上机作业 2：遍历目录

088 于海鑫
2017211240
name1e5s@bupt.edu.cn

版本：8
更新：*April 9, 2020*

目录

1 作业要求

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。

2 作业实现

2.1 获取选项

因为在这次作业中，使用 `getopt` 是不被允许的，因此我们首先需要一个用于处理选项的函数。在我们的实现中，我们决定基于 C 标准库函数 `strchr` 模拟实现一个 `getopt` 以支持我们的选项分析。核心代码如下：

```
    optdecl = strchr(optstring, optchar);
    if (optdecl) {
        if (optdecl[1] == ':') {
            optarg = ++optcursor;
            if (*optarg == '\0') {
                if (++optind < argc) {
                    optarg = argv[optind];
                } else {
                    optarg = NULL;
                    optchar = (optstring[0] == ':') ? ':' : '?';
                }
            }
            optcursor = NULL;
```

```
        }
    } else {
        optchar = '?';
    }
```

2.2 输出单个文件

在输出单个文件时，我们使用 `stat` 函数来获取单个文件的状态信息，之后根据这些状态信息打印出文件。核心代码如下：

```
static void list_node(const char *prefix, const char *file_name) {
    // 一些拼接代码

    struct stat status;
    if(stat(node_name, &status)) {
        fprintf(stderr, "%s - Can't access \"%s\": %s\n", elf_name,
            node_name, strerror(errno));
        return;
    }

    mode_t stat_mode = status.st_mode;
    off_t stat_size = status.st_size;
    time_t stat_mtime = status.st_mtime;

    // 一些格式化代码...

    int filter_flag = (!modify_time || time (NULL) - stat_mtime <=
        modify_time) &&
        (!lo_size || stat_size >= lo_size) &&
        (!hi_size || stat_size <= hi_size);
    if((filter_flag && !S_ISDIR(stat_mode)) || (S_ISDIR(stat_mode) &&
        recursive_flag)) {
        char stat_time_str[64];
        strftime(stat_time_str, 64,
            "%Y-%m-%d %H:%M", localtime(&stat_mtime));
        printf ("%s %8ld %s %s\n", mode_text, stat_size,
            stat_time_str, file_name);
    }
```

```
25
26      // 一些收尾代码
27  }
```

2.3 输出文件夹下的全部文件

基于上面输出单个文件的代码，我们可以实现输出某文件夹下全部文件的函数。其核心在于对 `readdir` 函数的使用。代码如下：

```
1   static void list_dir(const char *name) {
2       if(!init) {
3           printf("\n");
4       }
5       init = 0;
6
7       DIR *dir = opendir (name);
8       if (dir == NULL) {
9               fprintf(stderr, "%s - Can't access dir \"%s\": %s\n",
                    elf_name, name, strerror(errno));
10              return;
11          }
12
13      printf("%s:\n", name);
14
15      struct dirent *entry;
16      int count = 0;
17      while ((entry = readdir(dir)) != NULL) {
18          const char *entry_name = entry->d_name;
19          if(!all_flag && entry_name[0] == '.')
20              continue;
21
22          list_node(name, entry_name);
23          count++;
24      }
25      printf("%d files in total.\n", count);
26  }
```

2.4 递归输出

有时我们需要实现对于某文件夹的递归输出，此时我们使用广度优先搜索的方式，使用一个链表模拟的队列进行文件夹的递归输出。核心代码如下：

```
static void list_main(const char *name) {
    struct stat status;
    if(stat(name, &status)) {
        fprintf(stderr, "%s - Can't access \"%s\": %s\n", elf_name,
           name, strerror(errno));
        return;
    }
    if(S_ISDIR(status.st_mode)) {
        list_start = malloc(sizeof(list_node_t));
        list_start->node = name;
        list_start->next = NULL;
        list_end = list_start;
        while (list_start) {
            list_dir(list_start->node);
            list_start = list_start->next;
        }
    } else {
        list_node("", name);
    }
}
```

3 运行效果

3.1 基本使用

输出某一文件夹的内容时结果如下：

```
name1e5s@sumeru:~/Homework-2/build$ ./list ..
..:
-rw-rw-r--     1719 2020-04-09 17:31 getopt.c
-rw-rw-r--     5141 2020-04-09 17:31 list.c
-rw-rw-r--      123 2020-04-09 17:31 CMakeLists.txt
-rw-rw-r--      559 2020-04-09 17:31 homemade_getopt.h
```

```
5 files in total.
```

