**操作系统课程设计实验报告**

实验题目： 简单文件系统的实现

组 号： 周五下午16组

老师姓名： 赵伟华

日 期： 2018 年 06 月 18 日

目录·Content

[一 实验小组分工 1](#_Toc517118153)

[二 题目介绍 1](#_Toc517118154)

[三 实验思路 2](#_Toc517118155)

[3.1程序设计思想 2](#_Toc517118156)

[3.2关键数据结构设计： 3](#_Toc517118157)

[3.3全局变量定义 4](#_Toc517118158)

[3.4虚拟磁盘空间布局 4](#_Toc517118159)

[四 遇到问题及解决方法 4](#_Toc517118160)

[五 核心代码与实验结果及实验结果分析 5](#_Toc517118161)

[5.1 核心代码与实验结果 5](#_Toc517118162)

[5.2 实验结果分析 21](#_Toc517118163)

[六 实验改进与总结 21](#_Toc517118164)

[6.1 实验改进 21](#_Toc517118165)

[6.2 实验总结 21](#_Toc517118166)

[七 参考文献 22](#_Toc517118167)

# 一 实验小组分工

**对于实验3和4，书写小组分工，每个成员按如下格式：）**

**姓名，学号，专业，班级，负责题目X**

**模块一：文件系统总体规划、初始化函数、ls命令、cd命令、exit命令：**

**丁乃文，15198704，数字媒体技术，15222513**

**模块二：**

模块1：

模块2：

模块3：

模块4：

这样的形式，每个小组成员书写属于自己的那一块）

# 二 题目介绍

通过具体的文件存储空间的管理、文件物理结构、目录结构和文件操作的实现，加深对文件系统内部的数据结构、功能以及实现过程的理解。

本实验要求：

1. 在内存中开辟一个虚拟磁盘空间作为文件存储分区，在其上实现一个简单的基于多级目录的单用户单任务系统中的文件系统。在推出该文件系统的使用时，应将虚拟磁盘上的内容以一个文件的方式保存到磁盘上，一遍下次可以将它恢复到内存的虚拟磁盘中。
2. 文件物理结构可采用显式链接或其他结构
3. 空闲磁盘空间的管理可选择FAT表、位示图或其他办法。
4. 文件目录结构采用多级目录结构。为简单起见，可以不使用索引结点，每个目录项应包含文件名、物理地址、长度等信息，还可以通过目录项实现对文件的读和写的保护。
5. 需要提供一以下操作命令
   * 1. my\_format
     2. my\_mkdir
     3. my\_rmdir
     4. my\_ls
     5. my\_cd
     6. my\_create
     7. my\_open
     8. my\_close
     9. my\_write
     10. my\_read
     11. my\_rm
     12. my\_exitsys

# 三 实验思路

## 3.1程序设计思想

1. 在该虚拟文件系统启动时，申请一块内存作为磁盘空间。
2. 将这块内存空间进行格式化，本系统仿照的是FAT16文件系统，其结构如下

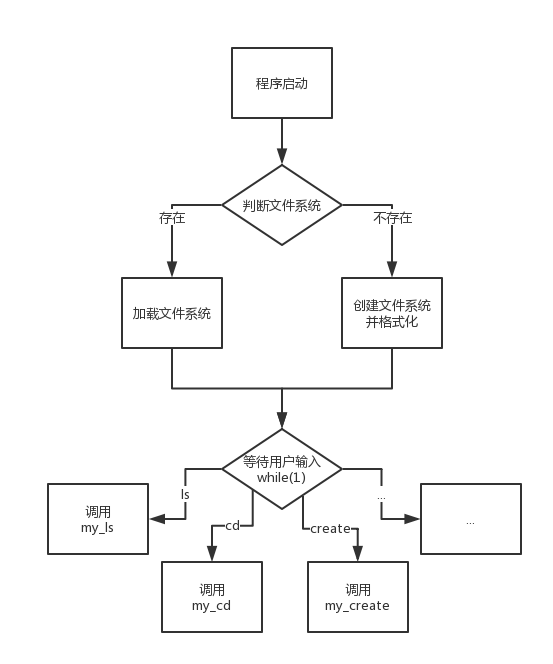
1块 2块 2块 995块

引导块 FAT1 FAT2 数据区

格式化时，主要包括引导块，FAT1，FAT2，的一些初始化工作。例如设置文件魔数，文件系统的信息，FAT1，FAT2的信息等等

1. 根据用户输入的命令，调用对应的my\_函数.

