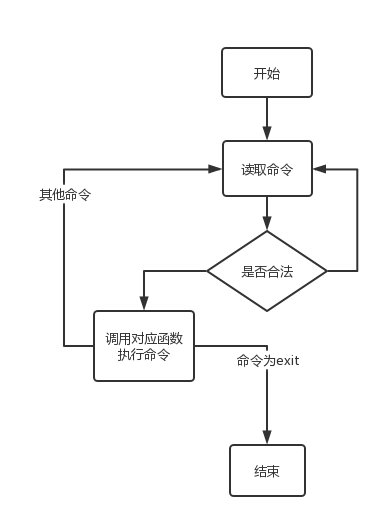
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 实习题目：文件系统 | | | | |
| 完成人 | 姓名： | 课序号： | 班级： | 学号： |
| 实习内容及要求简要描述 | 一.实验内容  准备一张fat格式的软盘，在Linux下编写一个用户程序，对软盘上的文件进行管理。  1. 设计并实现一个目录列表函数(类似Linux的shell命令ls)。该函数只需包含基本命令即可。  2. 设计并实现一个改变目录函数，即把当前目录切换到上一层目录或当前目录的子目录中(无须处理路径名)。  3. 设计并实现一个删除文件函数，该函数使用要删除的文件名(在当前目录中)作为参数  二．实验目的  1. 了解有关Linux文件管理的知识。  2. 设计一个简化的文件系统并实现一组操作。  Linux提供的文件管理适用于多种不同类型的磁盘文件系统，这些磁盘文件系统的组织结构各异，磁盘格式也不相同。虽然Linux为硬盘提供了自己的文件系统组织结构(ext2)，但是文件管理系统可以扩展为在任何文件系统格式上执行。  三．实验环境  Linux系统虚拟机，Windows2000及以上  四．实验要求  准备一张fat格式的软盘，在Linux下编写一个用户程序，对软盘上的文件进行管理，要求提供如下一组函数：  (1)设计并实现一个目录列表函数(类似Linux的shell命令ls)。该函数只需包含基本命令即可，不必像Linux中的ls那样支持各种操作。函数格式为：int fd\_ls();  (2)设计并实现一个改变目录函数，即把当前目录切换到上一层目录或当前目录的子目录中(无须处理路径名)，函数格式为：  int fd\_cd(char \*directory);  本函数假设软盘上已存在子目录，需要在文件系统中使用一个静态变量来代表当前目录。本函数要对当前目录变量进行操作。  (3)设计并实现一个删除文件函数，该函数使用要删除的文件名(在当前目录中)作为参数，函数格式为：  int fd\_rm(char \*name);  该函数需要查找文件，遍历FAT中的链接，设置FAT中的每个簇项并将其标志为未使用的，更新目录项。在删除的情况中，要注意文件的隐藏、只读和系统属性，任何具有这些设置的文件都不能删除。 | | | |
| 主要代码结构  （附注释） | （1）头文件  #include <stdio.h>  #include <malloc.h>  #include <string.h>  #include <time.h>  （2）宏定义  #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t  #define MAXTEXT 10000  （3）结构定义  /\* 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，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 long length; // 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;    /\* 文件分配表 \*/  typedef struct FAT  {  unsigned short id; // 磁盘块的状态（空闲的，最后的，下一个）  }fat;  /\* 用户打开文件表 \*/  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 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    unsigned short 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[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 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;    /\* 引导块 \*/  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;  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  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. 函数声明   void startsys(); // 进入文件系统  void my\_format(); // 磁盘格式化  void my\_cd(char \*dirname); // 更改当前目录  void my\_mkdir(char \*dirname); // 创建子目录  void my\_rmdir(char \*dirname); // 删除子目录  void my\_ls(); // 显示目录  void my\_create (char \*filename); // 创建文件  void my\_rm(char \*filename); // 删除文件  int my\_open(char \*filename); // 打开文件  int my\_close(int fd); // 关闭文件  int my\_write(int fd); // 写文件  int do\_write(int fd, char \*text, int len, char wstyle); // 实际写文件  int my\_read (int fd, int len); // 读文件  int do\_read (int fd, int len,char \*text); // 实际读文件  void my\_exitsys(); // 退出文件系统  unsigned short findblock(); // 寻找空闲盘块  int findopenfile(); // 寻找空闲文件表项 | | | |
| 结果分析（或错误原因分析） | 程序能够正常运行。能够执行create(新建文件), rm(删除文件), mkdir(新建目录), rmdir(删除目录), cd(打开目录), ls(显示当前目录下的文件和目录)等基础指令。  在这个基础上，还尝试添加文件的打开关闭与读写，但未能完全实现，仍有许多要改进的地方。 | | | |

