**操作系统课程设计实验报告**

——实验四：file system实验

负责人姓名：龚睿昊

学号：14061192

日期：2016.5.21

**小组成员**

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 姓名 | 学号 | 实验分工 |
| 1 | 王燕飞 | 14061154 | 实验一 |
| 2 | 林峥 | 14061160 | 实验二 |
| 3 | 叶俊辰 | 14061166 | 实验三 |
| 4 | 龚睿昊 | 14061192 | 实验四 |

目录

1.实验目的 4

2.需求说明 4

2.1基本要求 4

2.2 提高要求 4

2.3 完成情况 4

3.设计说明 5

3.1 程序流程图 5

3.2基本要求实现说明 5

3.3 提高要求实现说明 5

4.收获和感想 5

1.实验目的

1.了解文件管理系统的作用和工作方式

2.了解FAT文件系统的结构

3.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. 设计实现目录列表函数(ls)，显示当前目录下包含的信息
2. 设计实现改变目录函数(cd)，转换路径
3. 设计实现删除文件函数(df)，要删除的文件名作为参数，删除指定的文件
4. 设计实现创建文件函数(cf)，使用文件名和文件大小作为参数，创建文件

2.2 提高要求

1. 创建目录(mkdir)
2. 删除目录(rm (-r))
3. 路径转换支持多级目录和绝对路径
4. 修改cf，使其创建时可以写入实际内容
5. 修改ud\_ls，使其不只读一簇而是读出非根目录的所有内容

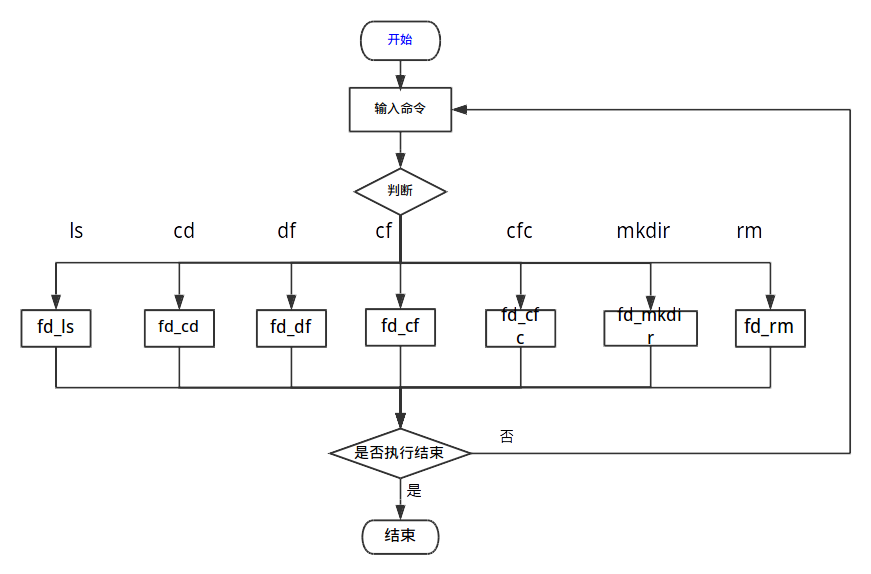
2.3 完成情况

【简述实验完成过程】。完成了以下功能：

1. cf filename size创建文件名为filename，大写为size的文件
2. cd path 路径转换，path可以是任意多级路径，或是绝对路径
3. ls 列出当前目录下所有内容，非根目录列出所有其占用的簇中的目录项内容
4. cfc filename content 创建文件名为filename,内容为content的文件
5. df filename 删除文件名为filename的文件
6. mkdir dirname 创建目录名为dirname的目录
7. rm filename/dirname 删除文件或目录，若要删除目录不为空，询问用户是否继续，继续则执行删除操作，否则不进行任何操作
8. rm -r dirname 直接递归删除目录

3.设计说明

3.1 程序流程图

3.2基本要求实现说明

配以详细的代码和实现步骤截图。

1.cf的修改和添加。

* 1. 添加创建时间函数setTime()和setDate，确保创建时间的正确
  2. 修改文件名格式的处理，增加formaName()函数使所有文件均以8.3格式，便于后续比较和管理

