**操作系统课程设计实验报告**

——实验四：文件系统实验

负责人姓名：韩青长

学号：14061183

日期：2016.5.24

**小组成员**

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 姓名 | 学号 | 实验分工 |
| 1 | 韩青长 | 14061183 | 源码编写 |
| 2 | 滕叶飞 | 14061189 | 分析设计 |
| 3 | 陈彦吉 | 14061195 | 文档撰写 |
| 4 | 叶青河 | 14061202 | BUG修复 |

目录

[1.实验目的 4](#_Toc446001831)

[2.需求说明 4](#_Toc446001832)

[2.1基本要求 4](#_Toc446001833)

[2.2 提高要求 4](#_Toc446001834)

[2.3 完成情况 4](#_Toc446001835)

[3.设计说明 5](#_Toc446001836)

[3.1 程序流程图 5](#_Toc446001837)

[3.2基本要求实现说明 5](#_Toc446001838)

[3.3 提高要求实现说明 5](#_Toc446001839)

[4.收获和感想 5](#_Toc446001840)

# 1.实验目的

1.了解文件管理系统的作用和工作方式。

2.学习Linux VFS的实现机制及inode、超级块等相关概念。

3.了解FAT文件系统的结构。

4.学习文件管理系统的一般开发方法。

# 2.需求说明

## 2.1基本要求

准备一个FAT16格式的U盘,在Linux下编写一个文件系统管理程序,对U盘上的文件进行管理。具体要求如下:

1.设计并实现一个目录列表函数(无须支持选项,如ls -a、ls -l等),用来显示当前目录下包含的文件信息。

2.设计并实现一个改变目录函数(无须处理路径名,如../../directoryName等),用来把当前目录切换到上一层目录或当前目录的子目录。

3.设计并实现一个删除文件函数,使用当前目录中的要删除的文件名作为参数,用来删除指定的文件,要注意文件的隐藏、只读和其他系统属性。

4.设计并实现一个创建文件函数,使用要创建的文件名和文件大小作为参数,用来创建一个新的文件。

## 2.2 提高要求

1.增加创建目录的功能。

2.增加删除目录的功能:通常需要先判断目录是否为空目录,若目录不为空,则需给出提示,并删除其包含的所有子目录和文件;若是空目录则可以直接删除。

3.增加绝对路径和多级目录的支持:这里需要对输入的目录路径字符串进行解析,然后逐级查找目录。

4.对ud\_cf()函数进行改进,使其可以向文件中写入实际内容,并根据写入的内容计算文件的实际大小。

5.对ud\_ls()函数进行改进,增加对全部非根目录信息的读取(基本要求中仅读取一个扇区的非根目录细信息)。

## 2.3 完成情况

第一周主要是阅读实验指导书和阅读源码，修复了关于时间和日期的BUG和df函数中多清除了一个目录项的BUG，完成了基本要求；第二周完成了大部分提高要求。完成了以下功能：

1.修复源码中的BUG，使基本要求得以完成。

2.添加了fd\_mkdir函数，增加了创建目录的功能。

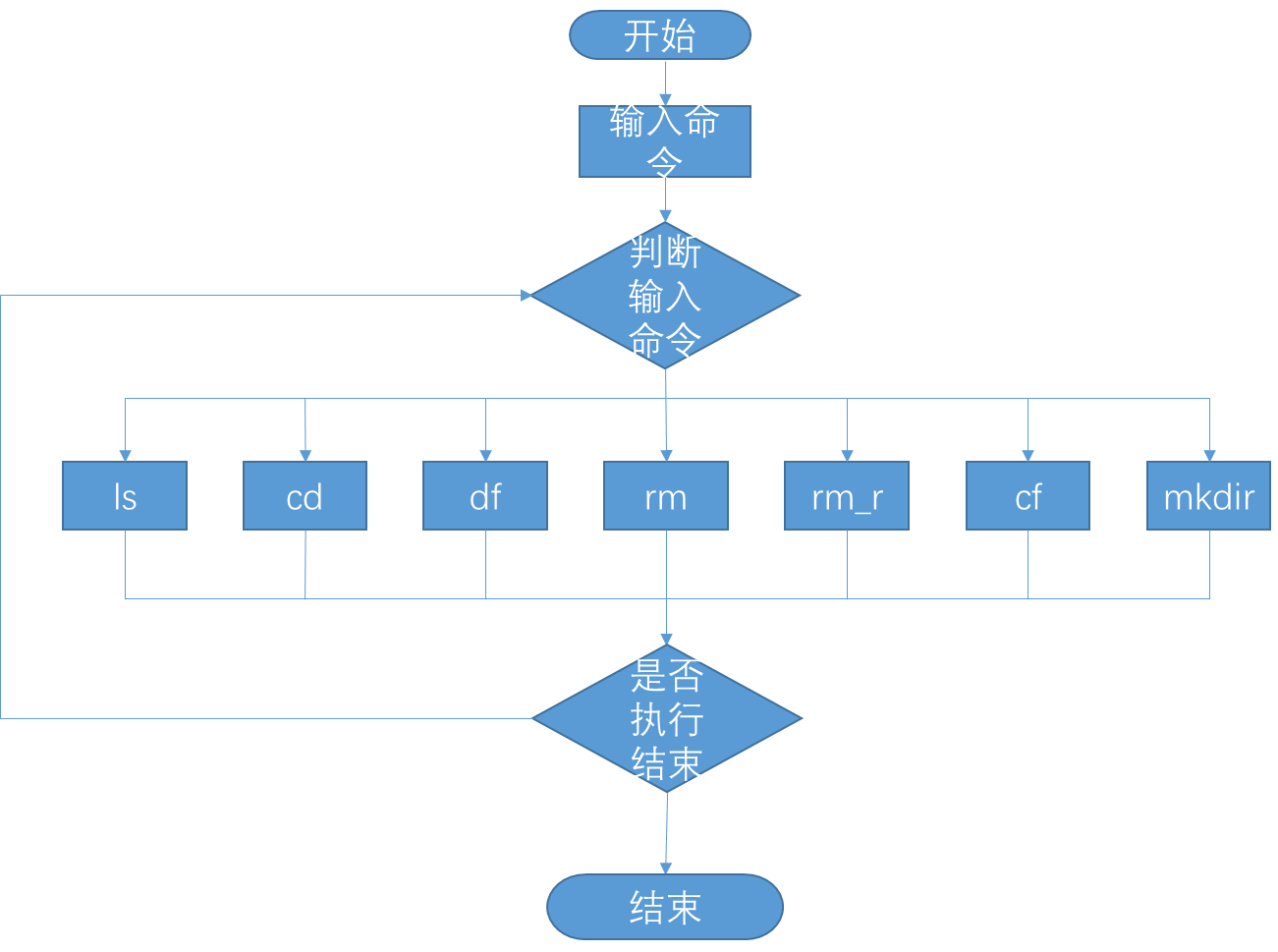
3.添加了fd\_rm函数和fd\_rm\_r函数，用rm\_dir函数辅助，增加了删除目录的功能。

4.添加了pathSwitch函数和pathBack函数，用来实现绝对路径和多级目录的功能。

5.修改了ls函数，使其可以读取全部非根目录的信息。

# 3.设计说明

## 3.1 程序流程图



## 3.2基本要求实现说明

代码：

#include<stdio.h>

#include<time.h>

#include<unistd.h>

#include<stdlib.h>

#include<sys/types.h>

#include<sys/stat.h>

#include<fcntl.h>

#include<string.h>

#include<ctype.h>

#include "filesys.h"

#define RevByte(low,high) ((high)<<8|(low))

#define RevWord(lowest,lower,higher,highest) ((highest)<< 24|(higher)<<16|(lower)<<8|lowest)

/\*

\*功能：打印启动项记录

\*/

void ScanBootSector()

{

unsigned char buf[SECTOR\_SIZE];

int ret,i;

if((ret = read(fd,buf,SECTOR\_SIZE))<0)

perror("read boot sector failed");

for(i = 0; i < 8; i++)

bdptor.Oem\_name[i] = buf[i+0x03];

bdptor.Oem\_name[i] = '\0';

bdptor.BytesPerSector = RevByte(buf[0x0b],buf[0x0c]);

bdptor.SectorsPerCluster = buf[0x0d];

bdptor.ReservedSectors = RevByte(buf[0x0e],buf[0x0f]);

bdptor.FATs = buf[0x10];

bdptor.RootDirEntries = RevByte(buf[0x11],buf[0x12]);

bdptor.LogicSectors = RevByte(buf[0x13],buf[0x14]);

bdptor.MediaType = buf[0x15];

bdptor.SectorsPerFAT = RevByte( buf[0x16],buf[0x17] );

bdptor.SectorsPerTrack = RevByte(buf[0x18],buf[0x19]);

bdptor.Heads = RevByte(buf[0x1a],buf[0x1b]);

bdptor.HiddenSectors = RevByte(buf[0x1c],buf[0x1d]);

printf("Oem\_name \t\t%s\n"

"BytesPerSector \t\t%d\n"

