算法的总体思想：

该文件系统主要用了结构体来实现超级块、i结点、组长块、用户打开表、系统打开表等数据结构，文件的物理索引采用多级索引的结构；空闲物理块的分配回收采用成组链分配法、空闲i结点的分配回收采用铭记i结点标记回收位置的方法、内存i结点的组织采用了哈希索引表的方法、目录结构为多级目录结构。

磁盘的物理结构通过一个二进制文件来进行模拟，初始化时将相应位置的超级块、i结点、根目录文件、空闲块的组长块信息、初始管理员用户信息写入磁盘文件。用户进程可以并发登录，每个用户登录后增加一个用户打开表表项；同时，用户进入根目录，并在路径栈中压入根目录。用户打开一个文件时，首先在系统打开文件表中进行注册，同时完成读写控制；对文件进行读写操作时，在路径栈中找到当前目录从而找到文件i结点，通过i结点物理索引表获取文件物理块信息并进行分配回收。用户进程目录的变化通过每个用户进程的路径栈的入栈出栈实现。目录的删除通过一个递归程序来实现删除目录下所有文件和子目录。

**初始化模块：**

1.功能

建立保存磁盘的二进制文件，并根据要求在虚拟磁盘块对应的位置写入超级块、空闲i结点、根目录文件、空闲块的组长块、一个初始管理员用户信息。

2.数据结构

a.磁盘I结点

struct dinode{ //磁盘I结点（每个32个字节,这里占用其中28个字节)

short inode\_no; //i结点号（内部标识符）(-1为空）

short inode\_type; //i结点类型（-1为空结点，0为目录文件，1为文本文件）

short i\_mode; //读写权限 (0为高级，管理员有所有权，1为低级，都可以读写）

short file\_size; //文本的字节数（目录文件写成5了）

short i\_addr[10]; //物理地址索引表

};

b.目录项

struct direct{ //目录项（每个16B)

short inode\_no; //i结点号（内存i结点)(2B)

char file\_name[14]; //文件名(14B)

};

c.超级块

struct filsys{ //超级块（管理快），能放在一个块里（超级块）(315个字节）

//int inode\_num; //剩余磁盘i结点个数

//int disk\_size; //物理磁盘块数 ????有毛用

int freeblock\_num; //剩余空闲块块数

int freeblock\_stack\_num; //空闲块栈中空闲块数

int freeblock\_stack[50]; //空闲块栈(成组链算法中每组块数为50

int freeinode\_stack\_num; //空闲i结点栈中空闲i结点数量

short freeinode\_stack[50]; //空闲i结点栈

short flag\_inode; //铭记i结点

char changed; //超级快在内存是否被修改的标识

};

d.目录文件

struct dir{ //目录文件 2048+4=2052B，5个数据块

struct direct direct\_list[DIRNUM]; //DIRNUM=128,每个目录包含的最大项数为128

int size; //目录文件表项数

};

e.struct block\_grouphead{ //组长块

short block\_index[50];

};

3.算法

首先建立用来模拟磁盘的二进制文件，然后构建初始时的超级块，包括初始时的空闲i结点和空闲块栈，写入1号物理块；接着构建根目录磁盘i结点，写入2号物理块起始位置;

随后用空闲i结点填充i结点区的其他位置；把根目录文件存入数据区第一个块（34号块）；根据成组链分配算法把初始时的组长块写入对应物理块中；把磁盘静态用户表写入39号块。

**打开读写模块**

功能：打开文件或目录，根据i结点的类型值判断是目录文件还是文本文件，如果是目录文件则路径栈入栈，用户进入该目录，否则修改用户打开表和系统打开表相关信息，然后根据用户输入判断要对文件读还是写，如果是读，就将文件中的字符串打印出来，如果是写就构造一个新文件，释放原来的块，重新根据输入文本大小计算出物理块数并分配块，将输入的文本写入分配的物理块。

数据结构：

包括上述的磁盘I结点，目录项，超级块，目录文件，组长块

struct inode{ //内存i结点NICFREE

struct inode \*last;

struct inode \*next;