3.2 递归输出

递归输出某一文件夹的内容时结果如下：

```
name1e5s@sumeru:~/Homework-2/build$ ./list ..
..:
-rw-rw-r--     1719 2020-04-09 17:31 getopt.c
-rw-rw-r--     5141 2020-04-09 17:31 list.c
-rw-rw-r--      123 2020-04-09 17:31 CMakeLists.txt
-rw-rw-r--      559 2020-04-09 17:31 homemade_getopt.h
5 files in total.
name1e5s@sumeru:~/Homework-2/build$ ./list -r ..
..:
-rw-rw-r--     1719 2020-04-09 17:31 getopt.c
-rw-rw-r--     5141 2020-04-09 17:31 list.c
-rw-rw-r--      123 2020-04-09 17:31 CMakeLists.txt
-rw-rw-r--      559 2020-04-09 17:31 homemade_getopt.h
drwxrwxr-x     4096 2020-04-09 17:31 build
5 files in total.

../build:
-rw-rw-r--     1502 2020-04-09 17:31 cmake_install.cmake
-rwxrwxr-x    13832 2020-04-09 17:31 list
drwxrwxr-x     4096 2020-04-09 17:31 CMakeFiles
-rw-rw-r--     5304 2020-04-09 17:31 Makefile
-rw-rw-r--    12576 2020-04-09 17:31 CMakeCache.txt
5 files in total.

../build/CMakeFiles:
-rw-rw-r--     3067 2020-04-09 17:31 Makefile2
-rw-rw-r--       85 2020-04-09 17:31 cmake.check_cache
-rwxrwxr-x    12312 2020-04-09 17:31 feature_tests.bin
drwxrwxr-x     4096 2020-04-09 17:31 list.dir
drwxrwxr-x     4096 2020-04-09 17:31 CMakeTmp
-rw-rw-r--      688 2020-04-09 17:31 feature_tests.c
drwxrwxr-x     4096 2020-04-09 17:31 3.10.2
-rw-rw-r--        2 2020-04-09 17:31 progress.marks
```

```
-rw-rw-r--    44929 2020-04-09 17:31 CMakeOutput.log
-rw-rw-r--      632 2020-04-09 17:31 CMakeDirectoryInformation.cmake
-rw-rw-r--    10011 2020-04-09 17:31 feature_tests.cxx
-rw-rw-r--     6639 2020-04-09 17:31 Makefile.cmake
-rw-rw-r--      171 2020-04-09 17:31 TargetDirectories.txt
13 files in total.

../build/CMakeFiles/list.dir:
-rw-rw-r--      524 2020-04-09 17:31 C.includecache
-rw-rw-r--      261 2020-04-09 17:31 cmake_clean.cmake
-rw-rw-r--       94 2020-04-09 17:31 link.txt
-rw-rw-r--     3136 2020-04-09 17:31 getopt.c.o
-rw-rw-r--      290 2020-04-09 17:31 depend.make
-rw-rw-r--      320 2020-04-09 17:31 depend.internal
-rw-rw-r--      171 2020-04-09 17:31 flags.make
-rw-rw-r--     7424 2020-04-09 17:31 list.c.o
-rw-rw-r--     5633 2020-04-09 17:31 build.make
-rw-rw-r--       64 2020-04-09 17:31 progress.make
-rw-rw-r--      657 2020-04-09 17:31 DependInfo.cmake
11 files in total.

../build/CMakeFiles/CMakeTmp:
0 files in total.

../build/CMakeFiles/3.10.2:
-rw-r--r--      402 2020-04-09 17:31 CMakeSystem.cmake
-rwxrwxr-x     8248 2020-04-09 17:31 CMakeDetermineCompilerABI_C.bin
-rw-r--r--     4849 2020-04-09 17:31 CMakeCXXCompiler.cmake
drwxrwxr-x     4096 2020-04-09 17:31 CompilerIdC
-rwxrwxr-x     8264 2020-04-09 17:31 CMakeDetermineCompilerABI_CXX.
   bin
drwxrwxr-x     4096 2020-04-09 17:31 CompilerIdCXX
-rw-r--r--     2219 2020-04-09 17:31 CMakeCCompiler.cmake
7 files in total.

../build/CMakeFiles/3.10.2/CompilerIdC:
-rwxrwxr-x     8408 2020-04-09 17:31 a.out
drwxrwxr-x     4096 2020-04-09 17:31 tmp
-rw-rw-r--    18076 2020-04-09 17:31 CMakeCCompilerId.c
```

```
3 files in total.

../build/CMakeFiles/3.10.2/CompilerIdCXX:
-rw-rw-r--    17631 2020-04-09 17:31 CMakeCXXCompilerId.cpp
-rwxrwxr-x     8416 2020-04-09 17:31 a.out
drwxrwxr-x     4096 2020-04-09 17:31 tmp
3 files in total.

../build/CMakeFiles/3.10.2/CompilerIdC/tmp:
0 files in total.

../build/CMakeFiles/3.10.2/CompilerIdCXX/tmp:
0 files in total.
```