附录一、流程图



附录二、源程序

#include <stdio.h>

#include <malloc.h>

#include <string.h>

#include <time.h>

#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t

#define MAXTEXT 10000

/\* 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，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 long length; // 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;

/\* 文件分配表 \*/

typedef struct FAT

{

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

}fat;

/\* 用户打开文件表 \*/

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 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

unsigned short 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[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 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;

/\* 引导块 \*/

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;

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

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

void startsys(); // 进入文件系统

void my\_format(); // 磁盘格式化

void my\_cd(char \*dirname); // 更改当前目录

void my\_mkdir(char \*dirname); // 创建子目录

void my\_rmdir(char \*dirname); // 删除子目录

void my\_ls(); // 显示目录

void my\_create (char \*filename); // 创建文件

void my\_rm(char \*filename); // 删除文件

int my\_open(char \*filename); // 打开文件

int my\_close(int fd); // 关闭文件

int my\_write(int fd); // 写文件

int do\_write(int fd, char \*text, int len, char wstyle); // 实际写文件

int my\_read (int fd, int len); // 读文件

int do\_read (int fd, int len,char \*text); // 实际读文件

void my\_exitsys(); // 退出文件系统

unsigned short findblock(); // 寻找空闲盘块

int findopenfile(); // 寻找空闲文件表项

void startsys()

{

FILE \*fp;

unsigned char buf[SIZE];

fcb \*root;

int i;

myvhard = (unsigned char \*)malloc(SIZE);//申请虚拟磁盘空间

memset(myvhard, 0, SIZE);//将myvhard中前SIZE个字节用 0 替换并返回 myvhard

if((fp = fopen(myfilename, "r")) != NULL)

{

fread(buf, SIZE, 1, fp);//将二进制文件读取到缓冲区

fclose(fp);//关闭打开的文件，缓冲区数据写入文件，释放系统提供文件资源

if(strcmp(((block0 \*)buf)->magic, "10101010"))//判断开始的8个字节内容是否为文件系统魔数

{

printf("myfilesys is not exist,begin to creat the file...\n");

my\_format();

}

else

{

for(i = 0; i < SIZE; i++)

myvhard[i] = buf[i];

}

}

else

{

printf("myfilesys is not exist,begin to creat the file...\n");

my\_format();

}

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

strcpy(openfilelist[0].filename, root->filename);

strcpy(openfilelist[0].exname, root->exname);

openfilelist[0].attribute = root->attribute;

openfilelist[0].time = root->time;

openfilelist[0].date = root->date;

openfilelist[0].first = root->first;

openfilelist[0].length = root->length;

openfilelist[0].free = root->free;

openfilelist[0].dirno = 5;

openfilelist[0].diroff = 0;

strcpy(openfilelist[0].dir, "\\root\\");

openfilelist[0].father = 0;

openfilelist[0].count = 0;

openfilelist[0].fcbstate = 0;

openfilelist[0].topenfile = 1;

for(i = 1; i < MAXOPENFILE; i++)

openfilelist[i].topenfile = 0;

curdir = 0;

strcpy(currentdir, "\\root\\");

startp = ((block0 \*)myvhard)->startblock;

}

void my\_format()

{

FILE \*fp;

fat \*fat1, \*fat2;

block0 \*blk0;

time\_t now;

struct tm \*nowtime;

fcb \*root;

int i;

blk0 = (block0 \*)myvhard;

