬盘上，当然 fat 表和根目录也进行了写入. 在这之后，这个硬盘就能被 winimage 识别（bochs 生成的 img 硬盘）.

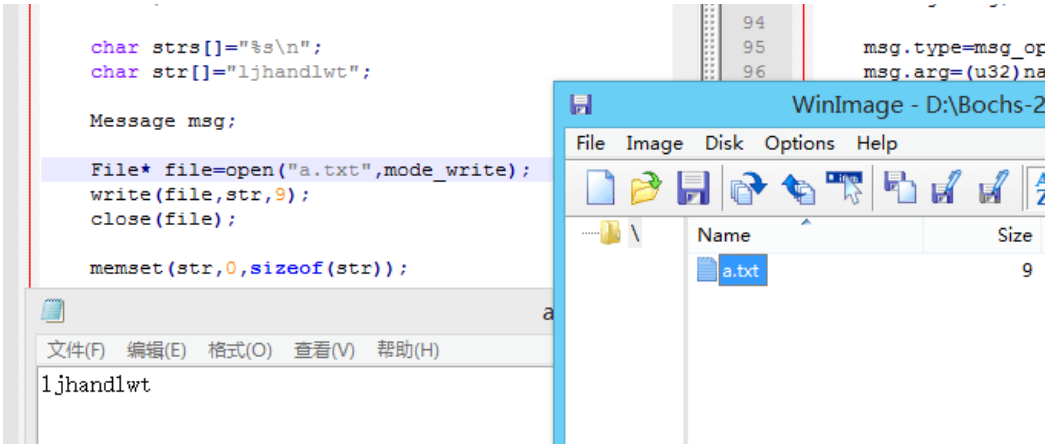

图 7 测试 write

上图为第一次使用文件的系统调用，向 a.txt 写入一个字符串. 从上图可以看出写入成功了.

```
//close(file);

memset(str,0,sizeof(str));
File* file=open("a.txt",mode_read);

//file=open("a.txt",mode_read);
read(file,str,3);
printf(strs,str);
read(file,str,3);
printf(strs,str);
//read(file,str,100);
//printf(strs,str);
close(file);
```

图 8 测试 read

上图代码是试图读入前面写入的 a.txt 文件，从屏幕可以看到读入成功.

```
char str[]="1234567890";

Message msg;

File* file=open("a.txt",mode_write);
write(file,str,strlen(str));
close(file);

memset(str,0,sizeof(str));
//File* file=open("a.txt",mode_read);

file=open("a.txt",mode_read);

a=read(file,str,3);
assert(a==3);
str[a]='\0';
printf(strs,str);

a=read(file,str,3);
str[a]='\0';
printf(strs,str);

a=read(file,str,100);
str[a]='\0';
printf(strs,str);

close(file);
```

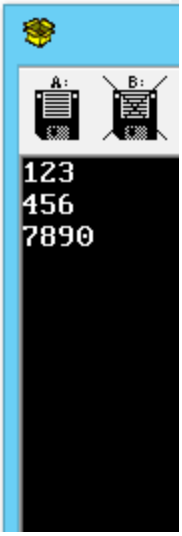

图 9 文件尾测试

上图为测试读入超量的字符，从结果可以看到，read 到了文件尾之后，fs 会进行特殊操作，提前返回.

结果：

本实验的展示流程:首先 3 号进程会打印一些信息，并提示你输入一个字符串，输入后 3 号进程会把字符串写到一个 a.txt 的文件里．之后 3 号进程会向 4 号进程发一个信息，4 号进程收到信息后，打开这个 a.txt 的文件，然后把里面的内容读取并输出．实验结果看上去可能不酷炫，但本实验的功能都用到了．

在完成上面工作后，3 号进程和 4 号进程会像之前的实验一样不断在 0 号终端打印字符，此时可切换到 0 号终端（Alt+F1）．0 号终端里，1 号进程和 2 号进程的行为和之前的实验一样，1 号进程不断 fork 和 wait 子进程，2 号进程和其子进程使用信号量同步输出．