"SectorsPerCluster \t%d\n"

"ReservedSector \t\t%d\n"

"FATs \t\t\t%d\n"

"RootDirEntries \t\t%d\n"

"LogicSectors \t\t%d\n"

"MedioType \t\t%d\n"

"SectorPerFAT \t\t%d\n"

"SectorPerTrack \t\t%d\n"

"Heads \t\t\t%d\n"

"HiddenSectors \t\t%d\n",

bdptor.Oem\_name,

bdptor.BytesPerSector,

bdptor.SectorsPerCluster,

bdptor.ReservedSectors,

bdptor.FATs,

bdptor.RootDirEntries,

bdptor.LogicSectors,

bdptor.MediaType,

bdptor.SectorsPerFAT,

bdptor.SectorsPerTrack,

bdptor.Heads,

bdptor.HiddenSectors);

}

/\*日期\*/

void findDate(unsigned short \*year,

unsigned short \*month,

unsigned short \*day,

unsigned char info[2])

{

int date;

date = RevByte(info[0],info[1]);

\*year = ((date & MASK\_YEAR)>> 9 )+1980;

\*month = ((date & MASK\_MONTH)>> 5);

\*day = (date & MASK\_DAY);

}

/\*时间\*/

void findTime(unsigned short \*hour,

unsigned short \*min,

unsigned short \*sec,

unsigned char info[2])

{

int time;

time = RevByte(info[0],info[1]);

\*hour = ((time & MASK\_HOUR )>>11);

\*min = (time & MASK\_MIN)>> 5;

\*sec = (time & MASK\_SEC) \* 2;

}

/\*

\*文件名格式化，便于比较

\*/

void FileNameFormat(unsigned char \*name)

{

unsigned char \*p = name;

while(\*p!='\0')

p++;

p--;

while(\*p==' ')

p--;

p++;

\*p = '\0';

}

/\*参数：entry，类型：struct Entry\*

\*返回值：成功，则返回偏移值；失败：返回负值

\*功能：从根目录或文件簇中得到文件表项

\*/

int GetEntry(struct Entry \*pentry)

{

int ret,i;

int count = 0;

unsigned char buf[DIR\_ENTRY\_SIZE], info[2];

/\*读一个目录表项，即32字节\*/

if( (ret = read(fd,buf,DIR\_ENTRY\_SIZE))<0)

perror("read entry failed");

count += ret;

if(buf[0]==0xe5 || buf[0]== 0x00)

return -1\*count;

else

{

/\*长文件名，忽略掉\*/

while (buf[11]== 0x0f)

{

if((ret = read(fd,buf,DIR\_ENTRY\_SIZE))<0)

perror("read root dir failed");

count += ret;

}

/\*命名格式化，主义结尾的'\0'\*/

for (i=0 ;i<=10;i++)

pentry->short\_name[i] = buf[i];

pentry->short\_name[i] = '\0';

FileNameFormat(pentry->short\_name);

info[0]=buf[22];

info[1]=buf[23];

findTime(&(pentry->hour),&(pentry->min),&(pentry->sec),info);

info[0]=buf[24];

info[1]=buf[25];

findDate(&(pentry->year),&(pentry->month),&(pentry->day),info);

pentry->FirstCluster = RevByte(buf[26],buf[27]);

pentry->size = RevWord(buf[28],buf[29],buf[30],buf[31]);

pentry->readonly = (buf[11] & ATTR\_READONLY) ?1:0;

pentry->hidden = (buf[11] & ATTR\_HIDDEN) ?1:0;

pentry->system = (buf[11] & ATTR\_SYSTEM) ?1:0;

pentry->vlabel = (buf[11] & ATTR\_VLABEL) ?1:0;

pentry->subdir = (buf[11] & ATTR\_SUBDIR) ?1:0;

pentry->archive = (buf[11] & ATTR\_ARCHIVE) ?1:0;

return count;

}

}

/\*

\*功能：显示当前目录的内容

\*返回值：1，成功；-1，失败

\*/

int fd\_ls()

{

int ret, offset,cluster\_addr;

struct Entry entry;

unsigned char buf[DIR\_ENTRY\_SIZE];

if( (ret = read(fd,buf,DIR\_ENTRY\_SIZE))<0)

perror("read entry failed");

if(curdir==NULL)

printf("Root\_dir\n");

else

printf("%s\_dir\n",curdir->short\_name);

printf("\tname\tdate\t\t time\t\tcluster\tsize\t\tattr\n");

if(curdir==NULL) /\*显示根目录区\*/

{

/\*将fd定位到根目录区的起始地址\*/

if((ret= lseek(fd,ROOTDIR\_OFFSET,SEEK\_SET))<0)

perror("lseek ROOTDIR\_OFFSET failed");

offset = ROOTDIR\_OFFSET;

/\*从根目录区开始遍历，直到数据区起始地址\*/

while(offset < (DATA\_OFFSET))

{

ret = GetEntry(&entry);

offset += abs(ret);

if(ret > 0)

{

printf("%12s\t"

"%d:%d:%d\t"

"%d:%d:%d \t"

"%d\t"

"%d\t\t"

"%s\n",

entry.short\_name,

entry.year,entry.month,entry.day,

entry.hour,entry.min,entry.sec,

entry.FirstCluster,

entry.size,

(entry.subdir) ? "dir":"file");

}

}

}

else /\*显示子目录\*/

{

cluster\_addr = DATA\_OFFSET + (curdir->FirstCluster-2) \* CLUSTER\_SIZE ;

if((ret = lseek(fd,cluster\_addr,SEEK\_SET))<0)

perror("lseek cluster\_addr failed");

offset = cluster\_addr;

/\*只读一簇的内容\*/

while(offset<cluster\_addr +CLUSTER\_SIZE)

{

ret = GetEntry(&entry);

offset += abs(ret);

if(ret > 0)

{

printf("%12s\t"

"%d:%d:%d\t"

"%d:%d:%d \t"

"%d\t"

"%d\t\t"

"%s\n",

entry.short\_name,

entry.year,entry.month,entry.day,

entry.hour,entry.min,entry.sec,

entry.FirstCluster,

entry.size,

(entry.subdir) ? "dir":"file");

}

}

}

return 0;

}

/\*

\*参数：entryname 类型：char

：pentry 类型：struct Entry\*

：mode 类型：int，mode=1，为目录表项；mode=0，为文件

\*返回值：偏移值大于0，则成功；-1，则失败

\*功能：搜索当前目录，查找文件或目录项

\*/

int ScanEntry (char \*entryname,struct Entry \*pentry,int mode)

{

int ret,offset,i;

int cluster\_addr;

char uppername[80];

for(i=0;i< strlen(entryname);i++)

uppername[i]= toupper(entryname[i]);

uppername[i]= '\0';

/\*扫描根目录\*/

if(curdir ==NULL)

{

if((ret = lseek(fd,ROOTDIR\_OFFSET,SEEK\_SET))<0)

perror ("lseek ROOTDIR\_OFFSET failed");

offset = ROOTDIR\_OFFSET;

while(offset<DATA\_OFFSET)

{

ret = GetEntry(pentry);

offset +=abs(ret);

if(pentry->subdir == mode &&!strcmp((char\*)pentry->short\_name,uppername))

return offset;

}

return -1;

}

/\*扫描子目录\*/

else

{

cluster\_addr = DATA\_OFFSET + (curdir->FirstCluster -2)\*CLUSTER\_SIZE;

if((ret = lseek(fd,cluster\_addr,SEEK\_SET))<0)

perror("lseek cluster\_addr failed");

offset= cluster\_addr;

while(offset<cluster\_addr + CLUSTER\_SIZE)

{

ret= GetEntry(pentry);

offset += abs(ret);

if(pentry->subdir == mode &&!strcmp((char\*)pentry->short\_name,uppername))

return offset;

}

return -1;

}

}

/\*

\*参数：dir，类型：char

\*返回值：1，成功；-1，失败

\*功能：改变目录到父目录或子目录

\*/

int fd\_cd(char \*dir)

{

struct Entry \*pentry;

int ret;

if(!strcmp(dir,"."))