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

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

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

strcpy(blk0->magic, "10101010");

strcpy(blk0->information, "My FileSystem Ver 1.0 \n Blocksize=1KB Whole size=1000KB Blocknum=1000 RootBlocknum=2\n");

blk0->root = 5;

blk0->startblock = (unsigned char \*)root;

for(i = 0; i < 5; i++)

{

fat1->id = END;

fat2->id = END;

fat1++;

fat2++;

}

fat1->id = 6;

fat2->id = 6;

fat1++;

fat2++;

fat1->id = END;

fat2->id = END;

fat1++;

fat2++;

for(i = 7; i < SIZE / BLOCKSIZE; i++)

{

fat1->id = FREE;

fat2->id = FREE;

fat1++;

fat2++;

}

now = time(NULL);

nowtime = localtime(&now);

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

strcpy(root->exname, "");

root->attribute = 0x28;

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

root->date = (nowtime->tm\_year - 80) \* 512 + (nowtime->tm\_mon + 1) \* 32 + nowtime->tm\_mday;

root->first = 5;

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

root->free = 1;

root++;

now = time(NULL);

nowtime = localtime(&now);

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

strcpy(root->exname, "");

root->attribute = 0x28;

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

root->date = (nowtime->tm\_year - 80) \* 512 + (nowtime->tm\_mon + 1) \* 32 + nowtime->tm\_mday;

root->first = 5;

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

root->free = 1;

fp = fopen(myfilename, "w");

fwrite(myvhard, SIZE, 1, fp);

fclose(fp);

}

void my\_cd(char \*dirname)

{

char \*dir;

int fd;

dir = strtok(dirname, "\\");//分解字符串为一组字符串。dirname为要分解的字符串，"\\"为分隔符字符串

if(strcmp(dir, ".") == 0)

return;

else if(strcmp(dir, "..") == 0)

{

if(curdir)

curdir = my\_close(curdir);

return;

}

else if(strcmp(dir, "root") == 0)

{

while(curdir)

curdir = my\_close(curdir);

dir = strtok(NULL, "\\");

}

while(dir)

{

fd = my\_open(dir);

if(fd != -1)

curdir = fd;

else

return;

dir = strtok(NULL, "\\");

}

}

void my\_mkdir(char \*dirname)

{

fcb \*fcbptr;

fat \*fat1, \*fat2;

time\_t now;

struct tm \*nowtime;

char text[MAXTEXT];

unsigned short blkno;

int rbn, fd, i;

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

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

openfilelist[curdir].count = 0;

rbn = do\_read(curdir, openfilelist[curdir].length, text);

fcbptr = (fcb \*)text;

for(i = 0; i < rbn / sizeof(fcb); i++)//在当前目录下找，是否有重名目录

{

if(strcmp(fcbptr->filename, dirname) == 0 && strcmp(fcbptr->exname, "") == 0)

{

printf("Error,the dirname is already exist!\n");

return;

}

fcbptr++;

}

fcbptr = (fcb \*)text;

for(i = 0; i < rbn / sizeof(fcb); i++)

{

if(fcbptr->free == 0)

break;

fcbptr++;

}

blkno = findblock();//寻找空闲盘块

if(blkno == -1)

return;

(fat1 + blkno)->id = END;

(fat2 + blkno)->id = END;

now = time(NULL);

nowtime = localtime(&now);

strcpy(fcbptr->filename, dirname);

strcpy(fcbptr->exname, "");

fcbptr->attribute = 0x30;

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

fcbptr->date = (nowtime->tm\_year - 80) \* 512 + (nowtime->tm\_mon + 1) \* 32 + nowtime->tm\_mday;

fcbptr->first = blkno;

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

fcbptr->free = 1;

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

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

fd = my\_open(dirname);//建立新目录的'.','..'目录

if(fd == -1)

return;

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

now = time(NULL);

nowtime = localtime(&now);

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

strcpy(fcbptr->exname, "");

fcbptr->attribute = 0x28;

