目录

[void startsys() 1](#_Toc534293776)

[void my\_format() 4](#_Toc534293777)

[void my\_cd 12](#_Toc534293778)

[my\_mkdir 13](#_Toc534293779)

[my\_rmdir 21](#_Toc534293780)

[my\_ls 24](#_Toc534293781)

[my\_create 26](#_Toc534293782)

# void startsys()

{

    FILE \*fp;

    int i;

    myvhard=(unsigned char \*)malloc(SIZE);

    memset(myvhard,0,SIZE);

    fp=fopen(filename,"r");

FILE \*fopen(char \*path,\*mode);

如果成功的打开一个文件, fopen()函数返回文件指针, 否则返回空指针 (NULL)。由此可判断文件打开是否成功。

mode:

r:  读打开， 不创建

w: 写打开，创建/清空

a: 写打开， 创建/追加

r+: 读/写打开，不创建

w+:读/写打开，创建/清空

a+:读/写打开， 创建/追加

    if(fp)

    {

        fread(buffer,SIZE,1,fp);

2、fread

size\_t fread(void \* ptr,size\_t size,size\_t nmemb,FILE \* stream);

参数size，表示一次读取的数据单元大小，nmemb表示读取的次数。

    成功时fread返回的值与nmemb相等；若小于nmemb但是大于0，则可能是到了文件末尾，不够次数；若返回0，则是文件读取错误，不满一个size的大小。

    所以循环读取文件的时候，while(可以判断是否大于等于0)作为条件，里面再基于feof函数判断是否到了文件末尾而退出循环。

        fclose(fp);

        if(buffer[0]==0xaa)如果最开始是字符串结束符

        {

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

            my\_format();

        }

        else

        {

            for(i=0;i<SIZE;i++)

                myvhard[i]=buffer[i];

        }

    }

    else

    {

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

        my\_format();

    }

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

    strcpy(openfilelist[0].exname,"di");

    openfilelist[0].attribute=0x2d;

    openfilelist[0].time=((fcb \*)(myvhard+5\*BLOCKSIZE))->time;

    openfilelist[0].date=((fcb \*)(myvhard+5\*BLOCKSIZE))->date;

    openfilelist[0].first=((fcb \*)(myvhard+5\*BLOCKSIZE))->first;

    openfilelist[0].length=((fcb \*)(myvhard+5\*BLOCKSIZE))->length;

    openfilelist[0].free=1;

    openfilelist[0].dirno=5;

    openfilelist[0].diroff=0;

    openfilelist[0].count=0;

    openfilelist[0].fcbstate=0;

    openfilelist[0].topenfile=0;

    openfilelist[0].father=0;

并填写根目录文件的相关信息

    memset(currentdir,0,sizeof(currentdir));

    strcpy(currentdir,"\\root\\");

    strcpy(openfilelist->dir[0],currentdir);

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

    ptrcurdir=&openfilelist[0];

    curfd=0;

}

# void my\_format()

/\*

原型声明: void my\_format()

功能描述： 对虚拟磁盘进行格式化，布局虚拟磁盘，建立根目录文件

输入：无

输出：无

函数功能实现算法描述：

虚拟磁盘空间布局

1块 2块    2块 995块

引导块 FAT1  FAT2 数据区

虚拟磁盘一共划分成1000个磁盘块

每块1024个字节，磁盘空间布局如上

将数据区的第一块（即虚拟磁盘的第6块）分配给根目录文件

\*/

void my\_format()

{

    FILE \*fp;

    fat \*fat1,\*fat2;

    block0 \*b0;

time\_t \*now;

time\_t 时间函数

　　time\_t 类型在time.h中定义：

　　　　#ifndef \_\_TIME\_\_T

　　　　#define \_\_TIME\_T

　　　　typedef long time\_t

　　　　#endif

可见， time\_t 实际上是一个长整型，其值表示从1970年1月1日00时00分00秒（linux系统的Epoch时间）到当前时刻的秒数，由于time\_t类型长度限制；

