**操作系统课程设计实验报告**

——实验四：FileSys实验

负责人姓名：岳明哲

学号：14061173

日期：2016.5.26

**19小组成员**

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 姓名 | 学号 | 实验分工 |
| 1 | 岳明哲 | 14061173 | 实验4 |
| 2 | 李文涛 | 14061217 | 实验1 |
| 3 | 李康宇 | 14061172 | 实验3 |
| 4 | 张东鑫 | 14061153 | 实验2 |

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、超级控制块等相关概念

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

1）ud\_ls函数的实现，实现一个目录列表函数

2）ud\_cd(char \*directory) 设计并实现改变目录函数

3) ud\_df(char \*name) 设计并实现一个删除文件的函数

4）ud\_cf(char \*filename,int seze)设计并实现一个创建文件的函数

## 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

【简述实验完成过程】。完成了以下功能：

1.增加创建目录的功能

2.完善了部分cd指令

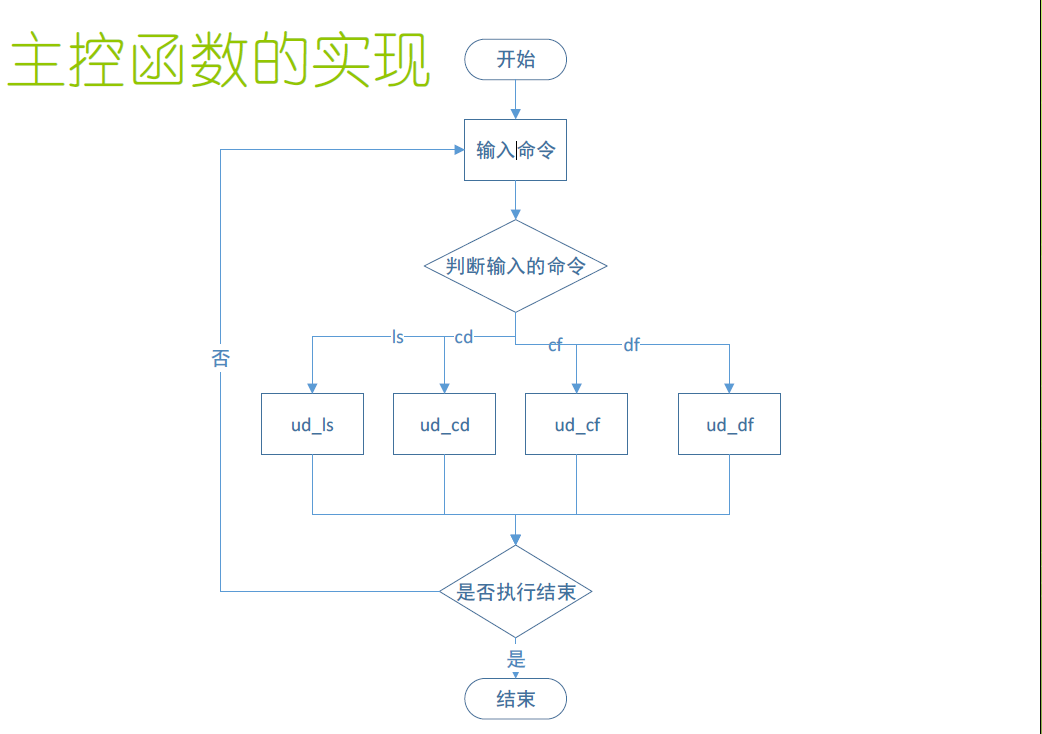
3.完善了cf指令，能够在创建文件的同时向文件中写入内容，并根据写入内容创建相应大小的文件

4.改进了ls指令，能够通过ls –a显示所有非根目录信息的读取

……

# 3.设计说明

## 3.1 程序流程图



## 3.2基本要求实现说明

1）ls指令，通过判断是根目录或者是子目录，根据根目录区的信息或者当前目录保存的起始簇号计算出在数据区的地址，读取改地址后面的内容，讲目录下的信息打印出来

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;

}

2)cd函数 根据传入的目录信息过滤不必要的操作，然后在该目录下查找参数名相吻合的目录项，找到就释放原来的curdir，并指向子目录项

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;

}

3)ud\_df 函数的实现，查找当前目录下是否存在要删除的文件，如果存在把文件对应的目录项内容删除，并在Fat表中标记为未使用

int fd\_df(char \*filename)

