**操作系统原理实验**

实验八

多终端、文件读写与操作

姓名： 吴侃

学号 : 14348134

班别： 2014级计算机系一班

日期： 2016.05.06 – 2016.05.19

目录

[零、特色先览 3](#_Toc1435499152)

[一、实验目的 3](#_Toc449432386)

[二、实验要求 3](#_Toc475082424)

[三、实验环境与工具 3](#_Toc426702818)

[（一）实验环境 3](#_Toc1489419291)

[（二）实验工具 4](#_Toc958484705)

[四、实验方案 4](#_Toc1411219622)

[(1) 多终端实现 4](#_Toc1358388224)

[(2) FAT12文件系统的文件读写 6](#_Toc1720540521)

[(3) FAT12文件系统的文件操作 7](#_Toc556843460)

[五、 实验操作 7](#_Toc989040878)

[多终端： 7](#_Toc1060246361)

[文件读写： 10](#_Toc1033569193)

[测试seekg, seekp 11](#_Toc1673450516)

[测试文件操作 12](#_Toc697977232)

[六、 小结 13](#_Toc575584213)

零、特色先览

本次实验的亮点包括:

1. 实现了多用户（终端）
2. 实现了文件读写（open, write, read, seekp, seekg）
3. 实现了文件操作（rm,cp,mv）

一、实验目的

实现多终端与FAT12文件系统中的文件读写、操作

二、实验要求

1. 实现多终端

2. 实现FAT12文件系统中的文件读写

3. 实现FAT12文件系统中的文件操作

三、实验环境与工具

## （一）实验环境

物理机操作系统: Arch Linux 4.5.4-1

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

## （二）实验工具

编辑器: Vim 7.4

汇编工具: NASM 2.11.08

C++编译器: g++ 6.1.1

链接工具: GNU ld 2.26.0.20160501

构建工具: GNU Make 4.1

调试工具: Bochs x86 Emulator 2.6.8

虚拟机: qemu-system-i386

VMWare Workstation 12 Pro

合并文件: dd

四、实验方案

## (1) 多终端实现

多终端的实现，需要修改进程调度过程，将shell分离出内核，保存和恢复用户的屏幕信息与光标位置。

1. 进程调度

在PCB表中新增一项UID, 用来标识该进程所属用户的ID，用UserID标记当前环境下的用户ID。当 UID等于0时，说明这是内核进程，该进程可以在任何用户环境下运行。其他用户进程，只在进程的UID等于UserID时，才被调度。

1. 将shell分离出内核

之前的版本将shell写在内核中，现在将其分离，成为一个用户程序。分离时最大的问题是RunProg函数的实现，由于RunProg函数涉及到很多内核参数，最终的解决方式为通过端口信息传输+信号量的方法，shell发送信息给内核，内核收到信息后执行程序。

1. 保存和恢复用户的屏幕信息与光标位置

这个比较简单，将段地址0xB800的80\*25个双字节拷贝到当前用户对应的储存屏幕数据的内存段，保存光标位置，然后切换用户（更改UserID）, 将新用户的原屏幕数据恢复到段0xB800处，恢复光标位置。储存屏幕数据的内存段，在kernel.asm中分配，使用与内核程序不一样的段（在C++中用char数组分配会导致链接出错，原因是链接时段越界）

D. 检测切换终端的按键

检测在内核中实现，当检测到对应按键时，更改UserID，并做屏幕信息、光标位置的保存与恢复操作。

## FAT12文件系统的文件读写

1. 文件读取

读取的过程为：

1. 从根目录找到对应的文件项，得到文件大小以及文件存储的第一个扇区。
2. 读取数据区中对应扇区的数据
3. 查找FAT，如果值大于等于0xFF8, 说明读取文件已经结束；否则为文件下一个扇区的扇区号，重复过程2，直至读完。需要注意，文件所在的最后一个扇区，数据可能不占满一个扇区。