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

fcbptr->date = (nowtime->tm\_year - 80) \* 512 + (nowtime->tm\_mon + 1) \* 32 + nowtime->tm\_mday;

fcbptr->first = blkno;

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

fcbptr->free = 1;

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

now = time(NULL);

nowtime = localtime(&now);

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

strcpy(fcbptr->exname, "");

fcbptr->attribute = 0x28;

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

fcbptr->date = (nowtime->tm\_year - 80) \* 512 + (nowtime->tm\_mon + 1) \* 32 + nowtime->tm\_mday;

fcbptr->first = blkno;

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

fcbptr->free = 1;

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

free(fcbptr);

my\_close(fd);

fcbptr = (fcb \*)text;

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

openfilelist[curdir].count = 0;

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

openfilelist[curdir].fcbstate = 1;

}

void my\_rmdir(char \*dirname)

{

fcb \*fcbptr,\*fcbptr2;

fat \*fat1, \*fat2, \*fatptr1, \*fatptr2;

char text[MAXTEXT], text2[MAXTEXT];

unsigned short blkno;

int rbn, rbn2, fd, i, j;

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

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

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

{

printf("Error,can't remove this directory.\n");

return;

}

openfilelist[curdir].count = 0;

rbn = do\_read(curdir, openfilelist[curdir].length, text);

fcbptr = (fcb \*)text;

for(i = 0; i < rbn / sizeof(fcb); i++)//查找要删除的目录

{

if(strcmp(fcbptr->filename, dirname) == 0 && strcmp(fcbptr->exname, "") == 0)

break;

fcbptr++;

}

if(i == rbn / sizeof(fcb))

{

printf("Error,the directory is not exist.\n");

return;

}

fd = my\_open(dirname);

rbn2 = do\_read(fd, openfilelist[fd].length, text2);

fcbptr2 = (fcb \*)text2;

for(j = 0; j < rbn2 / sizeof(fcb); j++)//判断要删除目录是否为空

{

if(strcmp(fcbptr2->filename, ".") && strcmp(fcbptr2->filename, "..") && strcmp(fcbptr2->filename, ""))

{

my\_close(fd);

printf("Error,the directory is not empty.\n");

return;

}

fcbptr2++;

}

blkno = openfilelist[fd].first;

while(blkno != END)

{

fatptr1 = fat1 + blkno;

fatptr2 = fat2 + blkno;

blkno = fatptr1->id;

fatptr1->id = FREE;

fatptr2->id = FREE;

}

my\_close(fd);

strcpy(fcbptr->filename, "");

fcbptr->free = 0;

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

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

openfilelist[curdir].fcbstate = 1;

}

void my\_ls()

{

fcb \*fcbptr;

char text[MAXTEXT];

int rbn, i;

openfilelist[curdir].count = 0;

rbn = do\_read(curdir, openfilelist[curdir].length, text);

fcbptr = (fcb \*)text;

for(i = 0; i < rbn / sizeof(fcb); i++)

{

if(fcbptr->free)

{

if(fcbptr->attribute & 0x20)

printf("%s\\\t\t<DIR>\t\t%d/%d/%d\t%02d:%02d:%02d\n", fcbptr->filename, (fcbptr->date >> 9) + 1980, (fcbptr->date >> 5) & 0x000f, fcbptr->date & 0x001f, fcbptr->time >> 11, (fcbptr->time >> 5) & 0x003f, fcbptr->time & 0x001f \* 2);

else

printf("%s.%s\t\t%dB\t\t%d/%d/%d\t%02d:%02d:%02d\t\n", fcbptr->filename, fcbptr->exname, (int)(fcbptr->length), (fcbptr->date >> 9) + 1980, (fcbptr->date >> 5) & 0x000f, fcbptr->date & 0x1f, fcbptr->time >> 11, (fcbptr->time >> 5) & 0x3f, fcbptr->time & 0x1f \* 2);

}

fcbptr++;

}

}

void my\_create(char \*filename)

{

fcb \*fcbptr;

fat \*fat1, \*fat2;

char \*fname, \*exname, text[MAXTEXT];

unsigned short blkno;

int rbn, i;

time\_t now;

struct tm \*nowtime;

