



Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«Московский государственный технический университет  
имени Н. Э. Баумана  
(национальный исследовательский университет)»  
(МГТУ им. Н. Э. Баумана)

---

ФАКУЛЬТЕТ

«Информатика и системы управления»

КАФЕДРА

«Программное обеспечение ЭВМ и информационные технологии»

## ОТЧЁТ

По лабораторной работе №2

По курсу: «Операционные системы»

Тема: «Дерево каталогов»

Студент: Керимов А. Ш.

Группа: ИУ7-64Б

Преподаватель: Рязанова Н. Ю.

Москва

2020

В строке 96 — рекурсивный вызов `dopath`. Выход из рекурсии осуществляется в 97 строке (при ошибке) или в 106 (по завершении обхода).

### Листинг 1: `main.c`

```
1 #include <dirent.h>
2 #include <errno.h>
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <string.h>
6 #include <sys/stat.h>
7 #include <unistd.h>
8 #include "color.h"
9
10 #define COLOR_REG    COLOR_FG_DEFAULT
11 #define COLOR_DIR    COLOR_BOLD COLOR_FG_BLUE
12 #define COLOR_BLK    COLOR_BOLD COLOR_FG_YELLOW COLOR_BG_DARKGRAY
13 #define COLOR_CHR    COLOR_BOLD COLOR_FG_YELLOW COLOR_BG_DARKGRAY
14 #define COLOR_FIFO    COLOR_FG_YELLOW COLOR_BG_DARKGRAY
15 #define COLOR_LINK    COLOR_BOLD COLOR_FG_CYAN
16 #define COLOR SOCK    COLOR_BOLD COLOR_FG_MAGENTA
17 #define COLOR_EXE    COLOR_BOLD COLOR_FG_GREEN
18
19 #define FTW_F 1 // файл, не являющийся каталогом
20 #define FTW_D 2 // каталог
21 #define FTW_DNR 3 // каталог, недоступный для чтения
22 #define FTW_NS 4 // файл, информацию о котором нельзя получить с помощью stat
23
24 // тип функции, которая будет вызываться для каждого встреченного файла
25 typedef int MyFunc(const char *filename, const struct stat *st, int depth, int type);
26
27 static MyFunc counter;
28 static int myftw(const char *, MyFunc *);
29 static int dopath(const char *filename, int depth, MyFunc *);
30
31 static size_t nreg, ndir, nblk, nchr, nfifo, nlink, nsock, ntot;
32
33 int main(int argc, char **argv)
34 {
35     if (argc != 2) {
36         fprintf(stderr, "Usage: %s <dir>\n", argv[0]);
37         return EXIT_FAILURE;
38     }
39
40     const int ret = myftw(argv[1], counter);
41
42     ntot = nreg + ndir + nblk + nchr + nfifo + nlink + nsock;