B. 文件写入

1. 在根目录查找文件，若文件存在，打开文件；若文件不存在，创建新的文件（在根目录加上新的一项）
2. 在当前处理的扇区写入数据，如果扇区被填满，申请新的数据区扇区。申请方式为，从头开始查找空闲的FAT项，项为0x000的代表该项为空，对应扇区即为空闲扇区。更新FAT，让原扇区对应的FAT项等于新扇区的FAT编号，新扇区的FAT项为0xFFF，代表文件结束。重复过程2, 直至文件完全写入。
3. 注意事项：更新FAT时，要使用位运算，根据编号的奇偶性更新1.5字节的FAT项。

## FAT12文件系统的文件操作

1. rm操作

找到对应项，将文件名的第一个字符改为0xE5，即标记被删除（可恢复）

B. mv操作

找到对应项，修改文件名

C. cp操作

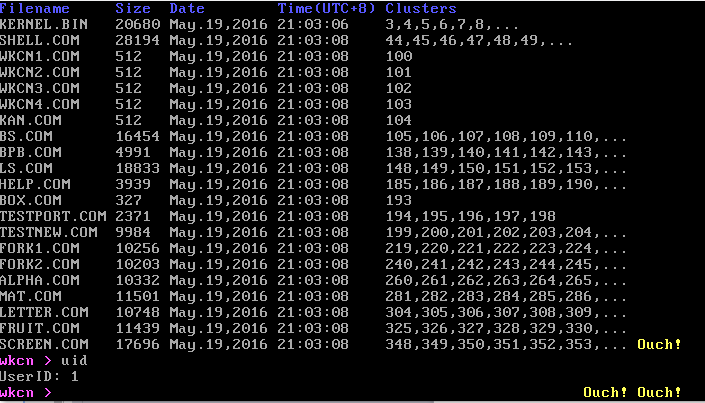
找到对应项，获取文件存储的第一个扇区，生成新的Entry并逐扇区写入新的空闲扇区，同时更新FAT，最终更新写入时间并将Entry写入根目录。

1. 实验操作

多终端：

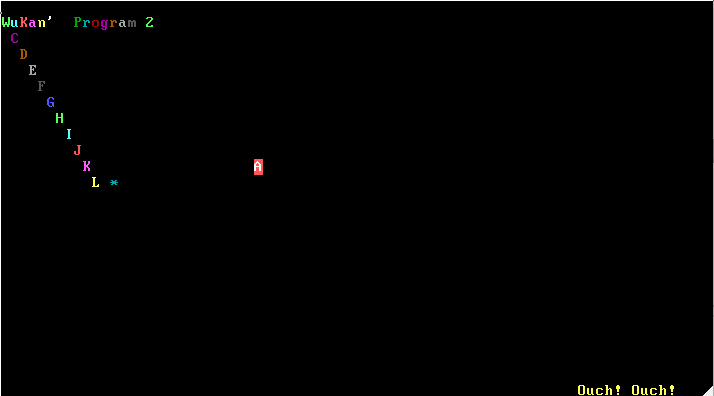
按下Alt + 1, Alt + 2, Alt + 3, Alt + 4可以切换用户（终端）

