#include <sys/types.h> #include <dirent.h>

主要实现ls -al 主要需要Linux下opendir()、readdir()和closedir()这三个函数主要用来遍历目录。

```
####    用DIR和stat函数来辅助实现读取目录文件信息。
```

```
int stat(const char *restrict pathname, struct stat *restrict buf);
```

opendir()函数

> 打开一个目录，在失败的时候返回NULL（如果path对应的是文件，则返回NULL）；
>
> opendir函数的作用是打开一个目录并建立一个目录流。如果成功,它返回一个指向DIR结构的指针,该指针用于读取目录数据项。
>
> opendir在失败时会返回一个空指针。注意,目录流用一个底层文件描述符来访问目录本身,所以如果打开的文件过多,opendir可能会失败。

> 功能：打开目录函数
>
> 头文件：#include <sys/types.h> #include <dirent.h>
>
> 定义函数：DIR * opendir(const char * name);
>
> 函数说明：opendir()用来打开参数name 指定的目录, 并返回DIR*形态的目录流, 和open()类似, 接下来对目录的读取和搜索都要使用此返回值.
>
> 返回值：成功则返回DIR* 型态的目录流, 打开失败则返回NULL.
>
> 错误代码： 1、EACCESS 权限不足。 2、EMFILE 已达到进程可同时打开的文件数上限。 3、ENFILE 已达到系统可同时打开的文件数上限。 4、ENOTDIR 参数name 非真正的目录。 5、ENOENT 参数name 指定的目录不存在, 或是参数name 为一空字符串。 6、ENOMEM 核心内存不足。

readdir()函数

> readdir函数将返回一个指针,指针指向的结构里保存着目录流drip中下一个目录项的有关资料。每调用一次
>
> readdir，就有一个文件名保存到d_name [NAME_MAX+1]中，后续的readdir调用将返回后续的目录项，如果发生错误或者到达目录尾,readdir将返回NULL。POSIX兼容的系统在到达目录尾时会返回NULL,但并不改变errno的值,在发生错误时才会设置errno。

> 注意,如果在readdir函数扫描目录的同时还有其他进程在该目录里创建或删除文件,readdir将不保证能够列出该目录里的所有文件(和子目录)。

> 本函数读取dir_handle目录下的目录项，如果有未读取的目录项，返回目录项，否则返回NULL。 循环读取dir_handle,目录和文件都读
>
> 函数原型：struct dirent * readdir(DIR * dir_handle);
>
> 返回dirent结构体指针，dirent结构体成员如下，（文件和目录都读）
>
> 函数说明：readdir()返回参数dir 目录流的下个目录进入点。结构dirent 定义如下：

```
struct dirent
{
  ino_t d_ino; //d_ino 此目录进入点的inode
  ff_t d_off; //d_off 目录文件开头至此目录进入点的位移
  signed short int d_reclen; //d_reclen _name 的长度, 不包含NULL 字符
  unsigned char d_type; //d_type d_name 所指的文件类型 d_name 文件名
  har d_name[256];
};
```

> 返回值：成功则返回下个目录进入点. 有错误发生或读取到目录文件尾则返回NULL.
>
> 附加说明：EBADF 参数dir 为无效的目录流。

closedir()函数

> 功能：关闭目录
>
> 相关函数：opendir()
>
> 头文件：#include <sys/types.h> #include <dirent.h>
>
> 定义函数：int closedir(DIR *dir);
>
> 函数说明：closedir()关闭参数dir 所指的目录流。
>
> 返回值：关闭成功则返回0,，失败返回-1,，错误原因存于errno 中。
>
> 错误代码：EBADF 参数dir 为无效的目录流。