43     if (ntot == 0) {
44         ntot = 1; // во избежание деления на 0; вывести 0 для всех счётчиков
45     }
46
47     printf("\n----- Summary ----- \n");
48     printf("regular files: %7ld, %5.2f %%\n", nreg, nreg * 100.0 / ntot);
49     printf("directories: %7ld, %5.2f %%\n", ndir, ndir * 100.0 / ntot);
50     printf("block devices: %7ld, %5.2f %%\n", nblk, nblk * 100.0 / ntot);
51     printf("char devices: %7ld, %5.2f %%\n", nchr, nchr * 100.0 / ntot);
52     printf("FIFOs: %7ld, %5.2f %%\n", nfifo, nfifo * 100.0 / ntot);
53     printf("symbolic links: %7ld, %5.2f %%\n", nlink, nlink * 100.0 / ntot);
54     printf("sockets: %7ld, %5.2f %%\n", nsock, nsock * 100.0 / ntot);
55     printf("Total: %7ld\n", ntot);
56
57     return ret;
58 }
59
60 // Обходит дерево каталогов, начиная с каталога pathname, применяя к каждому файлу функцию func.
61 static int myftw(const char *pathname, MyFunc *func)
62 {
63     return dopath(pathname, 0, func);
64 }
65
66 static int dopath(const char *filename, int depth, MyFunc *func)
67 {
68     struct stat statbuf;
69     struct dirent *entry;
70     DIR *dp;
71     int ret;
72
73     if (lstat(filename, &statbuf) == -1) {
74         return func(filename, &statbuf, depth, FTW_NS);
```

```

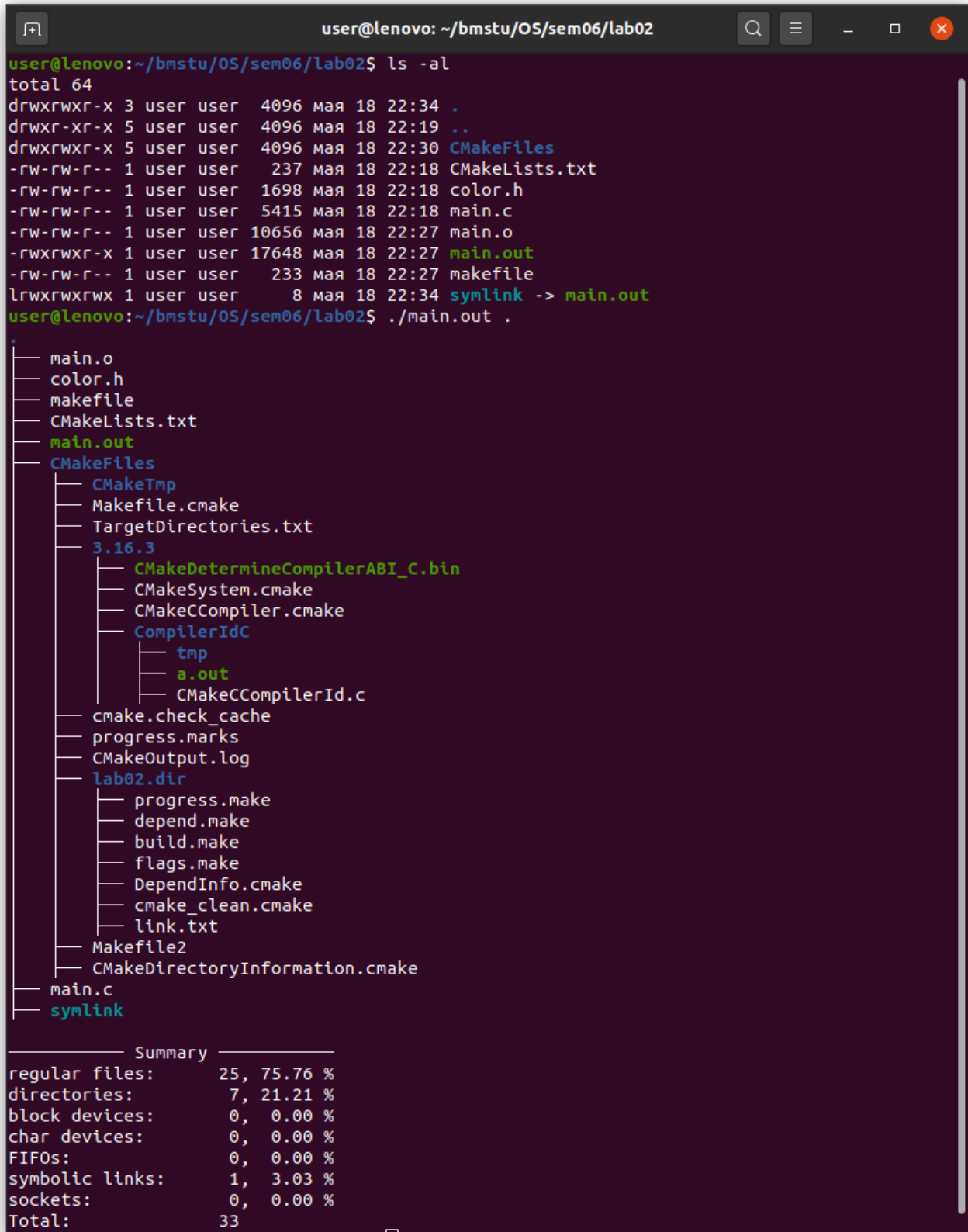
75     }
76
77     if (S_ISDIR(statbuf.st_mode) == 0) { // не каталог
78         return func(filename, &statbuf, depth, FTW_F);
79     }
80
81     if ((ret = func(filename, &statbuf, depth, FTW_D)) != EXIT_SUCCESS) {
82         return ret;
83     }
84
85     if ((dp = opendir(filename)) == NULL) { // каталог недоступен
86         return func(filename, &statbuf, depth, FTW_DNR);
87     }
88
89     chdir(filename);
90     while ((entry = readdir(dp)) != NULL) {
91         if (strcmp(entry->d_name, ".") == 0 || strcmp(entry->d_name, "..") == 0) {
92             continue;
93         }
94
95         /* рекурсивный вызов */
96         if ((ret = dopath(entry->d_name, depth + 1, func)) != EXIT_SUCCESS) {
97             return ret; // выход по ошибке
98         }
99     }
100     chdir("..");
101
102     if (closedir(dp) == -1) {