（long型），它所表示的时间不能晚于2038年1月19日03时14分07秒（UTC），（64位机器不存在此问题，想想why）

使用time（）函数获取当前时间的time\_t值，使用ctime（）函数将time\_t 转为当地时间字符串（返回值是字符串）；

    struct tm \*nowtime;

    unsigned char \*p;

    fcb \*root;

    int i;

    p=myvhard;//虚拟磁盘起始地址

    b0=(block0 \*)p;

    fat1=(fat \*)(p+BLOCKSIZE);

    fat2=(fat\*)(p+3\*BLOCKSIZE);//起始地址加三个磁盘块大小

int num=0xaaaa; /\*文件系统魔数 10101010\*/

魔数通常位于一个对象中，并用来判断对象（如文件）的类型。例如对于a.out格式的执行文件来说，其开始两个字节就是一个魔数，具有特定的值。对于一个文件系统来说，其超级块开始的字节就用来指定文件系统的类型，也是一个魔数。

    b0->root = 5;//根目录起始盘块号为5

    strcpy(b0->information,"My FileSystem Ver 1.0 \n Blocksize=1KB Whole size=1000KB Blocknum=1000 RootBlocknum=2\n");

    /\*

    FAT1,FAT2

        前面五个磁盘块已分配，标记为END

    \*/

    fat1->id=END;

    fat2->id=END;

    fat1++;fat2++;

    fat1->id=END;

    fat2->id=END;

    fat1++;fat2++;

    fat1->id=END;

    fat2->id=END;

    fat1++;fat2++;

    fat1->id=END;

    fat2->id=END;

    fat1++;fat2++;

    fat1->id=END;

    fat2->id=END;

    fat1++;fat2++;

    fat1->id=6;

    fat2->id=6;

    fat1++;fat2++;

    fat1->id=END;

    fat2->id=END;

    fat1++;fat2++;

    /\*

    将数据区的标记为空闲状态

    \*/

    for(i=7;i<SIZE/BLOCKSIZE;i++)

    {

        (\*fat1).id=FREE;

        (\*fat2).id=FREE;

        fat1++;

        fat2++;

    }//同时初始化两个FAT表

    /\*

    创建根目录文件root，将数据区的第一块分配给根目录区

    在给磁盘上创建两个特殊的目录项：".",".."，

    除了文件名之外，其它都相同

    \*/ if(strcmp(p,".")==0) //cd . 仍在当前目录

return;

if(strcmp(p,"..")==0) //cd .. 去往上层目录

    p+=BLOCKSIZE\*5;

    root=(fcb \*)p;

    strcpy(root->filename,".");

    strcpy(root->exname,"di");

    root->attribute=40;

    now=(time\_t \*)malloc(sizeof(time\_t));

    time(now);

    nowtime=localtime(now);

    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++;

    strcpy(root->filename,"..");

strcpy(root->exname,"di");

一个Linux文件能否被执行，与它的第一列的十个属性有关，与文件名一点关系也没有。这个跟windows有点不一样，win下的可执行文件扩展名通常是.com、.exe、.bat等，而在Linux下，只要文件具有x就代表这个文件具有可执行权限。

用ls查看文件属性时

一个文件能否被执行成功不仅与是否具有可执行权限有关，还与文件内容是否可执行有关。

在Linux中虽然文件后缀扩展名并没有什么实际的意义，但文件扩展名有利于我们了解文件的类型，是用来干嘛的，所以通常文件还是会加适当的扩展名。

    root->attribute=40;

    time(now);

    nowtime=localtime(now);

    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++;//初始化两个FCB里所有的值

    for(i=2;i<BLOCKSIZE\*2/sizeof(fcb);i++,root++)

    {

        root->filename[0]='\0';

    }

    fp=fopen(filename,"w");

    b0->startblock=p+BLOCKSIZE\*4;

    fwrite(myvhard,SIZE,1,fp);

    free(now);

    fclose(fp);

}

# void my\_cd

void my\_cd(char \*dirname)

{

char \*p,text[MAX\_TXT\_SIZE];

int fd,i;

p=strtok(dirname,"\\"); //提取出cd的命令

.原型 char \* strtok(char \* s,const char \* delim);

功能 对字符串进行分段处理，每次调用strtok，返回由delim中任意一个字符分隔的一段字符串。

参数说明：@s：输入字符串，且该字符串不能为const类型。

@delim：所有用于将@s进行分隔的字符

@return：每调用一次，返回字符串中的一段，以'\0'结束

使用注意：第一次调用，传s进去，后面的调用都传NULL。

函数实现原理：将@s中的所有delim字符都替换为‘\0'，即原来字符串自动被分为多段，当传入的@s为NULL时，自动调用下一段；@s不为NULL时，返回从@s开始的第一段

eg.