{

struct Entry \*pentry;

int ret;

unsigned char c;

unsigned short seed,next;

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

ret=ScanEntry(filename,pentry,0); //扫描当前目录查找文件

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;

}

4.cf函数的实现,先查找当前目录下有没有同名文件，如果没有依次查找fat表，找到空白簇，然后将簇信息依次写入int fd\_cf(char \*filename,int size)

{

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;

/\*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[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;

}

## 3.3 提高要求实现说明、

1)创建目录

mkdir 实现基本和创建文件相同，查找当前目录下是否有重名的，如果没有则创建目录，创建过程现在FAT表中找到未使用的簇，标记为使用，然后根据是在根目录下创建目录还是子目录下创建目录，将目录项写入不同的地方，属性为c[11]为0x10

c[28] = (0x00000000 & 0x000000ff);

c[29] = ((0x00000000 & 0x0000ff00)>>8);

c[30] = ((0x00000000& 0x00ff0000)>>16);

c[31] = ((0x00000000& 0xff000000)>>24);

int fd\_md(char \*dirname)

{

struct Entry \*pentry;

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

unsigned short cluster,clusterno[100];

unsigned char c[DIR\_ENTRY\_SIZE];

int index;

unsigned char buf[DIR\_ENTRY\_SIZE];

struct tm \*local;

time\_t t;

int cutime,cudate;

t=time(NULL);

local=localtime(&t);

cutime=local->tm\_hour\*2048+local->tm\_min\*32+local->tm\_sec/2;

cudate=(local->tm\_year+1900-1980)\*512+(local->tm\_mon+1)\*32+local->tm\_mday;

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

//扫描根目录，是否已存在该文件名

ret = ScanEntry(dirname,pentry,0);

if (ret<0)

{

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

{

index = cluster \*2;

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

{

/\*在fat表中写入xffff\*/

fatbuf[index]=0xff;

fatbuf[index+1]=0xff;

break;

}

}

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(dirname);i++)

{

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

}

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

c[i]=' ';

c[11] = 0x10;

/\*写时间和日期\*/

c[0x16]=(cutime&0x00ff);

c[0x17]=((cutime&0xff00)>>8);

c[0x18]=(cudate&0x00ff);

c[0x19]=((cudate&0xff00)>>8);

/\*写第一簇的值\*/

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

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

/\*写文件的大小\*/

c[28] = (0x00000000 & 0x000000ff);

c[29] = ((0x00000000 & 0x0000ff00)>>8);

c[30] = ((0x00000000& 0x00ff0000)>>16);

c[31] = ((0x00000000& 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(dirname);i++)

{

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

}

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

c[i]=' ';

c[11] = 0x10;

/\*写时间和日期\*/

c[0x16]=(cutime&0x00ff);

c[0x17]=((cutime&0xff00)>>8);

c[0x18]=(cudate&0x00ff);

c[0x19]=((cudate&0xff00)>>8);

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

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

c[28] = (0x00000000 & 0x000000ff);

c[29] = ((0x00000000 & 0x0000ff00)>>8);

c[30] = ((0x00000000& 0x00ff0000)>>16);

c[31] = ((0x00000000& 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 directory is exist\n");

free(pentry);

return -1;

}

return 1;

}

2) 增加绝对路径和多级目录的支持:只支持了部分内容./dirname/或者../direname/这样的能够支持，实现根据比较先进入当前或者上一级目录，然后再进入目录

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

{

scanf("%s", name);

if(strncmp(name,"./",2)==0)

{

fd\_cd(".");

for(i=2,num=2;i<30&&name[i]!='\0';i++)

{

if(name[i]=='/') //cd ./test1/截取test1

{

memset(cdname,0,30);

for(j=num,k=0;j<i;j++,k++)

cdname[k]=name[j];

if(fd\_cd(cdname)<0)

break;

num=i+1;

}

}

}

else if(strncmp(name,"../",3)==0)

{

fd\_cd("..");

for(i=3,num=3;i<30&&name[i]!='\0';i++)

{

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

{

memset(cdname,0,30);

for(j=num,k=0;j<i;j++,k++)

cdname[k]=name[j];

if(fd\_cd(cdname)<0)

break;

num=i+1;