//int user\_sharenum; //共享次数(打开用户数量） 在系统打开表中进行统计共享次数（读写次数）这就没用了

//int disk\_sharenum; //磁盘共享次数，写快捷方式时使用

char change\_flag; //修改标志 (0为未修改，1为修改）

//----以下为从磁盘i结点复制过来的部分

short inode\_no; //i结点号（内部标识符）

short inode\_type; //i结点类型（-1为空结点，0为目录文件，1为索引文件，2为普通文件）

int i\_mode; //读写权限 （0为高级，1为低级）

int file\_size; //文件大小（多少块）

int i\_addr[10]; //物理地址索引表

};

struct hinode{ //哈希链表表项

struct inode \*i\_forw;

};

struct hinode hinode\_index[NHINO]; //内存i结点哈希链表索引表,NHINO=128, 内存i结点哈希链表的大小（取余数法)

struct system\_file{ //系统打开文件表表项 《操作系统教程与实验》226

char open\_type; //打开方式（读还是写） 0为写，1为读

int read\_count; //读进程个数

int write\_count; //写进程个数route\_dinode

short inode\_no; //i结点号，用来计算内存i结点哈希链表索引位置

char filename\_open[14]; //系统打开的文件名

struct inode \*f\_inode; //打开文件内存i结点指针,通过在内存i结点哈希链表中搜索获得

//int f\_off; //文件读写指针？？？？？？？？？

};

struct system\_file sys\_ofile[SYSOPENFILE]; //系统打开文件表SYSOPENFILE=40最多允许打开文件数，系统打开文件最大表项数

struct user\_file{ //用户打开文件表表项

int u\_ofile[NOFILE]; //系统打开文件表的索引，NOFILE=20

short user\_id; //用户进程编号

short group\_id; //用户组号

short route\_stack[ROUTENUM]; //路径栈,存放目录文件的i结点号（栈顶为当前目录） 最多40层

char dirname\_stack[ROUTENUM][14]; //路径名栈，仅用于显示当前路径

int route\_stack\_head; //路径栈栈顶(以上两个栈共用一个栈指针)

};

struct user\_file user\_file\_list[USERNUM]; //当前已登录用户(最多10个）

算法：首先判断用户是否登录，然后根据文件名在当前目录的目录文件中找到该文件的类型，如果是目录文件，就把该目录压入该用户的路径栈；如果是文本文件，判断系统打开表中有没有该文件的i结点号，记录判断结果。用户选择读还是写，如果是读文件，如果该文件已经被系统打开，就判断有无读写冲突，然后修改用户表系统表打开文件信息。文件未被系统打开时在系统打开该文件的基础上修改用户打开表和系统打开表信息，然后根据文件i结点找到物理块位置和文件大小读出该文本文件并显示；如果是写文件，关于用户打开表和系统打开表的处理同上，之后输入要写入的文本，重新分配物理块并写入磁盘。

**空闲磁盘块分配回收模块：**

功能：对某个i结点的物理索引表分配指定个数个物理块或者回收所有该物理索引表对应的物理块；以及分配或回收单个物理块。

struct filsys{ //超级快（管理快），能放在一个块里（超级块）(315个字节）

//int inode\_num; //剩余磁盘i结点个数

//int disk\_size; //物理磁盘块数 ????有毛用

int freeblock\_num; //剩余空闲块块数

int freeblock\_stack\_num; //空闲块栈中空闲块数（将操作系统教程与实验p230组长块第一个表项移到这里

int freeblock\_stack[50]; //空闲块栈(成组链算法中每组块数为50

int freeinode\_stack\_num; //空闲i结点栈中空闲i结点数量

short freeinode\_stack[50]; //空闲i结点栈 (栈大小为50与磁盘i结点分配算法有关，

short flag\_inode; //铭记i结点

char changed; //超级快在内存是否被修改的标识

};

struct block\_grouphead{ //组长块

short block\_index[50];

};

struct index\_block{ //物理索引块

short index[256];

};