程序流程图如下：



图表 1 程序流程图

## 3.2关键数据结构设计：

文件控制块：

typedef struct FCB {

char filename[8]; *//文件名*

char exname[3]; *//文件扩展名*

unsigned char attribute; *//文件属性字段(目录or文件)*

unsigned short time; *//创建时间*

unsigned short date; *//创建日期*

unsigned short first; *//起始盘块号*

unsigned short length; *//文件长度*

char free; *//表示目录项是否为空*

} fcb;

文件分配表：

typedef struct FAT {

unsigned short id;

} fat;

用户打开文件表：

当打开一个文件时，必须将文件的目录项中的所有内容全部复制到内存中，同时还要记录有关文件操作的动态信息，如读写指针的值等。在本实例中实现的是一个用于单用户单任务系统的文件系统，为简单起见，我们把用户文件描述符表和内存FCB表合在一起，称为用户打开文件表，表项数目为10，即一个用户最多可同时打开10个文件。然后用一个数组来描述，则数组下标即某个打开文件的描述符。另外，我们在用户打开文件表中还设置了一个字段“char dir[80]”，用来记录每个打开文件所在的目录名，以方便用户打开不同目录下具有相同文件名的不同文件。

typedef struct USEROPEN {

char filename[8];

char exname[3];

unsigned char attribute;

unsigned short time;

unsigned short date;

unsigned short first;

unsigned short length;

char free;

int dirno; *// 父目录文件起始盘块号*

int diroff; *// 该文件对应的 fcb 在父目录中的逻辑序号*

char dir[MAXOPENFILE][80]; *// 全路径信息*

int count;

char fcbstate; *// 是否修改 1是 0否*

char topenfile; *// 0: 空 openfile*

} useropen;

引导块BLOCK0：

在引导块中主要存放逻辑磁盘的相关描述信息，比如磁盘块大小、磁盘块数量、文件分配表、根目录区、数据区在磁盘上的起始位置等。如果是引导盘，还要存放操作系统的引导信息。本实例是在内存的虚拟磁盘中创建一个文件系统，因此所包含的内容比较少，只有磁盘块大小、磁盘块数量、数据区开始位置、根目录文件开始位置等。

typedef struct BLOCK {

char magic\_number[8];

char information[200];

unsigned short root;

unsigned char\* startblock;

} block0;

## 3.3全局变量定义

指向虚拟磁盘的起始地址：

unsigned char\* myvhard;

用户打开文件表数组：

useropen openfilelist[MAXOPENFILE];

记录虚拟磁盘上的数据区开始位置：

unsigned char\* startp;

## 3.4虚拟磁盘空间布局

由于真正的磁盘操作需要涉及到设备的驱动程序，所以本实例是在内存中申请一块空间作为虚拟磁盘使用，我们的文件系统就建立在这个虚拟磁盘上。虚拟磁盘一共划分成1000个磁盘块，每个块1024个字节，其布局格式是模仿FAT文件系统设计的，其中引导块占一个盘块，两张FAT各占2个盘块，剩下的空间全部是数据区，在对虚拟磁盘进行格式化的时候，将把数据区第1块（即虚拟磁盘的第6块）分配给根目录文件，如下图所示：



图表 2 虚拟磁盘空间布局

# 四 遇到问题及解决方法

**（此处请同学们务必认真规范书写，对于每个实验，写几条遇到的问题，以条目化形式呈现，如 1. 2. 3. ）**

# 五 核心代码与实验结果及实验结果分析

## 5.1 核心代码与实验结果

格式化磁盘：

void my\_format()

{

*/\*\**

*\* 初始化前五个磁盘块*

*\* 设定第六个磁盘块为根目录磁盘块*

*\* 初始化 root 目录： 创建 . 和 .. 目录*

*\* 写入 FILENAME 文件 （写入磁盘空间）*

*\*/*

block0\* boot = (block0\*)myvhard;

strcpy(boot->magic\_number, "10101010");

strcpy(boot->information, "fat file system");

boot->root = 5;

boot->startblock = myvhard + BLOCKSIZE \* 5;

fat\* fat1 = (fat\*)(myvhard + BLOCKSIZE);

fat\* fat2 = (fat\*)(myvhard + BLOCKSIZE \* 3);

int i;

*for* (i = 0; i < 6; i++) {

fat1[i].id = END;

fat2[i].id = END;

}

*for* (i = 6; i < 1000; i++) {

fat1[i].id = FREE;

fat2[i].id = FREE;

}

*// 5th block is root*

fcb\* root = (fcb\*)(myvhard + BLOCKSIZE \* 5);

strcpy(root->filename, ".");

strcpy(root->exname, "di");

root->attribute = 0; *// dir file*

time\_t rawTime = time(NULL);

struct tm\* time = localtime(&rawTime);

*// 5 6 5 bits*

root->time = time->tm\_hour \* 2048 + time->tm\_min \* 32 + time->tm\_sec / 2;