char s[] = "111:222";

char \*p = strtok (s, ":");

printf ("%s\r\n", p); //打印出111

printf ("%s\r\n", s); //同样打印出111，因为原来@s中的":"被替换为'\0'，截断了字符串。

p = strtok (NULL, ":"); //注意传入参数为NULL。

printf ("%s\r\n", p); //打印出222，原@s的第二段字符串

if(strcmp(p,".")==0) //cd . 仍在当前目录

return;

if(strcmp(p,"..")==0) //cd .. 去往上层目录

{

fd=openfilelist[curfd].father;

my\_close(curfd);

curfd=fd;

ptrcurdir=&openfilelist[curfd];

return;

}

if(strcmp(p,"root")==0)

{

for(i=1;i<MAX\_OPEN\_FILE;i++)

{

if(openfilelist[i].free)

my\_close(i);

}

ptrcurdir=&openfilelist[0];

curfd=0;

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

}

while(p) //设置当前目录为该目录

{

fd=my\_open(p);

if(fd>0)

{

ptrcurdir=&openfilelist[fd];

curfd=fd;

}

else

return;

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

}

}

# my\_mkdir

/\*

原型声明: void my\_mkdir(char \*dirname)

功能描述： 创建子目录函数,在当前目录下创建名为dirname的目录

输入：

dirname 指向新建目录的名字的指针

输出：无

函数功能实现算法描述：

\*/

void my\_mkdir(char \*dirname) 创建子目录

{

    fcb \*dirfcb,\*pcbtmp;

    int rbn,i,fd;

    unsigned short bknum;

    char text[MAX\_TXT\_SIZE],\*p;

    time\_t \*now;

    struct tm \*nowtime;

    /\*

    将当前的文件信息读到text中

    rbn 是实际读取的字节数

    \*/

    openfilelist[curfd].count=0;

    rbn = do\_read(curfd,openfilelist[curfd].length,text); //将当前目录文件读入到内存中

    dirfcb=(fcb \*)text;

    //检测是否有相同的目录名

    for(i=0;i<rbn/sizeof(fcb);i++)

    {

        if(strcmp(dirname,dirfcb->filename)==0)

        {

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

            return;

        }

        dirfcb++;

    }

    dirfcb=(fcb \*)text;

    for(i=0;i<rbn/sizeof(fcb);i++)

    {

        if(strcmp(dirfcb->filename,"")==0)

            break;

        dirfcb++;

    }

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

    //寻找一个空闲文件表项

    fd=findFreeO();

    if(fd<0)

    {

        return;

    }

    //寻找空闲盘块

    bknum = findFree();

    if(bknum == END )

    {

        //回收文件占据的用户打开表表项

        openfilelist[fd].attribute=0;

        openfilelist[fd].count=0;

        openfilelist[fd].date=0;

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

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

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

        openfilelist[fd].length=0;

        openfilelist[fd].time=0;

        openfilelist[fd].free=0;

        openfilelist[fd].topenfile=0;

        return -1;

    }

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

    now=(time\_t \*)malloc(sizeof(time\_t));

    //在当前目录下新建目录项

    pcbtmp->attribute=0x30;  //0x30在ASCII码上代表0，即表示文件属性为0（目录文件）

    //获取时间，调整时间格式

    time(now);

    nowtime=localtime(now);

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

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

    strcpy(pcbtmp->filename,dirname);

    strcpy(pcbtmp->exname,"di");

    pcbtmp->first=bknum;

    pcbtmp->length=2\*sizeof(fcb);

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

    openfilelist[fd].count=0;

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

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

    p=openfilelist[fd].dir[fd];

    while(\*p!='\0')

        p++;

    strcpy(p,dirname);

    while(\*p!='\0') p++;

    \*p='\\';p++;

    \*p='\0';

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

    openfilelist[fd].diroff=i;

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

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

    openfilelist[fd].fcbstate=1;

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

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

    openfilelist[fd].free=1;

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

    openfilelist[fd].topenfile=1;

    do\_write(curfd,(char \*)pcbtmp,sizeof(fcb),2);

    pcbtmp->attribute=0x28;

    time(now);

    nowtime=localtime(now);

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

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

    strcpy(pcbtmp->filename,".");

    strcpy(pcbtmp->exname,"di");

    pcbtmp->first=bknum;

    pcbtmp->length=2\*sizeof(fcb);

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

    pcbtmp->attribute=0x28;

    time(now);

    nowtime=localtime(now);

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

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

    strcpy(pcbtmp->filename,"..");

    strcpy(pcbtmp->exname,"di");

    pcbtmp->first=openfilelist[curfd].first;

    pcbtmp->length=openfilelist[curfd].length;

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

    openfilelist[curfd].count=0;

    do\_read(curfd,openfilelist[curfd].length,text);

    pcbtmp=(fcb \*)text;

    pcbtmp->length=openfilelist[curfd].length;

    my\_close(fd);

    openfilelist[curfd].count=0;

    do\_write(curfd,text,pcbtmp->length,2);