采用成组链分配法：

1.分配物理块：

分配物理块时对每个物理块单独分配，分配时首先判断空闲块栈大小是否大于1，如果大于1，就直接在空闲块栈中获得分配的物理块号，否则把空闲块栈0号位置对应的最后一组的组长块写入超级块的空闲块栈，同时将该组长块作为分配的物理块。随后将获得的一组物理块号写入i结点的物理索引表，如果块数大于9，还需要分配一个索引块，把超过9的块号放入该索引块。

2.释放物理块

释放物理块时首先从i结点的物理索引表中获取要释放的物理块号，如果含有一级索引，还要访问索引块获得其余物理块号；对获得的待释放的物理块块号表中每一个块单独释放，如果空闲物理块栈没有满，就直接将该块号压栈，否则将该释放的物理块中写入超级块中的空闲块栈，而原来的空闲块栈清空，栈底压入该释放块块号。

i结点分配和回收模块：

功能：通过空闲i结点栈对空闲i结点进行分配和回收

数据结构：

struct filsys{ //超级快（管理快），能放在一个块里（超级块）(315个字节）

//int inode\_num; //剩余磁盘i结点个数

//int disk\_size; //物理磁盘块数 ????有毛用

int freeblock\_num; //剩余空闲块块数

int freeblock\_stack\_num; //空闲块栈中空闲块数（将操作系统教程与实验p230组长块第一个表项移到这里

int freeblock\_stack[50]; //空闲块栈(成组链算法中每组块数为50

int freeinode\_stack\_num; //空闲i结点栈中空闲i结点数量

short freeinode\_stack[50]; //空闲i结点栈 (栈大小为50与磁盘i结点分配算法有关，

short flag\_inode; //铭记i结点

char changed; //超级快在内存是否被修改的标识

};

算法：

1.分配i结点

如果空闲i结点栈不为空，就从栈中弹出一个i结点号，如果空闲i结点栈空了，就从铭记i结点开始按顺序找够50个空闲i结点压入栈中，并且弹出一个i结点号。

2.回收i结点号

如果要释放的i结点小于铭记i结点，就把该i结点作为新的i结点；如果空闲i结点栈满，仅将原来的空闲i结点的位置写入空闲i结点，否则再次基础上将该释放的i结点写入空闲i结点栈。

新建文本文件模块：

功能：在指定用户的当前目录下建立一个空的文本文件

数据结构：

struct dinode{ //磁盘I结点（每个32个字节,这里占用其中28个字节)

short inode\_no; //i结点号（内部标识符）(-1为空）

short inode\_type; //i结点类型（-1为空结点，0为目录文件，1为文本文件）

short i\_mode; //读写权限 (0为高级，管理员有所有权，1为低级，都可以读写）

short file\_size; //文本的字节数（目录文件写成5了）

short i\_addr[10]; //物理地址索引表

};

struct direct{ //目录项文件（每个16B)

short inode\_no; //i结点号（内存i结点)(2B)

char file\_name[14]; //文件名(14B)

};

struct dir{ //目录文件 2048+4=2052B，5个数据块

struct direct direct\_list[DIRNUM]; //DIRNUM=128,每个目录包含的最大项数 16\*128=2048B=4个数据块

int size; //目录文件表项数

};

算法：

首先分配一个磁盘i结点，然后从当前路径栈得到当前目录的目录文件i结点，通过i结点找到物理位置读取当期目录文件，将新建的文件信息写入该目录文件中并写回磁盘；然后把该文件的i结点写入分配的磁盘空闲i结点中。

新建目录模块：

功能：在当前目录下新建一个目录

数据结构：

struct dinode{ //磁盘I结点（每个32个字节,这里占用其中28个字节)

short inode\_no; //i结点号（内部标识符）(-1为空）

short inode\_type; //i结点类型（-1为空结点，0为目录文件，1为文本文件）

short i\_mode; //读写权限 (0为高级，管理员有所有权，1为低级，都可以读写）

short file\_size; //文本的字节数（目录文件写成5了）

short i\_addr[10]; //物理地址索引表

};

struct direct{ //目录项文件（每个16B)

short inode\_no; //i结点号（内存i结点)(2B)

char file\_name[14]; //文件名(14B)