103         fprintf(stderr, "closedir(%s): %s", filename, strerror(errno));
104     }
105
106     return ret; // выход по завершении обхода
107 }
108
109 static int counter(const char *filename, const struct stat *st, int depth, int type)
110 {
111     for (int i = 0; i + 1 < depth; ++i) {
112         printf("|   ");
113     }
114
115     const char *s = "";
116
117     switch (type) {
118     case FTW_F:
119         switch (st->st_mode & S_IFMT) {
120             case S_IFREG: ++nreg; s = st->st_mode & S_IXUSR ? COLOR_EXE : COLOR_REG; break;
121             case S_IFBLK: ++nblk; s = COLOR_BLK; break;
122             case S_IFCHR: ++nchr; s = COLOR_CHR; break;
123             case S_IFIFO: ++nfifo; s = COLOR_FIFO; break;
124             case S_IFLNK: ++nlink; s = COLOR_LINK; break;
125             case S_IFSOCK: ++nsock; s = COLOR SOCK; break;
126             case S_IFDIR:
127                 fprintf(stderr, "Dir file type is not expected\n");
128                 return EXIT_FAILURE;
129             default:
130                 fprintf(stderr, "Unknown file type '%s': %d\n", filename, st->st_mode &
131                     S_IFMT);
132                 return EXIT_FAILURE;
133         }
134         printf("├─ %s%s\n", s, filename, COLOR_RESET);
135         break;
136     case FTW_D:
137         ++ndir;
138         printf("%s%s%s\n", depth ? "├─ " : "", COLOR_DIR, filename, COLOR_RESET);
139         break;
140     case FTW_DNR:
141         fprintf(stderr, "Can't read directory %s: %s\n", filename, strerror(errno));
142         return EXIT_FAILURE;
143     case FTW_NS:
144         switch (errno) {
145             case EACCES: s = "Permission denied"; break;
146             case EFAULT: s = "Bad address"; break;
147             case ELOOP: s = "Too many levels of symbolic links"; break;
148             case ENAMETOOLONG: s = "File name too long"; break;
149             case ENOENT: s = "No such file or directory"; break;
150             case ENOMEM: s = "Cannot allocate memory"; break;
151             case ENOTDIR: s = "Not a directory"; break;
152             default: s = strerror(errno); break;
153         }