{

return 1;

}

if(!strcmp(dir,"..") && curdir==NULL)

return 1;

/\*返回上一级目录\*/

if(!strcmp(dir,"..") && curdir!=NULL)

{

curdir = fatherdir[dirno];

dirno--;

return 1;

}

pentry = (struct Entry\*)malloc(sizeof(struct Entry));

ret = ScanEntry(dir,pentry,1);

if(ret < 0)

{

printf("no such dir\n");

free(pentry);

return -1;

}

dirno ++;

fatherdir[dirno] = curdir;

curdir = pentry;

return 1;

}

/\*

\*参数：prev，类型：unsigned char

\*返回值：下一簇

\*在fat表中获得下一簇的位置

\*/

unsigned short GetFatCluster(unsigned short prev)

{

unsigned short next;

int index;

index = prev \* 2;

next = RevByte(fatbuf[index],fatbuf[index+1]);

return next;

}

/\*

\*参数：cluster，类型：unsigned short

\*返回值：void

\*功能：清除fat表中的簇信息

\*/

void ClearFatCluster(unsigned short cluster)

{

int index;

index = cluster \* 2;

fatbuf[index]=0x00;

fatbuf[index+1]=0x00;

}

/\*

\*将改变的fat表值写回fat表

\*/

int WriteFat()

{

if(lseek(fd,FAT\_ONE\_OFFSET,SEEK\_SET)<0)

{

perror("lseek failed");

return -1;

}

if(write(fd,fatbuf,512\*250)<0)

{

perror("read failed");

return -1;

}

if(lseek(fd,FAT\_TWO\_OFFSET,SEEK\_SET)<0)

{

perror("lseek failed");

return -1;

}

if((write(fd,fatbuf,512\*250))<0)

{

perror("read failed");

return -1;

}

return 1;

}

/\*

\*读fat表的信息，存入fatbuf[]中

\*/

int ReadFat()

{

if(lseek(fd,FAT\_ONE\_OFFSET,SEEK\_SET)<0)

{

perror("lseek failed");

return -1;

}

if(read(fd,fatbuf,512\*250)<0)

{

perror("read failed");

return -1;

}

return 1;

}

/\*

\*参数：filename，类型：char

\*返回值：1，成功；-1，失败

\*功能;删除当前目录下的文件

\*/

int fd\_df(char \*filename)

{

struct Entry \*pentry;

int ret;

unsigned char c;

unsigned short seed,next;

pentry = (struct Entry\*)malloc(sizeof(struct Entry));

/\*扫描当前目录查找文件\*/

ret = ScanEntry(filename,pentry,0);

if(ret<0)

{

printf("no such file\n");

free(pentry);

return -1;

}

/\*清除fat表项\*/

seed = pentry->FirstCluster;

while((next = GetFatCluster(seed))!=0xffff)

{

ClearFatCluster(seed);

seed = next;

}

ClearFatCluster( seed );

/\*清除目录表项\*/

c=0xe5;

if(lseek(fd,ret-0x20,SEEK\_SET)<0)

perror("lseek fd\_df failed");

if(write(fd,&c,1)<0)

perror("write failed");

// if(lseek(fd,ret-0x40,SEEK\_SET)<0)

// perror("lseek fd\_df failed");

// if(write(fd,&c,1)<0)

// perror("write failed");

free(pentry);

if(WriteFat()<0)

exit(1);

return 1;

}

/\*

\*参数：filename，类型：char，创建文件的名称

size， 类型：int，文件的大小

\*返回值：1，成功；-1，失败

\*功能：在当前目录下创建文件

\*/

int fd\_cf(char \*filename,int size)

{

time\_t timep;

struct tm \*p;

time (&timep);

p=gmtime(&timep);

struct Entry \*pentry;

int ret,i=0,cluster\_addr,offset;

unsigned short cluster,clusterno[100];

unsigned char c[DIR\_ENTRY\_SIZE];

int index,clustersize;

unsigned char buf[DIR\_ENTRY\_SIZE];

pentry = (struct Entry\*)malloc(sizeof(struct Entry));

clustersize = (size / (CLUSTER\_SIZE));

if(size % (CLUSTER\_SIZE) != 0)

clustersize ++;

//扫描根目录，是否已存在该文件名

ret = ScanEntry(filename,pentry,0);

if (ret<0)

{

/\*查询fat表，找到空白簇，保存在clusterno[]中\*/

for(cluster=2;cluster<1000;cluster++)

{

index = cluster \*2;

if(fatbuf[index]==0x00&&fatbuf[index+1]==0x00)

{

clusterno[i] = cluster;

i++;

if(i==clustersize)

break;

}

}

/\*在fat表中写入下一簇信息\*/

for(i=0;i<clustersize-1;i++)

{

index = clusterno[i]\*2;

fatbuf[index] = (clusterno[i+1] & 0x00ff);

fatbuf[index+1] = ((clusterno[i+1] & 0xff00)>>8);

}

/\*最后一簇写入0xffff\*/

index = clusterno[i]\*2;

fatbuf[index] = 0xff;

fatbuf[index+1] = 0xff;

if(curdir==NULL) /\*往根目录下写文件\*/

{

if((ret= lseek(fd,ROOTDIR\_OFFSET,SEEK\_SET))<0)

perror("lseek ROOTDIR\_OFFSET failed");

offset = ROOTDIR\_OFFSET;

while(offset < DATA\_OFFSET)

{

if((ret = read(fd,buf,DIR\_ENTRY\_SIZE))<0)

perror("read entry failed");

offset += abs(ret);

if(buf[0]!=0xe5&&buf[0]!=0x00)

{

while(buf[11] == 0x0f)

{

if((ret = read(fd,buf,DIR\_ENTRY\_SIZE))<0)

perror("read root dir failed");

offset +=abs(ret);

}

}

/\*找出空目录项或已删除的目录项\*/

else

{

offset = offset-abs(ret);

for(i=0;i<=strlen(filename);i++)

{

c[i]=toupper(filename[i]);

}

for(;i<=10;i++)

c[i]=' ';

c[11] = 0x01;

c[22] = (p->tm\_min << 5) | (p->tm\_sec / 2);

c[23] = (p->tm\_hour+8 << 3) | (p->tm\_min >> 3);

c[24] = ((p->tm\_mon+1) << 5) | p->tm\_mday;

c[25] = ((p->tm\_year-80) << 1) | ((p->tm\_mon+1) >> 3);

/\*写第一簇的值\*/

c[26] = (clusterno[0] & 0x00ff);

c[27] = ((clusterno[0] & 0xff00)>>8);

/\*写文件的大小\*/

c[28] = (size & 0x000000ff);

c[29] = ((size & 0x0000ff00)>>8);

c[30] = ((size& 0x00ff0000)>>16);

c[31] = ((size& 0xff000000)>>24);

if(lseek(fd,offset,SEEK\_SET)<0)

perror("lseek fd\_cf failed");

if(write(fd,&c,DIR\_ENTRY\_SIZE)<0)

perror("write failed");

free(pentry);

if(WriteFat()<0)

exit(1);

return 1;

}

}

}

else

{

cluster\_addr = (curdir->FirstCluster -2 )\*CLUSTER\_SIZE + DATA\_OFFSET;

if((ret= lseek(fd,cluster\_addr,SEEK\_SET))<0)

perror("lseek cluster\_addr failed");

offset = cluster\_addr;

while(offset < cluster\_addr + CLUSTER\_SIZE)

{

if((ret = read(fd,buf,DIR\_ENTRY\_SIZE))<0)

perror("read entry failed");

offset += abs(ret);

if(buf[0]!=0xe5&&buf[0]!=0x00)

{

while(buf[11] == 0x0f)

{

if((ret = read(fd,buf,DIR\_ENTRY\_SIZE))<0)

perror("read root dir failed");

offset +=abs(ret);

}

}

else

{

offset = offset - abs(ret);

for(i=0;i<=strlen(filename);i++)

{

c[i]=toupper(filename[i]);

}

for(;i<=10;i++)

c[i]=' ';

c[11] = 0x01;

c[22] = (p->tm\_min << 5) | (p->tm\_sec / 2);

c[23] = (p->tm\_hour+8 << 3) | (p->tm\_min >> 3);

c[24] = ((p->tm\_mon+1) << 5) | p->tm\_mday;

c[25] = ((p->tm\_year-80) << 1) | ((p->tm\_mon+1) >> 3);

c[26] = (clusterno[0] & 0x00ff);

c[27] = ((clusterno[0] & 0xff00)>>8);

c[28] = (size & 0x000000ff);

c[29] = ((size & 0x0000ff00)>>8);

c[30] = ((size& 0x00ff0000)>>16);

c[31] = ((size& 0xff000000)>>24);

if(lseek(fd,offset,SEEK\_SET)<0)

perror("lseek fd\_cf failed");

if(write(fd,&c,DIR\_ENTRY\_SIZE)<0)

perror("write failed");

free(pentry);

if(WriteFat()<0)

exit(1);

return 1;

}

}

}

}

else

{

printf("This filename is exist\n");

free(pentry);

return -1;

}

return 1;

}

void do\_usage()

{

printf("please input a command, including followings:\n\tls\t\t\tlist all files\n\tcd <dir>\t\tchange direcotry\n\tcf <filename> <size>\tcreate a file\n\tmkdir <dir>\t\tcreate a dir\n\tdf <file>\t\tdelete a file\n\t");

printf("rm <dir>\t\tdelete a dir with hint\n\trm -r <dir>\t\tdelete a dir\n\texit\t\t\texit this system\n");

}

int main()

{

char input[10];

int size=0, num = 0;

char name[12];

if((fd = open(DEVNAME,O\_RDWR))<0)

perror("open failed");

ScanBootSector();

if(ReadFat()<0)

exit(1);

do\_usage();

while (1)

{

printf(">");

scanf("%s",input);

if (strcmp(input, "exit") == 0)

break;

else if (strcmp(input, "ls") == 0)

fd\_ls();

else if(strcmp(input, "cd") == 0)