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

{

struct Entry \*pentry;

int ret,i=0,j=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);

formatName(filename,c,0);

printf("create:");

for(i=0;i<11;i++){

printf("%c",c[i]);

}

printf("\n");

//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);

/\*写文件时间\*/

//time

setTime(&c[22],&c[23]);

//date

setDate(&c[24],&c[25]);

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);

formatName(filename,c,0);

// 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);

/\*写文件时间\*/

//time

setTime(&c[22],&c[23]);

//date

setDate(&c[24],&c[25]);

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;

}

1. df的修改
   1. 修改fd\_df函数的置目录项前两位为0xe5的操作，源代码会把要删除文件和要删除文件的前一个文件的目录项均置为0xe5，经讨论群内讨论，猜测第二个置位操作可能是想连同该文件对应的长文件名文件一并删除，故此处修改，加了判断，如果 该目录项对应的attribute属性为0x0F，则一并删除，否则不删

int fd\_df(char \*filename)

{

struct Entry \*pentry;

int ret;

unsigned char c;

unsigned short seed,next;

unsigned char attribute;

//unsigned char filename[12];

//formatName(name,filename);

//filename[11] = '\0';

//FileNameFormat(filename);

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)

{

if(next==seed){

perror("wrong file info");

return -1;

}

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+11,SEEK\_SET)<0)

perror("lseek fd\_df failed");

if(read(fd,&attribute,1)<0)

perror("read failed");

if(attribute==0x0f)

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;

}

3.3 提高要求实现说明

配以详细的代码和实现步骤截图。

1.cfc函数的实现使创建文件时可添加具体内容

* 1. 找到空簇更新fat表，并添加目录项，然后，向占用的簇中循环写入文件内容

int fd\_cfc(char \*filename,char \* content)

{

struct Entry \*pentry;

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

unsigned short cluster,clusterno[100];

unsigned char c[DIR\_ENTRY\_SIZE];

unsigned short index1 = 0xffff;

int index,clustersize;

unsigned char buf[DIR\_ENTRY\_SIZE];

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

size = strlen(content)+1;

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);

formatName(filename,c,0);

printf("create:");

for(i=0;i<11;i++){

printf("%c",c[i]);

}

printf("\n");

//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);

/\*写文件时间\*/

//time

setTime(&c[22],&c[23]);

//date

setDate(&c[24],&c[25]);

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);

j = 0;

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

ClearCluster(clusterno[i]);

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

perror("lseek fd\_cf failed");

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

perror("write failed");

}

ClearCluster(clusterno[clustersize-1]);

if(lseek(fd,DATA\_OFFSET+(clusterno[clustersize-1]-2)\*CLUSTER\_SIZE,SEEK\_SET)<0)

perror("lseek fd\_cf failed");

if(write(fd,content+(clustersize-1)\*CLUSTER\_SIZE,strlen(content)-(clustersize-1)\*CLUSTER\_SIZE)<0)

perror("write failed");

return 1;

}

}

}

else

{

index1 = curdir->FirstCluster;

do{

cluster\_addr = (index1 -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);

formatName(filename,c,0);

// 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);

/\*写文件时间\*/

//time

setTime(&c[22],&c[23]);

//date

setDate(&c[24],&c[25]);

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);

j = 0;

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

ClearCluster(clusterno[i]);

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

perror("lseek fd\_cf failed");

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

perror("write failed");

}

ClearCluster(clusterno[clustersize-1]);

if(lseek(fd,DATA\_OFFSET+(clusterno[clustersize-1]-2)\*CLUSTER\_SIZE,SEEK\_SET)<0)

perror("lseek fd\_cf failed");

if(write(fd,content+(clustersize-1)\*CLUSTER\_SIZE,strlen(content)-(clustersize-1)\*CLUSTER\_SIZE)<0)

perror("write failed");

return 1;

}

}

index1 = GetFatCluster(index1);

}while(index1!=0xffff);

}

}

else

{

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

free(pentry);

return -1;

}

return 1;