另外，HD 进程在 2 号终端打印了硬盘信息，可键入 Alt+F3 查看．

还有一点，为了证明文件的确写入了硬盘，实验会同时在 bochs 下展示

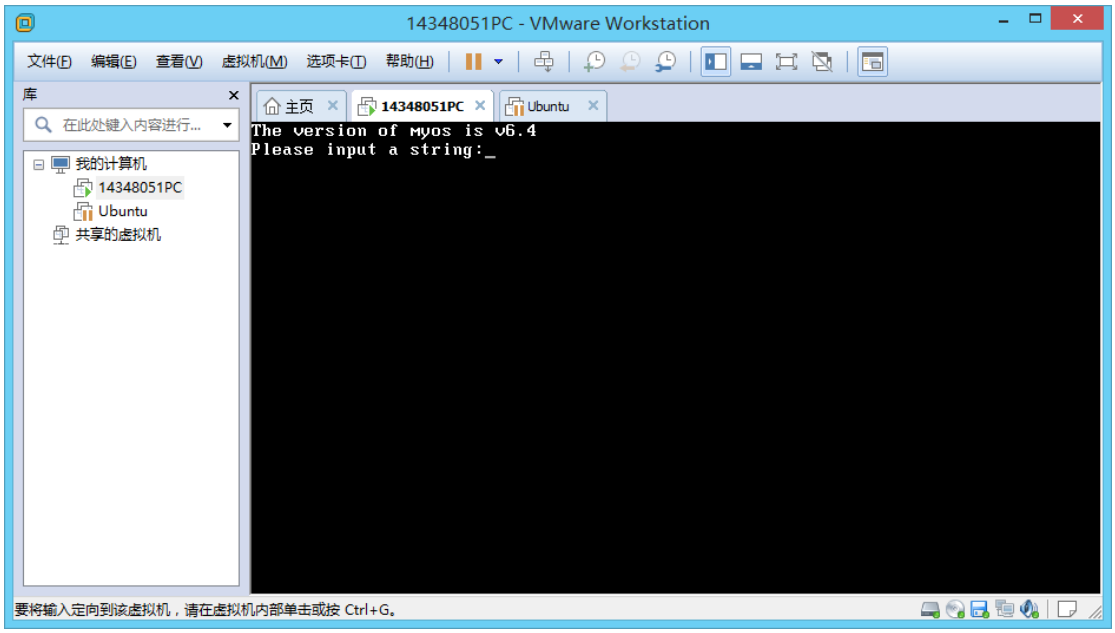

图 10 初始画面（终端 1）

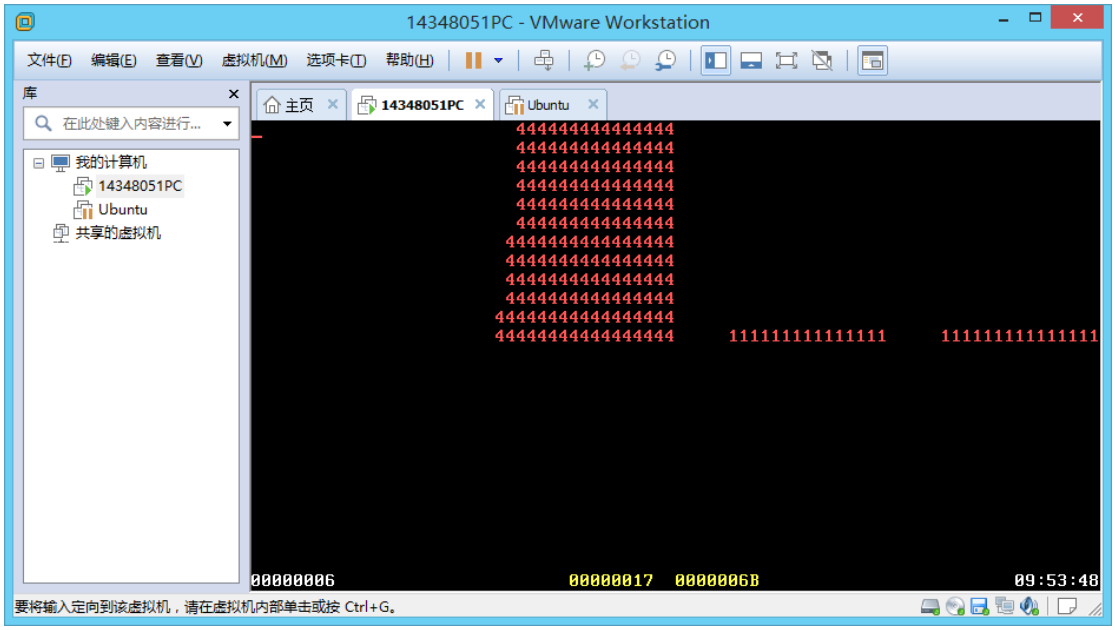

图 11 此时 3，4 号进程被阻塞（终端 0）

