### 实验五 文件系统

#### 实验目的

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

#### 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。在退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在磁盘上，以便下次可以再次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写保护。

(5) 要求提供以下操作命令：

my\_format：对文件存储器进行格式化。

my\_mkdir：用于创建子目录。

my\_rmdir：用于删除子目录。

my\_ls：用于显示目录中的内容。

my\_cd：用于更改当前目录。

my\_create：用于创建文件。

my\_open：用于打开文件。

my\_close：用于关闭文件。

my\_write：用于写文件。

my\_read：用于读文件。

my\_rm：用于删除文件。

exitsys：用于退出文件系统。

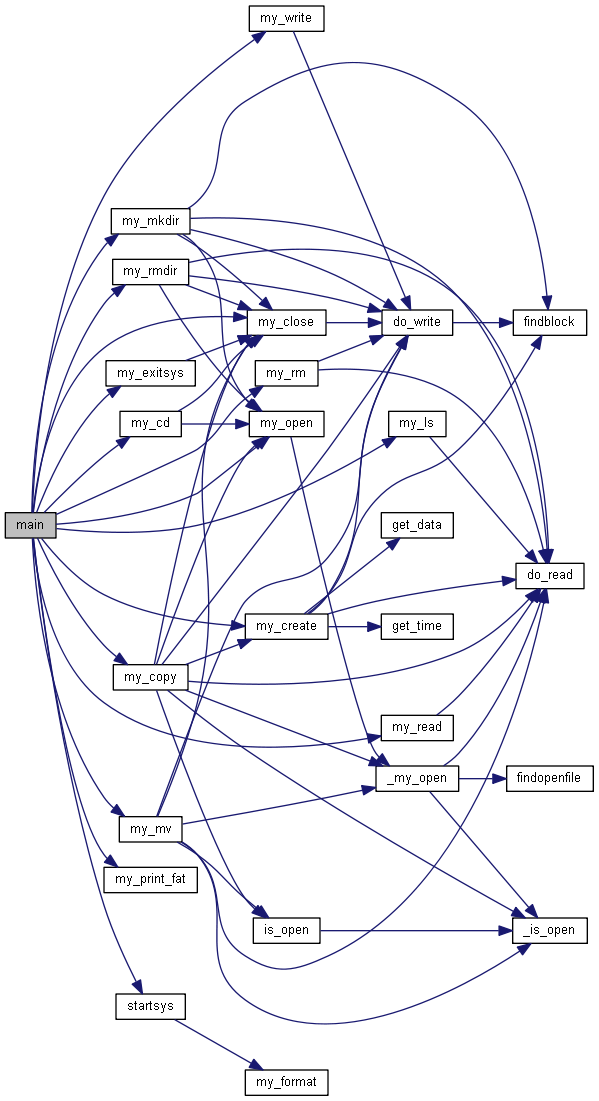
#### 实验设计

##### 3.1总体设计

程序的总体流程如下图所示：



程序的函数调用关系如下图所示：

****

##### 3.2数据结构设计

1. 定义的常量

#define BLOCKSIZE 1024 // 磁盘块大小

#define SIZE 1024000 // 虚拟磁盘空间大小

#define END 65535 // FAT中的文件结束标志

#define FREE 0 // FAT中盘块空闲标志

#define ROOTBLOCKNUM 2 // 根目录区所占盘块数

#define MAXOPENFILE 10 // 最多同时打开文件个数

#define MAXTEXT 102400 //文件的最大长度

我们仿照fat16进行文件系统的设计，,则可以将fat表的表项设置为unsign short类型，则该文件系统可以支持2^16个磁盘块，我们设置磁盘块大小为1KB，规定该虚拟磁盘空间大小为1000KB,则fat表的表项为1000，所以存储fat表需要两个磁盘块；我们规定给根目录分配两个磁盘块的空间大小。由于磁盘块最大为1000，可将unsigned short的最大值作为文件结束标志END。

1. 文件控制块FCB

用于记录文件的描述和控制信息，每个文件设置一个FCB，它也是文件的目录性内容；

1. 仿照fat16设置fcb,文件名由八个字节的filename和三个字节的exname组成；

目录文件的扩展名为””;

1. FCB中的time字段为16位的unsigned short类型，b11-b15表示小时,b5-b10 表示分钟,b0~b4表示秒,实际上second字段分配5位只能表示0-31，对实际的系统时间除以2得到该字段，实际上时间不是准确记录的；
2. FCB中的time字段为16位的unsigned short类型，b9-b15年份，b5-b8月份，

b0-b4 日期；

1. 设置文件最大的长度为102400Byte,所以fcb中文件长度的类型可以设置成四个字节大小的unsigned int;
2. 一个FCB结构体占用28Byte的空间，所以一个磁盘块中可以存储36个页表项；
3. 本程序中对文件属性字段进行了简化，将数据文件设置为0x0表示为读写文件，目录文件设置为0x10。

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，值为0x10时表示目录文件，值为0x00时表示数据文件

unsigned short time; // 文件创建时间

unsigned short date; // 文件创建日期

unsigned short first; // 文件起始盘块号

unsigned short int; // 文件长度

char free; // 表示目录项是否为空，若值为0，表示空，值为1，表示已分配

}fcb;