}

# my\_rmdir

void my\_rmdir(char \*dirname)

{

    int rbn,fd;

    char text[MAX\_TXT\_SIZE];

    fcb \*fcbptr,\*fcbtmp,\*fcbtmp2;

    unsigned short bknum;

    int i,j;

    fat \*fat1,\*fatptr;

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

    {

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

        return;

    }

    fat1=(fat \*)(myvhard+BLOCKSIZE);

    openfilelist[curfd].count=0;

    //

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

    fcbptr=(fcb \*)text;

    for(i=0;i<rbn/sizeof(fcb);i++)

    {

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

        {

            break;

        }

        fcbptr++;

    }

    if(i >= rbn/sizeof(fcb))

    {

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

        return;

    }

    bknum=fcbptr->first;

    fcbtmp2=fcbtmp=(fcb \*)(myvhard+bknum\*BLOCKSIZE);

    for(j=0;j<fcbtmp->length/sizeof(fcb);j++)

    {

        if(strcmp(fcbtmp2->filename,".") && strcmp(fcbtmp2->filename,"..") && fcbtmp2->filename[0]!='\0')

        {

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

            return;

        }

        fcbtmp2++;

    }

    while(bknum!=END)

    {

        fatptr=fat1+bknum;

        bknum=fatptr->id;

        fatptr->id=FREE;

    }

    strcpy(fcbptr->filename,"");

    strcpy(fcbptr->exname,"");

    fcbptr->first=END;

    openfilelist[curfd].count=0;

    do\_write(curfd,text,openfilelist[curfd].length,2);

}

# my\_ls

/\*

原型声明: void my\_ls()

功能描述： 显示目录函数

输入： 无

输出： 无

函数功能实现算法描述：

\*/

void my\_ls()  {

    fcb \*fcbptr;

    int i;

    char text[MAX\_TXT\_SIZE];

    unsigned short bknum;

    openfilelist[curfd].count=0;

    do\_read(curfd,openfilelist[curfd].length,text);

    fcbptr=(fcb \*)text;

    for(i=0;i<(int)(openfilelist[curfd].length/sizeof(fcb));i++)

    {

        if(fcbptr->filename[0]!='\0')

        {

            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,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++;

    }

    openfilelist[curfd].count=0;

}

# my\_create

/\*

原型声明: void my\_create(char \*filename)

功能描述： 创建文件函数

输入：

filename 指向文件名的指针

输出： 无

函数功能实现算法描述：

\*/

void my\_create(char \*filename)  //ysh

{

    char \*fname,\*exname,text[MAX\_TXT\_SIZE];

    int fd,rbn,i;

    fcb \*filefcb,\*fcbtmp;

    time\_t \*now;

    struct tm \*nowtime;

    unsigned short bknum;

    fat \*fat1,\*fatptr;

    fat1=(fat \*)(myvhard+BLOCKSIZE);

    fname=strtok(filename,".");  //文件名

    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[curfd].count=0;

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

    filefcb=(fcb \*)text;

    for(i=0;i<rbn/sizeof(fcb);i++)

    {

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

        {

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

            return;

        }

        filefcb++;

    }

    filefcb=(fcb \*)text;

    for(i=0;i<rbn/sizeof(fcb);i++)

    {

        if(strcmp(filefcb->filename,"")==0)

            break;

        filefcb++;

    }

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

    bknum=findFree();

    if(bknum==END)

    {

        return ;

    }

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

    now=(time\_t \*)malloc(sizeof(time\_t));

    fcbtmp->attribute=0x00;

    time(now);

    nowtime=localtime(now);

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

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

    strcpy(fcbtmp->filename,fname);

    strcpy(fcbtmp->exname,exname);

    //\*fcbtmp->exname=\*exname;

    fcbtmp->first=bknum;

    fcbtmp->length=0;

    do\_write(curfd,(char \*)fcbtmp,sizeof(fcb),2);

    free(fcbtmp);

    free(now);

    openfilelist[curfd].count=0;

    do\_read(curfd,openfilelist[curfd].length,text);

    fcbtmp=(fcb \*)text;

    fcbtmp->length=openfilelist[curfd].length;

    openfilelist[curfd].count=0;

    do\_write(curfd,text,openfilelist[curfd].length,2);

    openfilelist[curfd].fcbstate=1;

    fatptr=(fat \*)(fat1+bknum);

    fatptr->id=END;

}