*// 7 4 5 bits; year from 2000*

root->date = (time->tm\_year - 100) \* 512 + (time->tm\_mon + 1) \* 32 + (time->tm\_mday);

root->first = 5;

root->free = 1;

root->length = 2 \* sizeof(fcb);

fcb\* root2 = root + 1;

memcpy(root2, root, sizeof(fcb));

strcpy(root2->filename, "..");

*for* (i = 2; i < (int)(BLOCKSIZE / sizeof(fcb)); i++) {

root2++;

strcpy(root2->filename, "");

root2->free = 0;

}

FILE\* fp = fopen(FILENAME, "w");

fwrite(myvhard, SIZE, 1, fp);

fclose(fp);

}

创建目录：

a) 调用do\_read读入当前目录文件到内存，检查新建文件目录是否重名

b) 分配一个空闲的打开文件表项

c) 分配一个空闲的盘块

d) 在当前目录中问新建目录寻找一个空闲的目录项

e) 设置FCB，文件的属性信息

f) 创建特殊的两个目录项‘.’,‘..’

g) 返回

void my\_mkdir(char\* dirname)

{

*/\*\**

*\* 当前目录：当前打开目录项表示的目录*

*\* 该目录：以下指创建的目录*

*\* 父目录：指该目录的父目录*

*\* 如:*

*\* 我现在在 root 目录下， 输入命令 mkdir a/b/bb*

*\* 表示 在 root 目录下的 a 目录下的 b 目录中创建 bb 目录*

*\* 这时，父目录指 b，该目录指 bb，当前目录指 root*

*\* 以下都用这个表达，简单情况下，当前目录和父目录是一个目录*

*\* 来不及了，先讨论简单情况，即 mkdir bb*

*\*/*

int i = 0;

char text[MAX\_TEXT\_SIZE];

char\* fname = strtok(dirname, ".");

char\* exname = strtok(NULL, ".");

*if* (exname != NULL) {

printf("you can not use extension\n");

*return*;

}

*// 读取父目录信息*

openfilelist[currfd].count = 0;

int filelen = do\_read(currfd, openfilelist[currfd].length, text);

fcb\* fcbptr = (fcb\*)text;

*// 查找是否重名*

*for* (i = 0; i < (int)(filelen / sizeof(fcb)); i++) {

*if* (strcmp(dirname, fcbptr[i].filename) == 0 && fcbptr->attribute == 0) {

printf("dir has existed\n");

*return*;

}

}

*// 申请一个打开目录表项*

int fd = get\_free\_openfilelist();

*if* (fd == -1) {

printf("openfilelist is full\n");

*return*;

}

*// 申请一个磁盘块*

unsigned short int block\_num = get\_free\_block();

*if* (block\_num == END) {

printf("blocks are full\n");

openfilelist[fd].topenfile = 0;

*return*;

}

*// 更新 fat 表*

fat\* fat1 = (fat\*)(myvhard + BLOCKSIZE);

fat\* fat2 = (fat\*)(myvhard + BLOCKSIZE \* 3);

fat1[block\_num].id = END;

fat2[block\_num].id = END;

*// 在父目录中找一个空的 fcb，分配给该目录 ??未考虑父目录满的情况??*

*for* (i = 0; i < (int)(filelen / sizeof(fcb)); i++) {

*if* (fcbptr[i].free == 0) {

*break*;

}

}

openfilelist[currfd].count = i \* sizeof(fcb);

openfilelist[currfd].fcbstate = 1;

*// 初始化该 fcb*

fcb\* fcbtmp = (fcb\*)malloc(sizeof(fcb));

fcbtmp->attribute = 0;

time\_t rawtime = time(NULL);

struct tm\* time = localtime(&rawtime);

fcbtmp->date = (time->tm\_year - 100) \* 512 + (time->tm\_mon + 1) \* 32 + (time->tm\_mday);

fcbtmp->time = (time->tm\_hour) \* 2048 + (time->tm\_min) \* 32 + (time->tm\_sec) / 2;

strcpy(fcbtmp->filename, dirname);

strcpy(fcbtmp->exname, "di");

fcbtmp->first = block\_num;

fcbtmp->length = 2 \* sizeof(fcb);

fcbtmp->free = 1;

do\_write(currfd, (char\*)fcbtmp, sizeof(fcb), 2);