1. 文件分配表

FAT表用于记录虚拟磁盘上每个文件所占据的磁盘块的块号，并且虚拟磁盘上哪些块已经分配出去了，哪些是空闲的，系统中有两张fat表互为备份，每个fat表有1000项，每项有需要两个字节，所以需要占据两个磁盘块。

typedef struct FAT

{

unsigned short id; // 磁盘块的状态（空闲的，最后的，下一个）

}fat;

1. 用户打开文件表USEROPEN

当打开一个文件时，必须将文件的目录项中所有内容全部复制到内存中，同时还要记录有关文件使用的动态信息，入读写指针的值、FCB的修改状态等。我们参考了书本上的例子，在此基础上加了dirno,diroff表示该该文件的父目录的第一块磁盘块号和该FCB位于目录文件的目录项序号，在修改FCB时，可以快速定位到FCB的地址。

typedef struct USEROPEN

{

char filename[8]; // 文件名

char exname[3]; // 文件扩展名

unsigned char attribute;//文件属性字段，值为0时表示目录文件，值为1时表示数据文件

unsigned short time; // 文件创建时间

unsigned short date; // 文件创建日期

unsigned short first; // 文件起始盘块号

unsigned int length;//文件长度（对数据文件是字节数，对目录文件可以是目录项个数）

char free; // 表示目录项是否为空，若值为0，表示空，值为1，表示已分配

unsigned char dirno; // 父目录文件中的第一块盘块号

int diroff; // 相应打开文件的目录项在父目录文件的dirno盘块中的目录项序号

char dir[80]; // 相应打开文件所在的路径名，这样方便快速检查出指定文件是否已经打开

int father; // 父目录在打开文件表项的位置

int count; // 读写指针在文件中的位置,文件的总字符数

char fcbstate; // 是否修改了文件的FCB的内容，如果修改了置为1，否则为0

char topenfile; // 表示该用户打开表项是否为空，若值为0，表示为空，否则表示已被某打开文件占据

}useropen;

在USEROPEN结构体中记录

1. 引导块BLOCK0

在引导块中存放虚拟磁盘的相关描述信息，磁盘块的大小，磁盘块的数量，最大打开文件数，磁盘上数据区开始位置在该文件系统在，引导块占用一个磁盘块，两张fat表各占用两个磁盘块，所以数据开始磁盘块为5；，其中magic魔数是文件系统起始的几个字节。

typedef struct BLOCK0

{

char magic[10]; // 文件系统魔数

char information[200];//存储一些描述信息，如磁盘块大小、磁盘块数量、最多打开文件数等

unsigned short root; // 根目录文件的起始盘块号

unsigned char \*startblock; // 虚拟磁盘上数据区开始位置

}block0;

1. 全局变量

unsigned char \*myvhard; // 指向虚拟磁盘的起始地址

useropen openfilelist[MAXOPENFILE]; // 用户打开文件表数组

int curdir; // 用户打开文件表中的当前目录所在打开文件表项的位置，openfilelist的下标

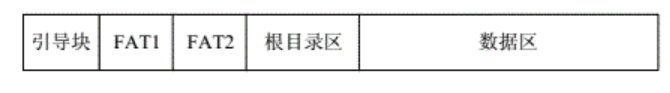
char currentdir[80]; // 记录当前目录的目录名（包括目录的路径）

unsigned char\* startp; // 记录虚拟磁盘上数据区开始位置

char myfilename[] = "myfilesys";//文件系统的文件名

1. 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，将把数据区前两块（即虚拟磁盘的第6块和第7块）分配给根目录文件，如图所示：



##### 3.3主要函数设计

1、进入文件系统函数startsys()

1）函数说明：

原型：void startsys()

功能：由main()函数调用，进入并初始化我们所建立的文件系统，以供用户使用。

2）函数设计思路：

1.申请虚拟磁盘空间；

2. 检查是否存在文件系统文件myfilesys，若存在，读取文件内容恢复文件系统；若不存在创建之；

3. 调用format函数对虚拟磁盘进行格式化；

4. 初始化用户打开文件表；

5. 将当前目录设置为根目录。

2、磁盘格式化函数my\_format()

1）函数说明：

原型：void my\_format()

功能：对虚拟磁盘进行格式化，布局虚拟磁盘，建立根目录文件（或根目录区）。

2）函数设计思路：

将虚拟磁盘第一个块作为引导块，写入文件系统魔数以及文件系统的描述信息。在引导块后建立两张FAT表，同时对FAT表进行初始化，建立根目录文件root，同时创建根目录下的“.”和“..”目录项。

3、更改当前目录函数my\_cd()

1）函数说明：

原型：void my\_cd(char \*dirname)

功能：改变当前目录到指定的名为dirname的目录。

2）函数设计思路：

1. 调用my\_open()打开指定目录名的父目录文件，并调用do\_read()读入该父目录文件内容到内存中；

2. 在父目录文件中检查新的当前目录名是否存在，如果存在则转3，否则返回，并显示出错信息；

3. 调用my\_close()关闭1中打开的父目录文件；

4. 调用my\_close()关闭原当前目录文件；

5. 如果新的当前目录文件没有打开，则打开该目录文件；并将ptrcurdir指向该打开文件表项；

6. 设置当前目录为该目录。

4、创建子目录函数my\_mkdir()

1）函数说明：

原型：void my\_mkdir(char \*dirname)