DIR结构体：

> DIR结构体类似于FILE，是一个内部结构，以下几个函数用这个内部结构保存当前正在被读取的目录的有关信息。
>
> 函数 DIR *opendir(const char *pathname)，即打开文件目录，返回的就是指向DIR结构体的指针，而该指针由以下几个函数使用:

```
struct dirent *readdir(DIR *dp);

void rewinddir(DIR *dp);

int closedir(DIR *dp);

long telldir(DIR *dp);

void seekdir(DIR *dp,long loc);
```

stat结构体：

> 目标：得到文件的属性
>
> 头文件： #include<sys/stat.h>
>
> 函数原型： int stat(const char * file_name, struct stat *buf);
>
> 参数： file_name 文件名, buf 指向buffer的指针
>
> 返回值： -1 错误， 0 成功

> 通过readdir函数读取到的文件名存储在结构体dirent的d_name成员中，而函数 int stat(const char *file_name, struct stat *buf);
>
> 的作用就是获取文件名为d_name的文件的详细信息，存储在stat结构体中。以下为stat结构体的定义：

```
struct stat {

        mode_t     st_mode;        //文件访问权限

        ino_t      st_ino;        //索引节点号

        dev_t      st_dev;         //文件使用的设备号

        dev_t      st_rdev;        //设备文件的设备号

        nlink_t    st_nlink;       //文件的硬连接数

        uid_t      st_uid;         //所有者用户识别号

        gid_t      st_gid;         //组识别号

        off_t      st_size;        //以字节为单位的文件容量

        time_t     st_atime;       //最后一次访问该文件的时间

        time_t     st_mtime;       //最后一次修改该文件的时间
```

```
        time_t     st_ctime;       //最后一次改变该文件状态的时间
        blksize_t st_blksize;     //包含该文件的磁盘块的大小
        blkcnt_t   st_blocks;      //该文件所占的磁盘块
      };
```

编译时，ceil未识别：

> 函数库中未相关设置。
>
> 解决：ceil()是#include <math.h>中的函数,虽然程序中已经包含了该头文件,但是编译的时候还是说这个函数没有定义(也就是"对ceil未定义的引用)，需要重新编译,并在最后加上"-lm"
>
> 补充：lm选项告诉编译器，我们程序中用到的数学函数要到这个库文件里找. 同时，常见的库链接方法为：数学库 -lm ; posix线程 -lpthread lc 是link libc lm 是link libm lz 是link libz

