Linux 开发环境及应用实验报告

实验二：遍历目录

付容天

学号 2020211616

班级 2020211310

计算机学院（国家示范性软件学院）

2023 年 3 月 31 日

1 实验目的

在本实验中，我们需要完成：

（1） 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；

（2） 使用 vi 编辑文件，熟悉工具 vi；

（3） 使用 Linux 的系统调用和库函数；

（4） 体会 Shell 文件通配符处理方式以及命令对选项的处理方式。其中，对选项的处理，要求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。

2 实验要求

本实验要求实现处理对象数量的不同情况（类似 ls 命令），如下所示：

（1） 0 个：列出当前目录下的所有文件；

（2） 普通文件：列出文件；

（3） 目录：列出目录下的所有文件。

并且，本实验要求实现以下自定义选项（类似 ls 命令）：

（1） r：递归方式给出子目录（每项要包含路径，类似 find 和-print 风格），需要设计递归程序；

（2） a：列出文件名第一个字符为圆点的普通文件（默认情况下不列出文件名首字符为圆点的文件）；

（3） l：后跟一个整数，限定文件大小的最小值（字节）；

（4） h：后跟一个整数，限定文件大小的最大值（字节）；

（5） m：后跟一个整数，限定文件最近修改时间（天数）；

（6） （双横杠）--：显示地终止命令选项分析。

3 实验分析与实验过程

3.1 获取命令行参数

在该实验中，我们需要对目录进行遍历，而且需要实现要求的功能，这里我们就需要获取到输入的命令行参数。命令行参数的获取通过 main 函数的两个参数即可实现。同时，我们设计相应的数据结构表示是否读入了对应的标识符。而且，我们还需要记录由命令行输入的路径，观察可得，所有输入路径都是由字符“/”开始的，故可以利用这一特点实现路径识别。代码如下所示：

```
struct flag() {
    bool r_flag;      // r命令
    bool a_flag;      // a命令
    bool l_flag;      // l命令
    bool h_flag;      // h命令
    bool m_flag;      // m命令
    int l_length;     // l命令指定最小程度
    int h_length;     // h命令指定最大长度
    int m_days;       // m命令指定时间
    int f_d_sum;      // "目录/文件名"的个数
    int f_d[M_LEN];   // 存储"目录/文件名"
};

struct flag flagInit() {
    struct flag FLAG;
    FLAG.r_flag = false;
    FLAG.a_flag = false;
    FLAG.l_flag = false;
    FLAG.h_flag = false;
    FLAG.m_flag = false;
    FLAG.l_length = 0;
    FLAG.h_length = 0;
    FLAG.m_days = 0;
    FLAG.f_d_sum = 0;
    return FLAG;
};

void getFlag(int argc, char *argv[]) {
    int i = 0;
    bool flagCancle = false;
    for (i = 1; i < argc; i++) {
```

```
        if (argv[i][0] == '-' && argv[i][1] == '-') {
            flagCancle = true; // 识别到--选项则终止分析
            continue;
        }
        if (argv[i][0] == '-' && !flagCancle) {
            switch (argv[i][1]) {
            case 'r':
                FLAG.r_flag = true;
                break;
            case 'a':
                FLAG.a_flag = true;
                break;
            case 'l':
                FLAG.l_flag = true;
                i++;
                FLAG.l_length = atoi(argv[i]);
                break;
            case 'h':
                FLAG.h_flag = true;
                i++;
                FLAG.h_length = atoi(argv[i]);
                break;
            case 'm':
                FLAG.m_flag = true;
                i++;
                FLAG.m_days = atoi(argv[i]);
                break;
            default:
                break;
            }
        }
        else {
            FLAG.f_d[FLAG.f_d_sum] = i;
            FLAG.f_d_sum++;
        }
    }
};
```

代码段 1：命令行输入参数的获取与处理

3.2 路径信息处理

在这一部分中，我们需要实现的是路径信息的打印。我们编写了相应的函数 print_path，该函数根据前一阶段收集到的 flag 信息，进行筛选和处理后，打印出文件相关信息，打印格式为“文件大小 文件路径/文件名”。代码如下所示：

```
void print_path(char *path) {
    struct stat st;
    time_t t_now;
    char *fileName;
    time(&t_now);
    int ret = stat(path, &st);
    if (ret == -1)
        printf("%s: No such file or directory\n", path);
    else {
        if (FLAG.l_flag)
            if (st.st_size < FLAG.l_length)
                return;
        if (FLAG.h_flag)
            if (st.st_size > FLAG.h_length)
                return;
        if (FLAG.m_flag)
            if (t_now - st.st_mtime > FLAG.m_days * 24 * 60 * 60)
                return;
        printf("%10ld  %s\n", st.st_size, path);
    }
}
```