3.3 添加限制

有限制的输出某一文件夹的内容时结果如下：

```
name1e5s@sumeru:~/Homework-2/build$  ./list -l 5000 -m 2 .
.:
-rwxrwxr-x    13832 2020-04-09 17:31 list
-rw-rw-r--     5304 2020-04-09 17:31 Makefile
-rw-rw-r--    12576 2020-04-09 17:31 CMakeCache.txt
5 files in total.
```

A 代码清单

A.1 CMakeLists.txt

```
cmake_minimum_required(VERSION 3.0)
project(list)

set(PROJ_FILES list.c getopt.c)

add_executable(list ${PROJ_FILES})
```

A.2 homemade_getopt.h

```
#ifndef HOMEMADE_GETOPT_H
#define HOMEMADE_GETOPT_H

#ifndef _HOMEMADE_SRC
#include <getopt.h>
#else

#if defined(__cplusplus)
extern "C" {
#endif


#define no_argument 1
#define required_argument 2
#define optional_argument 3

extern char* optarg;
extern int optind;

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

int homemade_getopt(int argc, char* const argv[], const char*
   optstring);
#if defined(__cplusplus)
}
#endif

#define getopt homemade_getopt

#endif

#endif // HOMEMADE_GETOPT_H
```

A.3 getopt.c

```c
#include <stddef.h>
#include <stdio.h>
#include <string.h>

#define _HOMEMADE_SRC
#include "homemade_getopt.h"

// Global definitions
char *optarg;
int optind = 1;

static char *optcursor;

int homemade_getopt(int argc, char* const argv[], const char*
   optstring) {
    int optchar = EOF;

    const char *optdecl = NULL;

    // Initialize global vars
    optarg = NULL;

    // Arguments overflow #1
    if(optind >= argc)
        goto no_more_argument;

    // Arguments overflow #2
    if(argv[optind] == NULL)
        goto no_more_argument;

    // Invalid arguments #1
    if(argv[optind][0] != '-')
        goto no_more_argument;

    // Invalid arguments #2
    if(strcmp(argv[optind], "-") == 0)
        goto no_more_argument;
```

```
37
38      // Long arguments #1
39      // Skip it here
40      if(strcmp(argv[optind], "--") == 0) {
41          optind++;
42          goto no_more_argument;
43      }
44
45      if (optcursor == NULL || optcursor[0] == '\0')
46          optcursor = argv[optind] + 1;
47
48      optchar = optcursor[0];
49
50      optdecl = strchr(optstring, optchar);
51      if (optdecl) {
52          if (optdecl[1] == ':') {
53              optarg = ++optcursor;
54              if (*optarg == '\0') {
55                  if (++optind < argc) {
56                      optarg = argv[optind];
57                  } else {
58                      optarg = NULL;
59                      optchar = (optstring[0] == ':') ? ':' : '?';
60                  }
61              }
62              optcursor = NULL;
63          }
64      } else {
65          optchar = '?';
66      }
67
68      if (optcursor == NULL || *++optcursor == '\0')
69          ++optind;
70
71      return optchar;
72  no_more_argument:
73      optcursor = NULL;
74      return EOF;
75  }
```