{

scanf("%s", name);

fd\_cd(name);

}

else if(strcmp(input, "df") == 0)

{

scanf("%s", name);

fd\_df(name);

}

else if(strcmp(input, "cf") == 0)

{

scanf("%s", name);

scanf("%s", input);

size = atoi(input);

fd\_cf(name, size);

}

else

do\_usage();

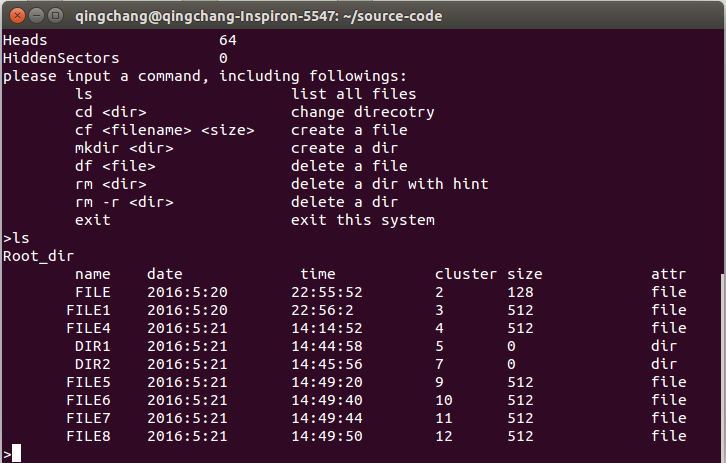
}

return 0;

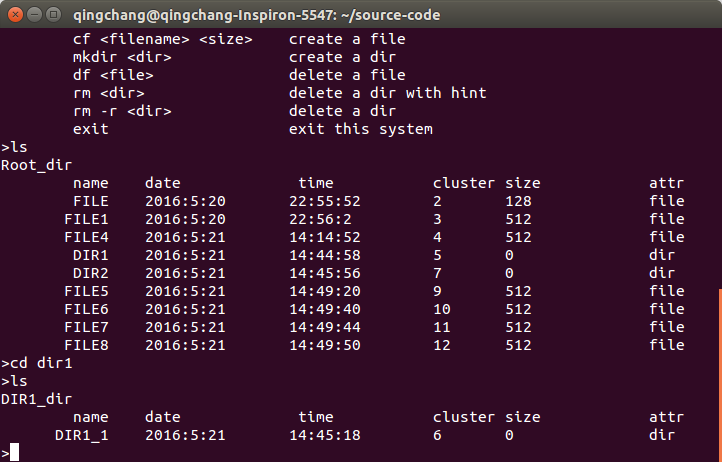
}

实现步骤截图：

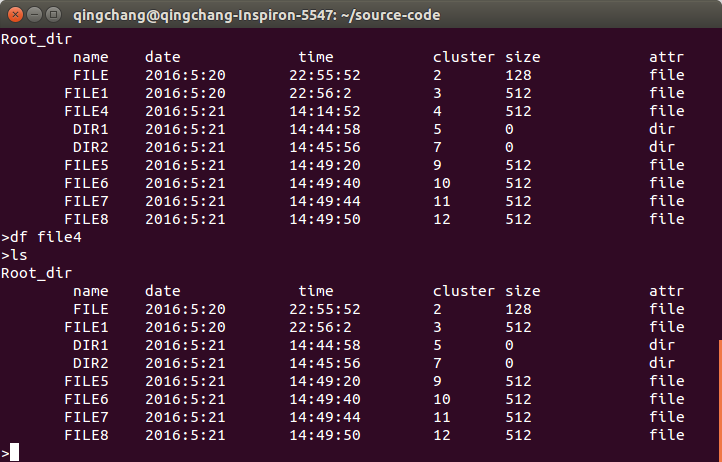
1. ls命令



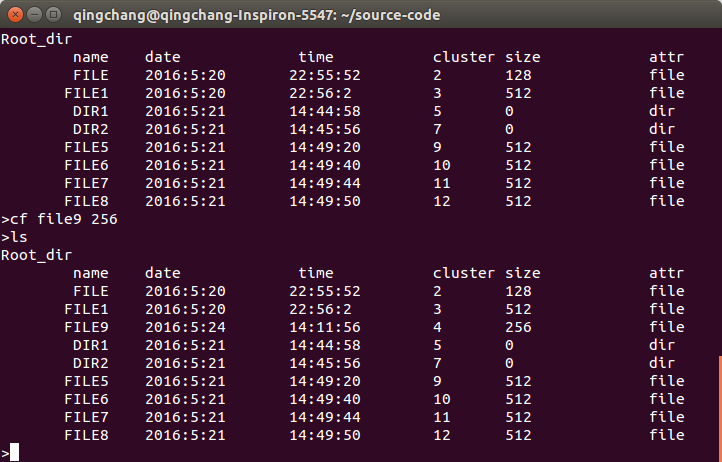
1. cd命令



1. df命令



1. cf命令



## 3.3 提高要求实现说明

#include<stdio.h>

#include<time.h>

#include<unistd.h>

#include<stdlib.h>

#include<sys/types.h>

#include<sys/stat.h>

#include<fcntl.h>

#include<string.h>

#include<ctype.h>

#include "filesys.h"

#define RevByte(low,high) ((high)<<8|(low))

#define RevWord(lowest,lower,higher,highest) ((highest)<< 24|(higher)<<16|(lower)<<8|lowest)

/\*

\*功能：打印启动项记录

\*/

void ScanBootSector()

{

unsigned char buf[SECTOR\_SIZE];

int ret,i;

if((ret = read(fd,buf,SECTOR\_SIZE))<0)

perror("read boot sector failed");

for(i = 0; i < 8; i++)

bdptor.Oem\_name[i] = buf[i+0x03];

bdptor.Oem\_name[i] = '\0';

bdptor.BytesPerSector = RevByte(buf[0x0b],buf[0x0c]);

bdptor.SectorsPerCluster = buf[0x0d];

bdptor.ReservedSectors = RevByte(buf[0x0e],buf[0x0f]);

bdptor.FATs = buf[0x10];

bdptor.RootDirEntries = RevByte(buf[0x11],buf[0x12]);

bdptor.LogicSectors = RevByte(buf[0x13],buf[0x14]);

bdptor.MediaType = buf[0x15];

bdptor.SectorsPerFAT = RevByte( buf[0x16],buf[0x17] );

bdptor.SectorsPerTrack = RevByte(buf[0x18],buf[0x19]);

bdptor.Heads = RevByte(buf[0x1a],buf[0x1b]);

bdptor.HiddenSectors = RevByte(buf[0x1c],buf[0x1d]);

printf("Oem\_name \t\t%s\n"

"BytesPerSector \t\t%d\n"

"SectorsPerCluster \t%d\n"

"ReservedSector \t\t%d\n"

"FATs \t\t\t%d\n"

"RootDirEntries \t\t%d\n"

"LogicSectors \t\t%d\n"

"MedioType \t\t%d\n"

"SectorPerFAT \t\t%d\n"

"SectorPerTrack \t\t%d\n"

"Heads \t\t\t%d\n"

"HiddenSectors \t\t%d\n",

bdptor.Oem\_name,

bdptor.BytesPerSector,

bdptor.SectorsPerCluster,

bdptor.ReservedSectors,

bdptor.FATs,

bdptor.RootDirEntries,

bdptor.LogicSectors,

bdptor.MediaType,

bdptor.SectorsPerFAT,

bdptor.SectorsPerTrack,

bdptor.Heads,

bdptor.HiddenSectors);

}

/\*日期\*/

void findDate(unsigned short \*year,

unsigned short \*month,

unsigned short \*day,

unsigned char info[2])

{

int date;

date = RevByte(info[0],info[1]);

\*year = ((date & MASK\_YEAR)>> 9 )+1980;

\*month = ((date & MASK\_MONTH)>> 5);

\*day = (date & MASK\_DAY);

}

/\*时间\*/

void findTime(unsigned short \*hour,

unsigned short \*min,

unsigned short \*sec,

unsigned char info[2])

{

int time;

time = RevByte(info[0],info[1]);

\*hour = ((time & MASK\_HOUR )>>11);

\*min = (time & MASK\_MIN)>> 5;

\*sec = (time & MASK\_SEC) \* 2;

}

/\*

\*文件名格式化，便于比较

\*/

void FileNameFormat(unsigned char \*name)

{

unsigned char \*p = name;

while(\*p!='\0')

p++;

p--;

while(\*p==' ')

p--;

p++;

\*p = '\0';

}

/\*参数：entry，类型：struct Entry\*

\*返回值：成功，则返回偏移值；失败：返回负值

\*功能：从根目录或文件簇中得到文件表项

\*/

int GetEntry(struct Entry \*pentry)

{

int ret,i;

int count = 0;

unsigned char buf[DIR\_ENTRY\_SIZE], info[2];

/\*读一个目录表项，即32字节\*/

if( (ret = read(fd,buf,DIR\_ENTRY\_SIZE))<0)

perror("read entry failed");

count += ret;

if(buf[0]==0xe5 || buf[0]== 0x00)

return -1\*count;

else

{

/\*长文件名，忽略掉\*/

while (buf[11]== 0x0f)

