Linux 开发环境及应用上机作业二：遍历目录
实验报告

毛子恒
2019211397
北京邮电大学 计算机学院

日期：2022 年 4 月 18 日

1 实验内容

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 编辑文件，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。

2 实验步骤

命令行选项获取 实现一个类似于 Linux 的 `getopt` 函数 `getOpt`，用于获取命令行选项，该函数依次遍历命令行参数，将获取到的选项返回。全局变量 `optind` 和 `optarg` 分别用于表示遍历到的参数下标以及获取到的参数值。

相比较 `getopt` 函数，我实现的函数稍作简化，无法通过传入参数定制需要解析的命令行选项，但是可以获取到匿名的参数（也就是路径选项）。

打开目录 `match` 函数尝试打开传入的路径，并且根据传入的路径是否以'/' 结尾做了一些特殊处理，如果指定的路径可以打开，那么调用 `output` 函数输出。

输出 `output` 函数判断指定的文件是否满足筛选条件，如果满足则输出，其中还涉及递归输出的过程。

getopt_long 函数的使用 `getopt_long` 函数声明如下：

```
int getopt_long(int argc, char * const argv[], const char *optstring, const struct option *longopts, int *longindex);
```

`getopt_long` 接受长、短的命令行选项，长选项以`--`开头。如果程序只接受长选项，那么 `optstring` 应指定为空字符串。如果缩写是唯一的，那么长选项名称可以缩写。长选项可以采用两种形式：`--arg=param` 或`--arg param`。`longopts` 是结构体 `option` 的数组。

结构体 option 的声明如下:

```
struct option {
   const char *name;
   int         has_arg;
   int        *flag;
   int         val;
};
```

- name：长选项的名字。
- has_arg：0，不需要参数；1，需要参数；2，参数是可选的。
- flag：指定如何为长选项返回结果。如果是 NULL，那么函数返回 val（设置为等效的短选项字符），否则返回 0。
- val：要返回的值。

logopts 数组的最后一个元素必须用零填充。

当一个短选项字符被识别时，getopt_long 也返回选项字符。

3 代码

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include <stdlib.h>
#include <ctype.h>
#include <time.h>
#include <unistd.h>
#include <dirent.h>
#include <sys/stat.h>

#ifndef MAXNAMLEN
#define MAXNAMLEN 255
#endif // MAXNAMLEN

char *opt_arg;
int opt_ind = 1;

struct Flags
{
    bool recursive;
    bool all;
    long low;
    long high;
    double modified;
} flags;

char pwd[MAXNAMLEN];
```

```
28
29  int getOpt(int argc, char *argv[]);
30
31  bool match(const char *pattern);
32
33  void output(const char *path);
34
35  void help_info();
36
37  int main(int argc, char *argv[])
38  {
39      int path_args_cnt = 0;
40      char **path_args_list = (char **)malloc(argc * sizeof(char *));
41
42      flags.high = flags.modified = 0x7fffffff;
43      int ch;
44      while ((ch = getOpt(argc, argv)) != -1)
45      {
46          switch (ch)
47          {
48          case 0:
49              path_args_list[++path_args_cnt] = opt_arg;
50              break;
51          case 'r':
52              flags.recursive = true;
53              break;
54          case 'a':
55              flags.all = true;
56              break;
57          case 'l':
58              flags.low = atoll(opt_arg);
59              if (strcmp(opt_arg, "0") && !flags.low)
60                  help_info();
61              break;
62          case 'h':
63              flags.high = atoll(opt_arg);
64              if (strcmp(opt_arg, "0") && !flags.high)
65                  help_info();
66              break;
67          case 'm':
68              flags.modified = atoll(opt_arg) * 3600 * 24;
69              if (strcmp(opt_arg, "0") && !flags.modified)
70                  help_info();
71              break;
72          case '?':
73              help_info();
74              break;
75          }
76      }
77
78      if (getcwd(pwd, MAXNAMLEN) == NULL)
```

