* 1. **简单文件系统的实现**

**3.1 设计目的和内容要求**

**1. 设计目的**

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实现，加深对文件系统内部数据结构、功能以及实现过程的理解。

**2．内容要求**

* 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，应将该虚拟文件系统以一个Windows 文件的方式保存到磁盘上，以便下次可以再将它恢复到内存的虚拟磁盘空间中。
  2. 文件存储空间的分配可采用显式链接分配或其他的办法。
  3. 空闲磁盘空间的管理可选择位示图或其他的办法。如果采用位示图来管理文件存储空间，并采用显式链接分配方式，那么可以将位示图合并到FAT中。
  4. 文件目录结构采用多级目录结构。为了简单起见，可以不使用索引结点，其中的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. 要求提供以下操作命令：
     + my\_format：对文件存储器进行格式化，即按照文件系统的结构对虚拟磁盘空间进行布局，并在其上创建根目录以及用于管理文件存储空间等的数据结构。
     + 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。
     + my\_exitsys：用于退出文件系统。

**3．学时安排**

授课2学时，上机9学时。

**4． 开发平台**

C或C++均可。

**5．思考**

* + - 1. 我们的数据结构中的文件物理地址信息是使用C语言的指针类型、还是整型，为什么？
      2. 如果引入磁盘索引结点，上述实现过程需要作哪些修改？
      3. 如果设计的是一个单用户多任务文件系统，则系统需要进行哪些扩充（尤其要考虑读写指针问题）？如果设计的是一个多用户文件系统，则又要进行哪些扩充？

**3.2 预备知识**

3.2.1 FAT文件系统介绍

**1．概述**

FAT文件系统是微软公司在其早期的操作系统MS-DOS及Windows9x中采用的文件系统，它被设计用来管理小容量的磁盘空间。FAT文件系统是以他的文件组织方式——文件分配表（file allocation table，FAT）命名的，文件分配表的每个表项中存放某文件的下一个盘块号，而该文件的起始盘块号则保存在它的文件控制块FCB中。在文件分配表中，一般用FFFF来标识文件的结束；用0000来标识某个逻辑块未被分配，即是空闲块。为了提高文件系统的可靠性，在逻辑磁盘上通常设置两张文件分配表，它们互为备份。此外，文件分配表必须存放在逻辑磁盘上的固定位置，而根目录区通常位于FAT2之后，以便操作系统在启动时能够定位所需的文件，其磁盘布局如图3-1所示：



图3-1 FAT文件系统磁盘布局

上述磁盘布局中，引导块中主要存放了用于描述分区的各种信息，包括逻辑块的大小、文件分配表的大小及位置、根目录的大小及位置等。除此之外，用于加载操作系统内核的引导程序也存储在引导块中。

FAT文件系统家族又分为FAT12、FAT16、FAT32三种类型，这里的数字表示文件分配表中每个表项（即簇号）所占的位数，即FAT12中每个表项占1.5个字节（12位），FAT16中每个表项占2个字节（16位），FAT32中每个表项占4个字节（32位）。由于FAT文件系统是以簇为单位为文件分配磁盘空间的（一个簇是一组连续的扇区，通常包含2n个扇区），因此，FAT32比FAT12和FAT16支持更多的簇数、更小的簇大小和更大的磁盘容量，从而大大提高磁盘空间的利用率。通常，FAT12适用于小容量磁盘，如软盘；FAT16是MS-DOS的文件系统；FAT32是Windows9x中的主要文件系统，开始支持大容量磁盘。

**2．文件控制块FCB**

为了正确、方便地操作文件，必须设置相应的数据结构用于存放文件的描述和控制信息，常用的数据结构有文件控制块（简称FCB）和索引节点（简称i节点）。在FAT文件系统中使用文件控制块。文件与文件控制块一一对应，而文件控制块的有序集合就称为文件目录，即一个文件控制块就是一个文件目录项。

虽然不同文件系统的文件控制块的内容和格式不完全相同，但通常都包括以下三类信息：**基本信息、存取控制信息和使用信息**。

（1）基本信息。包括文件名、用户名、文件类型、文件的物理地址、文件长度、文件的逻辑结构和物理结构等。

（2）存取控制信息。一般分别给出文件主、伙伴用户、一般用户的存取权限。

（3）使用信息。包括文件的建立日期及时间、上次存取文件的日期及时间、当前的使用信息等。

以MS-DOS（使用FAT16文件系统）为例，它的每个文件控制块包括32个字节，其字节分配情况如图3-2所示：



图3-2 MS-DOS的文件控制块

其中属性字段占一个字节，它的每一位用来表示该文件是否具有某种属性，如果某一位的值为1，则表示该文件具有该属性。各位所表示的属性如表3-1所示：

表3-1 文件属性对照表

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 属性 | 保留 | 保留 | 存档 | 子目录 | 卷标 | 系统文件 | 隐藏 | 只读 |
|  |  |  |  |  |  |  |  |  |