*// 设置打开文件表项*

openfilelist[fd].attribute = 0;

openfilelist[fd].count = 0;

openfilelist[fd].date = fcbtmp->date;

openfilelist[fd].time = fcbtmp->time;

openfilelist[fd].dirno = openfilelist[currfd].first;

openfilelist[fd].diroff = i;

strcpy(openfilelist[fd].exname, "di");

strcpy(openfilelist[fd].filename, dirname);

openfilelist[fd].fcbstate = 0;

openfilelist[fd].first = fcbtmp->first;

openfilelist[fd].free = fcbtmp->free;

openfilelist[fd].length = fcbtmp->length;

openfilelist[fd].topenfile = 1;

strcat(strcat(strcpy(openfilelist[fd].dir, (char\*)(openfilelist[currfd].dir)), dirname), "/");

*// 设置 . 和 .. 目录*

fcbtmp->attribute = 0;

fcbtmp->date = fcbtmp->date;

fcbtmp->time = fcbtmp->time;

strcpy(fcbtmp->filename, ".");

strcpy(fcbtmp->exname, "di");

fcbtmp->first = block\_num;

fcbtmp->length = 2 \* sizeof(fcb);

do\_write(fd, (char\*)fcbtmp, sizeof(fcb), 2);

fcb\* fcbtmp2 = (fcb\*)malloc(sizeof(fcb));

memcpy(fcbtmp2, fcbtmp, sizeof(fcb));

strcpy(fcbtmp2->filename, "..");

fcbtmp2->first = openfilelist[currfd].first;

fcbtmp2->length = openfilelist[currfd].length;

fcbtmp2->date = openfilelist[currfd].date;

fcbtmp2->time = openfilelist[currfd].time;

do\_write(fd, (char\*)fcbtmp2, sizeof(fcb), 2);

*// 关闭该目录的打开文件表项，close 会修改父目录中对应该目录的 fcb 信息*

*/\*\**

*\* 这里注意，一个目录存在 2 个 fcb 信息，一个为该目录下的 . 目录文件，一个为父目录下的 fcb。*

*\* 因此，这俩个fcb均需要修改，前一个 fcb 由各个函数自己完成，后一个 fcb 修改由 close 完成。*

*\* 所以这里，需要打开文件表，再关闭文件表，实际上更新了后一个 fcb 信息。*

*\*/*

my\_close(fd);

free(fcbtmp);

free(fcbtmp2);

*// 修改父目录 fcb*

fcbptr = (fcb\*)text;

fcbptr->length = openfilelist[currfd].length;

openfilelist[currfd].count = 0;

do\_write(currfd, (char\*)fcbptr, sizeof(fcb), 2);

openfilelist[currfd].fcbstate = 1;

}

进入目录：

a) Open指定的目录名，调用read读入该父目录到内存

b) 检查新的当前目录名是否存在

c) 关闭原当前目录

d) 设置当前目录为该目录

void my\_cd(char\* dirname)

{

int i = 0;

int tag = -1;

int fd;

*if* (openfilelist[currfd].attribute == 1) {

*// if not a dir*

printf("you are in a data file, you could use 'close' to exit this file\n");

*return*;

}

char\* buf = (char\*)malloc(10000);

openfilelist[currfd].count = 0;

do\_read(currfd, openfilelist[currfd].length, buf);

fcb\* fcbptr = (fcb\*)buf;

*// 查找目标 fcb*

*for* (i = 0; i < (int)(openfilelist[currfd].length / sizeof(fcb)); i++, fcbptr++) {

*if* (strcmp(fcbptr->filename, dirname) == 0 && fcbptr->attribute == 0) {

tag = 1;

*break*;

}

}

*if* (tag != 1) {

printf("my\_cd: no such dir\n");

*return*;

} *else* {

*// . 和 .. 检查*

*if* (strcmp(fcbptr->filename, ".") == 0) {

*return*;

} *else* *if* (strcmp(fcbptr->filename, "..") == 0) {

*if* (currfd == 0) {

*// root*

*return*;

} *else* {

currfd = my\_close(currfd);

*return*;

}

} *else* {

*// 其他目录*

fd = get\_free\_openfilelist();

*if* (fd == -1) {

*return*;

}

openfilelist[fd].attribute = fcbptr->attribute;

openfilelist[fd].count = 0;

openfilelist[fd].date = fcbptr->date;

openfilelist[fd].time = fcbptr->time;

strcpy(openfilelist[fd].filename, fcbptr->filename);

strcpy(openfilelist[fd].exname, fcbptr->exname);

openfilelist[fd].first = fcbptr->first;

openfilelist[fd].free = fcbptr->free;

openfilelist[fd].fcbstate = 0;

openfilelist[fd].length = fcbptr->length;

strcat(strcat(strcpy(openfilelist[fd].dir, (char\*)(openfilelist[currfd].dir)), dirname), "/");

openfilelist[fd].topenfile = 1;

openfilelist[fd].dirno = openfilelist[currfd].first;

openfilelist[fd].diroff = i;

currfd = fd;