功能：在当前目录下创建名为dirname的子目录。

2）函数设计思路：

1. 调用do\_read()读入当前目录文件内容到内存，检查当前目录下新建目录文件是否重名，若重名则返回，并显示错误信息；

2. 为新建子目录文件分配一个空闲打开文件表项，如果没有空闲表项则返回-1，并显示错误信息；

3. 检查FAT是否有空闲的盘块，如有则为新建目录文件分配一个盘块，否则释放1中分配的打开文件表项，返回，并显示错误信息；

4. 在当前目录中为新建目录文件寻找一个空闲的目录项或为其追加一个新的目录项;需修改当前目录文件的长度信息，并将当前目录文件的用户打开文件表项中的fcbstate置为1；

5. 准备好新建目录文件的FCB的内容，文件的属性为目录文件，以覆盖写方式调用do\_write()将其填写到对应的空目录项中；

6. 在新建目录文件所分配到的磁盘块中建立两个特殊的目录项“.”和“..”目录项，方法是：首先在用户空间中准备好内容，然后以截断写或者覆盖写方式调用do\_write()将其写到3中分配到的磁盘块中；

5、删除子目录函数rmdir()

1）函数说明：

原型：void my\_rmdir(char \*dirname)

功能：在当前目录下删除名为dirname的子目录。

2）函数设计思路：

1. 读入当前目录文件到内存；

2. 检查当前目录下是否存在欲删除的目录文件，若不存在则报错返回；3.若存在，则检查该目录文件是否为空（即目录文件中仅含“.”和“..”

目录项）；

4. 检查该目录文件是否已经打开，若已打开则首先将其关闭；

5. 回收该目录文件所占据的磁盘块，修改FAT；

6. 从当前目录文件中清空该目录文件的目录项，且free字段置为0：

7. 修改当前目录文件的用户打开表项中的长度信息，并将表项中的

fcbstate置为1。

6、显示目录函数my\_ls()

1）函数说明：

原型：void my\_ls(void)

功能：显示当前目录的内容（子目录和文件信息）。

2）函数设计思路：

1. 读取当前目录文件内容到内存；

2. 将读出的目录文件的信息按照一定的格式显示到屏幕上。

7、创建文件函数my\_create()

1）函数说明：

原型：void my\_create(char \*filenm)

功能：在当前目录下创建文件名为filenm的文件

2）函数设计思路：

1. 利用strtok函数分解文件名，得到文件名和文件扩展名；

2. 根据curdir全局函数，读取当前目录文件中的内容，根据文件名找出是否存在同名文件，若存在则输出错误提示，若不存在，则创建；

3. 利用findblock找到虚拟磁盘中空闲的一个磁盘块，读取当前目录的内容，找到文件中fcb->free=0的fcb,将该空闲磁盘块给该fcb->first,表示该磁盘分配给fcb,填充fcb的其他信息，将fcb写入到目录文件中，修改当前文件的USEOPENFILE信息。

8、删除文件函数my\_rm()

1）函数说明：

原型：void my\_rm(char \*filename)

功能：在当前目录下删除文件名为filename的文件

1. 函数设计思路：

1. 利用strtok函数分解文件名，得到文件名和文件扩展名，若文件名不合法，则输出错误，若合法则删除；

2．根据currdir全局函数，读取当前目录文件中的内容，根据文件名找出是否存在同名文件，若不存在则输出错误提示，若存在，则找到该文件对应的fcb；

3. 根据fcb找到该文件的第一块磁盘块号，根据fat->id找到记录下一块磁盘块号，在fat表中将其置为0，表示该磁盘块已经被释放，再释放下一块磁盘块，直到fat->id为END,表示为最后一块磁盘号；

4. 将fcb中的free置为0,表示该fcb已经被释放，将该fcb写入目录文件中，修改

5. 修改当前文件的USEOPENFILE信息。

9、打开文件函数my\_open()

1）函数说明：

原型：int \_my\_open(char \*filenm,int curdir)

功能：在指定目录curdir下，打开一个文件名为filename的文件或文件夹，若打开错误返回-1，成功打开则返回该文件的fd;

2）函数设计思路：

1. 利用strtok函数分解文件名，得到文件名和扩展名，判断该文件名是否合法；

2. 利用\_is\_open函数判断是否已经打开该文件，若打开，则输出错误信息；

3. 根据fd，读取指定目录文件中的内容，根据文件名找出是否存在同名文件，若不存在则输出错误提示，若存在，则找到该文件对应的fcb；

4. 根据findopenfile找到一个空闲的openfilelist返回该打开文件的下标fd,复制fcb的信息到USEOPENFILE中，且将根据当前打开文件的USEOPENFILE的信息填充到该打开文件的中，将该打开文件的topenfile字段设置为1表示该打开文件已经被占用；

10、关闭文件函数my\_close()

1）函数说明：

原型：void my\_close（int fd）；

功能：关闭前面打开的文件描述符为fd的文件

2）函数设计思想：

检查fd是否超出用户打开文件表所在数组的最大下标，如果超出则返回-1；检查用户打开文件表表项中的fcbstate字段的值，如果为1则需要将该文件的FCB的内容保存到虚拟磁盘上该文件的目录项中，方法是：打开该文件的父目录文件，以覆盖写方式调用do\_write()将欲关闭文件的FCB写入父目录文件的相应盘块中；回收该文件占据的用户打开文件表表项（进行清空操作），并将topenfile字段置为0；