**3．根目录区**

FAT12、FAT16的根目录区是固定区域、固定大小的，位于第二个FAT之后，如图3-1所示，且占据若干连续扇区，其中FAT12占14个扇区，一共224个根目录项；而FAT16占32个扇区，最多保存512个目录项，作为系统区的一部分。FAT32的根目录是作为文件处理的，采用与子目录文件相同的管理方式，其位置不是固定的，不过一般情况也是位于第二个FAT之后的，其大小可视需要增加，因此根目录下的文件数目不再受最多512个的限制。

3.2.2 几个C语言库函数介绍

由于我们的文件系统是建立在内存的虚拟磁盘上的，在退出文件系统的时候必须以一个文件的形式保存到磁盘上；而在启动文件系统的时候必须从磁盘上将该文件读入到内存的虚拟磁盘中。下面介绍几个可能会用到的C库函数，在使用这些库函数之前必须包含头文件“stdio.h”。

**1．打开文件函数fopen()**

（1）格式：FILE \*fopen(const char \*filename,const char \*mode)

（2）功能：按照指定打开方式打开指定文件。

（3）输入参数说明：

filename：待打开的文件名，如果不存在就创建该文件。

mode： 文件打开方式，常用的有：

* + - "r"：为读而打开文本文件（不存在则出错）。
    - "w"：为写而打开文本文件（若不存在则创建该文件；反之，则从文件起始位置写,原内容将被覆盖）。
    - "a"：为在文件末尾添加数据而打开文本文件。（若不存在则创建该文件；反之,在原文件末尾追加）。
    - "r+"：为读和写而打开文本文件。（读时，从头开始；在写数据时,新数据只覆盖所占的空间，其后不变) 。
    - "w+"：首先建立一个新文件，进行写操作，随后可以从头开始读。(若文件存在，原内容将全部消失) 。
    - "a+"：功能与"a"相同；只是在文件末尾添加新的数据后，可以从头开始读。

另外，上述模式字符串中都可以加一个“b”字符，如rb、wb、ab、rb+、wb+、ab+等组合，字符“b”表示fopen() 函数打开的文件为二进制文件，而非纯文字文件。

（4）输出：一个指向FILE类型的指针。

**2．关闭文件函数fclose()**

（1）格式：int fclose(FILE \* stream);

（2）功能：用来关闭先前fopen()打开的一个文件。此动作会让缓冲区内的数据写入文件中，并释放系统所提供的文件资源。

（3）输入参数说明：

stream：指向要关闭文件的指针，它是先前执行fopen()函数的返回值。

（4）输出：若关闭文件成功则返回0；有错误发生时则返回EOF并把错误代码存到errno。

**3．读文件函数fread()**

（1）格式：size\_t fread( void \*buffer, size\_t size, size\_t count, FILE \*stream )；

（2）功能：读二进制文件到内存。

（3）输入参数说明：

buffer：用于存放输入数据的缓冲区的首地址；

stream：使用fopen()打开的文件的指针，用于指示要读取的文件；

size： 每个数据块的字节数；

count： 要读入的数据块的个数；

size\*count：表示要求读取的字节数。

（4）输出：实际读取的数据块的个数。

**4．写文件函数fwrite()**

（1）格式：size\_t fwite(const void \*buffer,size\_t size,size\_t count,FILE \*stream)；

（2）功能：将数据写到二进制文件中。

（3）输入参数说明：

buffer：用于存放输出数据的缓冲区的首地址；

stream：使用fopen()打开的文件的指针，用于指示要写出的文件；

size： 每个数据块的字节数；

count： 要写出的数据块的个数；

size\*count：表示要求写出的字符数。

（4）输出：实际写出的数据块的个数。

**5．判断文件结束函数feof ()**

（1）格式：int feof(FILE \* stream)