}

}

}

删除目录：

a) Read读入当前目录文件内容到内存，检查要删除的文件目录是否存在

b) 检查该目录是否为空

c) 检查是否已经打开，打开用close则关闭

d) 回收给目录文件的磁盘块

e) 修改该目录文件的目录项

f) 修改用户打开表项的长度信息

g) 返回

void my\_rmdir(char\* dirname)

{

int i, tag = 0;

char buf[MAX\_TEXT\_SIZE];

*// 排除 . 和 .. 目录*

*if* (strcmp(dirname, ".") == 0 || strcmp(dirname, "..") == 0) {

printf("can not remove . and .. special dir\n");

*return*;

}

openfilelist[currfd].count = 0;

do\_read(currfd, openfilelist[currfd].length, buf);

*// 查找要删除的目录*

fcb\* fcbptr = (fcb\*)buf;

*for* (i = 0; i < (int)(openfilelist[currfd].length / sizeof(fcb)); i++, fcbptr++) {

*if* (fcbptr->free == 0)

*continue*;

*if* (strcmp(fcbptr->filename, dirname) == 0 && fcbptr->attribute == 0) {

tag = 1;

*break*;

}

}

*if* (tag != 1) {

printf("no such dir\n");

*return*;

}

*// 无法删除非空目录*

*if* (fcbptr->length > 2 \* sizeof(fcb)) {

printf("can not remove a non empty dir\n");

*return*;

}

*// 更新 fat 表*

int block\_num = fcbptr->first;

fat\* fat1 = (fat\*)(myvhard + BLOCKSIZE);

int nxt\_num = 0;

*while* (1) {

nxt\_num = fat1[block\_num].id;

fat1[block\_num].id = FREE;

*if* (nxt\_num != END) {

block\_num = nxt\_num;

} *else* {

*break*;

}

}

fat1 = (fat\*)(myvhard + BLOCKSIZE);

fat\* fat2 = (fat\*)(myvhard + BLOCKSIZE \* 3);

memcpy(fat2, fat1, BLOCKSIZE \* 2);

*// 更新 fcb*

fcbptr->date = 0;

fcbptr->time = 0;

fcbptr->exname[0] = '\0';

fcbptr->filename[0] = '\0';

fcbptr->first = 0;

fcbptr->free = 0;

fcbptr->length = 0;

openfilelist[currfd].count = i \* sizeof(fcb);

do\_write(currfd, (char\*)fcbptr, sizeof(fcb), 2);

*// 删除目录需要相应考虑可能删除 fcb，也就是修改父目录 length*

*// 这里需要注意：因为删除中间的 fcb，目录有效长度不变，即 length 不变*

*// 因此需要考虑特殊情况，即删除最后一个 fcb 时，极有可能之前的 fcb 都是空的，这是需要*

*// 循环删除 fcb (以下代码完成)，可能需要回收 block 修改 fat 表等过程(do\_write 完成)*

int lognum = i;

*if* ((lognum + 1) \* sizeof(fcb) == openfilelist[currfd].length) {

openfilelist[currfd].length -= sizeof(fcb);

lognum--;

fcbptr = (fcb \*)buf + lognum;

*while* (fcbptr->free == 0) {

fcbptr--;

openfilelist[currfd].length -= sizeof(fcb);

}

}

*// 更新父目录 fcb*

fcbptr = (fcb\*)buf;

fcbptr->length = openfilelist[currfd].length;

openfilelist[currfd].count = 0;

do\_write(currfd, (char\*)fcbptr, sizeof(fcb), 2);

openfilelist[currfd].fcbstate = 1;

}

列出目录和文件项命令：

a) Read当前目录到内存

b) 读出目录文件的信息，显示到屏幕上

c) 返回

void my\_ls()

{

*// 判断是否是目录*

*if* (openfilelist[currfd].attribute == 1) {

printf("data file\n");

*return*;

}

char buf[MAX\_TEXT\_SIZE];

int i;

*// 读取当前目录文件信息(一个个fcb), 载入内存*

openfilelist[currfd].count = 0;

do\_read(currfd, openfilelist[currfd].length, buf);

*// 遍历当前目录 fcb*

fcb\* fcbptr = (fcb\*)buf;

*for* (i = 0; i < (int)(openfilelist[currfd].length / sizeof(fcb)); i++, fcbptr++) {

*if* (fcbptr->free == 1) {

*if* (fcbptr->attribute == 0) {

printf("<DIR> %-8s\t%d/%d/%d %d:%d\n",

fcbptr->filename,

(fcbptr->date >> 9) + 2000,

(fcbptr->date >> 5) & 0x000f,

(fcbptr->date) & 0x001f,

(fcbptr->time >> 11),

(fcbptr->time >> 5) & 0x003f);

