Linux 上机作业——“遍历目录”实验报告

一、 实验题目

编程实现程序 list.c，列表普通磁盘文件， 包括文件名和文件大小。

1) 使用 vi 编辑工具，熟悉工具 vi。

2) 使用 Linux 的系统调用和库函数。

3) 体会 shell 文件通配符的处理方式以及命令对选项的处理方式。

对选项的处理，自行编程逐个分析命令行参数，不考虑多选项挤在一个命令行参数内的情况。

二、 处理对象和选项

1. 处理对象可以有 0 个到多个
   1) 0 个：列出当前目录下所有文件
   2) 普通文件：列出文件
   3) 目录：列出目录下所有文件
2. 实现自定义选项 r,a,l,h,m 以及--
   1) r：递归方式列出子目录（每一项需要包含路径）
   2) a：列出文件名第一个字符为圆点的普通文件
   3) l：后面跟一个整数，限定文件大小的最小值（字节）
   4) h：后面跟一个整数，限定文件大小的最大值（字节）
   5) m：后面跟一个整数，限定文件的最近修改时间必须在 n 天内
   6) --：显式地终止命令选项分析

三、 整体思路

1. dirent 结构体和 stat 结构体。

   次实验需要通过路径对文件及文件夹的一些属性进行访问，根据条件筛选，而获取这些信息我们需要使用到 dirent 结构体以及 stat 结构体，

查询到这两个结构体内所包含的信息为：

```
struct dirent
{
    long d_ino; /*索引节点号 */
    off_t d_off; /*在目录文件中的偏移 */
    unsigned short d_reclen; /*文件名长 */
    unsigned char d_type; /*文件类型 */
    char d_name [NAME_MAX+1]; /*文件名，最长 255 字符 */
}
```

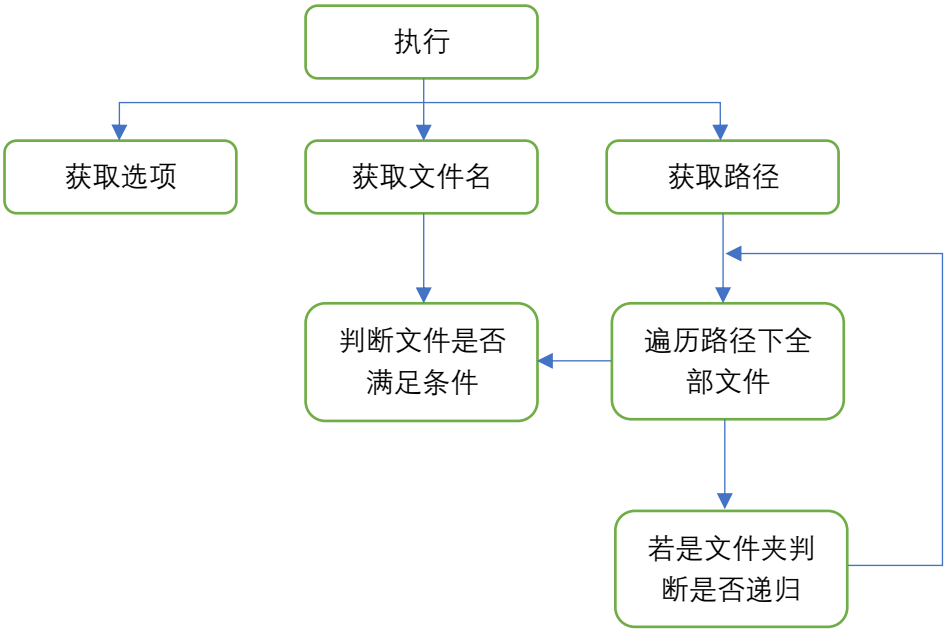
```
struct stat {
        mode_t     st_mode;    //文件访问权限
        ino_t      st_ino;     //索引节点号
        dev_t      st_dev;     //文件使用的设备号
        dev_t      st_rdev;    //设备文件的设备号
        nlink_t    st_nlink;   //文件的硬连接数
        uid_t      st_uid;     //所有者用户识别号
        gid_t      st_gid;     //组识别号
        off_t      st_size;    //以字节为单位的文件容量
        time_t     st_atime;   //最后一次访问文件的时间
        time_t     st_mtime;   //最后一次修改文件的时间
        time_t     st_ctime;  //最后一次改变文件状态的时间
        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
};
```