代码段 2：文件路径信息处理与打印

在上面的代码中，我们用到了结构 stat，这个结构体在 stat.h 和 types.h 下有相应的声明，专门用来处理路径问题。stat 结构体内容如下：

```
struct stat {
    dev_t     st_dev;           // 存储该文件的块设备的设备号ID
    ino_t     st_ino;           // inode号
    mode_t    st_mode;          // 访问权限及文件类型
    nlink_t   st_nlink;         // link数
    uid_t     st_uid;           // 文件主ID
    gid_t     st_gid;           // 组ID
    dev_t     st_rdev;          // device ID (if special file)
```

```
    off_t     st_size;          // 文件大小（字节数
    blksize_t st_blksize;       // blocksize for filesystem I/O
    blkcnt_t  st_blocks;        // 分配的512字节尺寸块个数
    struct timespec st_atim;    // access时间
    struct timespec st_mtim;    // modification时间
    struct timespec st_ctim;    // change时间
};
```

代码段 3：头文件中对 stat 结构体的定义

并且，我们可以利用 stat 结构体中的 st_mode 属性来判断当前路径的真实意义，该属性具有如下的可能性：

判断变量	变量意义
S_ISREG	普通磁盘文件
S_ISDIR	目录文件
S_ISCHR	字符设备文件
S_ISIFO	管道文件
S_ISLNK	符号连接文件

表 1：st_mode 的不同属性及相应意义

3.3 递归遍历目录

有了上面的功能后，现在我们只需要实现遍历目录的功能，该功能结合输入的目录以及对应的深度进行遍历，从而对所有目录及其子目录进行完整地处理。该部分代码如下所示：

```
void scan_dir(char *dir, int depth) {
    DIR *dp;                    // 子目录流指针
    struct dirent *entry;       // 该指针保存后续目录
    struct stat statbuf;        // 保存文件属性
    char path[512] = {0};
    if ((dp = opendir(dir)) == NULL) {
        printf("%s\n:No such file or directory!!!\n", dir);
        return;
    }
    while ((entry = readdir(dp)) != NULL) {
        if (strcmp(".", entry->d_name) == 0 ||
            strcmp("..", entry->d_name) == 0)
            continue;
        if (entry->d_name[0] == '.' && FLAG.a_flag == false)
```

```
            continue;
        if (depth == 0)
            sprintf(path, "%s", entry->d_name);
        else
            sprintf(path, "%s/%s", dir, entry->d_name);

        lstat(path, &statbuf);
        if (S_ISDIR(statbuf.st_mode))
            if (FLAG.r_flag) // 递归处理
                scan_dir(path, depth + 1);
        else
            print_path(path);
    }
    closedir(dp);
}
```