};

struct dir{ //目录文件 2048+4=2052B，5个数据块

struct direct direct\_list[DIRNUM]; //DIRNUM=128,每个目录包含的最大项数 16\*128=2048B=4个数据块

int size; //目录文件表项数

};

算法：

首先分配一个磁盘i结点，然后从当前路径栈得到当前目录的目录文件i结点，通过i结点找到物理位置读取当期目录文件，将要新建的目录的信息写入该目录文件并写回磁盘；为该目录文件的i结点的物理索引表分配5个物理块，把该i结点写入分配的磁盘i结点位置；在之前为目录文件分配的物理块中写入空目录文件。用户输入是否进入该目录，如果要进入就调用打开文件函数。

删除文件模块：

功能：将指定用户的当前目录下的一个文本文件或目录删除

数据结构：

struct dinode{ //磁盘I结点（每个32个字节,这里占用其中28个字节)

short inode\_no; //i结点号（内部标识符）(-1为空）

short inode\_type; //i结点类型（-1为空结点，0为目录文件，1为文本文件）

short i\_mode; //读写权限 (0为高级，管理员有所有权，1为低级，都可以读写）

short file\_size; //文本的字节数（目录文件写成5了）

short i\_addr[10]; //物理地址索引表

}; //还需要一个类型字段标记是不是空的i结点

struct direct{ //目录项文件（每个16B)

short inode\_no; //i结点号（内存i结点)(2B)

char file\_name[14]; //文件名(14B)

};

struct dir{ //目录文件 2048+4=2052B，5个数据块

struct direct direct\_list[DIRNUM]; //DIRNUM=128,每个目录包含的最大项数 16\*128=2048B=4个数据块

int size; //目录文件表项数

};

struct system\_file{ //系统打开文件表表项 《操作系统教程与实验》226

char open\_type; //打开方式（读还是写） 0为写，1为读

int read\_count; //读进程个数

int write\_count; //写进程个数route\_dinode

short inode\_no; //i结点号，用来计算内存i结点哈希链表索引位置

char filename\_open[14]; //系统打开的文件名

struct inode \*f\_inode; //打开文件内存i结点指针,通过在内存i结点哈希链表中搜索获得

//int f\_off; //文件读写指针？？？？？？？？？

};

struct system\_file sys\_ofile[SYSOPENFILE]; //系统打开文件表SYSOPENFILE=40最多允许打开文件数，系统打开文件最大表项数

struct user\_file{ //用户打开文件表表项

//int default\_mode; //默认打开方式？？？？？？???????????

int u\_ofile[NOFILE]; //系统打开文件表的索引，NOFILE=20

short user\_id; //用户进程编号

short group\_id; //用户组号

short route\_stack[ROUTENUM]; //路径栈,存放目录文件的i结点号（栈顶为当前目录） 最多40层

char dirname\_stack[ROUTENUM][14]; //路径名栈，仅用于显示当前路径

int route\_stack\_head; //路径栈栈顶(以上两个栈共用一个栈指针)

};

struct user\_file user\_file\_list[USERNUM]; //当前已登录用户(最多10个）

算法：

从该用户路径栈得到当前目录i结点号从而获取当前目录文件，在当前目录文件中找到要删除的文件的i结点号，如果i结点对应的文件类型是文本文件，就判断系统打开表是否打开该文件，如果打开就拒绝操作，否则就释放该文件的物理块，然后释放该文件的i结点；如果i结点对应的文件类型是目录，进入该目录，获取该目录的目录文件，递归调用本模块删除该目录文件中每一个目录项对应的文本文件或目录，然后退出当前目录，获取当前目录的目录文件，去掉要删除的目录对应的目录项，并将该目录文件写回磁盘。