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

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

fname = strtok(filename, ".");

exname = strtok(NULL, ".");

if(strcmp(fname, "") == 0)

{

printf("Error,creating file must have a right name.\n");

return;

}

if(!exname)

{

printf("Error,creating file must have a extern name.\n");

return;

}

openfilelist[curdir].count = 0;

rbn = do\_read(curdir, openfilelist[curdir].length, text);

fcbptr = (fcb \*)text;

for(i = 0; i < rbn / sizeof(fcb); i++)

{

if(strcmp(fcbptr->filename, fname) == 0 && strcmp(fcbptr->exname, exname) == 0)

{

printf("Error,the filename is already exist!\n");

return;

}

fcbptr++;

}

fcbptr = (fcb \*)text;

for(i = 0; i < rbn / sizeof(fcb); i++)

{

if(fcbptr->free == 0)

break;

fcbptr++;

}

blkno = findblock();

if(blkno == -1)

return;

(fat1 + blkno)->id = END;

(fat2 + blkno)->id = END;

now = time(NULL);

nowtime = localtime(&now);

strcpy(fcbptr->filename, fname);

strcpy(fcbptr->exname, exname);

fcbptr->attribute = 0x00;

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

fcbptr->date = (nowtime->tm\_year - 80) \* 512 + (nowtime->tm\_mon + 1) \* 32 + nowtime->tm\_mday;

fcbptr->first = blkno;

fcbptr->length = 0;

fcbptr->free = 1;

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

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

fcbptr = (fcb \*)text;

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

openfilelist[curdir].count = 0;

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

openfilelist[curdir].fcbstate = 1;

}

void my\_rm(char \*filename)

{

fcb \*fcbptr;

fat \*fat1, \*fat2, \*fatptr1, \*fatptr2;

char \*fname, \*exname, text[MAXTEXT];

unsigned short blkno;

int rbn, i;

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

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

fname = strtok(filename, ".");

exname = strtok(NULL, ".");

if(strcmp(fname, "") == 0)

{

printf("Error,removing file must have a right name.\n");

return;

}

if(!exname)

{

printf("Error,removing file must have a extern name.\n");

return;

}

openfilelist[curdir].count = 0;

rbn = do\_read(curdir, openfilelist[curdir].length, text);

fcbptr = (fcb \*)text;

for(i = 0; i < rbn / sizeof(fcb); i++)

{

if(strcmp(fcbptr->filename, fname) == 0 && strcmp(fcbptr->exname, exname) == 0)

break;

fcbptr++;

}

if(i == rbn / sizeof(fcb))

{

printf("Error,the file is not exist.\n");

return;

}

openfilelist[curdir].count = 0;

rbn = do\_read(curdir, openfilelist[curdir].length, text);

fcbptr = (fcb \*)text;

for(i = 0; i < rbn / sizeof(fcb); i++)

{

if(strcmp(fcbptr->filename, fname) == 0 && strcmp(fcbptr->exname, exname) == 0)

break;

fcbptr++;

}

if(i == rbn / sizeof(fcb))

{

printf("Error,the file is not exist.\n");

return;

}

blkno = fcbptr->first;

while(blkno != END)

{

fatptr1 = fat1 + blkno;

fatptr2 = fat2 + blkno;

blkno = fatptr1->id;

fatptr1->id = FREE;

fatptr2->id = FREE;

}

strcpy(fcbptr->filename, "");

fcbptr->free = 0;

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

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

openfilelist[curdir].fcbstate = 1;

}

int my\_open(char \*filename)

{

fcb \*fcbptr;

char \*fname, exname[3], \*str, text[MAXTEXT];

int rbn, fd, i;

fname = strtok(filename, ".");

str = strtok(NULL, ".");

if(str)

strcpy(exname, str);

else

strcpy(exname, "");

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

{

if(strcmp(openfilelist[i].filename, fname) == 0 && strcmp(openfilelist[i].exname, exname) == 0 && i != curdir)

{

printf("Error,the file is already open.\n");

return -1;