```
79      {
80          fprintf(stderr, "getcwd(): Error.");
81          free(path_args_list);
82          exit(1);
83      }
84
85      if (!path_args_cnt) // no path args, list pwd
86      {
87          match(pwd);
88          free(path_args_list);
89          return 0;
90      }
91
92      for (int i = 1; i <= path_args_cnt; ++i)
93      {
94          char *pattern = (char *)malloc((strlen(path_args_list[i]) +
    ↪ MAXNAMLEN + 1) * sizeof(char));
95          if (!strlen(path_args_list[i]) || (path_args_list[i][0] == '.' &&
    ↪ path_args_list[i][1] != '.'))
96          {
97              strcpy(pattern, pwd);
98              strcpy(pattern + strlen(pwd), path_args_list[i] + 1);
99          }
100         else if ((path_args_list[i][0] == '.' && path_args_list[i][1] ==
    ↪ '.' && path_args_list[i][2] == '/') || strrchr(path_args_list[i], '/')
    ↪ == NULL)
101         {
102             strcpy(pattern, pwd);
103             pattern[strlen(pwd)] = '/';
104             strcpy(pattern + strlen(pwd) + 1, path_args_list[i]);
105         }
106         else
107             strcpy(pattern, path_args_list[i]);
108         // printf("%s\n", pattern);
109
110         if (!match(pattern))
111             fprintf(stderr, "list: cannot access '%s': No such file or
    ↪ directory\n", path_args_list[i]);
112         free(pattern);
113     }
114     free(path_args_list);
115     return 0;
116 }
117
118 int getOpt(int argc, char *argv[])
119 {
120     if (opt_ind >= argc)
121         return -1;
122     if (argv[opt_ind][0] != '-')
123     {
124         opt_arg = argv[opt_ind];
```

```
125         opt_ind++;
126         return 0;
127     }
128     if (argv[opt_ind][1] == '-')
129     {
130         opt_ind++;
131         return -1;
132     }
133     else if (argv[opt_ind][1] == 'r' || argv[opt_ind][1] == 'a')
134     {
135         opt_arg = NULL;
136         char opt = argv[opt_ind][1];
137         opt_ind++;
138         return opt;
139     }
140     else if (argv[opt_ind][1] == 'l' || argv[opt_ind][1] == 'h' ||
    ↪  argv[opt_ind][1] == 'm')
141     {
142         if (opt_ind + 1 >= argc)
143         {
144             opt_arg = NULL;
145             return '?';
146         }
147         else
148             opt_arg = argv[opt_ind + 1];
149         char opt = argv[opt_ind][1];
150         opt_ind += 2;
151         return opt;
152     }
153     else
154     {
155         opt_ind++;
156         return '?';
157     }
158 }
159
160 bool match(const char *pattern)
161 {
162     char *last_slash = strrchr(pattern, '/');
163     char *last_pattern = (char *)malloc(strlen(pattern) * sizeof(char));
164     char *file_name = (char *)malloc(MAXNAMLEN * sizeof(char));
165     strncpy(last_pattern, pattern, last_slash - pattern);
166     strcpy(file_name, last_slash + 1);
167
168     DIR *dir;
169     struct dirent *ent;
170     if ((dir = opendir(last_pattern)) == NULL)
171     {
172         free(last_pattern);
173         free(file_name);
174         return false;
```

```
175     }
176
177     if (!strlen(file_name)) // pattern end with '/'
178     {
179         *last_slash = 0;
180         last_slash = strrchr(last_pattern, '/');
181         *last_slash = 0;
182         strcpy(file_name, last_slash + 1);
183         closedir(dir);
184         dir = opendir(last_pattern);
185     }
186     // printf("%s %s\n", last_pattern, file_name);
187
188     while ((ent = readdir(dir)) != NULL)
189     {
190         if (!strcmp(file_name, ent->d_name))
191         {
192             output(pattern);
193             return true;
194         }
195     }
196     closedir(dir);
197     free(last_pattern);
198     free(file_name);
199     return false;
200 }
201
202 void output(const char *path)
203 {
204     struct stat statbuf;
205     stat(path, &statbuf);
206     if (S_ISDIR(statbuf.st_mode))
207     {
208         DIR *dir;
209         struct dirent *ent;
210         struct stat statbuf;
211
212         if ((dir = opendir(path)) == NULL)
213         {
214             fprintf(stderr, "Can`t open directory %s\n", path);
215             return;
216         }
217
218         while ((ent = readdir(dir)) != NULL)
219         {
220             if (ent->d_name[0] == '.')
221                 if (!flags.all)
222                     continue;
223             char *new_path = (char *)malloc((strlen(path) + MAXNAMLEN + 1)
    ↪  * sizeof(char));
224             strcpy(new_path, path);
```