（2）功能：用来判断是否已读取到文件末尾。

（3）输入参数说明：

stream：使用fopen()打开的文件的指针，用于指示要判断的文件。

（4）输出：如果已读到文件尾则返回非零值，其他情况返回0。

**6．定位文件函数fseek()**

（1）格式：int fseek( FILE \*stream, long offset, int origin )；

（2）功能： 移动文件读写指针在文件中的位置。

（3）输入参数说明：

stream：使用fopen()打开的文件的指针，用于指示要定位读写指针的文件；

offset：位移量，以字节为单位；

origin：初始位置，有三个常量：

SEEK\_CUR：读写指针当前位置；

SEEK\_SET：文件开头；

SEEK\_END：文件末尾。

　　 当origin值为SEEK\_CUR 或SEEK\_END时，参数offset可以为负值。

**3.3实例系统的设计与实现**

本实例系统是仿照FAT16文件系统来设计实现的，但根目录没有采用FAT16的固定位置、固定大小的根目录区，而是以根目录文件的形式来实现的，这也是目前主流文件系统对根目录的处理方式。

3.3.1 数据结构设计

**1．需要包含的头文件**

（1）#include <stdio.h>

（2）#include <malloc.h>

（3）#include <string.h>

（4）#include <time.h>

**2．定义的常量**

（1）#define BLOCKSIZE 1024 磁盘块大小

（2）#define SIZE 1024000 虚拟磁盘空间大小

（3）#define END 65535 FAT中的文件结束标志

（4）#define FREE 0 FAT中盘块空闲标志

（5）#define ROOTBLOCKNUM 2 根目录区所占盘块总数

（6）#define MAXOPENFILE 10 最多同时打开文件个数

**3．数据结构**

（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，它也是文件的目录项的内容。

typedef struct FCB //仿照FAT16设置的

{

char filename[8]; //文件名

char exname[3];//文件扩展名

unsigned char attribute;//文件属性字段：为简单起见，我们只为文件设置了两种属性：

//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 data;//文件创建日期

unsigned short first;//文件起始盘块号

unsigned long length;//文件长度（字节数）

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

}fcb;

（2）文件分配表FAT

在本实例中，文件分配表有两个作用：一是记录磁盘上每个文件所占据的磁盘块的块号；二是记录磁盘上哪些块已经分配出去了，哪些块是空闲的，即起到了位示图的作用。若FAT中某个表项的值为FREE，则表示该表项所对应的磁盘块是空闲的；若某个表项的值为END，则表示所对应的磁盘块是某文件的最后一个磁盘块；若某个表项的值是其他值，则该值表示某文件的下一个磁盘块的块号。为了提高系统的可靠性，本实例中设置了两张FAT表，它们互为备份，每个FAT占据两个磁盘块。

typedef struct FAT

{

unsigned short id;

}fat;

（3）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]; //文件名

char exname[3];//文件扩展名

unsigned char attribute;//文件属性：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 data;//文件创建日期

unsigned short first;//文件起始盘块号

unsigned long 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

//前面内容是文件的FCB中的内容。

// 下面设置的dirno和diroff记录了相应打开文件的目录项在父目录文件中的位置，//这样如果该文件的fcb被修改了，则要写回父目录文件时比较方便

int dirno; //相应打开文件的目录项在父目录文件中的盘块号

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[MAXOPENFILE][80]; //相应打开文件所在的目录名，这样方便快速检查出

//指定文件是否已经打开

int count; //读写指针在文件中的位置

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;

（4）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 BLOCK0 //引导块内容

{

//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、

char information[200];

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;

**4．全局变量定义**

（1）unsigned char \*myvhard： 指向虚拟磁盘的起始地址

（2）useropen openfilelist[MAXOPENFILE]： 用户打开文件表数组

（3）useropen \*ptrcurdir： 指向用户打开文件表中的当前目录所在打开文件表项的位置；

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

（5）unsigned char\* startp： 记录虚拟磁盘上数据区开始位置

**5．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，如图3-3所示：



图3-3 虚拟磁盘空间布局

当然，也可以仿照FAT16文件系统，设置根目录区，其位置紧跟第2张FAT后面，大小也是固定的，这个思路相对要简单一点，请同学们自己去实现。

3.3.2 实例主要命令及函数设计

**1．系统主函数main()**

（1）对应命令：无

（2）命令调用格式：无

（3）函数设计格式：void main()