红色为根据自定义选项功能发现主要需要查看的内容。

2. 逻辑结构

根据处理对象的数量，0 个或多个，需要分别讨论，如果具有多个处理对象，需根据处理对象的不同（路径或文件）分别保存。其中对于文件来

说只需要考虑这一个文件即可，但是对于路径而言，依次要进行打开目录，遍历文件，关闭目录这些步骤，而且如果需要递归查看文件目录就需要重复执行上述步骤，大致流程如下：



四、 函数功能及实现

1. 全局变量：

```
int r = 0;      //递归列出子目录，需要列出时为 1
int a = 0;      //列出隐藏文件，需要时为 1
int l = -1;     //限定文件最小字节，-1 表示不限制
int h = -1;     //限定文件最大字节，-1 表示不限制
int m = -1;     //限定文件最最近修改天数，-1 表示不限制
```

2. 主函数需要分析指令，将指令、文件、路径分隔开：

```
int main(int argc, char *argv[]){
    if(argc == 1){
        //没有参数，显示当前目录文件
    }
    else{
        //多个参数
```

```
        while(全部参数){
            if(argv[i][0] == '-')
                //分析指令
            else if(argv[i][0] == '/')
                //分析路径
            else
                //文件
        }
    }
}
```

3. 以 get_all_files_under_path 函数为例,这个函数用于遍历路径中的文件,同时借由全局变量判断是否将文件信息打印或是递归文件夹:

```
void get_all_files_under_path(char path[]){
    DIR *dp;
    struct dirent *file;
    dp = opendir(path);
    if(!dp)打开文件夹失败
    else{
        while((file = readdir(dp)) != NULL){
            if(file->d_type != 4)
                //普通文件
            else{
                //文件夹
                if(r==1 && 文件夹不为本文件夹或上级文件夹)
                    //递归调用此函数
            }
        }
    }
}
```

4. 在 limit 函数中，需要对文件夹信息进行判断从而决定需不需要打印：

```
void limit(struct dirent *file, char path[]){
    struct stat statbuf;
    stat(file->d_name, &statbuf);
    if(a==0)
        //文件名开头为'.'则不显示
    if(l!=-1)
        //文件大小小于 l 则不显示
    if(h!=-1)
        //文件大小大于 h 则不显示
    if(m!=-1)
        //文件距离最近修改时间天数大于 m 则不显示
}
```

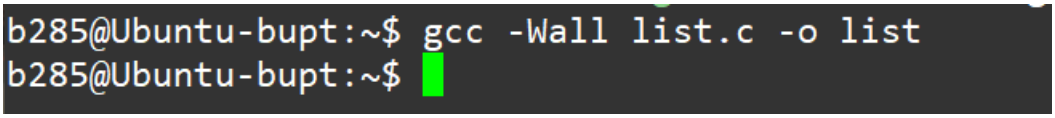
5. folder_size 函数用于计算文件夹的大小，包括其中的隐藏文件夹：

```
void folder_size(char path[]){
    while(遍历文件)
        if(普通文件)
            size+=普通文件大小
        else(文件夹)
            folder_size(char path[])
}
```