11、拷贝文件my\_copy()

1）函数说明：

原型：int my\_copy(char \*filename,char \*path)

功能：在当前目录下，拷贝一个文件名为filename的文件，到路径path中去，其中path可以为绝对路径，也可以是相对路径；若拷贝成功，返回1，若不成功返回-1；

2）函数设计思路：

1. 利用strtok函数分解文件名，得到文件名和扩展名，利用\_is\_open函

数判断该文件名是否被打开，若未打开则打开当前文件；

2. 利用strtok\_r函数分解路径名，判断目录文件是否打开，若未打开则

打开目录文件，按照路径打开目标路径目录文件，利用need\_close数组记录在操作过程中被打开的文件，在拷贝完成后，应关闭原本是关闭状态的文件，若打开文件fd最后需要关闭，则该数组下标fd为1；若该路径不存在，则关闭打开的文件，返回错误信息；

3. 读出该源文件的数据，存到临时变量中，关闭源文件；

4. 在目标路径下创建同名文件，打开该文件，写入源文件的数据；

5. 关闭新文件，从该新文件往上查找是否有需要关闭的目录文件，自底

向下关闭文件；

12、移动文件my\_mv()

1）函数说明：

原型：int my\_mv(char \*filename,char \*path)

功能：在当前目录下，移动一个文件名为filename的文件，到路径path中去，其中path可以为绝对路径，也可以是相对路径；若移动成功，返回1，若不成功返回-1；

1. 函数设计思路：

1. 利用strtok函数分解文件名，得到文件名和扩展名，利用\_is\_open函

数判断该文件名是否被打开，若打开则关闭当前文件，读取当前目录文件，判断源文件是否存在，若存在记录该文件的fcb,若不存在则返回错误；

2. 利用strtok\_r函数分解路径名，判断目录文件是否打开，若未打开则

打开目录文件，按照路径打开目标路径目录文件，利用need\_close数组记录在操作过程中被打开的文件，在拷贝完成后，应关闭原本是关闭状态的文件，若打开文件fd最后需要关闭，则该need\_close[fd]=1；若该路径不存在，则关闭打开的文件，返回错误；

3. 读取目标路径的文件内容，判断该路径下是否存在同名文件。若存在

则创建不成功，若不存在，创建;

4. 在目标文件目录下找到一块空闲fcb,将源文件的fcb所有信息复制到

目标目录文件目录中，修改openfilelsit的信息，并修改目标目录文件的第一个FCB的信息；

5. 在当前目录下删除源文件目录项，读取当前文件的内容，找到源文件

的FCB,修改FCB写入当前目录文件中，修改openfilelsit的信息，并修改当前目录文件的第一个FCB的信息；

6. 从目标路径下开始关闭需要关闭的文件；

13、写文件函数my\_write()

1）函数说明：

原型：int my\_write（int fd）；

功能：将用户通过键盘输入的内容写到fd所指定的文件中。截断写、覆盖写和追加写。截断写是放弃原来文件的内容，重新写文件；覆盖写是修改文件在当前读写指针所指的位置开始的部分内容；追加写是在原文件的最后添加新的内容。在本实例中，输入写文件命令后，系统会出现提示让用户选择其中的一种写方式，并将随后键盘输入的内容按照所选的方式写到文件中，键盘输入内容通过CTRL+D键（或其他设定的键）结束。

2）函数设计思路：

1. 三种写的方式

截断写：释放文件除第一块外的其他磁盘空间内容（查找并修改FAT表），将用户打开文件表项中文件长度修改为0，将读写指针置为0。

追加写：修改文件的当前读写指针位置到文件的末尾。

覆盖写：读写指针指向文件开头位置。

2. 以CTRL+D结束后，调用do\_write()函数将通过键盘输入的内容写到文件中。

3. 如果do\_write()函数的返回值为非负值，则将实际写入字节数增加do\_write()函数返回值，否则显示出错信息。

4. 如果当前读写指针位置大于用户打开文件表项中的文件长度，则修改打开文件表项中的文件长度信息，并将fcbstate置1.

5. 返回写入的字数。

14、实际写文件函数do\_write()

1）函数说明：

原型：int my\_write(int fd，char \*text，int len，char wstyle)；

功能：被写文件函数my\_write()调用，用来将键盘输入的内容写到相应的文件中去。

2）函数设计思路：

1. 用malloc()申请1024B的内存空间作为读写磁盘的缓冲区buf，申请失败则返回-1，并显示出错信息。

2. 将读写指针转化为逻辑块块号和块内偏移off，并利用打开文件表表项中的首块号及FAT表的相关内容将逻辑块块号转换成对应的磁盘块块号blkno；如果找不到对应的磁盘块，则需要检索FAT为该逻辑块分配一新的磁盘块，并将对应的磁盘块块号blkno登记到FAT中，若分配失败，则返回-1，并显示出错信息；

3. 如果是覆盖写，或者如果当前读写指针所对应的块内偏移off不等于0，则将块号为blkno的虚拟磁盘块全部1024B的内容读到缓冲区buf中；否则便用ASCII码0清空buf；

4. 将text中未写入的内容暂存到缓冲区buff的第off字节开始的位置，直到缓冲区满，或者接收到结束字符CTRL+D为止；将本次写入字节数记录到tmplen中；