关闭文本文件模块：

功能：关闭某个用户进程打开的某个文本文件

数据结构：

struct system\_file{ //系统打开文件表表项 《操作系统教程与实验》226

char open\_type; //打开方式（读还是写） 0为写，1为读

int read\_count; //读进程个数

int write\_count; //写进程个数route\_dinode

short inode\_no; //i结点号，用来计算内存i结点哈希链表索引位置

char filename\_open[14]; //系统打开的文件名

struct inode \*f\_inode; //打开文件内存i结点指针,通过在内存i结点哈希链表中搜索获得

//int f\_off; //文件读写指针？？？？？？？？？

};

struct system\_file sys\_ofile[SYSOPENFILE]; //系统打开文件表SYSOPENFILE=40最多允许打开文件数，系统打开文件最大表项数

struct user\_file{ //用户打开文件表表项

//int default\_mode; //默认打开方式？？？？？？???????????

int u\_ofile[NOFILE]; //系统打开文件表的索引，NOFILE=20

short user\_id; //用户进程编号

short group\_id; //用户组号

short route\_stack[ROUTENUM]; //路径栈,存放目录文件的i结点号（栈顶为当前目录） 最多40层

char dirname\_stack[ROUTENUM][14]; //路径名栈，仅用于显示当前路径

int route\_stack\_head; //路径栈栈顶(以上两个栈共用一个栈指针)

};

算法：

在该用户的用户进程打开表中找到该文件的系统打开表打开索引，去掉该索引；然后根据获得的索引更改系统打开表中该文件的访问计数，如果访问计数值为0就关闭系统打开表对该文件的打开，并且释放内存i结点。

多用户登录登出注册模块：

功能：实现用户并发登录，保持多个用户进程在线的功能，新增用户。

数据结构：

struct user\_file{ //用户打开文件表表项

//int default\_mode; //默认打开方式？？？？？？???????????

int u\_ofile[NOFILE]; //系统打开文件表的索引，NOFILE=20

short user\_id; //用户进程编号

short group\_id; //用户组号

short route\_stack[ROUTENUM]; //路径栈,存放目录文件的i结点号（栈顶为当前目录） 最多40层

char dirname\_stack[ROUTENUM][14]; //路径名栈，仅用于显示当前路径

int route\_stack\_head; //路径栈栈顶(以上两个栈共用一个栈指针)

};

struct user\_file user\_file\_list[USERNUM]; //当前已登录用户(最多10个）

算法：

1.登录

首先在用户打开表中判断重复登录，然后从磁盘获得静态用户表，判断用户是否重复登录，如果没有登录就在用户打开表添加该用户

2.登出

首先关闭用户打开的所有文件，然后在用户打开表中删除该用户。

3.注册新用户

读取磁盘中的静态用户表，加入新增的用户，再写回磁盘。

文件的分级或和户的权限提升模块：

功能：管理员获得文件所有权，管理员释放文件所有权，管理员提升普通用户为管理员

数据结构：

struct user\_file{ //用户打开文件表表项

//int default\_mode; //默认打开方式？？？？？？???????????

int u\_ofile[NOFILE]; //系统打开文件表的索引，NOFILE=20

short user\_id; //用户进程编号

short group\_id; //用户组号

short route\_stack[ROUTENUM]; //路径栈,存放目录文件的i结点号（栈顶为当前目录） 最多40层

char dirname\_stack[ROUTENUM][14]; //路径名栈，仅用于显示当前路径

int route\_stack\_head; //路径栈栈顶(以上两个栈共用一个栈指针)

};

struct user\_file user\_file\_list[USERNUM]; //当前已登录用户(最多10个）

struct user\_process{ //用户进程表

short user\_id; //用户进程编号

short group\_id; //用户组号 0号为系统进程（管理员），1号为普通用户进程

char password[12]; //口令

};

算法：

1.管理员获得和释放文件所有权

先判断操作者是否是管理员，如果是管理员，在当前目录的目录文件中找到该文件的i结点号，获得该i结点，修改等级并写回。

2.管理员提升普通用户为管理员

先判断操作者是否是管理员，如果是管理员，获得磁盘静态用户表，修改用户等级，并写回磁盘；如果要提升权限的用户已经登录，还要修改其在用户打开表中的用户组别信息。

实验结果：

1.