代码段 4：递归扫描文件目录

3.4 主函数编写

在编写完本次实验所需的所有函数后，接下来我们编写主函数。本实验用到的所有功能之间的调用关系如下图所示：

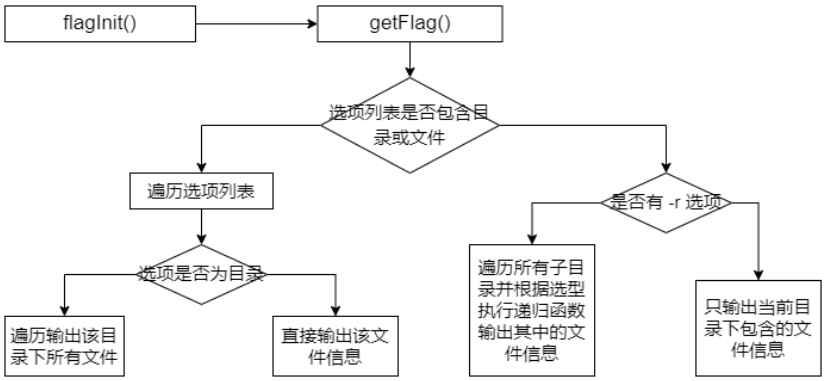

图 1：本实验功能总体调度方案

结合上图以及前面所完成的功能模块，我编写了如下所示的主函数：

```
int main(int argc, char *argv[]) {
    FLAG = flagInit();
    getFlag(argc, argv);

    if (FLAG.f_d_sum == 0) {
        char _path[M_LEN];
        getcwd(_path, M_LEN);//当前目录路径
```

```
        scan_dir(_path, 0);
    }
    else {
        for (int i = 0; i < FLAG.f_d_sum; i++) {
            struct stat st;
            lstat(argv[opt.f_d[i]], &st);
            if (S_ISDIR(st.st_mode))
                scan_dir(argv[FLAG.f_d[i]], 100);
            else
                print_path(argv[FLAG.f_d[i]]);
        }
    }
    return 0;
}
```

代码段 5：主函数

3.5 实验结果展示

在上述 5 个代码片段的基础上，增加必要的头文件，编写得到 list.c 文件，并用 gcc 编译出可执行文件。我预先在目录下创建了一些测试文件夹和测试文件，通过 WinSCP 软件查看其结构如下所示：

图 2：目录总体结构

图 3：两个子目录内容

首先测试“*”：输出所有不是“.”打头的文件及其大小，如下所示：

```
c1616@Ubuntu-bupt:~$ ./list *
        33  dir1/test1.txt
       134  dir2/test2.txt
        96  dir2/test3.txt
     15968  hello
        71  hello.cpp
     16560  list
      5125  list.c
       137  new.txt
       127  test.txt
        46  try.txt
```

图 4：“*”测试

然后测试-l 和-h 选项，列出文件大小在 1 到 100 的文件，如下图所示：

```
c1616@Ubuntu-bupt:~$ ./list -l 1 -h 100
        71  hello.cpp
        46  try.txt
```

图 5：“-l”与“-h”测试

然后测试-r 选项，递归列出当前目录树下大小超过 10 的文件，如下所示：

```
c1616@Ubuntu-bupt:~$ ./list -a -r -l 10
       696  .config/pulse/68a06708ce184970870837a446e9beb3-stream-volumes.tdb
       696  .config/pulse/68a06708ce184970870837a446e9beb3-card-database.tdb
      8192  .config/pulse/68a06708ce184970870837a446e9beb3-device-volumes.tdb
       256  .config/pulse/cookie
       807  .profile
       134  dir2/test2.txt
        96  dir2/test3.txt
       127  test.txt
        33  dir1/test1.txt
     16384  .beijing.html.swp
       220  .bash_logout
    390341  .cache/gstreamer-1.0/registry.x86_64.bin
       137  new.txt
        71  hello.cpp
        20  .lesshst
     16560  list
      3771  .bashrc
     14157  .viminfo
        46  try.txt
     15306  .bash_history
      5125  list.c
     15968  hello
       105  .local/share/keyrings/login.keyring
```

图 6：“-a”与“-r”选项测试

然后测试路径指定功能，结果如下图所示：

```
c1616@Ubuntu-bupt:~$ ./list -r dir2
       134  dir2/test2.txt
        96  dir2/test3.txt
c1616@Ubuntu-bupt:~$
```

图 7：路径指定功能测试

最后测试“--”选项，下图命令将“-l”作为目录参数而非命令行参数：

```
c1616@Ubuntu-bupt:~$ ./list -- -l
-l: No such file or directory
c1616@Ubuntu-bupt:~$
```

图 8：“--”选项测试

4 实验问题与实验总结

在本次实验中我也遇到了一些问题，现记录如下：

（1） 实验目标理解不到位：一开始我对于实验目标的理解是有一些偏差的，经过回看课堂 PPT、研读教材等环节，我正确地分析出了本次实验所要实现的需求；

（2） C 语言库不熟悉：在具体的处理中，遇到的主要问题就是 C 语言库文件使用不够熟练，对诸如 stat 等库文件内容含义不够熟悉。经过阅读文档，我解决了这个问题。

总的来说，在本次实验中，我通过分析和编写 list.c 文件，实现了 Linux 系统目录的遍历功能，学习了 vim 工具的使用以及 Shell 下文件通配符的处理方式，并且对于在 Linux 下文件的组织方式和 ls 命令有了更深入的理解，本次实验我收获满满。