}

}

}

4) 对ud\_cf()函数进行改进能够支持写内容，根据读入的信息，判断需要多少簇，然后将读入的内容写入文件中

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

{

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];

struct tm \*local;

time\_t t;

int cftime,cfdate;

unsigned char data[size];

t = time(NULL);

local = localtime(&t);

cftime = local->tm\_hour\*2048+local->tm\_min\*32+local->tm\_sec/2;

cfdate = (local->tm\_year+1900-1980)\*512+(local->tm\_mon+1)\*32+local->tm\_mday;

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

//扫描根目录，是否已存在该文件名

ret = ScanEntry(filename,pentry,0);

if (ret<0)

{

printf("输入信息:\n");

scanf("%s",data);

size=strlen(data);

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

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

clustersize ++; //需要多少个簇

/\*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);

if(lseek(fd,(clusterno[i]-2)\*CLUSTER\_SIZE+DATA\_OFFSET,SEEK\_SET)<0)

perror("lseek file data failed");

if(write(fd,data+i\*CLUSTER\_SIZE,CLUSTER\_SIZE)<0)

perror("write failed");

}

/\*最后一簇写入0xffff\*/

index = clusterno[i]\*2;

fatbuf[index] = 0xff;

fatbuf[index+1] = 0xff;

if(lseek(fd,(clusterno[i]-2)\*CLUSTER\_SIZE+DATA\_OFFSET,SEEK\_SET)<0)

perror("lseek file data failed");

if(write(fd,data+i\*CLUSTER\_SIZE,size-i\*CLUSTER\_SIZE)<0)

perror("write failed");

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;

/\*写时间和日期\*/

c[0x16]=(cftime&0x00ff);

c[0x17]=((cftime&0xff00)>>8);

c[0x18]=(cfdate&0x00ff);

c[0x19]=((cfdate&0xff00)>>8);

/\*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;

/\*写时间和日期\*/

c[0x16]=(cftime&0x00ff);

c[0x17]=((cftime&0xff00)>>8);

c[0x18]=(cfdate&0x00ff);

c[0x19]=((cfdate&0xff00)>>8);

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;

}

5)ls功能的实现，能够调用ls –a显示所有非根目录的信息，从数据区开始每次读一个簇的内容，如果其实簇号不为0则显示出来，若为0也显示则可验证cf中创建文件时写入的内容

if(funct==1)

{

printf("All\_Non-Root\_dir\n");

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

for(i=0;i<100;i++)

{

cluster\_addr=DATA\_OFFSET+i\*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&&entry.FirstCluster!=0)

{

printf("%16s\t"

"%d-%d-%d\t"

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

"%d\t"

"%d\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");