5. 将buf中1024B的内容写入到块号为blkno的虚拟磁盘块中；

6. 将当前读写指针修改为原来的值加上tmplen；并将本次实际写入的字节数增加tmplen；

7. 返回本次实际写入的字节数。

15、读文件函数my\_read()

1）函数说明：

原型：int my\_read(int fd)；

功能：读出指定文件中从读写指针开始的长度为len的内容到用户空间

中

2）函数设计思路：

定义一个字符型数组text[len]，用来接收用户从文件中读出的文件内容：检查fd的有效性（fd不能超出用户打开文件表所在数组的最大下标），如果无效则返回-1，并显示出错信息；调用do\_read()将指定文件中的len字节内容读出到text[]中；如果do\_read()的返回值为负，则显示出错信息；否则将text[]中的内容显示到屏幕上；返回。

16、实际读文件函数do\_read()

1）函数说明：

原型：int do\_read (int fd, int len,char \*text)

功能：被my\_read()调用，读出指定文件中从读写指针开始的长度为len的内容到用户空间的text中

1. 函数设计思路：
2. 使用malloc()申请1024B空间作为缓冲区buf，申请失败则返回-1，并

显示出错信息；

1. 将读写指针转化为逻辑块块号及块内偏移量off，利用打开文件表表

项中的首块号查找FAT表，找到该逻辑块所在的磁盘块块号；将该磁盘块块号转化为虚拟磁盘上的内存位置；

1. 将该内存位置开始的1024B（一个磁盘块）内容读入buf中；
2. 比较buf中从偏移量off开始的剩余字节数是否大于等于应读写的字

节数len，如果是，则将从off开始的buf中的len长度的内容读入到text[]中；否则，将从off开始的buf中的剩余内容读入到text[]中；

1. 使用free()释放1中申请的buf。
2. 返回实际读出的字节数。

17、退出文件系统函数my\_exitsys()

1）函数说明：

原型：void my\_exitsys()

功能：退出文件系统

2）函数设计思路：

首先将当前打开的文件关闭，将虚拟磁盘空间中的所有内容保存到磁盘上的myfilesys文件中，撤销用户打开文件表，释放其内存空间，释放虚拟磁盘空间。

18、打印当前fat中的内容my\_print\_fat()

1）函数说明：

原型：void my\_print\_fat()

功能：打印当前fat中的内容

2）函数设计思路：

将指针定位至fat1所在地址，循环输出fat1->id。

19、判断文件打开函数is\_open()

1）函数说明：

原型：int is\_open(char \*fnm,char \*exnm,int fd)

功能：判断一个目录下文件名为fnm的文件是否打开，若打开，返回该文件的fd,若未打开则返回-1；

2）函数设计思路：

在全局变量openfilelist数组中遍历是否含有同名文件，且该文件的father为指定目录fd。

20. 输出错误函数PRINT\_ERROR()

1）函数说明：

原型：void PRINT\_ERROR(int code\_error)

功能：当程序报错时输出错误原因。

2）函数设计思路：

定义六种错误类型，以错误代码调用该函数，输出相应的错误提示信息。

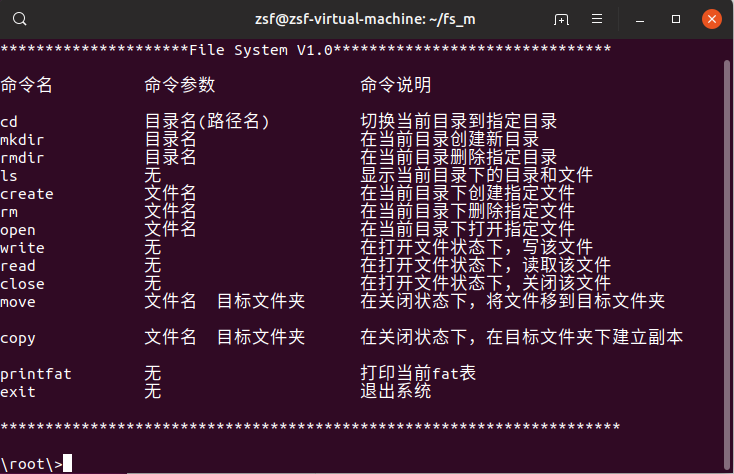
##### 3.4实验创新性

1. 功能扩展：除实现实验要求的命令外，程序还实现了打印fat表printfat，文件移动命令move以及文件复制命令copy。
2. 错误提示：将系统产生的错误进行分类，设置一个错误提示函数来打印错误信息，为每种错误定义一个宏，当产生错误需要打印信息时，调用错误提示函数。