}

1. ls的修改
   1. 修改fd\_ls函数，当前目录非根目录时，使其循环读取此目录所有的内容（即查找fat表依次读簇直到oxffff为止）

3.mkdir的实现

* 1. 找到空白簇，更新fat表，添加目录项
  2. 此目录中自动添加此目录的目录项和其父目录的目录项，并将文件名分别设置为”.”和”..”，以便实现多级目录

int fd\_mkdir(char\* filename)

{

struct Entry \*pentry;

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

unsigned short cluster,clusterno = -1;

unsigned char c[DIR\_ENTRY\_SIZE];

int index,clustersize;

unsigned char buf[DIR\_ENTRY\_SIZE];

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

int size = 0;

memset(c, 0, sizeof(char) \* DIR\_ENTRY\_SIZE);

ret = ScanEntry(filename,pentry,1);

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 = cluster;

ClearCluster(clusterno);

break;

}

}

if(clusterno < 0)

{

perror("not find blank cluster");

return -1;

}

index = clusterno \*2;

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);

formatName(filename,c,1);

c[11] = 0x11;

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

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

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

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

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

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

/\*写文件时间\*/

//time

setTime(&c[22],&c[23]);

//date

setDate(&c[24],&c[25]);

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

perror("lseek fd\_mkdir failed");

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

perror("write failed");

free(pentry);

if(WriteFat() < 0)

exit(1);

/\*向目录所在簇中写入此目录和父目录的目录项\*/

formatName(".",c,1);

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

perror("lseek fd\_mkdir failed");

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

perror("write failed");

memset(c, 0, sizeof(char) \* DIR\_ENTRY\_SIZE);

formatName("..",c,1);

c[11] = 0x11;

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

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

if(lseek(fd,DATA\_OFFSET+(clusterno-2)\*CLUSTER\_SIZE+0x20, SEEK\_SET) < 0 )

perror("lseek fd\_mkdir failed");

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

perror("write failed");

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 dir failed");

offset += abs(ret);

}

}

else

{

offset = offset - abs(ret);

formatName(filename,c,1);

c[11] = 0x11;

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

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

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

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

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

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

/\*写文件时间\*/

//time

setTime(&c[22],&c[23]);

//date

setDate(&c[24],&c[25]);

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

perror("lseek fd\_mkdir failed");

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

perror("write failed");

free(pentry);

if(WriteFat() < 0)

exit(1);

/\*向目录所在簇中写入此目录和父目录的目录项\*/

formatName(".",c,1);

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

perror("lseek fd\_mkdir failed");

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

perror("write failed");

memset(c, 0, sizeof(char) \* DIR\_ENTRY\_SIZE);

formatName("..",c,1);

c[11] = 0x11;

c[26] = (curdir->FirstCluster & 0x00ff);

c[27] = ((curdir->FirstCluster & 0xff00) >> 8);

if(lseek(fd,DATA\_OFFSET+(clusterno-2)\*CLUSTER\_SIZE+0x20, SEEK\_SET) < 0 )

perror("lseek fd\_mkdir failed");

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

perror("write failed");

return 1;

}

}

}

}

else

{

printf("This directory is existed\n");

free(pentry);

return -1;

}

return 1;

}

1. rm 的添加
   1. 增加fd\_dd函数，用于递归删除目录
   2. 增加checkDir()函数，用于检测目录是否为空
   3. 根据-r参数判断，若存在-r，则直接递归删除该目录内容，否则提示用户是否继续，根据用户输入删除与否

int CheckDir(struct Entry \*pentry)

{

int ret,offset,cluster\_addr;

struct Entry entry;

unsigned short index = 0xffff;

//int index;

unsigned char buf[DIR\_ENTRY\_SIZE];

index = pentry->FirstCluster;

do{

cluster\_addr = DATA\_OFFSET+(index - 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(&entry);

offset += abs(ret);

if (ret > 0 && strcmp(entry.short\_name,".")!=0 && strcmp(entry.short\_name,"..")!=0){

return 1;

}

}

index = GetFatCluster(index);

}while( index != 0xffff);

return 0;

}

int fd\_dd(char \*dirname,int flag)

