操作系统实验报告

学 生 姓 名　　苏靖博
学　　号　　20105050110
班　　级　　计实验20

实验名称	文件管理实验	实验序号	3
实验日期	Nov. 28 2022	实验人	苏靖博

一、实验目的和要求

1. 通过一个简单的多用户文件系统设计, 加深理解文件系统(File System)的内部功能及实现

2. 设计为二级文件系统

3. 具备基本文件系统功能; 具备保护码即可进行读写保护

二、相关背景知识

了解 xv6 源代码对文件系统 (file system) 的实现方法、数据结构的设计以及数据的存储方式. 能够模仿 xv6 源代码的思想和方法设计出满足要求的简易文件系统.

相关工具: 这次使用了 CLion 这款 IDE 工具, 使用 IDE 的好处是能够清晰的展示当前文件下函数及变量之间的结构关系, 能够更加容易的进行索引和跳转; 另外同时分屏参考相对复杂的 xv6 源代码更加清晰, 有助于提高编程效率.

三、实验内容

1. 为 Unix/Linux 系统设计一个简单的二级文件系统, 要求做到用户登录 (login)、登出 (logout)、创建文件 (create)、删除文件 (delete)、打开文件 (open)、关闭文件 (close)、读文件 (read)、写文件 (write)、列出目录 (ls / dir).

2. 列目录 (ls / dir)时要求文件名、物理地址、保护码及文件长度.

3. 文件可以进行读写保护.

项目链接🔗 https://github.com/Sue217/NCUT/tree/main/OS/lab3

四、关键数据结构与函数的说明

```
struct superblock {
  int size;     // size of disk blocks
  int nblocks;  // number of disk blocks
  int ninodes;  // number of inodes
};  // 12 bytes

struct inode {
  int size;
  int inum;              // inode number (index of inodes array)
  int first_block;       // first block of inode
};  // 12 bytes

struct diskblock {
  int next_block;        // next block number
  char data[BLOCK_SIZE];  // block content
```

```
};  // 516 bytes


struct file {
  short type;           // folder or just file
  short readable;
  short writable;
  short executable;
  struct inode* ip;      // index of inode
  char name[NAME_SIZE];   // file name
};  // 32 bytes

struct superblock sb;
struct inode* inodes;
struct diskblock* dbs;
struct file* files;
```

项目链接🔗 https://github.com/Sue217/NCUT/tree/main/OS/lab3

五、编译与执行过程截图

编译执行: `gcc -g getb.c fixsize.c finde.c print.c user.c fs.c main.c -std=c11 -Wall -Werror && ./a.out`

valid command:

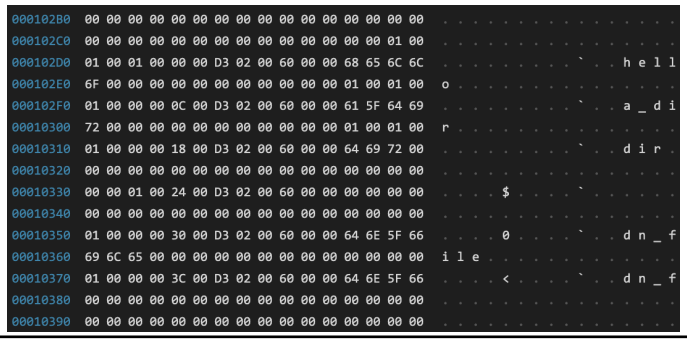
```
login
create: create file_name mode, type
write: write file_name content
read: read file_name
ls: ls file_name
delete: delete file_name
logout
clear
```

```
Login Successfully!

#sujingbo $
#sujingbo $
#sujingbo $ sue
command not found: sue
#sujingbo $ create hello 7 0
#sujingbo $ write hello hello, world
#sujingbo $ read hello

hello, world
```

```
#sujingbo $ create dir 6 1
#sujingbo $ write dir can I write it?
Not a file [Write failed!]
#sujingbo $ read dir
Not a file [Read failed!]
#sujingbo $ create dn_f 2 0
#sujingbo $ write dn_f can I write it?
#sujingbo $ read dn_f
[Read failed!]
#sujingbo $ ls
-rwx    0x600002d30000    12    hello
drw-    0x600002d3000c    0    a_dir
drw-    0x600002d30018    0    dir
---x    0x600002d30024    0    dn_file
--w-    0x600002d30030    0    dn_file
--w-    0x600002d3003c    15    dn_f
#sujingbo $ clear
#sujingbo $ delete dn_file
#sujingbo $ ls
-rwx    0x600002d30000    12    hello
drw-    0x600002d3000c    0    a_dir
drw-    0x600002d30018    0    dir
---x    0x600002d30024    0
--w-    0x600002d30030    0    dn_file
--w-    0x600002d3003c    15    dn_f
#sujingbo $ logout

Bye 👋🏻
```

```
⚙       00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded Text
00000000  04 02 00 00 80 00 00 00 10 00 00 00 0C 00 00 00  ................
00000010  00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00  ................
00000020  01 00 00 00 00 00 00 00 02 00 00 00 02 00 00 00  ................
00000030  00 00 00 00 03 00 00 00 03 00 00 00 00 00 00 00  ................
00000040  04 00 00 00 04 00 00 00 0F 00 00 00 05 00 00 00  ................
00000050  05 00 00 00 00 00 00 00 06 00 00 00 FF FF FF FF  ................
00000060  00 00 00 00 07 00 00 00 FF FF FF FF 00 00 00 00  ................
00000070  08 00 00 00 FF FF FF FF 00 00 00 00 09 00 00 00  ................
00000080  FF FF FF FF 00 00 00 00 0A 00 00 00 FF FF FF FF  ................
00000090  00 00 00 00 0B 00 00 00 FF FF FF FF 00 00 00 00  ................
000000A0  0C 00 00 00 FF FF FF FF 00 00 00 00 0D 00 00 00  ................
000000B0  FF FF FF FF 00 00 00 00 0E 00 00 00 FF FF FF FF  ................
000000C0  00 00 00 00 0F 00 00 00 FF FF FF FF FE FF FF FF  ................
000000D0  68 65 6C 6C 6F 2C 20 77 6F 72 6C 64 00 00 00 00  hello, world....
000000E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

```
000000AD0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000AE0  FE FF FF FF 63 61 6E 20 49 20 77 72 69 74 65 20  ....can I write
00000AF0  69 74 3F 00 00 00 00 00 00 00 00 00 00 00 00 00  it?.............
00000B00  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000B10  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000B20  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000B30  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```



六、实验结果与分析

1. 能够实现用户登录并创建文件与进行初始化.

2. 能够顺利开关文件并在磁盘上进行读写操作.

3. 能够列出全部文件并区分文件类型 (文件?文件夹).

4. 能够进行读写保护, 并且对文件夹不能够进行读写.

七、调试时遇到的问题及解决方法

起初遇到较难的问题是如何对磁盘进行读写. 以往我们会选择开辟栈空间 (array) 实现存储与读写, 但是这样不能够实现持久化. 在进行测试的时候我遇到了不能够持久化存储的问题, 当我一次 create 文件并初始化后, 可以看到磁盘已经写入, 但是再次重新运行该程序并注释掉 create 操作时, 出现了 segmentation fault.

在实现时发现困难的点在于确定写磁盘的位置. 如果没有确定的结构或是规范读写是很容易出现由于非法访问导致的 segmentation fault. 于是引入了 file 这个结构来配合 inode 存储文件, 并在 file 结构中留一个指向 inode 的指针 *ip, 这样我们的文件系统就可以顺利工作了.

对于 binary file 的观测我选择使用 vscode 中专门用来检查 binary file 的插件 Hex Editor 完成对我的文件系统读写的检查.

八、调试后的程序源代码

🔗 https://github.com/Sue217/NCUT/tree/main/OS/lab3

九、实验体会

1. 本次实验让我熟悉了 xv6 file system 基本实现逻辑.

2. 本次实验学会了如何在磁盘上进行读写操作, 并完成内容检查及调试

3. 加强了代码和调试的功底, 相比前两次实验, 本次实验代码量更大, 维护更加复杂, 让我获益匪浅.