#### 实验结果

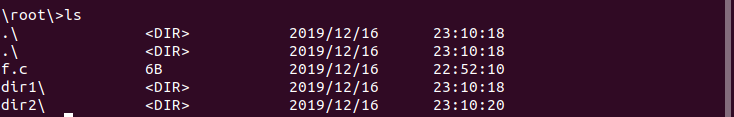
##### 4.1实验测试

###### 登录界面：



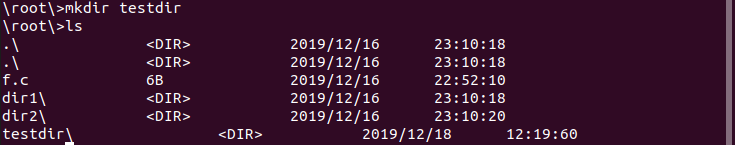
###### ls命令

使用ls命令显示当前目录下的所有目录和文件：



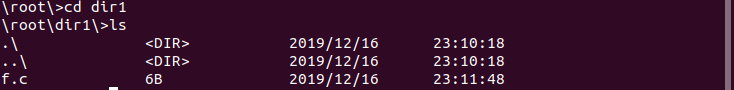
###### mkdir命令

使用mkdir命令在root目录下创建新的子目录testdir，然后使用ls命令再次显示跟目录下的所有子目录，发现相较之前多了一个testdir目录，说明mkdir命令创建子目录成功：