{

if((ret = read(fd,buf,DIR\_ENTRY\_SIZE))<0)

perror("read root dir failed");

count += ret;

}

/\*命名格式化，主义结尾的'\0'\*/

for (i=0 ;i<=10;i++)

pentry->short\_name[i] = buf[i];

pentry->short\_name[i] = '\0';

FileNameFormat(pentry->short\_name);

info[0]=buf[22];

info[1]=buf[23];

findTime(&(pentry->hour),&(pentry->min),&(pentry->sec),info);

info[0]=buf[24];

info[1]=buf[25];

findDate(&(pentry->year),&(pentry->month),&(pentry->day),info);

pentry->FirstCluster = RevByte(buf[26],buf[27]);

pentry->size = RevWord(buf[28],buf[29],buf[30],buf[31]);

pentry->readonly = (buf[11] & ATTR\_READONLY) ?1:0;

pentry->hidden = (buf[11] & ATTR\_HIDDEN) ?1:0;

pentry->system = (buf[11] & ATTR\_SYSTEM) ?1:0;

pentry->vlabel = (buf[11] & ATTR\_VLABEL) ?1:0;

pentry->subdir = (buf[11] & ATTR\_SUBDIR) ?1:0;

pentry->archive = (buf[11] & ATTR\_ARCHIVE) ?1:0;

return count;

}

}

/\*

\*功能：显示当前目录的内容

\*返回值：1，成功；-1，失败

\*/

int fd\_ls()

{

int ret, offset,cluster\_addr;

struct Entry entry;

unsigned char buf[DIR\_ENTRY\_SIZE];

if( (ret = read(fd,buf,DIR\_ENTRY\_SIZE))<0)

perror("read entry failed");

if(curdir==NULL)

printf("Root\_dir\n");

else

printf("%s\_dir\n",curdir->short\_name);

printf("\tname\tdate\t\t time\t\tcluster\tsize\t\tattr\n");

if(curdir==NULL) /\*显示根目录区\*/

{

/\*将fd定位到根目录区的起始地址\*/

if((ret= lseek(fd,ROOTDIR\_OFFSET,SEEK\_SET))<0)

perror("lseek ROOTDIR\_OFFSET failed");

offset = ROOTDIR\_OFFSET;

/\*从根目录区开始遍历，直到数据区起始地址\*/

while(offset < (DATA\_OFFSET))

{

ret = GetEntry(&entry);

offset += abs(ret);

if(ret > 0)

{

printf("%12s\t"

"%d:%d:%d\t"

"%d:%d:%d \t"

"%d\t"

"%d\t\t"

"%s\n",

entry.short\_name,

entry.year,entry.month,entry.day,

entry.hour,entry.min,entry.sec,

entry.FirstCluster,

entry.size,

(entry.subdir) ? "dir":"file");

}

}

}

else /\*显示子目录\*/

{

cluster\_addr = DATA\_OFFSET + (curdir->FirstCluster-2) \* CLUSTER\_SIZE ;

if((ret = lseek(fd,cluster\_addr,SEEK\_SET))<0)

perror("lseek cluster\_addr failed");

offset = cluster\_addr;

/\*只读一簇的内容\*/

while(offset<cluster\_addr +CLUSTER\_SIZE)

{

ret = GetEntry(&entry);

offset += abs(ret);

if(ret > 0)

{

printf("%12s\t"

"%d:%d:%d\t"

"%d:%d:%d \t"

"%d\t"

"%d\t\t"

"%s\n",

entry.short\_name,

entry.year,entry.month,entry.day,

entry.hour,entry.min,entry.sec,

entry.FirstCluster,

entry.size,

(entry.subdir) ? "dir":"file");

}

}

}

return 0;

}

/\*

\*参数：entryname 类型：char

：pentry 类型：struct Entry\*

：mode 类型：int，mode=1，为目录表项；mode=0，为文件

\*返回值：偏移值大于0，则成功；-1，则失败

\*功能：搜索当前目录，查找文件或目录项

\*/

int ScanEntry (char \*entryname,struct Entry \*pentry,int mode)

{

int ret,offset,i;

int cluster\_addr;

char uppername[80];

for(i=0;i< strlen(entryname);i++)

uppername[i]= toupper(entryname[i]);

uppername[i]= '\0';

/\*扫描根目录\*/

if(curdir ==NULL)

{

if((ret = lseek(fd,ROOTDIR\_OFFSET,SEEK\_SET))<0)

perror ("lseek ROOTDIR\_OFFSET failed");

offset = ROOTDIR\_OFFSET;

while(offset<DATA\_OFFSET)

{

ret = GetEntry(pentry);

offset +=abs(ret);

if(pentry->subdir == mode &&!strcmp((char\*)pentry->short\_name,uppername))

return offset;

}

return -1;

}

/\*扫描子目录\*/

else

{

cluster\_addr = DATA\_OFFSET + (curdir->FirstCluster -2)\*CLUSTER\_SIZE;

if((ret = lseek(fd,cluster\_addr,SEEK\_SET))<0)

perror("lseek cluster\_addr failed");

offset= cluster\_addr;

while(offset<cluster\_addr + CLUSTER\_SIZE)

{

ret= GetEntry(pentry);

offset += abs(ret);

if(pentry->subdir == mode &&!strcmp((char\*)pentry->short\_name,uppername))

return offset;

}

return -1;

}

}

/\*

\*参数：dir，类型：char

\*返回值：1，成功；-1，失败

\*功能：改变目录到父目录或子目录

\*/

int fd\_cd(char \*dir)

{

struct Entry \*pentry;

int ret;

if(!strcmp(dir,"."))

{

return 1;

}

if(!strcmp(dir,"..") && curdir==NULL)

return 1;

/\*返回上一级目录\*/

if(!strcmp(dir,"..") && curdir!=NULL)

{

curdir = fatherdir[dirno];

dirno--;

return 1;

}

pentry = (struct Entry\*)malloc(sizeof(struct Entry));

ret = ScanEntry(dir,pentry,1);

if(ret < 0)

{

printf("no such dir\n");

free(pentry);

return -1;

}

dirno ++;

fatherdir[dirno] = curdir;

curdir = pentry;

return 1;

}

/\*

\*参数：prev，类型：unsigned char

\*返回值：下一簇

\*在fat表中获得下一簇的位置

\*/

unsigned short GetFatCluster(unsigned short prev)

{

unsigned short next;

int index;

index = prev \* 2;

next = RevByte(fatbuf[index],fatbuf[index+1]);

return next;

}

/\*

\*参数：cluster，类型：unsigned short

\*返回值：void

\*功能：清除fat表中的簇信息

\*/

void ClearFatCluster(unsigned short cluster)

{

int index;

index = cluster \* 2;

fatbuf[index]=0x00;

fatbuf[index+1]=0x00;

}

/\*

\*将改变的fat表值写回fat表

\*/

int WriteFat()

{

if(lseek(fd,FAT\_ONE\_OFFSET,SEEK\_SET)<0)

{

perror("lseek failed");

return -1;

}

if(write(fd,fatbuf,512\*250)<0)

{

perror("read failed");

return -1;

}

if(lseek(fd,FAT\_TWO\_OFFSET,SEEK\_SET)<0)

{

perror("lseek failed");

return -1;

}

if((write(fd,fatbuf,512\*250))<0)

{

perror("read failed");

return -1;

}

return 1;

}

/\*

\*读fat表的信息，存入fatbuf[]中

\*/

int ReadFat()

{

if(lseek(fd,FAT\_ONE\_OFFSET,SEEK\_SET)<0)

{

perror("lseek failed");

return -1;

}

if(read(fd,fatbuf,512\*250)<0)

{

perror("read failed");

return -1;

}

return 1;

}

/\*

\*参数：filename，类型：char

\*返回值：1，成功；-1，失败

\*功能;删除当前目录下的文件

\*/

int fd\_df(char \*filename)

{

struct Entry \*pentry;

int ret;

unsigned char c;

unsigned short seed,next;

pentry = (struct Entry\*)malloc(sizeof(struct Entry));

/\*扫描当前目录查找文件\*/

ret = ScanEntry(filename,pentry,0);

if(ret<0)

{

printf("no such file\n");

free(pentry);

return -1;

}

/\*清除fat表项\*/

seed = pentry->FirstCluster;

while((next = GetFatCluster(seed))!=0xffff)

{

ClearFatCluster(seed);

seed = next;

}

ClearFatCluster( seed );

/\*清除目录表项\*/

c=0xe5;

if(lseek(fd,ret-0x20,SEEK\_SET)<0)

perror("lseek fd\_df failed");

if(write(fd,&c,1)<0)

perror("write failed");

// if(lseek(fd,ret-0x40,SEEK\_SET)<0)

// perror("lseek fd\_df failed");

// if(write(fd,&c,1)<0)

// perror("write failed");

free(pentry);

if(WriteFat()<0)

exit(1);

return 1;