（4）功能：系统主函数

（5）输入：无

（6）输出：无

（7）函数需完成的工作：

① 对前面定义的全局变量进行初始化；

② 调用startsys()进入文件系统；

③ 列出文件系统提供的各项功能及命令调用格式；

④ 显示命令行提示符，等待用户输入命令；

⑤ 将用户输入的命令保存到一个buf中；

⑥ 对buf中的内容进行命令解析，并调用相应的函数执行用户键入的命令；

⑦ 如果命令不是“my\_exitsys”，则命令执行完毕后转④。

**2. 进入文件系统函数startsys()**

（1）对应命令：无

（2）命令调用格式：无

（3）函数设计格式：void startsys()

（4）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。

（5）输入：无

（6）输出：无。

（7）函数需完成的工作：

① 申请虚拟磁盘空间；

② 使用c语言的库函数fopen()打开myfsys文件：若文件存在，则转③；若文件不存在，则创建之，转⑤

③ 使用c语言的库函数fread()读入myfsys文件内容到用户空间中的一个缓冲区中，并判断其开始的8个字节内容是否为“10101010”（文件系统魔数），如果是，则转④；否则转⑤；

④ 将上述缓冲区中的内容复制到内存中的虚拟磁盘空间中；转⑦

⑤ 在屏幕上显示“myfsys文件系统不存在，现在开始创建文件系统”信息，并调用my\_format()对①中申请到的虚拟磁盘空间进行格式化操作。转⑥；

⑥ 将虚拟磁盘中的内容保存到myfsys文件中；转⑦

⑦ 使用c语言的库函数fclose()关闭myfsys文件；

⑧ 初始化用户打开文件表，将表项0分配给根目录文件使用，并填写根目录文件的相关信息，由于根目录没有上级目录，所以表项中的dirno和diroff分别置为5（根目录所在起始块号）和0；并将ptrcurdir指针指向该用户打开文件表项。

⑨ 将当前目录设置为根目录。

**3．磁盘格式化函数my\_format()**

（1）对应命令：my\_format

（2）命令调用格式：my\_format

（3）函数设计格式：void my\_format()

（4）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）。

（5）输入：无

（6）输出：无。

（7）函数需完成的工作：

① 将虚拟磁盘第一个块作为引导块，开始的8个字节是文件系统的魔数，记为“10101010”；在之后写入文件系统的描述信息，如FAT表大小及位置、根目录大小及位置、盘块大小、盘块数量、数据区开始位置等信息；

② 在引导块后建立两张完全一样的FAT表，用于记录文件所占据的磁盘块及管理虚拟磁盘块的分配，每个FAT占据两个磁盘块；对于每个FAT中，前面5个块设置为已分配，后面995个块设置为空闲；

③ 在第二张FAT后创建根目录文件root，将数据区的第1块（即虚拟磁盘的第6块）分配给根目录文件，在该磁盘上创建两个特殊的目录项：“.”和“..”，其内容除了文件名不同之外，其他字段完全相同。

**4．更改当前目录函数my\_cd()**

（1）对应命令：my\_cd

（2）命令调用格式：my\_cd dirname

（3）函数设计格式：void my\_cd(char \*dirname)

（4）功能：改变当前目录到指定的名为dirname的目录。

（5）输入：

dirname：新的当前目录的目录名；

（6）输出：无

（7）函数需完成的工作：

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。

**5．创建子目录函数my\_mkdir()**

（1）对应命令：my\_mkdir

（2）命令调用格式：my\_ mkdir dirname

（3）函数设计格式：void my\_mkdir(char \*dirname)

（4）功能：在当前目录下创建名为dirname的子目录。

（5）输入：

dirname：新建目录的目录名。

（6）输出：无。

（7）函数需完成的工作：

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；

⑦ 返回。

**6．删除子目录函数rmdir()**

（1）对应命令：my\_ rmdir

（2）命令调用格式：my\_ rmdir dirname

（1）函数设计格式：void my\_rmdir(char \*dirname)

（2）功能：在当前目录下删除名为dirname的子目录。

（3）输入：

dirname：欲删除目录的目录名。

（4）输出：无。

（5）函数需完成的工作：

① 调用do\_read()读入当前目录文件内容到内存，检查当前目录下欲删除目录文件是否存在，若不存在则返回，并显示错误信息；

② 检查欲删除目录文件是否为空（除了“.”和“..”外没有其他子目录和文件），可根据其目录项中记录的文件长度来判断，若不为空则返回，并显示错误信息；

③ 检查该目录文件是否已经打开，若已打开则调用my\_close()关闭掉；

④ 回收该目录文件所占据的磁盘块，修改FAT；

⑤ 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：以覆盖写方式调用do\_write()来实现；

⑥ 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；

⑦ 返回。

**7．显示目录函数my\_ls()**

（1）对应命令：my\_ls

（2）命令调用格式：my\_ls

（3）函数设计格式：void my\_ls(void)