6. 除此之外，还有 get_file(char file[]), string_to_number(char str[]) 等函数用于其他功能。

五、 用例测试

首先编译 list.c 文件：



1. ./list:列出当前工作目录下全部文件。

```
b285@Ubuntu-bupt:~$ ./list
805                 beijing.csv         /home/b285
47                  awk1.txt            /home/b285
75                  awk2.txt            /home/b285
21792               list                /home/b285
4431                folder              /home/b285
21831               beijing.html        /home/b285
0                   -a                  /home/b285
2041                list1.c             /home/b285
135                 sed.txt             /home/b285
17272               list1               /home/b285
6353                list.c              /home/b285
b285@Ubuntu-bupt:~$
```

2. ./list -l 10 -h 1000：列出目录下大小在 10-100 字节的文件。

```
b285@Ubuntu-bupt:~$ ./list -l 10 -h 1000
805                 beijing.csv         /home/b285
47                  awk1.txt            /home/b285
75                  awk2.txt            /home/b285
135                 sed.txt             /home/b285
b285@Ubuntu-bupt:~$
```

3. ./list -a:列出隐藏文件。

```
b285@Ubuntu-bupt:~$ ./list -a
13938                   .config                 /home/b285
805                     beijing.csv             /home/b285
183                     .wget-hsts              /home/b285
4096                    .                       /home/b285
807                     .profile                /home/b285
220                     .bash_logout            /home/b285
47                      awk1.txt                /home/b285
4096                    .cache                  /home/b285
20480                   ..                      /home/b285
75                      awk2.txt                /home/b285
21792                   list                    /home/b285
4431                    folder                  /home/b285
3771                    .bashrc                 /home/b285
21831                   beijing.html            /home/b285
10906                   .viminfo                /home/b285
0                       -a                      /home/b285
2041                    list1.c                 /home/b285
135                     sed.txt                 /home/b285
26983                   .bash_history           /home/b285
17272                   list1                   /home/b285
6353                    list.c                  /home/b285
4201                    .local                  /home/b285
```

4. ./list -r -a:递归列出全部文件。

```
220                     .bash_logout            /home/b285
47                      awk1.txt                /home/b285
4096                    .                       /home/b285/.cache
20480                   ..                      /home/b285/.cache
20480                   motd.legal-displayed/home/b285/.cache
20480                   ..                      /home/b285
75                      awk2.txt                /home/b285
17640                   list                    /home/b285
928184403               3.txt                   /home/b285/folder
4096                    .                       /home/b285/folder
4096                    2.txt                   /home/b285/folder
20480                   ..                      /home/b285/folder
20480                   1.txt                   /home/b285/folder
4096                    .                       /home/b285/folder/folder1
4096                    one.txt                 /home/b285/folder/folder1
20480                   ..                      /home/b285/folder/folder1
20480                   two.txt                 /home/b285/folder/folder1
3771                    .bashrc                 /home/b285
21831                   beijing.html            /home/b285
10754                   .viminfo                /home/b285
0                       -a                      /home/b285
135                     sed.txt                 /home/b285
23689                   .bash_history           /home/b285
5260                    list.c                  /home/b285
4096                    .                       /home/b285/.local
20480                   ..                      /home/b285/.local
932184383               user.keystore           /home/b285/.local/share/keyrings
4096                    .                       /home/b285/.local/share/keyrings
20480                   ..                      /home/b285/.local/share/keyrings
20480                   login.keyring           /home/b285/.local/share/keyrings
4096                    .                       /home/b285/.local/share
20480                   ..                      /home/b285/.local/share
b285@Ubuntu-bupt:~$
```

5. ./list -- -a：显示终止命令分析

```
b285@Ubuntu-bupt:~$ ./list -- -a
0                       -a                              /home/b285
b285@Ubuntu-bupt:~$
```