} *else* {

printf("<---> %-8s\t%d/%d/%d %d:%d\t%d\n",

fcbptr->filename,

(fcbptr->date >> 9) + 2000,

(fcbptr->date >> 5) & 0x000f,

(fcbptr->date) & 0x001f,

(fcbptr->time >> 11),

(fcbptr->time >> 5) & 0x003f,

fcbptr->length);

}

}

}

}

新建文件命令：

a) 分配一个空闲的打开文件表项

b) 检查新文件的父目录是否打开

c) Read该父目录的文件到内存，并检测新建的文件名是否重名

d) 检查是否有空闲盘块

e) 寻找空闲的目录项

f) 准备好新文件的FCB

g) 调用close关闭打开的父目录文件

h) 返回

int my\_create(char\* filename)

{

*// 非法判断*

*if* (strcmp(filename, "") == 0) {

printf("please input filename\n");

*return* -1;

}

*if* (openfilelist[currfd].attribute == 1) {

printf("you are in data file now\n");

*return* -1;

}

openfilelist[currfd].count = 0;

char buf[MAX\_TEXT\_SIZE];

do\_read(currfd, openfilelist[currfd].length, buf);

int i;

fcb\* fcbptr = (fcb\*)buf;

*// 检查重名*

*for* (i = 0; i < (int)(openfilelist[currfd].length / sizeof(fcb)); i++, fcbptr++) {

*if* (fcbptr->free == 0) {

*continue*;

}

*if* (strcmp(fcbptr->filename, filename) == 0 && fcbptr->attribute == 1) {

printf("the same filename error\n");

*return* -1;

}

}

*// 申请空 fcb;*

fcbptr = (fcb\*)buf;

*for* (i = 0; i < (int)(openfilelist[currfd].length / sizeof(fcb)); i++, fcbptr++) {

*if* (fcbptr->free == 0)

*break*;

}

*// 申请磁盘块并更新 fat 表*

int block\_num = get\_free\_block();

*if* (block\_num == -1) {

*return* -1;

}

fat\* fat1 = (fat\*)(myvhard + BLOCKSIZE);

fat\* fat2 = (fat\*)(myvhard + BLOCKSIZE \* 3);

fat1[block\_num].id = END;

memcpy(fat2, fat1, BLOCKSIZE \* 2);

*// 修改 fcb 信息*

strcpy(fcbptr->filename, filename);

time\_t rawtime = time(NULL);

struct tm\* time = localtime(&rawtime);

fcbptr->date = (time->tm\_year - 100) \* 512 + (time->tm\_mon + 1) \* 32 + (time->tm\_mday);

fcbptr->time = (time->tm\_hour) \* 2048 + (time->tm\_min) \* 32 + (time->tm\_sec) / 2;

fcbptr->first = block\_num;

fcbptr->free = 1;

fcbptr->attribute = 1;

fcbptr->length = 0;

openfilelist[currfd].count = i \* sizeof(fcb);

do\_write(currfd, (char\*)fcbptr, sizeof(fcb), 2);

*// 修改父目录 fcb*

fcbptr = (fcb\*)buf;

fcbptr->length = openfilelist[currfd].length;

openfilelist[currfd].count = 0;

do\_write(currfd, (char\*)fcbptr, sizeof(fcb), 2);

openfilelist[currfd].fcbstate = 1;

}

删除文件命令

a) 检查要删除的文件的父目录是否已打开

b) Read父目录到内存

c) 检查文件是否打开

d) 回收磁盘快

e) 清空该文件的目录项

f) 修改用户打开文件表项中的长度信息

g) 返回

void my\_rm(char\* filename)

{

char buf[MAX\_TEXT\_SIZE];

openfilelist[currfd].count = 0;

do\_read(currfd, openfilelist[currfd].length, buf);

int i, flag = 0;

fcb\* fcbptr = (fcb\*)buf;

*// 查询*

*for* (i = 0; i < (int)(openfilelist[currfd].length / sizeof(fcb)); i++, fcbptr++) {

*if* (strcmp(fcbptr->filename, filename) == 0 && fcbptr->attribute == 1) {

flag = 1;

*break*;

}

}

*if* (flag != 1) {

printf("no such file\n");

*return*;

}

*// 更新 fat 表*

int block\_num = fcbptr->first;

fat\* fat1 = (fat\*)(myvhard + BLOCKSIZE);

int nxt\_num = 0;

*while* (1) {

nxt\_num = fat1[block\_num].id;

fat1[block\_num].id = FREE;

*if* (nxt\_num != END)

block\_num = nxt\_num;

*else*

*break*;