（4）功能：显示当前目录的内容（子目录和文件信息）。

（5）输入：无

（6）输出：无

（7）函数需完成的工作：

① 调用do\_read()读出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；

③ 返回。

**8．创建文件函数my\_create()**

（1）对应命令：my\_create

（2）命令调用格式：my\_create filename

（3）函数设计格式：int my\_create (char \*filename)

（4）功能：创建名为filename的新文件。

（5）输入：

filename：新建文件的文件名，可能包含路径。

（6）输出：若创建成功，返回该文件的文件描述符（文件打开表中的数组下标）；否则返回-1。

（7）函数需完成的工作：

① 为新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；

② 若新文件的父目录文件还没有打开，则调用my\_open()打开；若打开失败，则释放①中为新建文件分配的空闲文件打开表项，返回-1，并显示错误信息；

③ 调用do\_read()读出该父目录文件内容到内存，检查该目录下新文件是否重名，若重名则释放①中分配的打开文件表项，并调用my\_close()关闭②中打开的目录文件；然后返回-1，并显示错误信息；

④ 检查FAT是否有空闲的盘块，如有则为新文件分配一个盘块，否则释放①中分配的打开文件表项，并调用my\_close()关闭②中打开的目录文件；返回-1，并显示错误信息；

⑤ 在父目录中为新文件寻找一个空闲的目录项或为其追加一个新的目录项;需修改该目录文件的长度信息，并将该目录文件的用户打开文件表项中的fcbstate置为1；

⑥ 准备好新文件的FCB的内容，文件的属性为数据文件，长度为0，以覆盖写方式调用do\_write()将其填写到⑤中分配到的空目录项中；

⑦ 为新文件填写①中分配到的空闲打开文件表项，fcbstate字段值为0，读写指针值为0；

⑧ 调用my\_close()关闭②中打开的父目录文件；

⑨ 将新文件的打开文件表项序号作为其文件描述符返回。

**9．删除文件函数my\_rm()**

（1）对应命令：my\_rm

（2）命令调用格式：my\_rm filename

（3）函数设计格式：void my\_rm(char \*filename)

（4）功能：删除名为filename的文件。

（5）输入：

filename：欲删除文件的文件名，可能还包含路径。

（6）输出：无。

（7）函数需完成的工作：

① 若欲删除文件的父目录文件还没有打开，则调用my\_open()打开；若打开失败，则返回，并显示错误信息；

② 调用do\_read()读出该父目录文件内容到内存，检查该目录下欲删除文件是否存在，若不存在则返回，并显示错误信息；

③ 检查该文件是否已经打开，若已打开则关闭掉；

④ 回收该文件所占据的磁盘块，修改FAT；

⑤ 从文件的父目录文件中清空该文件的目录项，且free字段置为0：以覆盖写方式调用do\_write()来实现；；

⑥ 修改该父目录文件的用户打开文件表项中的长度信息，并将该表项中的fcbstate置为1；

⑦ 返回。

**10．打开文件函数my\_open()**

（1）对应命令：my\_open

（2）命令调用格式：my\_open filename

（3）函数设计格式：int my\_open(char \*filename)

（4）功能：打开当前目录下名为filename的文件。

（5）输入：

filename：欲打开文件的文件名

（6）输出：若打开成功，返回该文件的描述符（在用户打开文件表中表项序号）；否则返回-1。

（7）函数需完成的工作：

① 检查该文件是否已经打开，若已打开则返回-1，并显示错误信息；

② 调用do\_read()读出父目录文件的内容2到内存，检查该目录下欲打开文件是否存在，若不存在则返回-1，并显示错误信息；

③ 检查用户打开文件表中是否有空表项，若有则为欲打开文件分配一个空表项，若没有则返回-1，并显示错误信息；

④ 为该文件填写空白用户打开文件表表项内容，读写指针置为0；

⑤ 将该文件所分配到的空白用户打开文件表表项序号（数组下标）作为文件描述符fd返回。

**11．关闭文件函数my\_close()**

（1）对应命令：my\_close

（2）命令调用格式：my\_close fd

（3）函数设计格式：void my\_close(int fd)