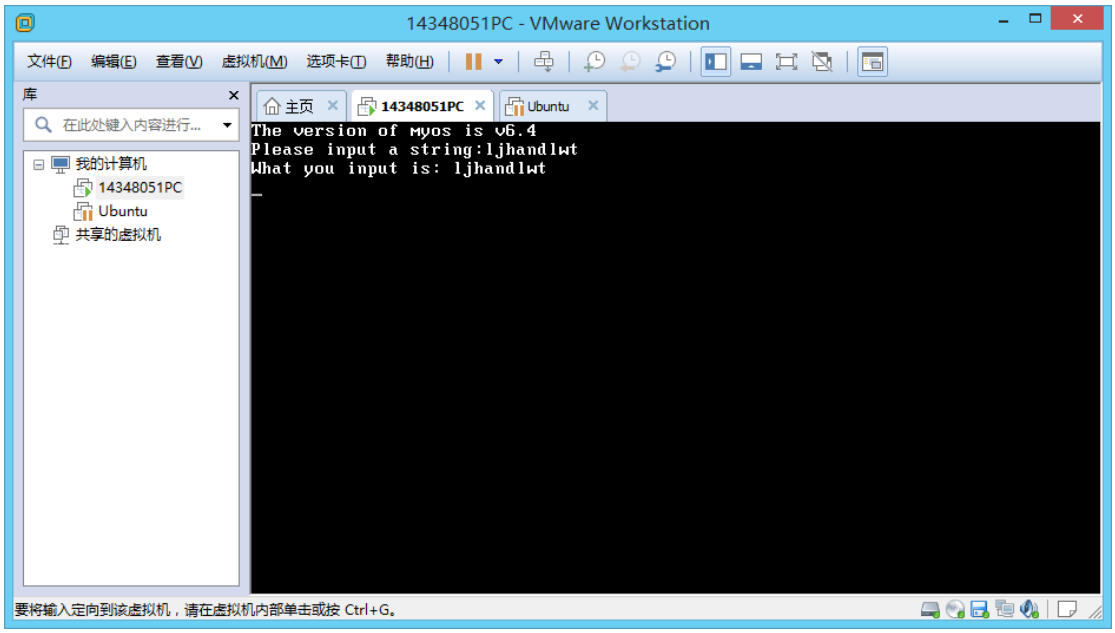

图 12 在 3 号进程输入字符串，4 号进程通过文件打印出来（终端 1）

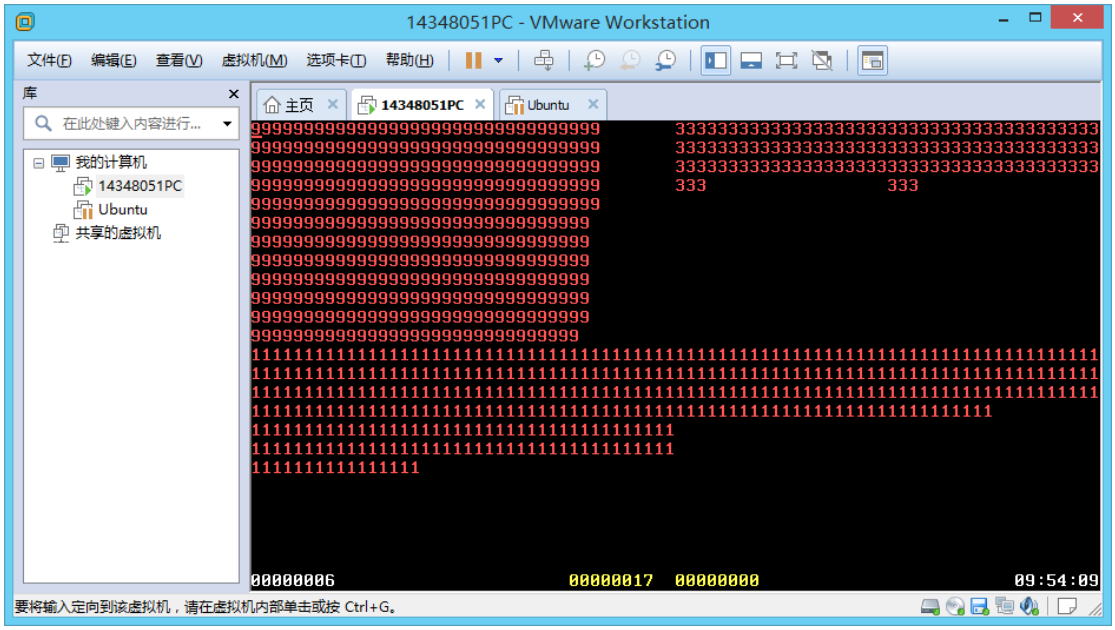

图 13 此时 3，4 号进程开始打印字符（终端 0）

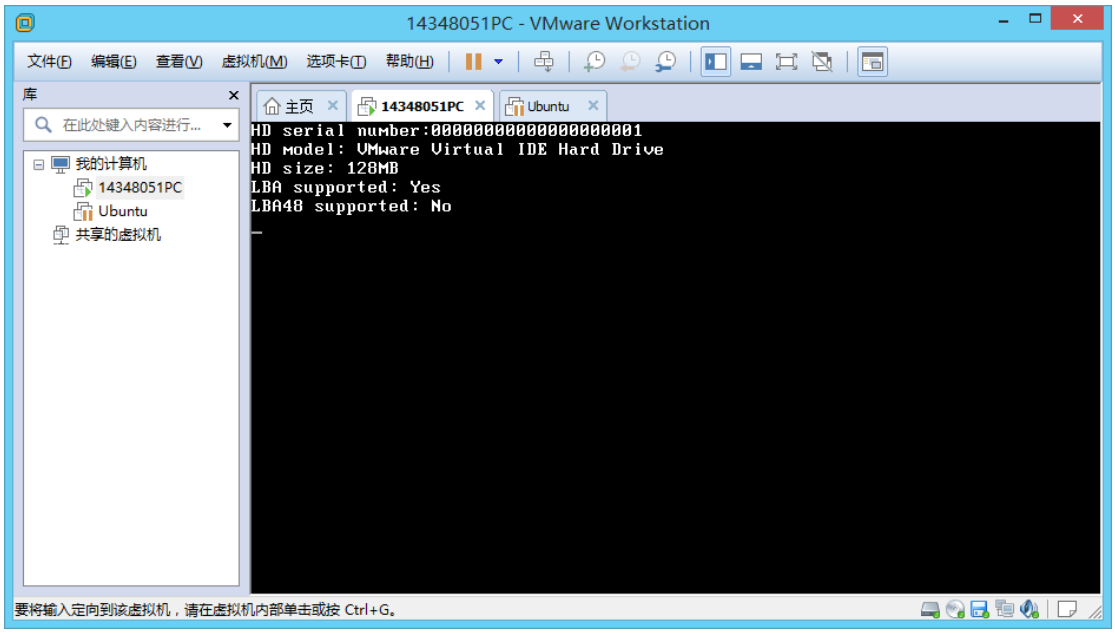

图 14 HD 输出硬盘信息（终端 2）