}

fat1 = (fat\*)(myvhard + BLOCKSIZE);

fat\* fat2 = (fat\*)(myvhard + BLOCKSIZE \* 3);

memcpy(fat2, fat1, BLOCKSIZE \* 2);

*// 清空 fcb*

fcbptr->date = 0;

fcbptr->time = 0;

fcbptr->exname[0] = '\0';

fcbptr->filename[0] = '\0';

fcbptr->first = 0;

fcbptr->free = 0;

fcbptr->length = 0;

openfilelist[currfd].count = i \* sizeof(fcb);

do\_write(currfd, (char\*)fcbptr, sizeof(fcb), 2);

*//*

int lognum = i;

*if* ((lognum + 1) \* sizeof(fcb) == openfilelist[currfd].length) {

openfilelist[currfd].length -= sizeof(fcb);

lognum--;

fcbptr = (fcb \*)buf + lognum;

*while* (fcbptr->free == 0) {

fcbptr--;

openfilelist[currfd].length -= sizeof(fcb);

}

}

*// 修改父目录 . 目录文件的 fcb*

fcbptr = (fcb\*)buf;

fcbptr->length = openfilelist[currfd].length;

openfilelist[currfd].count = 0;

do\_write(currfd, (char\*)fcbptr, sizeof(fcb), 2);

openfilelist[currfd].fcbstate = 1;

}

打开文件命令：

a) 检查该文件名是否存在

b) Read该父目录到内存

c) 检查用户打开的文件表中是否有空闲表项

d) 为该文件填写空白用户打开文件表项内容

e) 返回

int my\_open(char\* filename)

{

char buf[MAX\_TEXT\_SIZE];

openfilelist[currfd].count = 0;

do\_read(currfd, openfilelist[currfd].length, buf);

int i, flag = 0;

fcb\* fcbptr = (fcb\*)buf;

*// 重名检查*

*for* (i = 0; i < (int)(openfilelist[currfd].length / sizeof(fcb)); i++, fcbptr++) {

*if* (strcmp(fcbptr->filename, filename) == 0 && fcbptr->attribute == 1) {

flag = 1;

*break*;

}

}

*if* (flag != 1) {

printf("no such file\n");

*return* -1;

}

*// 申请新的打开目录项并初始化该目录项*

int fd = get\_free\_openfilelist();

*if* (fd == -1) {

printf("my\_open: full openfilelist\n");

*return* -1;

}

openfilelist[fd].attribute = 1;

openfilelist[fd].count = 0;

openfilelist[fd].date = fcbptr->date;

openfilelist[fd].time = fcbptr->time;

openfilelist[fd].length = fcbptr->length;

openfilelist[fd].first = fcbptr->first;

openfilelist[fd].free = 1;

strcpy(openfilelist[fd].filename, fcbptr->filename);

strcat(strcpy(openfilelist[fd].dir, (char\*)(openfilelist[currfd].dir)), filename);

openfilelist[fd].dirno = openfilelist[currfd].first;

openfilelist[fd].diroff = i;

openfilelist[fd].topenfile = 1;

openfilelist[fd].fcbstate = 0;

currfd = fd;

*return* 1;

}

关闭文件命令：

a) 检查fd的有效性

b) 检查用户打开文件表表项的fcbstate字段

c) 回收该文件占据的用户打开文件表表项

d) 返回

int my\_close(int fd)

{

*if* (fd > MAXOPENFILE || fd < 0) {

printf("my\_close: fd error\n");

*return* -1;

}

int i;

char buf[MAX\_TEXT\_SIZE];

int father\_fd = -1;

fcb\* fcbptr;

*for* (i = 0; i < MAXOPENFILE; i++) {

*if* (openfilelist[i].first == openfilelist[fd].dirno) {

father\_fd = i;

*break*;

}

}

*if* (father\_fd == -1) {

printf("my\_close: no father dir\n");

*return* -1;

}

*if* (openfilelist[fd].fcbstate == 1) {

do\_read(father\_fd, openfilelist[father\_fd].length, buf);

*// update fcb*

fcbptr = (fcb\*)(buf + sizeof(fcb) \* openfilelist[fd].diroff);

strcpy(fcbptr->exname, openfilelist[fd].exname);

strcpy(fcbptr->filename, openfilelist[fd].filename);

fcbptr->first = openfilelist[fd].first;

fcbptr->free = openfilelist[fd].free;

fcbptr->length = openfilelist[fd].length;

fcbptr->time = openfilelist[fd].time;

fcbptr->date = openfilelist[fd].date;

fcbptr->attribute = openfilelist[fd].attribute;

openfilelist[father\_fd].count = openfilelist[fd].diroff \* sizeof(fcb);

do\_write(father\_fd, (char\*)fcbptr, sizeof(fcb), 2);