}

void initEntry(struct Entry \*p)

{

int i;

for(i = 0; i < 12; i++)

{

p->short\_name[i] = 0x00;

}

p->year = 0;

p->month = 0;

p->day = 0;

p->hour = 0;

p->min = 0;

p->sec = 0;

p->FirstCluster = 0;

p->size = 0;

p->readonly = 0x00;

p->hidden = 0x00;

p->system = 0x00;

p->vlabel = 0x00;

p->subdir = 0x00;

p->archive = 0x00;

}

int rm\_dir(struct Entry \*pentry)

{

int cluster\_addr, ret, offset, i;

unsigned char buf[DIR\_ENTRY\_SIZE];

struct Entry \*p;

unsigned char c;

unsigned short seed,next;

cluster\_addr = (pentry->FirstCluster -2 )\*CLUSTER\_SIZE + DATA\_OFFSET;

if((ret= lseek(fd,cluster\_addr,SEEK\_SET))<0)

perror("lseek cluster\_addr failed");

offset = cluster\_addr;

while(offset < cluster\_addr + CLUSTER\_SIZE)

{

p = (struct Entry\*)malloc(sizeof(struct Entry));

initEntry(p);

ret = GetEntry(p);

offset += abs(ret);

if(p->short\_name[0] == 0xe5 || p->short\_name[0] == 0x00)

continue;

/\*clear FAT\*/

seed = p->FirstCluster;

while((next = GetFatCluster(seed))!=0xffff)

{

ClearFatCluster(seed);

seed = next;

}

ClearFatCluster( seed );

/\*清除目录表项\*/

c=0xe5;

if(lseek(fd,offset-0x20,SEEK\_SET)<0)

perror("lseek rm\_dir failed");

if(write(fd,&c,1)<0)

perror("write failed");

if(p->subdir == 1)

{

rm\_dir(p);

}

free(p);

if(lseek(fd,offset,SEEK\_SET)<0)

perror("lseek rm\_dir failed");

}

return 1;

}

int fd\_rm\_r(char \*dirname)

{

struct Entry \*pentry;

int ret, offset, cluster\_addr, ifEmpty = 1;

char input[5];

unsigned char c;

unsigned short seed,next;

unsigned char buf[DIR\_ENTRY\_SIZE];

pentry = (struct Entry\*)malloc(sizeof(struct Entry));

ret = ScanEntry(dirname,pentry,1);

if(ret<0)

{

printf("no such dir\n");

free(pentry);

return -1;

}

/\*clear FAT\*/

seed = pentry->FirstCluster;

while((next = GetFatCluster(seed))!=0xffff)

{

ClearFatCluster(seed);

seed = next;

}

ClearFatCluster( seed );

/\*清除目录表项\*/

c=0xe5;

if(lseek(fd,ret-0x20,SEEK\_SET)<0)

perror("lseek fd\_rm failed");

if(write(fd,&c,1)<0)

perror("write failed");

rm\_dir(pentry);

free(pentry);

if(WriteFat()<0)

exit(1);

return 1;

}

int fd\_rm(char \*dirname)

{

struct Entry \*pentry;

int ret, offset, cluster\_addr, temp, ifEmpty = 1;

char input[5];

unsigned char c;

unsigned short seed,next;

unsigned char buf[DIR\_ENTRY\_SIZE];

pentry = (struct Entry\*)malloc(sizeof(struct Entry));

ret = ScanEntry(dirname,pentry,1);

if(ret<0)

{

printf("no such dir\n");

free(pentry);

return -1;

}

temp = ret;

cluster\_addr = (pentry->FirstCluster -2 )\*CLUSTER\_SIZE + DATA\_OFFSET;

if((ret= lseek(fd,cluster\_addr,SEEK\_SET))<0)

perror("lseek cluster\_addr failed");

offset = cluster\_addr;

while(offset < cluster\_addr + CLUSTER\_SIZE)

{

if((ret = read(fd,buf,DIR\_ENTRY\_SIZE))<0)

perror("read entry failed");

offset += abs(ret);

if(buf[0] != 0xe5 && buf[0] != 0x00)

{

ifEmpty = 0;

break;

}

}

if(ifEmpty == 1)

{

printf("空目录，直接删除\n");

seed = pentry->FirstCluster;

while((next = GetFatCluster(seed))!=0xffff)

{

ClearFatCluster(seed);

seed = next;

}

ClearFatCluster( seed );

/\*清除目录表项\*/

c=0xe5;

if(lseek(fd,temp-0x20,SEEK\_SET)<0)

perror("lseek fd\_rm failed");

if(write(fd,&c,1)<0)

perror("write failed");

free(pentry);

if(WriteFat()<0)

exit(1);

return 1;

}

else

{

printf("确定删除%s目录及其包含的所有子目录和文件吗？y/n\t", dirname);

scanf("%s", input);

while(strcmp(input, "y") != 0 && strcmp(input, "Y") != 0 && strcmp(input, "n") != 0 && strcmp(input, "N") != 0)

{

printf("输入错误，请重新输入\t");

scanf("%s", input);

}

if(strcmp(input, "y") == 0 || strcmp(input, "Y") == 0)

{

seed = pentry->FirstCluster;

while((next = GetFatCluster(seed))!=0xffff)

{

ClearFatCluster(seed);

seed = next;

}

ClearFatCluster( seed );

/\*清除目录表项\*/

c=0xe5;

if(lseek(fd,temp-0x20,SEEK\_SET)<0)

perror("lseek fd\_rm failed");

if(write(fd,&c,1)<0)

perror("write failed");

rm\_dir(pentry) ;

free(pentry);

if(WriteFat()<0)

exit(1);

return 1;

}

else

{

printf("删除失败\n");

free(pentry);

return -1;

}

}

}

/\*

\*参数：filename，类型：char，创建文件的名称

size， 类型：int，文件的大小

\*返回值：1，成功；-1，失败

\*功能：在当前目录下创建文件

\*/

int fd\_cf(char \*filename,int size)

{

time\_t timep;

struct tm \*p;

time (&timep);

p=gmtime(&timep);

struct Entry \*pentry;

int ret,i=0,cluster\_addr,offset;

unsigned short cluster,clusterno[100];

unsigned char c[DIR\_ENTRY\_SIZE];

int index,clustersize;

unsigned char buf[DIR\_ENTRY\_SIZE];

pentry = (struct Entry\*)malloc(sizeof(struct Entry));

clustersize = (size / (CLUSTER\_SIZE));

if(size % (CLUSTER\_SIZE) != 0)

clustersize ++;

//扫描根目录，是否已存在该文件名

ret = ScanEntry(filename,pentry,0);

if (ret<0)

{

/\*查询fat表，找到空白簇，保存在clusterno[]中\*/

for(cluster=2;cluster<1000;cluster++)

{

index = cluster \*2;

if(fatbuf[index]==0x00&&fatbuf[index+1]==0x00)

{

clusterno[i] = cluster;

i++;

if(i==clustersize)

break;

}

}

/\*在fat表中写入下一簇信息\*/

for(i=0;i<clustersize-1;i++)

{

index = clusterno[i]\*2;

fatbuf[index] = (clusterno[i+1] & 0x00ff);

fatbuf[index+1] = ((clusterno[i+1] & 0xff00)>>8);

}

/\*最后一簇写入0xffff\*/

index = clusterno[i]\*2;

fatbuf[index] = 0xff;

fatbuf[index+1] = 0xff;

if(curdir==NULL) /\*往根目录下写文件\*/

{

if((ret= lseek(fd,ROOTDIR\_OFFSET,SEEK\_SET))<0)

perror("lseek ROOTDIR\_OFFSET failed");

offset = ROOTDIR\_OFFSET;

while(offset < DATA\_OFFSET)

{

if((ret = read(fd,buf,DIR\_ENTRY\_SIZE))<0)

perror("read entry failed");

offset += abs(ret);

if(buf[0]!=0xe5&&buf[0]!=0x00)

{

while(buf[11] == 0x0f)

{

if((ret = read(fd,buf,DIR\_ENTRY\_SIZE))<0)

perror("read root dir failed");

offset +=abs(ret);

}

}

/\*找出空目录项或已删除的目录项\*/

else

{

offset = offset-abs(ret);

for(i=0;i<=strlen(filename);i++)

{

c[i]=toupper(filename[i]);

}

for(;i<=10;i++)

c[i]=' ';

c[11] = 0x01;

c[22] = (p->tm\_min << 5) | (p->tm\_sec / 2);

c[23] = (p->tm\_hour+8 << 3) | (p->tm\_min >> 3);

c[24] = ((p->tm\_mon+1) << 5) | p->tm\_mday;

c[25] = ((p->tm\_year-80) << 1) | ((p->tm\_mon+1) >> 3);

/\*写第一簇的值\*/

c[26] = (clusterno[0] & 0x00ff);

c[27] = ((clusterno[0] & 0xff00)>>8);