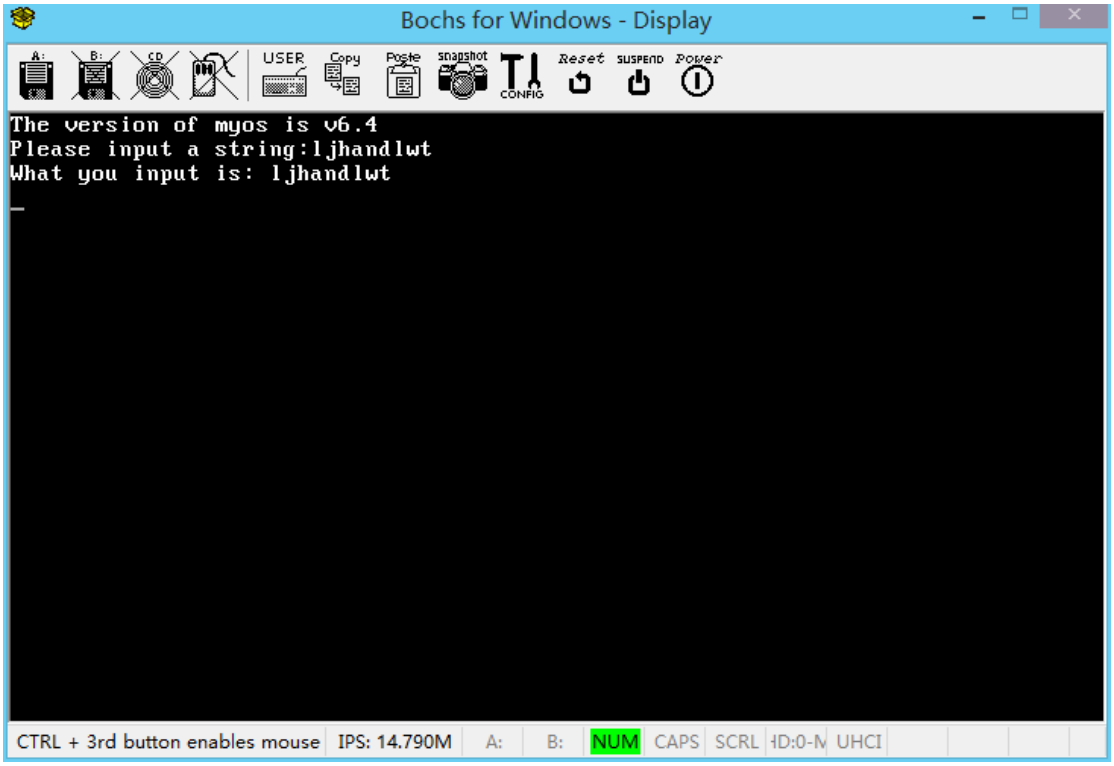

图 15 bochs 下的情况

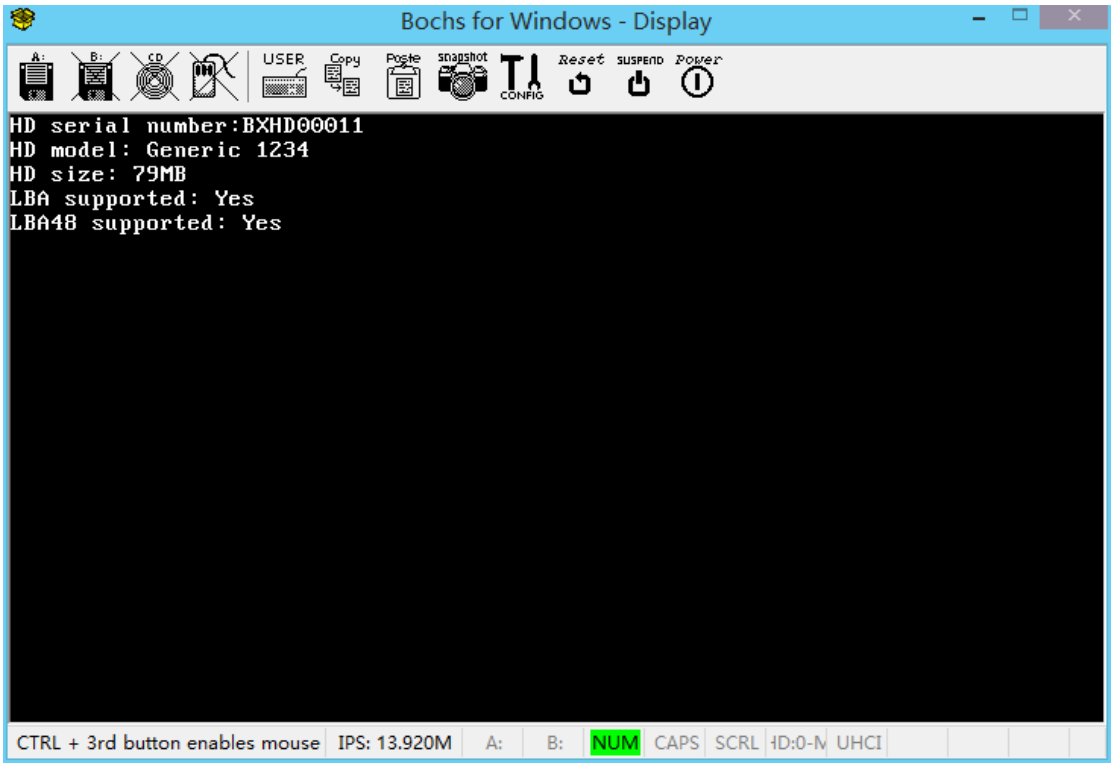

图 16 bochs 下硬盘信息

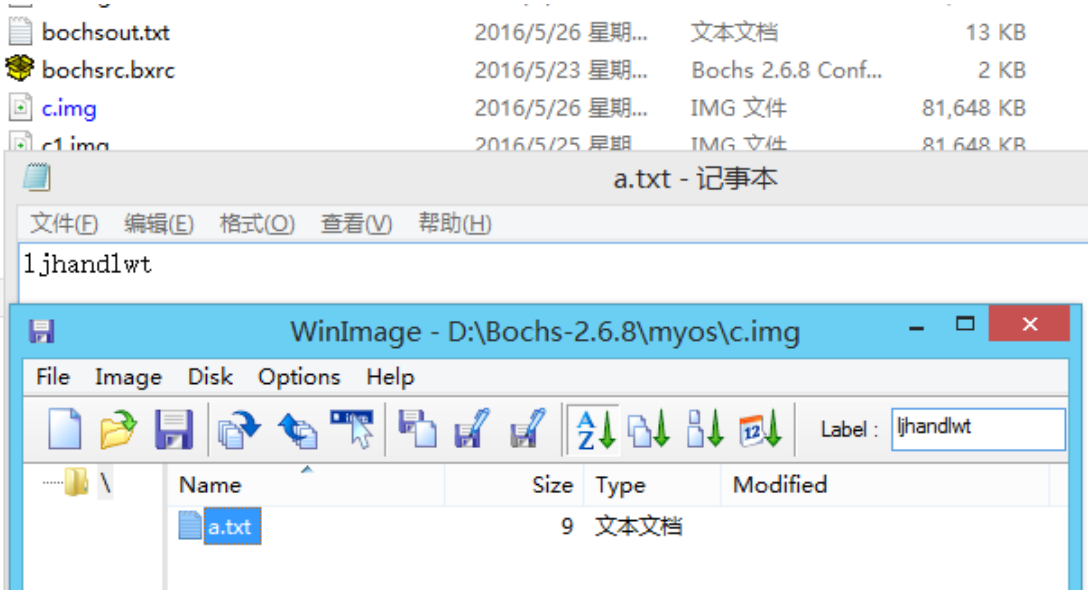

图 17 bochs 下的硬盘确实创建了这个文件

五. 实验总结

1. 理论上有了文件系统，各种东西都可以做了．像运行独立文件的进程，文件 log，输入输出做成文件，做一个 shell 等等．
2. 如果有时间的话，我应该会学学 FAT16 或者 FAT32 的，FAT12 只能储存 1.44M，即使对目前只有几十 M 的虚拟硬盘来说，都太浪费了．可惜时间不多，而且怕掉进大坑．FAT12 写过一次，哪里水浅哪里水深基本知道．
3. 文件系统这个词，它是有双重含义的．首先它是硬盘上的一个数据结构，像 FAT12，FAT16 等．其次它还是指操作系统对文件操作的一个模块，在本实验就是 fs 进程．
4. 还有一点的是，把硬盘控制器集成在硬盘内真是一个天才般的想法．这样做操作系统就无需考虑不同的物理硬盘所造成的差别，只要有一个统一的接口就可以了．

六. 参考资料

[1] Orange'S：一个操作系统的实现（自己动手做操作系统 第 2 版） 于谦 2009．6

[2] x86 汇编语言-从实模式到保护模式 李忠