}

}

openfilelist[curdir].count = 0;

rbn = do\_read(curdir, openfilelist[curdir].length, text);

fcbptr = (fcb \*)text;

for(i = 0; i < rbn / sizeof(fcb); i++)

{

if(strcmp(fcbptr->filename, fname) == 0 && strcmp(fcbptr->exname, exname) == 0)

break;

fcbptr++;

}

if(i == rbn / sizeof(fcb))

{

printf("Error,the file is not exist.\n");

return -1;

}

fd = findopenfile();

if(fd == -1)

return -1;

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

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

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

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

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

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

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

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

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

openfilelist[fd].diroff = i;

strcpy(openfilelist[fd].dir, openfilelist[curdir].dir);

strcat(openfilelist[fd].dir, filename);

if(fcbptr->attribute & 0x20)

strcat(openfilelist[fd].dir, "\\");

openfilelist[fd].father = curdir;

openfilelist[fd].count = 0;

openfilelist[fd].fcbstate = 0;

openfilelist[fd].topenfile = 1;

return fd;

}

int my\_close(int fd)

{

fcb \*fcbptr;

int father;

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

{

printf("Error,the file is not exist.\n");

return -1;

}

if(openfilelist[fd].fcbstate)

{

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

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

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

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

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

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

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

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

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

father = openfilelist[fd].father;

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

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

free(fcbptr);

openfilelist[fd].fcbstate = 0;

}

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

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

openfilelist[fd].topenfile = 0;

return father;

}

int my\_write(int fd)

{

fat \*fat1, \*fat2, \*fatptr1, \*fatptr2;

int wstyle, len, ll, tmp;

char text[MAXTEXT];

unsigned short blkno;

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

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

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

{

printf("The file is not exist!\n");

return -1;

}

while(1)

{

printf("Please enter the number of write style:\n1.cut write\t2.cover write\t3.add write\n");

scanf("%d", &wstyle);

if(wstyle > 0 && wstyle < 4)

break;

printf("Input Error!");

}

getchar();

switch(wstyle)

{

case 1:

blkno = openfilelist[fd].first;

fatptr1 = fat1 + blkno;

fatptr2 = fat2 + blkno;

blkno = fatptr1->id;

fatptr1->id = END;

fatptr2->id = END;

while(blkno != END)

{

fatptr1 = fat1 + blkno;

fatptr2 = fat2 + blkno;

blkno = fatptr1->id;

fatptr1->id = FREE;

fatptr2->id = FREE;

}

openfilelist[fd].count = 0;

openfilelist[fd].length = 0;

break;

case 2:

openfilelist[fd].count = 0;

break;

case 3:

openfilelist[fd].count = openfilelist[fd].length;

break;

default:

break;

}

ll = 0;

printf("please input write data(end with Ctrl+Z):\n");

while(gets(text))

{

len = strlen(text);

text[len++] = '\n';

text[len] = '\0';

tmp = do\_write(fd, text, len, wstyle);

if(tmp != -1)

ll += tmp;

if(tmp < len)

{

printf("Wirte Error!");

break;

}

}

return ll;

}

int do\_write(int fd, char \*text, int len, char wstyle)

{

fat \*fat1, \*fat2, \*fatptr1, \*fatptr2;

unsigned char \*buf, \*blkptr;

unsigned short blkno, blkoff;

int i, ll;

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

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

buf = (unsigned char \*)malloc(BLOCKSIZE);

if(buf == NULL)

{

printf("malloc failed!\n");

return -1;

}

blkno = openfilelist[fd].first;

blkoff = openfilelist[fd].count;

fatptr1 = fat1 + blkno;

fatptr2 = fat2 + blkno;

while(blkoff >= BLOCKSIZE)

{

blkno = fatptr1->id;

if(blkno == END)

{

blkno = findblock();

if(blkno == -1)

{

free(buf);

return -1;

}

fatptr1->id = blkno;

fatptr2->id = blkno;

fatptr1 = fat1 + blkno;

fatptr2 = fat2 + blkno;

fatptr1->id = END;

fatptr2->id = END;