在终端1下输入ls等命令：

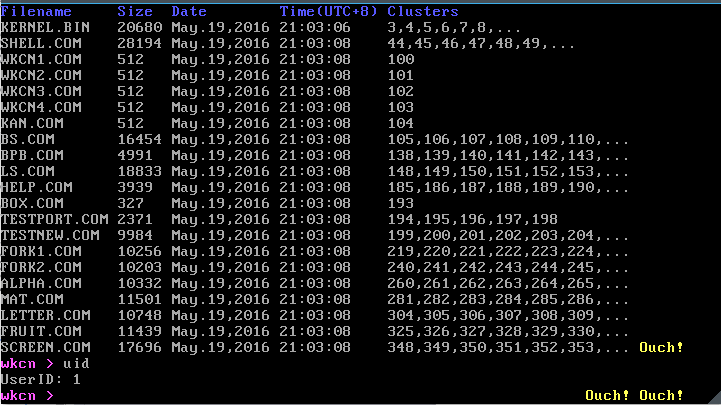


可以看到，UserID = 1

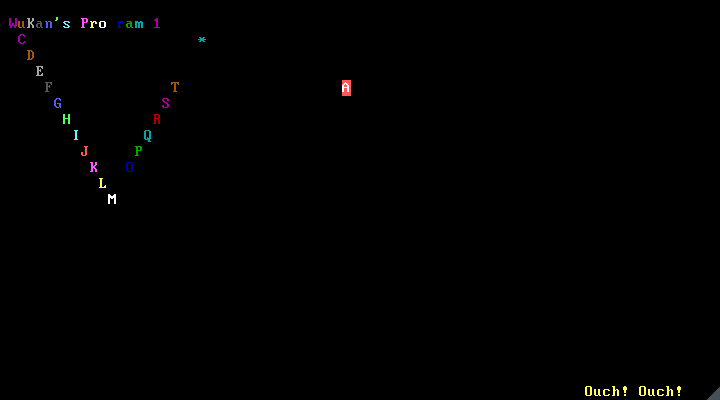
按ALT + 2, 切换到终端2, 执行用户程序1和2



再次输入ALT + 1, 切换为用户1

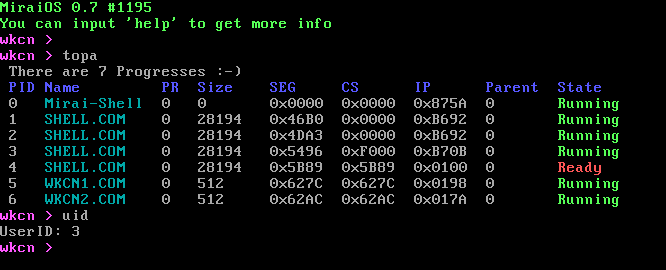


按ALT + 2, 切换到用户2



可以看到，不同终端运行的程序互不影响:-)

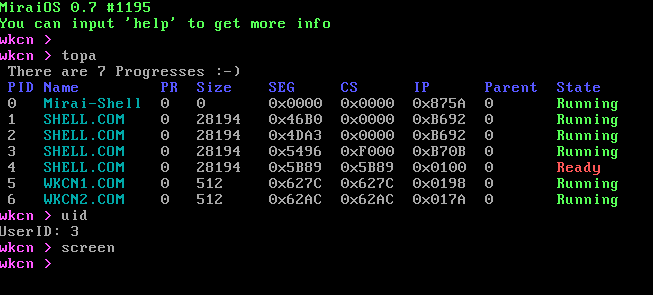
使用topa可以查看所有用户执行的进程：切换到终端3 (Alt + 3),输入topa



可以看到4个shell进程，代表4个终端，PID为5和6的进程为用户2的进程（这里没有显示UID），4号shell状态为Ready, 说明其还没有被使用过。这里的state指的是当UserID等于它们的UID时的state.