```

```

153         fprintf(stderr, "lstat(%s, &st): %s\n", filename, s);
154         return EXIT_FAILURE;
155     default:
156         fprintf(stderr, "Unknown file type: %d\n", type);
157         return EXIT_FAILURE;
158     }
159
160     return EXIT_SUCCESS;
161 }

```



```

user@lenovo: ~/bmstu/OS/sem06/lab02
user@lenovo:~/bmstu/OS/sem06/lab02$ ls -al
total 64
drwxrwxr-x 3 user user 4096 мая 18 22:34 .
drwxr-xr-x 5 user user 4096 мая 18 22:19 ..
drwxrwxr-x 5 user user 4096 мая 18 22:30 CMakeFiles
-rw-rw-r-- 1 user user 237 мая 18 22:18 CMakeLists.txt
-rw-rw-r-- 1 user user 1698 мая 18 22:18 color.h
-rw-rw-r-- 1 user user 5415 мая 18 22:18 main.c
-rw-rw-r-- 1 user user 10656 мая 18 22:27 main.o
-rwxrwxr-x 1 user user 17648 мая 18 22:27 main.out
-rw-rw-r-- 1 user user 233 мая 18 22:27 makefile
lrwxrwxrwx 1 user user 8 мая 18 22:34 symlink -> main.out
user@lenovo:~/bmstu/OS/sem06/lab02$ ./main.out .
.
├── main.o
├── color.h
├── makefile
├── CMakeLists.txt
├── main.out
├── CMakeFiles
│   ├── CMakeTmp
│   ├── Makefile.cmake
│   ├── TargetDirectories.txt
│   ├── 3.16.3
│   │   ├── CMakeDetermineCompilerABI_C.bin
│   │   ├── CMakeSystem.cmake
│   │   ├── CMakeCCompiler.cmake
│   │   ├── CompilerIdC
│   │   │   ├── tmp
│   │   │   ├── a.out
│   │   │   └── CMakeCCompilerId.c
│   ├── cmake.check_cache
│   ├── progress.marks
│   ├── CMakeOutput.log
│   ├── lab02.dir
│   │   ├── progress.make
│   │   ├── depend.make
│   │   ├── build.make
│   │   ├── flags.make
│   │   ├── DependInfo.cmake
│   │   ├── cmake_clean.cmake
│   │   └── link.txt
│   ├── Makefile2
│   └── CMakeDirectoryInformation.cmake
├── main.c
└── symlink

Summary
regular files: 25, 75.76 %
directories: 7, 21.21 %
block devices: 0, 0.00 %
char devices: 0, 0.00 %
FIFOs: 0, 0.00 %
symbolic links: 1, 3.03 %
sockets: 0, 0.00 %
Total: 33

```

Рис. 1: Демонстрация работы программы

# Приложение А

Листинг 2: color.h

```
1 #ifndef COLOR_H_
2 #define COLOR_H_
3
4 #define COLOR_RESET      "\033[0m"
5 #define COLOR_BOLD      "\033[1m"
6 #define COLOR_DIM        "\033[2m"
7 #define COLOR_UNDERLINE  "\033[4m"
8 #define COLOR_BLINK      "\033[5m"
9 #define COLOR_REVERSE    "\033[7m"
10 #define COLOR_HIDDEN     "\033[8m"
11
12 #define COLOR_FG_DEFAULT  "\033[39m"
13 #define COLOR_FG_BLACK   "\033[30m"
14 #define COLOR_FG_RED     "\033[31m"
15 #define COLOR_FG_GREEN   "\033[32m"
16 #define COLOR_FG_YELLOW  "\033[33m"
17 #define COLOR_FG_BLUE    "\033[34m"
18 #define COLOR_FG_MAGENTA "\033[35m"
19 #define COLOR_FG_CYAN    "\033[36m"
20 #define COLOR_FG_LIGHTGRAY "\033[37m"
21 #define COLOR_FG_DARKGRAY "\033[90m"
22 #define COLOR_FG_LIGHTRED "\033[91m"
23 #define COLOR_FG_LIGHTGREEN "\033[92m"
24 #define COLOR_FG_LIGHTYELLOW "\033[93m"
25 #define COLOR_FG_LIGHTBLUE "\033[94m"
26 #define COLOR_FG_LIGHTMAGENTA "\033[95m"
27 #define COLOR_FG_LIGHTCYAN "\033[96m"
28 #define COLOR_FG_WHITE   "\033[97m"
29
30 #define COLOR_BG_DEFAULT  "\033[49m"
31 #define COLOR_BG_BLACK   "\033[40m"
32 #define COLOR_BG_RED     "\033[41m"
33 #define COLOR_BG_GREEN   "\033[42m"
34 #define COLOR_BG_YELLOW  "\033[43m"
35 #define COLOR_BG_BLUE    "\033[44m"
36 #define COLOR_BG_MAGENTA "\033[45m"
37 #define COLOR_BG_CYAN    "\033[46m"
38 #define COLOR_BG_LIGHTGRAY "\033[47m"
39 #define COLOR_BG_DARKGRAY "\033[100m"
40 #define COLOR_BG_LIGHTRED "\033[101m"
41 #define COLOR_BG_LIGHTGREEN "\033[102m"
42 #define COLOR_BG_LIGHTYELLOW "\033[103m"
43 #define COLOR_BG_LIGHTBLUE "\033[104m"
44 #define COLOR_BG_LIGHTMAGENTA "\033[105m"
45 #define COLOR_BG_LIGHTCYAN "\033[106m"
46 #define COLOR_BG_WHITE   "\033[107m"
47
48 #endif // COLOR_H_
```