}

else

{

fatptr1 = fat1 + blkno;

fatptr2 = fat2 + blkno;

}

blkoff = blkoff - BLOCKSIZE;

}

ll = 0;

while(ll < len)

{

blkptr = (unsigned char \*)(myvhard + blkno \* BLOCKSIZE);

for(i = 0; i < BLOCKSIZE; i++)

buf[i] = blkptr[i];

for(;blkoff < BLOCKSIZE; blkoff++)

{

buf[blkoff] = text[ll++];

openfilelist[fd].count++;

if(ll == len)

break;

}

for(i = 0; i < BLOCKSIZE; i++)

blkptr[i] = buf[i];

if(ll < len)

{

blkno = fatptr1->id;

if(blkno == END)

{

blkno = findblock();

if(blkno == -1)

break;

fatptr1->id = blkno;

fatptr2->id = blkno;

fatptr1 = fat1 + blkno;

fatptr2 = fat2 + blkno;

fatptr1->id = END;

fatptr2->id = END;

}

else

{

fatptr1 = fat1 + blkno;

fatptr2 = fat2 + blkno;

}

blkoff = 0;

}

}

if(openfilelist[fd].count > openfilelist[fd].length)

openfilelist[fd].length = openfilelist[fd].count;

openfilelist[fd].fcbstate = 1;

free(buf);

return ll;

}

int my\_read(int fd, int len)

{

char text[MAXTEXT];

int ll;

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

{

printf("The File is not exist!\n");

return -1;

}

openfilelist[fd].count = 0;

ll = do\_read(fd, len, text);

if(ll != -1)

printf("%s", text);

else

printf("Read Error!\n");

return ll;

}

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

{

fat \*fat1, \*fatptr;

unsigned char \*buf, \*blkptr;

unsigned short blkno, blkoff;

int i, ll;

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

buf = (unsigned char \*)malloc(BLOCKSIZE);

if(buf == NULL)

{

printf("malloc failed!\n");

return -1;

}

blkno = openfilelist[fd].first;

blkoff = openfilelist[fd].count;

if(blkoff >= openfilelist[fd].length)

{

puts("Read out of range!");

free(buf);

return -1;

}

fatptr = fat1 + blkno;

while(blkoff >= BLOCKSIZE)

{

blkno = fatptr->id;

blkoff = blkoff - BLOCKSIZE;

fatptr = fat1 + blkno;

}

ll = 0;

while(ll < len)

{

blkptr = (unsigned char \*)(myvhard + blkno \* BLOCKSIZE);

for(i = 0; i < BLOCKSIZE; i++)

buf[i] = blkptr[i];

for(; blkoff < BLOCKSIZE; blkoff++)

{

text[ll++] = buf[blkoff];

openfilelist[fd].count++;

if(ll == len || openfilelist[fd].count == openfilelist[fd].length)

break;

}

if(ll < len && openfilelist[fd].count != openfilelist[fd].length)

{

blkno = fatptr->id;

if(blkno == END)

break;

blkoff = 0;

fatptr = fat1 + blkno;

}

}

text[ll] = '\0';

free(buf);

return ll;

}

void my\_exitsys()

{

FILE \*fp;

while(curdir)

curdir = my\_close(curdir);

fp = fopen(myfilename, "w");

fwrite(myvhard, SIZE, 1, fp);

fclose(fp);

free(myvhard);

}

unsigned short findblock()

{

unsigned short i;

fat \*fat1, \*fatptr;

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

for(i = 7; i < SIZE / BLOCKSIZE; i++)

{

fatptr = fat1 + i;

if(fatptr->id == FREE)

return i;

}

printf("Error,Can't find free block!\n");

return -1;

}

int findopenfile()

{

int i;

for(i = 0; i < MAXTEXT; i++)

{

if(openfilelist[i].topenfile == 0)

return i;

}

printf("Error,open too many files!\n");

return -1;

}

int main()

{

char cmd[15][10] = {"cd", "mkdir", "rmdir", "ls", "create", "rm", "open", "close", "write", "read", "exit"};

char s[30], \*sp;

int cmdn, flag = 1, i;

startsys();