文件读写：

输入screen, 将当前屏幕信息保存到screen.txt中

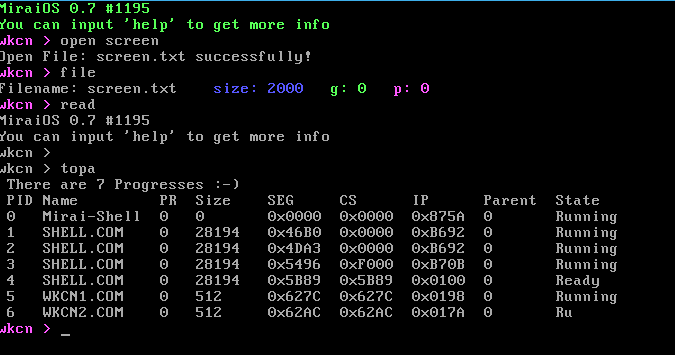


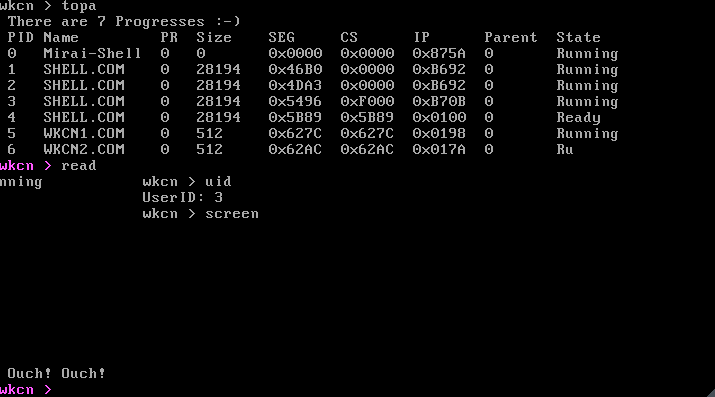
切换到终端4,

open scree: 打开screen.txt

file 查看文件状态

read 读取文件，每次最多读取1024个字节，有maxBufferSize决定



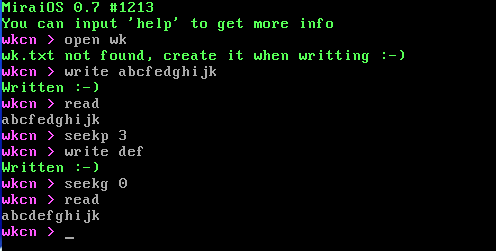


可以看到，屏幕信息被正确保存并且能够读写screen.

测试seekg, seekp

输入错误序列abcfedghijk

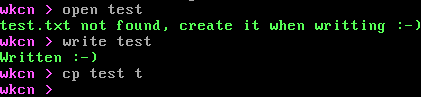
再进行更正



挂载镜像后，也可以看见创建了一个叫wk.txt的文本文件。

测试文件操作

创建一个新文件test.txt, 将其拷贝得到t.txt



ls

2016-05-20-121105_433x54_scrot

重命名t.txt为k.txt

2016-05-20-121131_149x22_scrot

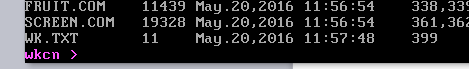
ls

2016-05-20-121140_181x54_scrot

删除test.txt和k.txt

2016-05-20-121152_155x39_scrot

ls



文件被删除

1. 小结

本次实验，我实现了多终端和文件读写及操作。

多终端的实现，困难之处在于将shell移除内核后，如何让shell根据指令执行程序，这涉及到进程间通信。我之前也采取过其它方式，比如把执行程序的函数写成系统中断，问题在于传递字符串比较困难，实现起来很复杂。另一种方式是将执行程序的函数放进一个头文件，问题在于读磁盘时栈不够大，并且涉及很多内核参数。

多终端的另一个问题是保存屏幕数据的内存的位置，最好的方式是另外开辟一个段，保证内核代码的偏移量不超过64k.

对于文件读写，我的方法是先用C++在Linux系统下实现FAT12文件系统的读写，然后移植到我的操作系统。在Linux系统下，我的主要问题为位运算写错了，导致写入FAT时出错。在我的操作系统下，主要问题为16位NASM与32位G++结合，C++的结构体无法使用内部函数（由于涉及到压栈），栈空间过小，不能在用户程序内通过栈获得缓冲区。我开始采用了进程通信的方法（include/os\_msg.h），无奈出现了错误。由于时间原因，我最终采取的方法是将缓冲区写在数据段内，而不是使用栈。

这次实验，我也了解了宏内核与微内核的区别以及各自的实现方式。我觉得我在操作系统的实验中已经很尽力了，从开始记录编译次数到现在，总共编译了1213次，不包括reset代码时的次数。我学到了很多，还有更多的东西需要探索～

附上Github的提交记录：

刚好一百次提交