{

struct Entry \*pentry;

int ret;

unsigned char c;

unsigned short seed,next;

char mark[10];

unsigned char attribute;

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

ret = ScanEntry(dirname,pentry,1);

if(ret<0){

perror("no such dir");

free(pentry);

return -1;

}

/\*目录为空\*/

if(CheckDir(pentry)==0){

ClearDir(pentry);

seed = pentry->FirstCluster;

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

{

ClearFatCluster(seed);

seed = next;

}

ClearFatCluster(seed);

c = 0xe5;

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

perror("lseek fd\_dd failed");

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

perror("write failed");

if(lseek(fd,ret-0x40+11,SEEK\_SET)<0)

perror("lseek fd\_dd failed");

if(read(fd,&attribute,1)<0)

perror("read failed");

if(attribute==0x0f)

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

perror("lseek fd\_dd failed");

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

perror("write failed");

free(pentry);

if (WriteFat()<0)

exit(1);

return 1;

}

if(flag==0){

printf("warning: this directory is not empty.Do you want to continue?(Y/N)\n");

scanf("%s",mark);

if(!(strcmp(mark,"y")==0||strcmp(mark,"Y")==0))

return 1;

}

ClearDir(pentry);

//printf("clear over\n");

seed = pentry->FirstCluster;

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

{

ClearFatCluster(seed);

seed = next;

}

ClearFatCluster(seed);

c = 0xe5;

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

perror("lseek fd\_dd failed");

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

perror("write failed");

if(lseek(fd,ret-0x40+11,SEEK\_SET)<0)

perror("lseek fd\_dd failed");

if(read(fd,&attribute,1)<0)

perror("read failed");

if(attribute==0x0f)

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

perror("lseek fd\_dd failed");

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

perror("write failed");

free(pentry);

if (WriteFat()<0)

exit(1);

return 1;

}

int ClearDir(struct Entry \*pentry)

{

int ret,offset,cluster\_addr;

struct Entry entry;

unsigned char c;

unsigned short seed,next;

unsigned short index = 0xffff;

unsigned char buf[DIR\_ENTRY\_SIZE];

unsigned char attribute;

index = pentry->FirstCluster;

do{

cluster\_addr = DATA\_OFFSET+(index - 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(&entry);

offset += abs(ret);

if (ret > 0){

if(entry.subdir == 1 && CheckDir(&entry) == 1)

{

if(strcmp(entry.short\_name,".")!=0 && strcmp(entry.short\_name,"..")!=0)

ClearDir(&entry);

}

/\*清空Fat\*/

if(strcmp(entry.short\_name,".")!=0 && strcmp(entry.short\_name,"..")!=0)

{

seed = entry.FirstCluster;

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

{

ClearFatCluster(seed);

seed = next;

}

ClearFatCluster(seed);

}

/\*清空目录项\*/

c = 0xe5;

if(ret>=32 && lseek(fd,offset-0x20,SEEK\_SET) < 0)

perror("lseek uf\_ClearDir1 failed");

if(ret>=32 && write(fd,&c,1) < 0)

perror("write failed");

if(ret>=64 && lseek(fd,offset-0x40+11,SEEK\_SET)<0)

perror("lseek fd\_ClearDir2 failed");

if(ret>=64 && read(fd,&attribute,1)<0)

perror("read failed");

if(ret>=64 && attribute==0x0f)

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

perror("lseek fd\_ClearDir3 failed");

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

perror("write failed");

if (WriteFat()<0)

exit(1);

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

perror("lseek fd\_ClearDir4 failed");

}

}

index = GetFatCluster(index);

}while(index != 0xffff);

return 1;

}

int fd\_dd(char \*dirname,int flag)

{

struct Entry \*pentry;

int ret;

unsigned char c;

unsigned short seed,next;

char mark[10];

unsigned char attribute;

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

ret = ScanEntry(dirname,pentry,1);

if(ret<0){

perror("no such dir");

free(pentry);

return -1;

}

/\*目录为空\*/

if(CheckDir(pentry)==0){

ClearDir(pentry);

seed = pentry->FirstCluster;