/\*写文件的大小\*/

c[28] = (size & 0x000000ff);

c[29] = ((size & 0x0000ff00)>>8);

c[30] = ((size& 0x00ff0000)>>16);

c[31] = ((size& 0xff000000)>>24);

if(lseek(fd,offset,SEEK\_SET)<0)

perror("lseek fd\_cf failed");

if(write(fd,&c,DIR\_ENTRY\_SIZE)<0)

perror("write failed");

free(pentry);

if(WriteFat()<0)

exit(1);

return 1;

}

}

}

else

{

cluster\_addr = (curdir->FirstCluster -2 )\*CLUSTER\_SIZE + DATA\_OFFSET;

if((ret= lseek(fd,cluster\_addr,SEEK\_SET))<0)

perror("lseek cluster\_addr failed");

offset = cluster\_addr;

while(offset < cluster\_addr + CLUSTER\_SIZE)

{

if((ret = read(fd,buf,DIR\_ENTRY\_SIZE))<0)

perror("read entry failed");

offset += abs(ret);

if(buf[0]!=0xe5&&buf[0]!=0x00)

{

while(buf[11] == 0x0f)

{

if((ret = read(fd,buf,DIR\_ENTRY\_SIZE))<0)

perror("read root dir failed");

offset +=abs(ret);

}

}

else

{

offset = offset - abs(ret);

for(i=0;i<=strlen(filename);i++)

{

c[i]=toupper(filename[i]);

}

for(;i<=10;i++)

c[i]=' ';

c[11] = 0x01;

c[22] = (p->tm\_min << 5) | (p->tm\_sec / 2);

c[23] = (p->tm\_hour+8 << 3) | (p->tm\_min >> 3);

c[24] = ((p->tm\_mon+1) << 5) | p->tm\_mday;

c[25] = ((p->tm\_year-80) << 1) | ((p->tm\_mon+1) >> 3);

c[26] = (clusterno[0] & 0x00ff);

c[27] = ((clusterno[0] & 0xff00)>>8);

c[28] = (size & 0x000000ff);

c[29] = ((size & 0x0000ff00)>>8);

c[30] = ((size& 0x00ff0000)>>16);

c[31] = ((size& 0xff000000)>>24);

if(lseek(fd,offset,SEEK\_SET)<0)

perror("lseek fd\_cf failed");

if(write(fd,&c,DIR\_ENTRY\_SIZE)<0)

perror("write failed");

free(pentry);

if(WriteFat()<0)

exit(1);

return 1;

}

}

}

}

else

{

printf("This filename is exist\n");

free(pentry);

return -1;

}

return 1;

}

void clear(int clusterno)

{

int cluster\_addr, ret, offset;

unsigned char c;

c = 0xe5;

cluster\_addr = (clusterno -2 )\*CLUSTER\_SIZE + DATA\_OFFSET;

if((ret= lseek(fd,cluster\_addr,SEEK\_SET))<0)

perror("lseek cluster\_addr failed");

offset = cluster\_addr;

while(offset < cluster\_addr + CLUSTER\_SIZE)

{

if(lseek(fd,offset,SEEK\_SET)<0)

perror("lseek fd\_df failed");

if(write(fd,&c,1)<0)

perror("write failed");

offset += 0x20;

}

}

int fd\_mkdir(char \*dirname)

{

time\_t timep;

struct tm \*p;

time (&timep);

p=gmtime(&timep);

struct Entry \*pentry;

int ret,i=0,cluster\_addr,offset;

unsigned short cluster,clusterno;

unsigned char c[DIR\_ENTRY\_SIZE];

int index;

unsigned char buf[DIR\_ENTRY\_SIZE];

pentry = (struct Entry\*)malloc(sizeof(struct Entry));

ret = ScanEntry(dirname,pentry,1);

if(ret < 0)

{

for(cluster=2;cluster<1000;cluster++)

{

index = cluster \*2;

if(fatbuf[index]==0x00&&fatbuf[index+1]==0x00)

{

clusterno = cluster;

break;

}

}

fatbuf[index] = 0xff;

fatbuf[index+1] = 0xff;

if(curdir==NULL)

{

if((ret= lseek(fd,ROOTDIR\_OFFSET,SEEK\_SET))<0)

perror("lseek ROOTDIR\_OFFSET failed");

offset = ROOTDIR\_OFFSET;

while(offset < DATA\_OFFSET)

{

if((ret = read(fd,buf,DIR\_ENTRY\_SIZE))<0)

perror("read entry failed");

offset += abs(ret);

if(buf[0]!=0xe5&&buf[0]!=0x00)

{

while(buf[11] == 0x0f)

{

if((ret = read(fd,buf,DIR\_ENTRY\_SIZE))<0)

perror("read root dir failed");

offset +=abs(ret);

}

}

else

{

offset = offset-abs(ret);

for(i=0;i<=strlen(dirname);i++)

{

c[i]=toupper(dirname[i]);

}

for(;i<=10;i++)

c[i]=' ';

c[11] = 0x10;

c[22] = (p->tm\_min << 5) | (p->tm\_sec / 2);

c[23] = (p->tm\_hour+8 << 3) | (p->tm\_min >> 3);

c[24] = ((p->tm\_mon+1) << 5) | p->tm\_mday;

c[25] = ((p->tm\_year-80) << 1) | ((p->tm\_mon+1) >> 3);

c[26] = (clusterno & 0x00ff);

c[27] = ((clusterno & 0xff00)>>8);

if(lseek(fd,offset,SEEK\_SET)<0)

perror("lseek fd\_mkdir failed");

if(write(fd,&c,DIR\_ENTRY\_SIZE)<0)

perror("write failed");

clear(clusterno);

free(pentry);

if(WriteFat()<0)

exit(1);

return 1;

}

}

}

else

{

cluster\_addr = (curdir->FirstCluster -2 )\*CLUSTER\_SIZE + DATA\_OFFSET;

if((ret= lseek(fd,cluster\_addr,SEEK\_SET))<0)

perror("lseek cluster\_addr failed");

offset = cluster\_addr;

while(offset < cluster\_addr + CLUSTER\_SIZE)

{

if((ret = read(fd,buf,DIR\_ENTRY\_SIZE))<0)

perror("read entry failed");

offset += abs(ret);

if(buf[0]!=0xe5&&buf[0]!=0x00)

{

while(buf[11] == 0x0f)

{

if((ret = read(fd,buf,DIR\_ENTRY\_SIZE))<0)

perror("read root dir failed");

offset +=abs(ret);

}

}

else

{

offset = offset - abs(ret);

for(i=0;i<=strlen(dirname);i++)

{

c[i]=toupper(dirname[i]);

}

for(;i<=10;i++)

c[i]=' ';

c[11] = 0x10;

c[22] = (p->tm\_min << 5) | (p->tm\_sec / 2);

c[23] = (p->tm\_hour+8 << 3) | (p->tm\_min >> 3);

c[24] = ((p->tm\_mon+1) << 5) | p->tm\_mday;

c[25] = ((p->tm\_year-80) << 1) | ((p->tm\_mon+1) >> 3);

c[26] = (clusterno & 0x00ff);

c[27] = ((clusterno & 0xff00)>>8);

if(lseek(fd,offset,SEEK\_SET)<0)

perror("lseek fd\_mkdir failed");

if(write(fd,&c,DIR\_ENTRY\_SIZE)<0)

perror("write failed");

clear(clusterno);

free(pentry);

if(WriteFat()<0)

exit(1);

return 1;

}

}

}

}

else

{

printf("This dirname is exist\n");

free(pentry);

return -1;

}

}

int pathSwitch(char \*pathname, char \*name)//duo ji mu lu

{

int num = 0, i, j, k = 0, ret;

struct Entry \*pentry;

if(pathname[0] == '/')

{

k = 1;

if(curdir != NULL)

{

dirno++;

fatherdir[dirno] = curdir;

curdir = NULL;

}

}

for(i = k; i < strlen(pathname); i++)

{

if(pathname[i] == '/')

num++;

}

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

{

j = 0;

while(pathname[k] != '/')

name[j++] = pathname[k++];

k++;

name[j] = '\0';

pentry = (struct Entry\*)malloc(sizeof(struct Entry));

ret = ScanEntry(name,pentry,1);

if(ret < 0)

{

printf("no such dir\n");

free(pentry);

return -1;

}

dirno ++;

fatherdir[dirno] = curdir;

curdir = pentry;

}

for(j = 0; pathname[k] != '\0'; k++, j++)

name[j] = pathname[k];

name[j] = '\0';

if(pathname[0] == '/')

num++;

return num;

}

void pathBack(int num)

{

int i;

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

{

curdir = fatherdir[dirno];

dirno--;

}

}

void do\_usage()

{

printf("please input a command, including followings:\n\tls\t\t\tlist all files\n\tcd <dir>\t\tchange direcotry\n\tcf <filename> <size>\tcreate a file\n\tmkdir <dir>\t\tcreate a dir\n\tdf <file>\t\tdelete a file\n\t");

printf("rm <dir>\t\tdelete a dir with hint\n\trm -r <dir>\t\tdelete a dir\n\texit\t\t\texit this system\n");

}

int main()