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*File System V1.0\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n");

printf("命令名\t\t命令参数\t\t命令说明\n\n");

printf("cd\t\t目录名(路径名)\t\t切换当前目录到指定目录\n");

printf("mkdir\t\t目录名\t\t\t在当前目录创建新目录\n");

printf("rmdir\t\t目录名\t\t\t在当前目录删除指定目录\n");

printf("ls\t\t无\t\t\t显示当前目录下的目录和文件\n");

printf("create\t\t文件名\t\t\t在当前目录下创建指定文件\n");

printf("rm\t\t文件名\t\t\t在当前目录下删除指定文件\n");

printf("open\t\t文件名\t\t\t在当前目录下打开指定文件\n");

printf("write\t\t无\t\t\t在打开文件状态下，写该文件\n");

printf("read\t\t无\t\t\t在打开文件状态下，读取该文件\n");

printf("close\t\t无\t\t\t在打开文件状态下，读取该文件\n");

printf("exit\t\t无\t\t\t退出系统\n\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n");

while(flag)

{

printf("%s>", openfilelist[curdir].dir);

gets(s);

cmdn = -1;

if(strcmp(s, ""))

{

sp=strtok(s, " ");

for(i = 0; i < 15; i++)

{

if(strcmp(sp, cmd[i]) == 0)

{

cmdn = i;

break;

}

}

// printf("%d\n", cmdn);

switch(cmdn)

{

case 0:

sp = strtok(NULL, " ");

if(sp && (openfilelist[curdir].attribute & 0x20))

my\_cd(sp);

else

printf("Please input the right command.\n");

break;

case 1:

sp = strtok(NULL, " ");

if(sp && (openfilelist[curdir].attribute & 0x20))

my\_mkdir(sp);

else

printf("Please input the right command.\n");

break;

case 2:

sp = strtok(NULL, " ");

if(sp && (openfilelist[curdir].attribute & 0x20))

my\_rmdir(sp);

else

printf("Please input the right command.\n");

break;

case 3:

if(openfilelist[curdir].attribute & 0x20)

my\_ls();

else

printf("Please input the right command.\n");

break;

case 4:

sp = strtok(NULL, " ");

if(sp && (openfilelist[curdir].attribute & 0x20))

my\_create(sp);

else

printf("Please input the right command.\n");

break;

case 5:

sp = strtok(NULL, " ");

if(sp && (openfilelist[curdir].attribute & 0x20))

my\_rm(sp);

else

printf("Please input the right command.\n");

break;

case 6:

sp = strtok(NULL, " ");

if(sp && (openfilelist[curdir].attribute & 0x20))

{

if(strchr(sp, '.'))//查找sp中'.'首次出现的位置

curdir = my\_open(sp);

else

printf("the openfile should have exname.\n");

}

else

printf("Please input the right command.\n");

break;

case 7:

if(!(openfilelist[curdir].attribute & 0x20))

curdir = my\_close(curdir);

else

printf("No files opened.\n");

break;

case 8:

if(!(openfilelist[curdir].attribute & 0x20))

my\_write(curdir);

else

printf("No files opened.\n");

break;

case 9:

if(!(openfilelist[curdir].attribute & 0x20))

my\_read(curdir, openfilelist[curdir].length);

else

printf("No files opened.\n");

break;

case 10:

if(openfilelist[curdir].attribute & 0x20)

{

my\_exitsys();

flag = 0;

}

else

printf("Please input the right command.\n");

break;

default:

printf("Please input the right command.\n");

break;

}

}

}

return 0;

}

附录三、运行结果截屏

测试流程：

使用ls查看当前目录下的文件/目录

使用mkdir新建目录haha

再次使用ls查看当前目录下的文件/目录，发现创建目录haha成功

使用create新建文件test.txt

使用ls查看当前目录下的文件/目录，发现创建文件test.txt成功

使用rm删除文件test.txt

使用ls查看当前目录下的文件/目录，发现删除文件test.txt成功

使用cd打开目录haha

成功打开