```
225             if (!strcmp(path, "/"))
226                 strcpy(new_path + 1, ent->d_name);
227             else
228             {
229                 strcpy(new_path + strlen(path) + 1, ent->d_name);
230                 *(new_path + strlen(path)) = '/';
231             }
232             stat(new_path, &statbuf);
233             if (statbuf.st_size >= flags.low && statbuf.st_size <=
    ↪  flags.high && difftime(time(NULL), statbuf.st_mtime) <= flags.modified)
234                 printf("%10ld %s\n", statbuf.st_size, new_path);
235             if (!flags.recursive)
236             {
237                 free(new_path);
238                 continue;
239             }
240             if (S_ISDIR(statbuf.st_mode))
241             {
242                 if (!strcmp(ent->d_name, ".") || !strcmp(ent->d_name,
    ↪  ".."))
243                     continue;
244                 output(new_path);
245             }
246             free(new_path);
247         }
248     }
249     else
250     {
251         stat(path, &statbuf);
252         if (statbuf.st_size >= flags.low && statbuf.st_size <= flags.high
    ↪  && difftime(time(NULL), statbuf.st_mtime) <= flags.modified)
253             printf("%10ld %s\n", statbuf.st_size, path);
254     }
255 }
256
257 void help_info()
258 {
259     printf("usage: list [-r] [-a] [-l <minimum_size>] [-h <maximum_size>]
    ↪  [-m <modified_days>] [file ...]\n");
260     exit(1);
261 }
```

4 运行结果

采用如下命令编译:

```
gcc list.c -o list -std=c11 -Wall
```

运行结果如图 1。

```
b397@Ubuntu-bupt:~/work2$ ./list .
     21992 /home/b397/work2/list
      6682 /home/b397/work2/list.c
b397@Ubuntu-bupt:~/work2$ ./list ./
     21992 /home/b397/work2/list
      6682 /home/b397/work2/list.c
b397@Ubuntu-bupt:~/work2$ ./list ../work1
       805 /home/b397/work2/../work1/result.csv
        96 /home/b397/work2/../work1/work.awk
     21830 /home/b397/work2/../work1/beijing.html
b397@Ubuntu-bupt:~/work2$ ./list ../work1/*.awk
        96 /home/b397/work2/../work1/work.awk
b397@Ubuntu-bupt:~/work2$ ./list ../work* -r -l 100 -h 10000 -a
       805 /home/b397/work2/../work1/result.csv
      4096 /home/b397/work2/../work1/.
      4096 /home/b397/work2/../work1/..
      4096 /home/b397/work2/../work2/.
      4096 /home/b397/work2/../work2/..
      6682 /home/b397/work2/../work2/list.c
b397@Ubuntu-bupt:~/work2$ ./list /home/b397/work1 /home/b397/work2 -- -l
       805 /home/b397/work1/result.csv
        96 /home/b397/work1/work.awk
     21830 /home/b397/work1/beijing.html
     21992 /home/b397/work2/list
      6682 /home/b397/work2/list.c
b397@Ubuntu-bupt:~/work2$ ./list ../work1 -m 1
b397@Ubuntu-bupt:~/work2$ ./list
     21992 /home/b397/work2/list
      6682 /home/b397/work2/list.c
```

图 1: 运行结果

5 实验总结

实验期间我观察到 Shell 会先对通配符进行解析，将其展开为多个参数传递给程序。

本次实验中我应用 Linux 目录和文件信息访问的库函数，实现了对指定目录的遍历操作，使我对 Linux 的文件系统理解更加深刻。