6. ./list -r -m 3：修改时间必须在三天内

```
b285@Ubuntu-bupt:~$ ./list -r -m 3
4096                68a06708ce184970870837a446e9beb3-default-source/home/
b285/.config/pulse
4096                68a06708ce184970870837a446e9beb3-stream-volumes.tdb/h
ome/b285/.config/pulse
4096                68a06708ce184970870837a446e9beb3-default-sink/home/b2
85/.config/pulse
20480               68a06708ce184970870837a446e9beb3-card-database.tdb/ho
me/b285/.config/pulse
20480               68a06708ce184970870837a446e9beb3-device-volumes.tdb/h
ome/b285/.config/pulse
20480               cookie              /home/b285/.config/pulse
20480               motd.legal-displayed/home/b285/.cache
17640               list                /home/b285
4096                2.txt               /home/b285/folder
20480               1.txt               /home/b285/folder
4096                one.txt             /home/b285/folder/folder1
20480               two.txt             /home/b285/folder/folder1
0                   -a                  /home/b285
5260                list.c              /home/b285
20480               login.keyring       /home/b285/.local/share/keyrings
b285@Ubuntu-bupt:~$
```

六、 实验总结

本次实验通过对一条 ls 命令的简单实现，让我体会到了程序员与使用者是如何进行信息交换的，每一条看似简单的命令背后都有大量代码作为支撑，从而让我们方便快捷使用计算机。其中我认识到了 dirent 和 stat 结构体，两者相互配合可以达到遍历文件夹的目的，归根到底这两者背后也得有大量代码才能实现，其实我在这里既扮演程序员也扮演使用者的角色。