}

*// 释放打开文件表*

memset(&openfilelist[fd], 0, sizeof(useropen));

currfd = father\_fd;

*return* father\_fd;

}

写文件命令：

a) 检查fd的有效性

b) 提示用户输入写方式

c) 提示用户输入内容

d) 调用do\_write（）将键入的内容写入到文件中

e) 返回写入的字节数

int my\_write(int fd)

{

*if* (fd < 0 || fd >= MAXOPENFILE) {

printf("my\_write: no such file\n");

*return* -1;

}

int wstyle;

*while* (1) {

*// 1: 截断写，清空全部内容，从头开始写*

*// 2. 覆盖写，从文件指针处开始写*

*// 3. 追加写，字面意思*

printf("1:Truncation 2:Coverage 3:Addition\n");

scanf("%d", &wstyle);

*if* (wstyle > 3 || wstyle < 1) {

printf("input error\n");

} *else* {

*break*;

}

}

char text[MAX\_TEXT\_SIZE] = "\0";

char texttmp[MAX\_TEXT\_SIZE] = "\0";

printf("please input data, line feed + $$ to end file\n");

getchar();

*while* (gets(texttmp)) {

*if* (strcmp(texttmp, "$$") == 0) {

*break*;

}

texttmp[strlen(texttmp)] = '\n';

strcat(text, texttmp);

}

text[strlen(text)] = '\0';

do\_write(fd, text, strlen(text) + 1, wstyle);

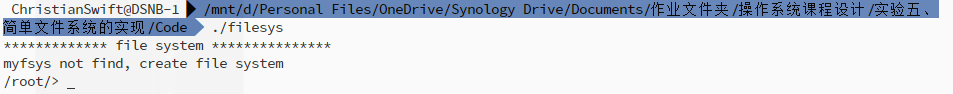
openfilelist[fd].fcbstate = 1;

*return* 1;

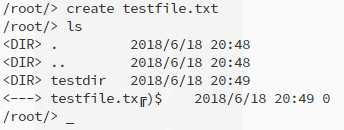
}

## 5.2 实验结果分析

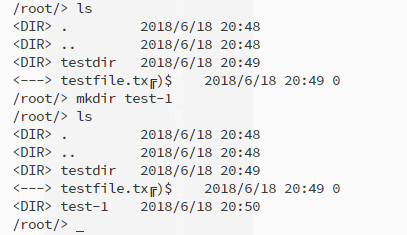
创建虚拟磁盘并格式化：



显示工作目录的所有文件和文件夹：



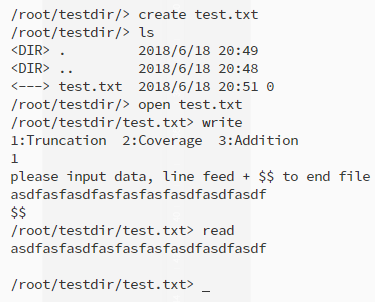
创建文件夹



进入文件夹并创建文件



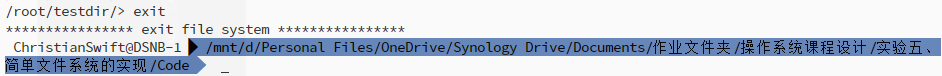
打开文件并写入文件并读取文件



关闭文件



退出虚拟文件系统



# 六 实验改进与总结

## 6.1 实验改进

**（大部分同学的代码都是参考网上资料，或者借鉴已完成同学的，此处重点突出自己如何改进，为什么这样改进，说出自己改进部分的优缺点）**

## 6.2 实验总结

**（此处类似于写观后感，同学们辛辛苦苦完成实验，单独编译内核可能就折腾很久，相信会有很多的感触和收获，让你的笔在第一时间记录这美好的瞬间）**

# 七 参考文献

[0] 赵伟华等.计算机操作系统[M].杭州电子科技大学计算机学院,2018,7(7.6)300-307

[1] mushroom-x. OS-Assignment-2016/3\_Simple\_File\_System at master · mushroom-x/OS-Assignment-2016[EB/OL]. https://github.com/mushroom-x/OS-Assignment-2016/tree/master/3\_Simple\_File\_System, 2016-05-26

[2] wanghaofeng. 操作系统课程设计--简单文件系统的实现 - CSDN博客[EB/OL]. https://blog.csdn.net/wanghaofeng/article/details/5720235, 2010-07-08

[3] DDUPzy. [操作系统][简单文件系统实现] - DDUPzy - 博客园[EB/OL]. http://www.cnblogs.com/zy691357966/p/5565699.html, 2016-06-07