{

char input[10];

int size=0, num = 0;

char name[12];

char temp[120];

char pathname[120];

if((fd = open(DEVNAME,O\_RDWR))<0)

perror("open failed");

ScanBootSector();

if(ReadFat()<0)

exit(1);

do\_usage();

while (1)

{

printf(">");

scanf("%s",input);

if (strcmp(input, "exit") == 0)

break;

else if (strcmp(input, "ls") == 0)

fd\_ls();

else if(strcmp(input, "cd") == 0)

{

scanf("%s", pathname);

if(pathname[0] == '/')

{

dirno = -1;

curdir = NULL;

}

if((num = pathSwitch(pathname, name)) == -1)

{

continue;

}

fd\_cd(name);

}

else if(strcmp(input, "df") == 0)

{

scanf("%s", pathname);

if((num = pathSwitch(pathname, name)) == -1)

{

continue;

}

fd\_df(name);

pathBack(num);

}

else if(strcmp(input, "rm") == 0)

{

scanf("%s", temp);

if(strcmp(temp, "-r") == 0)

{

scanf("%s", pathname);

if((num = pathSwitch(pathname, name)) == -1)

{

continue;

}

fd\_rm\_r(name);

pathBack(num);

}

else

{

if((num = pathSwitch(temp, name)) == -1)

{

continue;

}

fd\_rm(name);

pathBack(num);

}

}

else if(strcmp(input, "cf") == 0)

{

scanf("%s", pathname);

scanf("%s", input);

size = atoi(input);

if((num = pathSwitch(pathname, name)) == -1)

{

continue;

}

fd\_cf(name, size);

pathBack(num);

}

else if(strcmp(input, "mkdir") == 0)

{

scanf("%s", pathname);

if((num = pathSwitch(pathname, name)) == -1)

{

continue;

}

fd\_mkdir(name);

pathBack(num);

}

else

do\_usage();

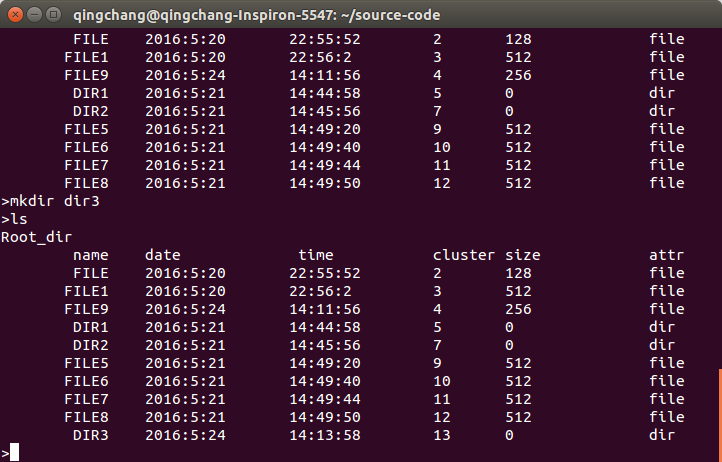
}

return 0;

}

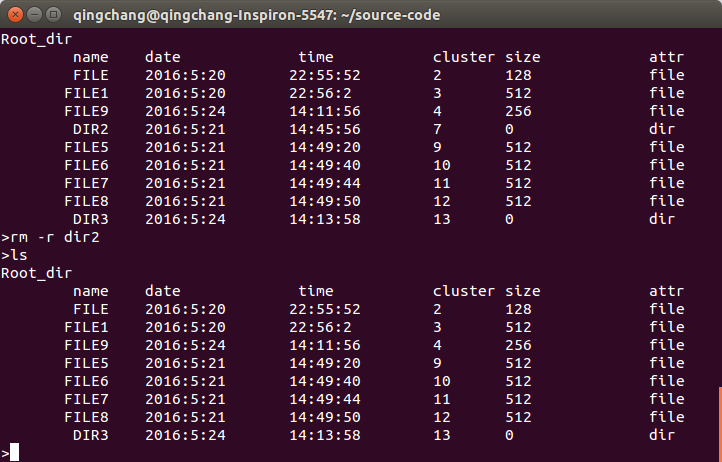
实现截图：

1. mkdir命令

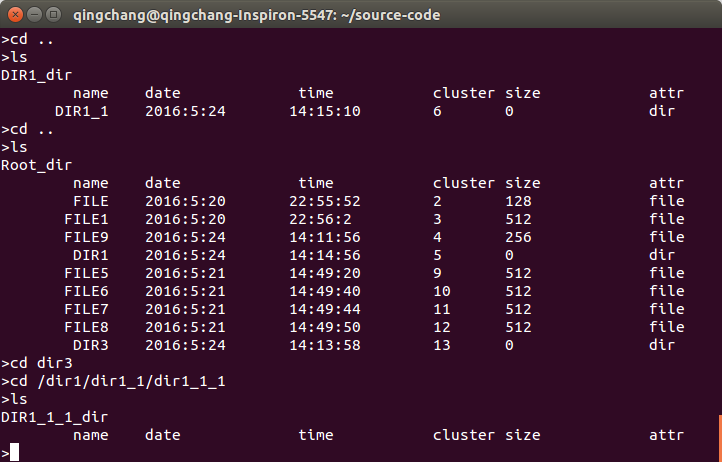


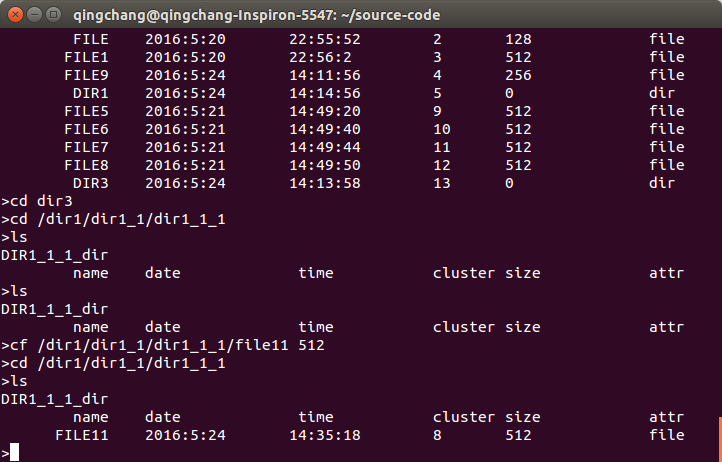
1. rm命令和rm\_r命令



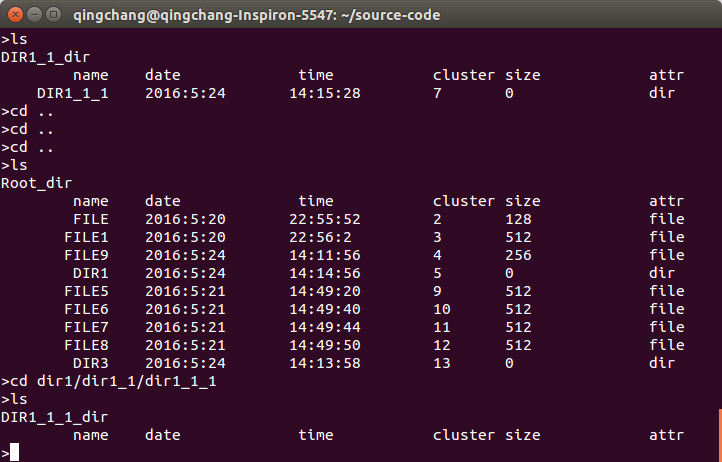


1. 绝对路径

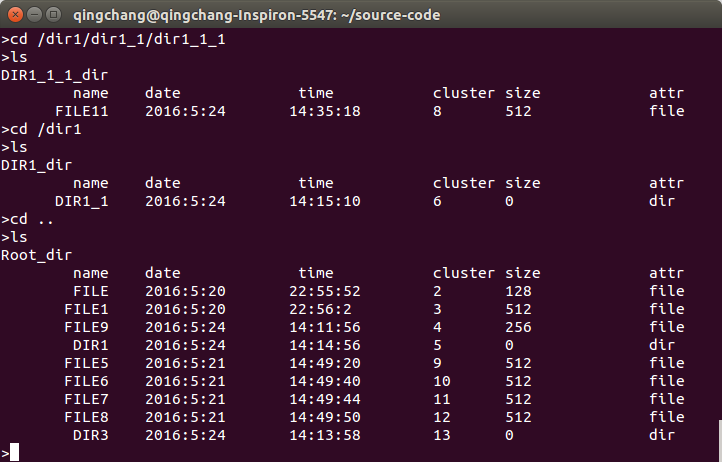




1. 多级目录



1. ls命令



# 4.收获和感想

这次文件系统实验让我们组每个人都重新认真学习了文件系统的相关知识，弄清楚了文件管理系统的工作方式和FAT文件系统的结构，还有超级块和inode等相关概念。另外，在完成实验的过程中，我们分工明确，共同攻克难题，感受到了浓浓的团队精神。总之，这次实验不仅让我们学习到了很多知识，更是让我们在精神层面有很多收获。期待未来更多学习成长的机会！