c语言实现ls -al:

```
#include <stdio.h>
#include <dirent.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <string.h>
#include <math.h> // ceil

/*操作获得文件的数据*/
void do_ls (char []);
void dostat(char *);
void show_file_info(char *, struct stat *);
void mode_to_letters(int , char []);
char *uid_to_name(uid_t);
char *gid_to_name(gid_t);

/*下面三个声明是关于总量输出操作的*/
void do_ls1(char []);
void dostat1(char *);
void show_file_info1(struct stat *);

/*用于计算总量的数据*/
long int value;
float ans;

void main(int ac, char *av[]) { //主函数
    int ac1 = ac;    // 记录先输出总量的个数
```

```
    if (ac1 == 1) {    // 输出总量的入口
        do_ls1(".");    // 当文件只有1时， 那么就只有个点
    } else {
        while(--ac1) {
            do_ls1(*av);
        }
    }
    printf("总量为 %ld\n",value);

    if (ac == 1) {//操作获得其他信息
        do_ls(".");
    } else {
        while(--ac) {
            do_ls(*av);
        }
    }

}

void do_ls1(char dirname1[]) {
    DIR *dir_list1; //目录操作指针
    struct dirent *dirfile1;//各个目录

    if (( dir_list1 = opendir(dirname1)) == NULL) { //要是目录本身不存在
        fprintf(stderr, "no open the %s lists!\n", dirname1);
    } else {    //有能操作的目录 那么循环取操作
        while(( dirfile1 = readdir(dir_list1)) != NULL) {//readdir调用后下一次调用就是下一
个数据
            dostat1(dirfile1->d_name);
        }
        closedir(dir_list1);//关闭目录
    }
}

//功能同上
void do_ls(char dirname[]) {
    DIR *dir_list; //目录操作指针
    struct dirent *dirfile;//各个目录

    if (( dir_list = opendir(dirname)) == NULL) {
        fprintf(stderr, "no open the %s lists!\n", dirname);
    } else {
        while(( dirfile = readdir(dir_list)) != NULL) {
            dostat(dirfile->d_name);  // 传入文件名为d_name的详细信息
        }
        closedir(dir_list);
    }
}


void dostat1(char *filename) {
    struct stat info;
    if (stat(filename, &info) == -1)//打开错误
```

```
    perror(filename);                   //perror(s) 用来将上一个函数发生错误的原因输出到标准设备(stderr)。参数 s 所指的字符串会先打印出，后面再加上错误原因字符串。此错误原因依照全局变量errno的值来决定要输出的字符串。
    else {                              //打开
          show_file_info1(&info);     // 传入
    }
}

void show_file_info1(struct stat *info_open) {
    ans = ceil ((info_open->st_size) * 1.0 / 4096); // 每个文件都被计算block块数 不足1块按一块占算
    value += (int)ans * 4; // 1个block是4k 累加计算total的总量
}

//功能同上
void dostat(char *filename) {
    struct stat info;
    if (stat(filename, &info) == -1)//打开错误
    perror(filename);
    else {                              //打开
          show_file_info(filename, &info);//多传入了一个d_name的详细信息

    }
}

void show_file_info(char *filename, struct stat *info_open) {
    char *uid_to_name(), *ctime(), *gid_to_name(), *filemode();
    void mode_to_letters();
    char modestr[11]; // 前10位做类型显示
    mode_to_letters(info_open->st_mode, modestr);//计算文件类型和许可权限

    /*输出各种数据*/
    printf("%s ", modestr);
    printf("%ld ", info_open->st_nlink);
    printf("%-8s", uid_to_name(info_open->st_uid));
    printf("%-8s", gid_to_name(info_open->st_gid));
    printf("%8ld\t", info_open->st_size);
    printf("%-8.12s ", 4 + ctime(&info_open->st_mtime));

    if (modestr[0] == 'd') {
        printf("\033[1m\033[34m %-16s\n\033[0m", filename);
    } else {
        printf("%-16s\n",filename);
    }

}

void  mode_to_letters(int mode, char str[]) {  // 利用掩码转化字符
    strcpy(str, "----------"); //填入字符
    /*下面的数值是#include <sys/stat.h> 里面的宏定义值

    /*文件类型*/
    if (S_ISDIR(mode)) str[0] = 'd';
```

```
    if (S_ISCHR(mode)) str[0] = 'c';
    if (S_ISBLK(mode)) str[0] = 'b';

    /*user*/
    if (mode & S_IRUSR) str[1] = 'r';
    if (mode & S_IWUSR) str[2] = 'w';
    if (mode & S_IXUSR) str[3] = 'x';

    /*group*/
    if (mode & S_IRGRP) str[4] = 'r';
    if (mode & S_IWGRP) str[5] = 'w';
    if (mode & S_IXGRP) str[6] = 'x';

    /*other*/
    if (mode & S_IROTH) str[7] = 'r';
    if (mode & S_IWOTH) str[8] = 'w';
    if (mode & S_IXOTH) str[9] = 'x';
}

#include <pwd.h> // 口令文件

char *uid_to_name(uid_t uid) { // 找出uid
    struct passwd *getpwuid(), *pw_ptr;
    static char numstr[10];

    if ((pw_ptr = getpwuid(uid)) == NULL) {
        sprintf(numstr, "%d", uid);
        return numstr;
    } else {
        return pw_ptr->pw_name;
    }
}

#include <grp.h> // 组文件

char *gid_to_name(gid_t gid) {// 找出gid
    struct group *getgrgid(), *grp_ptr;
    static char numstr[10];

    if ((grp_ptr = getgrgid(gid)) == NULL) {
        sprintf(numstr, "%d", gid);
        return numstr;
    } else {
        return grp_ptr->gr_name;
    }
}
```