（4）功能：关闭前面由my\_open()打开的文件描述符为fd的文件。

（5）输入：

fd：文件描述符。

（6）输出：无。

（7）函数需完成的工作：

① 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；

② 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；

③ 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；

④ 返回。

**12．写文件函数my\_write()**

（1）对应命令：my\_write

（2）命令调用格式：my\_write fd

（3）函数设计格式：int my\_write(int fd)

（4）功能：将用户通过键盘输入的内容写到fd所指定的文件中。磁盘文件的读写操作都必须以完整的数据块为单位进行，在写操作时，先将数据写在缓冲区中，缓冲区的大小与磁盘块的大小相同，然后再将缓冲区中的数据一次性写到磁盘块中；读出时先将一个磁盘块中的内容读到缓冲区中，然后再传送到用户区。本实例为了简便起见，没有设置缓冲区管理，只是在读写文件时由用户使用malloc()申请一块空间作为缓冲区，读写操作结束后使用free()释放掉。

写操作常有三种方式：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CTR+Z键（或其他设定的键）结束。

（5）输入：

fd： open()函数的返回值，文件的描述符；

（6）输出：实际写入的字节数。

（7）函数需完成的工作：

① 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；

② 提示并等待用户输入写方式：（1：截断写；2：覆盖写；3：追加写）

③ 如果用户要求的写方式是截断写，则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），将内存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并转④；如果用户要求的写方式是追加写，则修改文件的当前读写指针位置到文件的末尾，并转④；如果写方式是覆盖写，则直接转④；

④ 提示用户：整个输入内容通过CTR+Z键（或其他设定的键）结束；用户可分多次输入写入内容，每次用回车结束；

⑤ 等待用户从键盘输入文件内容，并将用户的本次输入内容保存到一临时变量text[]中，要求每次输入以回车结束，全部结束用CTR+Z键（或其他设定的键）；

⑥ 调用do\_write()函数将通过键盘键入的内容写到文件中。

⑦ 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，并转⑨；

⑧ 如果text[]中最后一个字符不是结束字符CTR+Z，则转⑦继续进行写操作；否则转⑨；

⑨ 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；

⑩ 返回实际写入的字节数。

**13．实际写文件函数do\_write()**

（1）对应命令：无

（2）命令调用格式：无

（3）函数设计格式：int my\_write(int fd，char \*text，int len，char wstyle)

（4）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。

（5）输入：

fd： open()函数的返回值，文件的描述符；

text：指向要写入的内容的指针；

len：本次要求写入字节数

wstyle：写方式

（6）输出：实际写入的字节数。

（7）函数需完成的工作：

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CTR+Z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 如果tmplen小于len，则转②继续写入；否则转⑧；

⑧ 返回本次实际写入的字节数。

**14．读文件函数my\_read()**

（1）对应命令：my\_read

（2）命令调用格式：my\_read fd len

（3）函数设计格式：int myread (int fd, int len)

（4）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。

（5）输入：

fd： open()函数的返回值，文件的描述符；

len: 要从文件中读出的字节数。

（6）输出：实际读出的字节数。

（7）函数需完成的工作：

① 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；

② 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；

③ 调用do\_read()将指定文件中的len字节内容读出到text[]中；

④ 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；

⑤ 返回。

**15．实际读文件函数do\_read()**

（1）对应命令：无

（2）命令调用格式：无

（3）函数设计格式：int do\_read (int fd, int len,char \*text)

（4）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。

（5）输入：

fd： open()函数的返回值，文件的描述符；

len: 要求从文件中读出的字节数。

text：指向存放读出数据的用户区地址

（6）输出：实际读出的字节数。

（7）函数需完成的工作：

① 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；

② 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；

③ 将该内存位置开始的1024B（一个磁盘块）内容读入buf中；

④ 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；

⑤ 将读写指针增加④中已读字节数，将应读写的字节数len减去④中已读字节数，若len大于0，则转②；否则转⑥；

⑥ 使用free()释放①中申请的buf。

⑦ 返回实际读出的字节数。

**16. 退出文件系统函数my\_exitsys()**

（1）对应命令：my\_exitsys

（2）命令调用格式：my\_ exitsys

（1）函数设计格式：void my\_exitsys()

（2）功能：退出文件系统。

（3）输入：无

（4）输出：无。

（5）函数需完成的工作：

① 使用C库函数fopen()打开磁盘上的myfsys文件；

② 将虚拟磁盘空间中的所有内容保存到磁盘上的myfsys文件中；

③ 使用c语言的库函数fclose()关闭myfsys文件；

④ 撤销用户打开文件表，释放其内存空间

⑤ 释放虚拟磁盘空间。