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

{

ClearFatCluster(seed);

seed = next;

}

ClearFatCluster(seed);

c = 0xe5;

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

perror("lseek fd\_dd failed");

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

perror("write failed");

if(lseek(fd,ret-0x40+11,SEEK\_SET)<0)

perror("lseek fd\_dd failed");

if(read(fd,&attribute,1)<0)

perror("read failed");

if(attribute==0x0f)

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

perror("lseek fd\_dd failed");

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

perror("write failed");

free(pentry);

if (WriteFat()<0)

exit(1);

return 1;

}

if(flag==0){

printf("warning: this directory is not empty.Do you want to continue?(Y/N)\n");

scanf("%s",mark);

if(!(strcmp(mark,"y")==0||strcmp(mark,"Y")==0))

return 1;

}

ClearDir(pentry);

//printf("clear over\n");

seed = pentry->FirstCluster;

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

{

ClearFatCluster(seed);

seed = next;

}

ClearFatCluster(seed);

c = 0xe5;

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

perror("lseek fd\_dd failed");

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

perror("write failed");

if(lseek(fd,ret-0x40+11,SEEK\_SET)<0)

perror("lseek fd\_dd failed");

if(read(fd,&attribute,1)<0)

perror("read failed");

if(attribute==0x0f)

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

perror("lseek fd\_dd failed");

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

perror("write failed");

free(pentry);

if (WriteFat()<0)

exit(1);

return 1;

}

int fd\_rm(char \*name,int flag)

{

int ret;

struct Entry \*pentry;

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

if(flag==1){

ret = ScanEntry(name,pentry,1);

if(ret>0){

fd\_dd(name,1);

return 1;

}

else{

free(pentry);

perror("no such directory");

return -1;

}

}

else{

//printf("name:%s\n",name);

ret = ScanEntry(name,pentry,0);

if(ret>0){

fd\_df(name);

return 1;

}

ret = ScanEntry(name,pentry,1);

if(ret>0){

fd\_dd(name,0);

return 1;

}

perror("no such file or directory");

free(pentry);

return -1;

}

}

1. cd的多级路径和绝对路径支持
   1. 增加parseDir()函数，分析路径并保存于二维数组中
   2. 增加findDir()和updateCurdir()函数，用于将最终找到的路径的目录项转换为标准的目录项，如最终找到的目录项名字为”..”，这个目录项显然不是标准的，则需找到他的父目录项中与其簇相同的目录项，设置为当前目录项，这项目录项的名字才是真实正确的

int fd\_cd(char \*dir)

{

struct Entry \*pentry;

int ret;

// struct Entry \*entries[20];

struct Entry \*temp;

char dirs[50][12];

int i,j;

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;

}

\*/

/\*转到根目录\*/

if(strcmp(dir,"/")==0){

/\* while(dirno>0){

free(fatherdir[dirno]);

dirno--;

}

\*/

free(curdir);

curdir=NULL;

return 1;

}

temp = curdir;

/\*解析目录并保存\*/

if(dir[0]=='/') curdir=NULL;

parseDir(dir,dirs);

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

for(i=0;dirs[i][0]!='\0';i++){

ret = ScanEntry(dirs[i],pentry,1);

if(ret==-1){

printf("no such directory:%s\n",dirs[i]);

free(pentry);

curdir = temp;

return -1;

}

curdir = pentry;

if(ret==-3 || pentry->FirstCluster==0){

curdir = NULL;

}

}

if(curdir==NULL)

free(pentry);

updateCurdir(&curdir);

return 1;

}

4.收获和感想

1.对FAT16格式的文件系统的结构有了更加真切的了解

2.通过尝试与探索，了解了linux自带的文件管理系统的实现方式，并借鉴其方式实现在我们的文件管理系统中。

3.意识到了申请内存后及时释放的重要性！！！在添加功能时，有个别位置申请内存后没有及时释放，导致随着路径转换的增加，申请的内存超过限制，自动冲去了全局变量fatbuf[]的空间，也就是我们存储的fat表的拷贝那部分内存被冲去部分，进而造成奇怪的bug，调了很长时间才发现问题所在，血的教训！