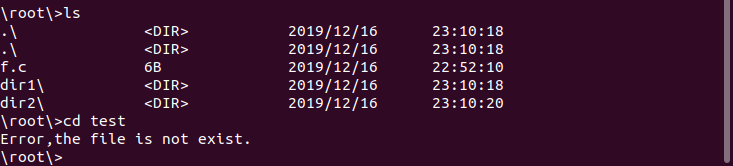


###### cd命令

使用cd 命令进入根目录下的子目录dir1，并使用ls命令进行查看，发现有一个文件f.c

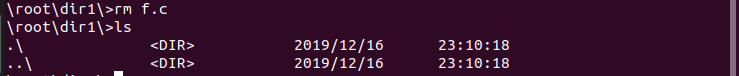


当输入的目录不存在时，报错：



###### rm命令

使用rm命令删除指定目录下的文件，删除后再使用ls命令查看，指定文件消失，说明删除操作成功。

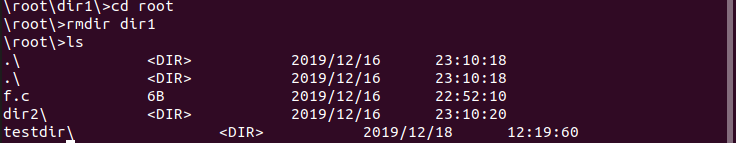


###### rmdir命令

使用rmdir命令尝试删除dir1目录，系统经检查发现dir1目录不为空，无法删除。

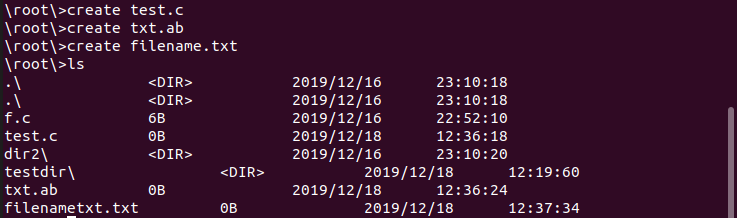


删除dir1中的文件之后，再次进行删除目录的操作，使用ls命令查看，发现删除成功。



###### create命令

先分别创建三个后缀名长度分别为1,2,3的文件，再使用ls命令查看，发现创建成功。



###### open和close



###### read操作

read前必须先将要读取的文件打开，否则无法读取

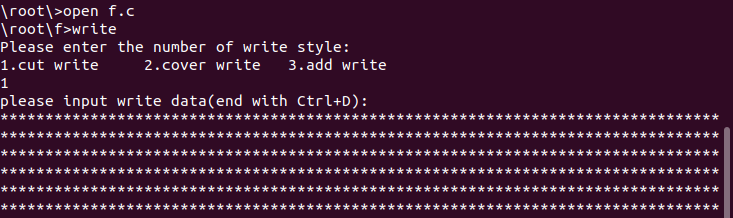


使用open操作将文件打开以后，在进行读取即可成功

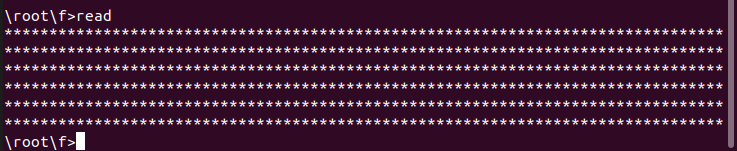


###### write操作

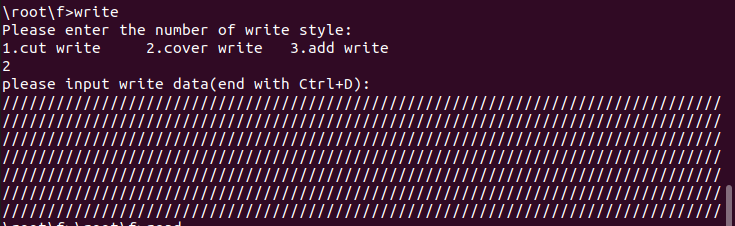
截断写：



使用read命令检验：



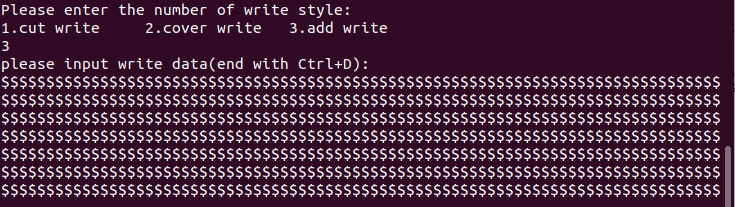
覆盖写：



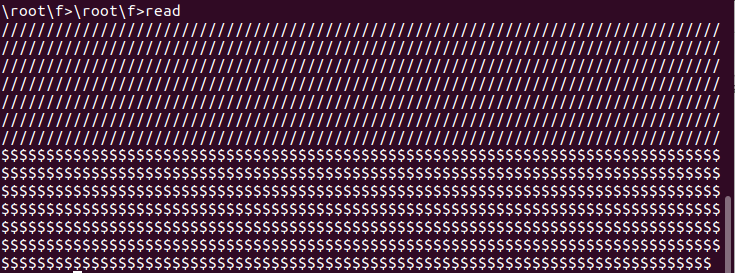
结果检验：



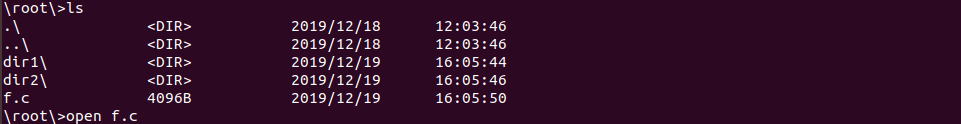
追加写：

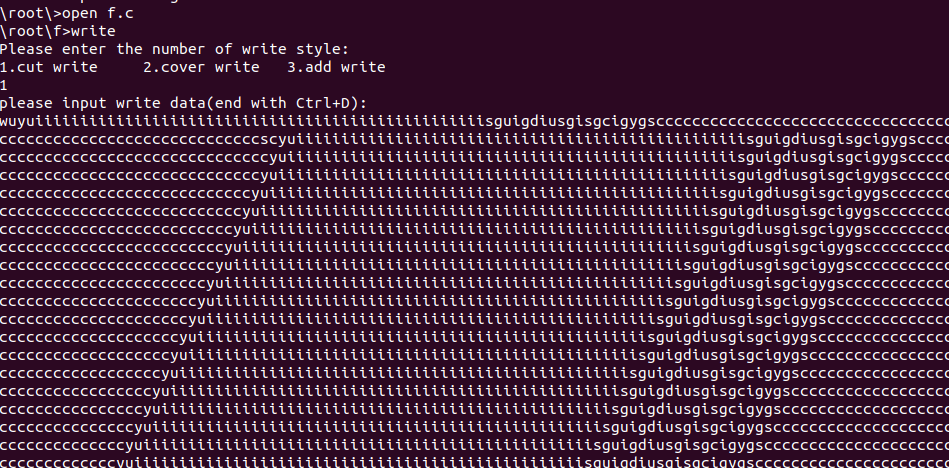


结果检验：

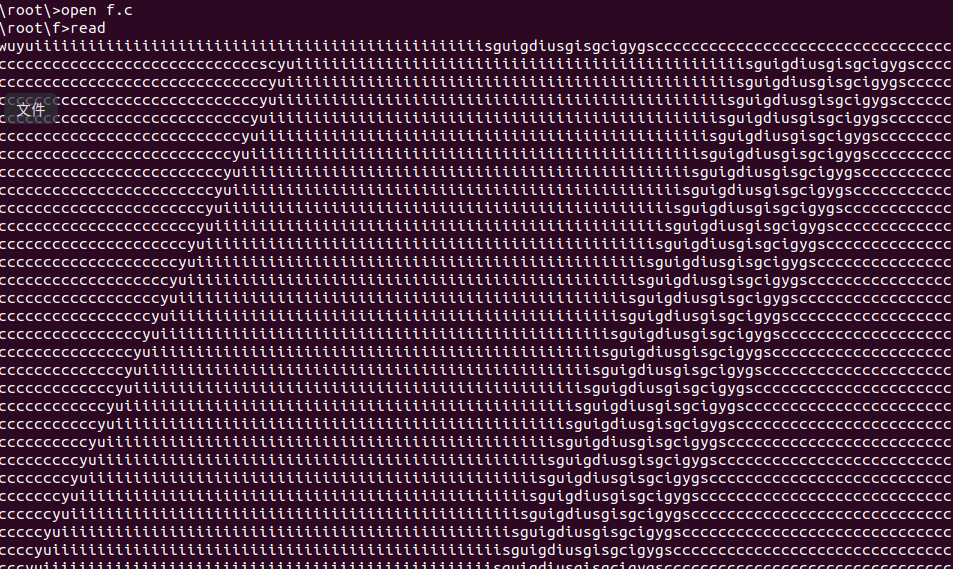


当写入大量数据时（这里写入4096B）：



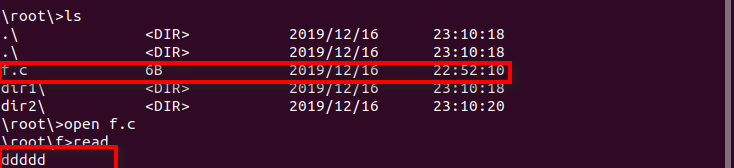


read操作：



###### move操作

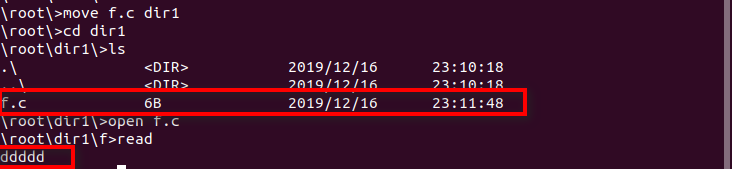
根目录下的文件情况如图，f.c的创建时间以及内容如下图所示：



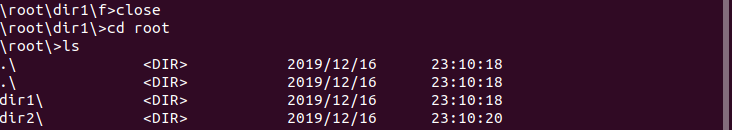
目标目录下的文件情况如下图所示：



使用Move命令将其移动到dir1目录下，进入dir1发现f.c成功移入，且创建时间以及其包含的内容未发生改变：



退回根目录，检查其此时此刻的文件情况，发现根目录下的f.c文件不见了，证明操作成功：