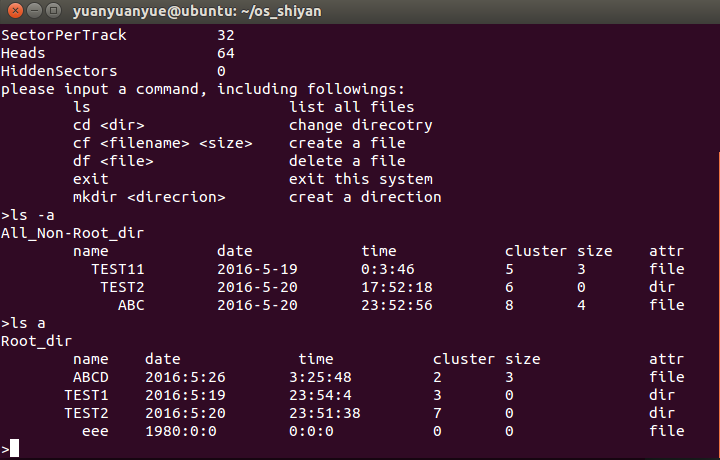
}

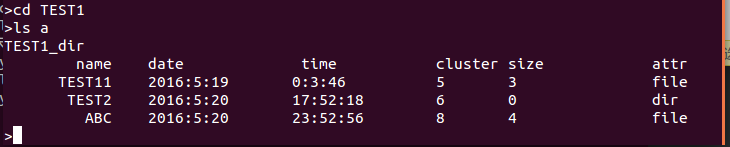
}

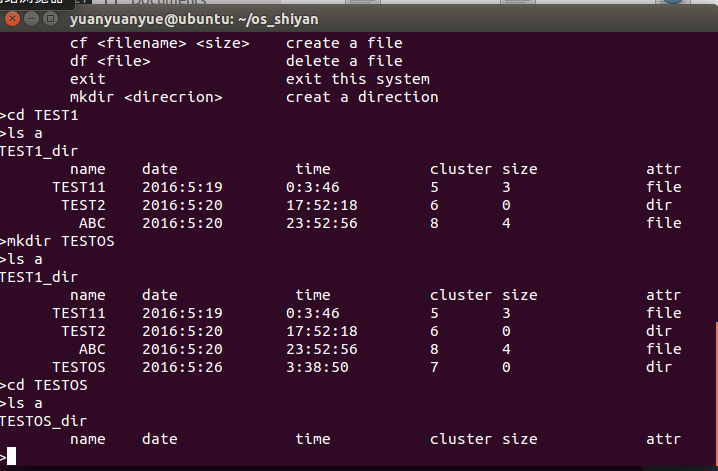
}

return 1;

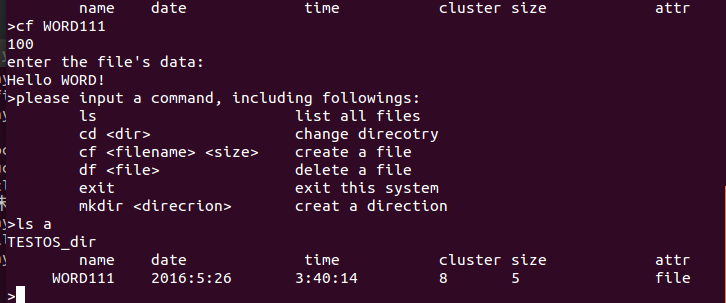
}



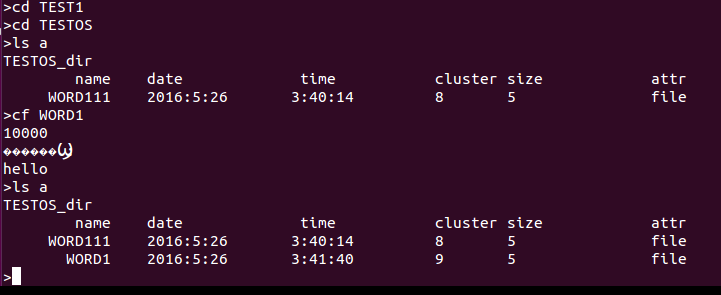
ls-a 显示除了在TEST1文件下的内容，即非根目录的内容



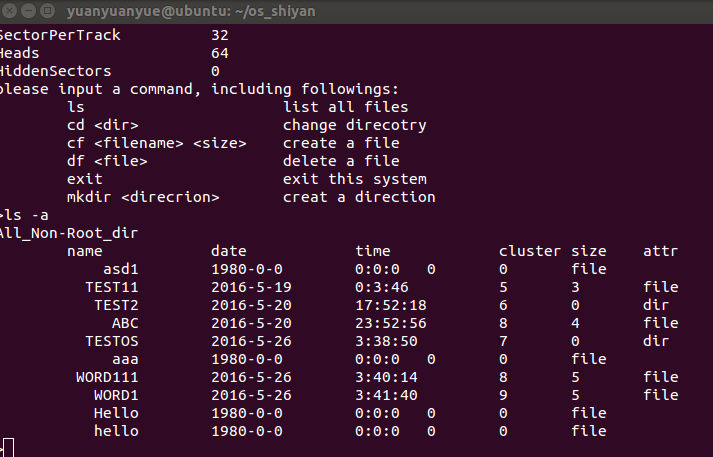
在TEST1目录下创建TESTOS目录，进入后用ls显示为空



创建文件并写入内容，根据写入内容大小创建文件



在删除掉判断是不是文件或者目录的ls –a指令后显示数据区的所有内容



可见hello 被成功写入

# 4.收获和感想

经过这次实验认识到了文件系统的，知道了FAT文件系统的结构，根据FAT表示如何分配管理文件的，以及不同对文件或者目录的操作的原理和实现的过程，但是有的内容还是没法深刻理解，还是需要之后的学习

在完成实验的过程中提高了自己的自学能力，同时也提高了团队能力，能够一个团队互相合作完成任务。

会议记录

第一次会议

阅读源代码，搞清楚要实现的功能，分配任务。

第二次会议

互相交流，讨论，完善相应的代码。

第三次会议

互相交流，讨论。提出合理的建议，处理bug。