A.4 list.c

```c
#define _HOMEMADE_SRC
#include "homemade_getopt.h"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <time.h>

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

// Global options
static const char *elf_name;
static int recursive_flag;
static int all_flag;
static off_t lo_size;
static off_t hi_size;
static time_t modify_time;
static int init = 1;

// Helper linked list
typedef struct lnode {
    const char *node;
    struct lnode *next;
} list_node_t;

typedef list_node_t *list_t;

static list_t list_start;
static list_t list_end;

static char get_type(mode_t mode) {
    if(S_ISREG(mode))
        return '-';
    if(S_ISDIR(mode))
```

```
38          return 'd';
39      if(S_ISCHR(mode))
40          return 'c';
41      if(S_ISBLK(mode))
42          return 'b';
43      if(S_ISLNK(mode))
44          return 'l';
45      if(S_ISFIFO(mode))
46          return 'p';
47      if(S_ISSOCK(mode))
48          return 's';
49  }
50
51  static void list_node(const char *prefix, const char *file_name) {
52      size_t name_size = strlen(prefix);
53      size_t entry_size = strlen(file_name);
54      char *node_name = malloc(name_size + entry_size + 2);
55
56      strcpy(node_name, prefix);
57      node_name[name_size] = '/';
58      strcpy (&node_name[name_size + 1], file_name);
59      node_name[name_size + 1 + entry_size] = 0;
60
61      struct stat status;
62      if(stat(node_name, &status)) {
63          fprintf(stderr, "%s - Can't access \"%s\": %s\n", elf_name,
                node_name, strerror(errno));
64          return;
65      }
66
67      mode_t stat_mode = status.st_mode;
68      off_t stat_size = status.st_size;
69      time_t stat_mtime = status.st_mtime;
70
71      char mode_text[] = {'-', '-', '-', '-', '-',
72                          '-', '-', '-', '-', '-', '\0'};
73      mode_text[0] = get_type(stat_mode);
74      if(stat_mode & S_IRUSR) {
75          mode_text[1] = 'r';
```

```
76      }
77      if(stat_mode & S_IWUSR) {
78          mode_text[2] = 'w';
79      }
80      if(stat_mode & S_IXUSR) {
81          mode_text[3] = 'x';
82      }
83      if(stat_mode & S_IRGRP) {
84          mode_text[4] = 'r';
85      }
86      if(stat_mode & S_IWGRP) {
87          mode_text[5] = 'w';
88      }
89      if(stat_mode & S_IXGRP) {
90          mode_text[6] = 'x';
91      }
92      if(stat_mode & S_IROTH) {
93          mode_text[7] = 'r';
94      }
95      if(stat_mode & S_IWOTH) {
96          mode_text[8] = 'w';
97      }
98      if(stat_mode & S_IXOTH) {
99          mode_text[9] = 'x';
100     }
101
102     int filter_flag = (!modify_time || time (NULL) - stat_mtime <=
            modify_time) &&
103         (!lo_size || stat_size >= lo_size) &&
104         (!hi_size || stat_size <= hi_size);
105     if((filter_flag && !S_ISDIR(stat_mode)) || (S_ISDIR(stat_mode) &&
            recursive_flag)) {
106         char stat_time_str[64];
107         strftime(stat_time_str, 64,
108             "%Y-%m-%d␣%H:%M", localtime(&stat_mtime));
109         printf ("%s␣%8ld␣%s␣%s\n", mode_text, stat_size,
                stat_time_str, file_name);
110     }
111
```

```
112     int dame_flag = 0;
113     size_t file_name_len = strlen(file_name);
114     if((file_name[0] == '.' && file_name_len == 1) ||
115        (file_name[0] == '.' && file_name[1] == '.' && file_name_len
             == 2)) {
116         dame_flag = 1;
117     }
118 
119     if(S_ISDIR(stat_mode) && recursive_flag && !dame_flag) {
120         list_t tmp = malloc(sizeof(list_node_t));
121         tmp->node = node_name;
122         tmp->next = NULL;
123         list_end->next = tmp;
124         list_end = tmp;
125     }
126 }
127 
128 static void list_dir(const char *name) {
129     if(!init) {
130         printf("\n");
131     }
132     init = 0;
133 
134     DIR *dir = opendir (name);
135     if (dir == NULL) {
136             fprintf(stderr, "%s - Can't access dir \"%s\": %s\n",
                    elf_name, name, strerror(errno));
137             return;
138         }
139 
140     printf("%s:\n", name);
141 
142     struct dirent *entry;
143     int count = 0;
144     while ((entry = readdir(dir)) != NULL) {
145         const char *entry_name = entry->d_name;
146         if(!all_flag && entry_name[0] == '.')
147             continue;
148 
```

```
149         list_node(name, entry_name);
150         count++;
151     }
152     printf("%d␣files␣in␣total.\n", count);
153 }
154 
155 static void list_main(const char *name) {
156     struct stat status;
157     if(stat(name, &status)) {
158         fprintf(stderr, "%s␣-␣Can't␣access␣\"%s\":␣%s\n", elf_name,
               name, strerror(errno));
159         return;
160     }
161     if(S_ISDIR(status.st_mode)) {
162         list_start = malloc(sizeof(list_node_t));
163         list_start->node = name;
164         list_start->next = NULL;
165         list_end = list_start;
166         while (list_start) {
167             list_dir(list_start->node);
168             list_start = list_start->next;
169         }
170     } else {
171         list_node("", name);
172     }
173 }
174 
175 int main(int argc, char **argv) {
176     elf_name = argv[0];
177     int c = 0;
178     while ((c = getopt (argc, argv, "ral:h:m:")) != -1) {
179         switch (c) {
180             case 'r':
181                 recursive_flag = 1;
182                 break;
183             case 'a':
184                 all_flag = 1;
185                 break;
186             case 'l':
```

```
187                 lo_size = atoi(optarg);
188                 break;
189             case 'h':
190                 hi_size = atoi(optarg);
191                 break;
192             case 'm':
193                 modify_time = atoi (optarg) * 24 * 60 * 60;
194                 break;
195             default:
196                 break;
197         }
198     }
199     for(int i = optind; i < argc; i++) {
200         list_main(argv[i]);
201     }
202     return 0;
203 }
```