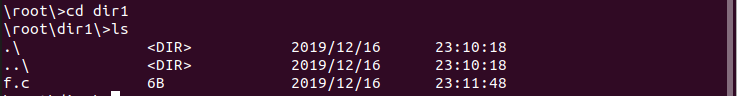


当移动的文件名称与目标目录下某文件重名时，系统提示重名不能进行移动：

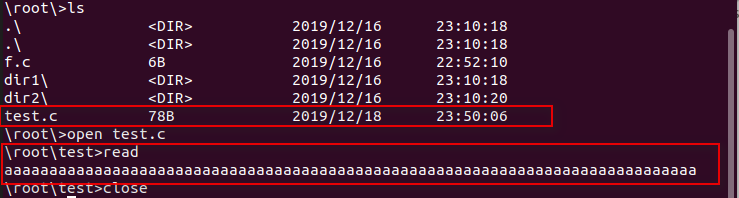


###### copy操作

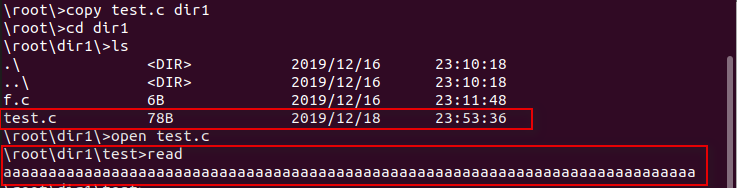
首先显示dir1下的文件：



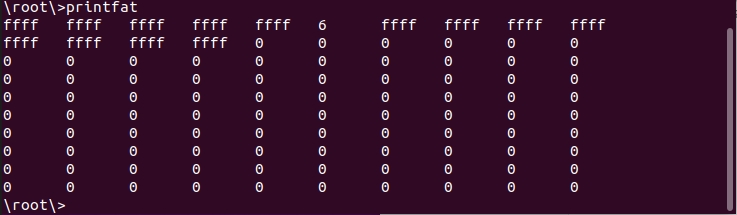
然后回到根目录，以dir1作为目标目录，将根目录下的test.c文件复制到dir1目录下:



复制后，可以看到此时dir1下已经有了test.c文件，且经过观察，dir1目录下的test.c文件的创建时间与根目录下的创建时间不同，为复制时的时间，且使用open命令查看，test.c文件中的内容依旧保持不变，说明操作成功：



###### Printfat



###### 退出



##### 4.2遇到的问题

1. 问题：对文件名进行分隔处理时产生错误；

原因：使用strtok函数进行字符串分解时，该函数对分隔结果不会新的字符串，而是在使用时会对源字符串进行处理，导致在分隔后，原指针不会指向原字符串导致错误；

解决办法：在使用该字符串前，复制字符串，使用源字符串的副本放入该函数中。

1. 问题：在开始写代码时，由于只是实现了书本中的基础功能，所以写的函数都是在当前目录下的操作，比如my\_open()函数中，是打开当前文件夹下的一个文件；

解决办法：在编写代码后，加上了移动文件，拷贝文件的功能，此时可能需要在根目录下打开另外一条路径，所以需要一个功能为在指定目录下打开文件函数，为了减少代码改动，将my\_open(char \*filename)改为\_my\_open(char \*filename,int curdir),可以将原来的全局变量curdir变为局部变量，实现该功能，然后my\_open(char \*filename)调用\_is\_open()函数，虽然在命名上会产生误解，但减少了代码的改动。

1. 问题：在调试my\_rm函数的功能时，若输入错误的目标路径，虽然系统会提示错误，且源文件也不会丢失，但是尝试进入该目标路径的正确部分时会提示该文件已经打开。

原因：由于产生路径找不到时，系统已经打开该路径正确的目录文件，这是发现路径错误就直接返回，则导致原来没有关闭的文件打开。

解决办法：发现该错误后，参考该函数在执行成功后的清理操作，将打开的文件关闭。

1. 问题：在使用调用my\_open函数是，会打开一个错误但filename相同的文件;

原因：该函数调用了\_is\_open函数，该函数在指定目录下，判断某文件是否打开。在比较文件是否打开时，遍历openfilelist中的每一个，比较一个被占用的的文件名与其文件名是否同名。若两个打开文件同名，但是在不同的目录项下，则该函数就会出错；

解决办法：在执行\_is\_open函数时，比较打开文件的文件名之外还比较了该打开文件父目录的fd,这样就能正确实现该功能；