七、 源代码

```
 #include <stdio.h>
#include <dirent.h>
#include <sys/stat.h>
```

```
#include <unistd.h>
#include <string.h>
#include <time.h>

int r = 0;
int a = 0;
int l = -1;
int h = -1;
int m = -1;
long size = 0;

int string_to_number(char str[]){
    int i = 0;
    int sum = 0;
    while(i < strlen(str)){
        sum += str[i] - '0';
        sum *= 10;
        i++;
    }
    return sum/10;
}

int is_num(char str[]){
    int i=0;
    while(i < strlen(str)){
        if(str[i] >= '0' && str[i] <= '9'){
            i++;
        }
        else{
            return 0;
        }
    }
    return 1;
}

void folder_size(char path[]){
    DIR *dp;
    struct dirent *file;
    dp = opendir(path);
    if(!dp){
        printf("%s 2open dir error\n", path);
    }
    else{
        while((file = readdir(dp)) != NULL){
```

```
            char old[1024];
            strcpy(old, path);
            strcat(path, "/");
            strcat(path, file->d_name);
            if(file->d_type != 4){
                struct stat statbuf;
                stat(path, &statbuf);
                size += statbuf.st_size;
            }
            else    if(strcmp(file->d_name,    ".")    !=    0    &&
strcmp(file->d_name, "..") != 0){
                folder_size(path);
            }
            strcpy(path, old);
        }
    }
    closedir(dp);
}

void limit(struct dirent *file, char path[]){
    char old[1024];
    struct stat statbuf;
    stat(file->d_name, &statbuf);
    //不显示隐藏文件
    if(a == 0){
        if(file->d_name[0] == '.'){
            return;
        }
    }
    //限制文件最小大小
    if(l != -1){
        if(statbuf.st_size < l){
            return;
        }
    }
    //限制文件最大大小
    if(h != -1){
        if(statbuf.st_size > h){
            return;
        }
    }
    //限制文件最近修改时间
    if(m != -1){
        time_t now = time(NULL);
```

```
        int interval = (now - statbuf.st_mtime) / 86400;
        if(interval > m){
            return;
        }
    }
    if(file->d_type  ==  4  &&  strcmp(file->d_name,  ".")  !=  0  &&
strcmp(file->d_name, "..") != 0){
        size = 0;
        strcpy(old, path);
        strcat(path, "/");
        strcat(path, file->d_name);
        folder_size(path);
        strcpy(path, old);
        printf("%-20ld%-20s%s\n", size + statbuf.st_size, file->d_name,
path);
    }
    else
        printf("%-20ld%-20s%s\n", statbuf.st_size, file->d_name, path);
}

void get_all_files_under_path(char path[]){
    DIR *dp;
    struct dirent *file;
    dp = opendir(path);
    if(!dp){
        printf("open dir error");
    }
    else{
        while((file = readdir(dp)) != NULL){
            /*普通文件*/
            if(file->d_type != 4){
                limit(file, path);
            }
            /*文件夹*/
            else{
                if(r  ==  1  &&  strcmp(file->d_name,  ".")  !=  0  &&
strcmp(file->d_name, "..") != 0){
                    char old_path[1024];
                    strcpy(old_path, path);
                    strcat(path, "/");
                    strcat(path, file->d_name);
                    get_all_files_under_path(path);
                    strcpy(path, old_path);
                }
```

```
                else{
                    limit(file, path);
                }
            }
        }
        closedir(dp);
    }
}

void get_file(char file[]){
    char pwd[1024];
    if(!getcwd(pwd, 1024)){
        return;
    }
    struct stat statbuf;
    if(stat(file, &statbuf) == -1){
        return;
    }
    //不显示隐藏文件
    if(a == 0){
        if(file[0] == '.'){
            return;
        }
    }
    //限制文件最小大小
    if(l != -1){
        if(statbuf.st_size < l){
            return;
        }
    }
    //限制文件最大大小
    if(h != -1){
        if(statbuf.st_size > h){
            return;
        }
    }
    //限制文件最近修改时间
    if(m != -1){
        time_t now = time(NULL);
        int interval = (now - statbuf.st_mtime) / 86400;
        if(interval > m){
            return;
        }
    }
```

```
    printf("%-20ld%-20s%s\n", statbuf.st_size, file, pwd);
}

void analyze(char *argv[], int *flag, int *i){
    *i = *i + 1;
    if(is_num(argv[*i]) == 1){
        *flag = string_to_number(argv[*i]);
    }
    else{
        *i = *i - 1;
    }
}

int main(int argc, char *argv[]){
    char path[1024][1024];
    char file[128][128];
    memset(path, '\0', sizeof(path));
    memset(file, '\0', sizeof(file));
    int pathindex = 0, fileindex = 0;
    //0个参数
    if(argc == 1){
        if(!getcwd(path[0], 1024)){
            return 0;
        }
        get_all_files_under_path(path[0]);
    }
    //多个参数
    else{
        int i = 1;
        while(i < argc){
            //指令
            if(argv[i][0] == '-'){
                if(argv[i][1] == 'r'){
                    r = 1;
                }
                else if(argv[i][1] == 'a'){
                    a = 1;
                }
                else if(argv[i][1] == 'l' && i < argc - 1){
                    analyze(argv, &l, &i);
                }
                else if(argv[i][1] == 'h' && i < argc - 1){
                    analyze(argv, &h, &i);
                }
```

```
                else if(argv[i][1] == 'm' && i < argc - 1){
                    analyze(argv, &m, &i);
                }
                else if(argv[i][1] == '-'){
                    i++;
                    strcpy(file[fileindex], argv[i]);
                    fileindex++;
                }
            }
            //路径
            else if(argv[i][0] == '/') {
                strcpy(path[pathindex], argv[i]);
                pathindex++;
            }
            //文件
            else{
                strcpy(file[fileindex], argv[i]);
                fileindex++;
            }
            i++;
        }
        //输出结果
            if((path[0][0] == '\0' && file[0][0] == '\0') || file[0][0]
== '*'){
                if(!getcwd(path[0], 1024)){
                    return 0;
                }
                get_all_files_under_path(path[0]);
            }
            else{
                int i = 0;
                while(path[i][0] != '\0'){
                get_all_files_under_path(path[i]);
                i++;
            }
            i = 0;
            while(file[i][0] != '\0'){
                get_file(file[i]);
                i++;
            }
        }
    }
}
```