



# Unix程序设计

实验（五）编写程序myfind

|      |                |
|------|----------------|
| 姓 名  | 熊恪峥            |
| 学 号  | 22920202204622 |
| 日 期  | 2022年11月23日    |
| 学 院  | 信息学院           |
| 课程名称 | Unix程序设计       |

## 实验（五）编写程序myfind

### 目录

|                    |                    |              |
|--------------------|--------------------|--------------|
| <b>1</b>           | <b>实验内容</b>        | <b>1</b>     |
| 1.1                | 命令语法 . . . . .     | 1            |
| 1.2                | 命令语义 . . . . .     | 1            |
| <b>2</b>           | <b>程序设计与实现</b>     | <b>1</b>     |
| 2.1                | 程序设计 . . . . .     | 1            |
| 2.2                | 程序实现 . . . . .     | 1            |
| <b>3</b>           | <b>程序测试</b>        | <b>1</b>     |
| <br><b>附录：代码清单</b> |                    | <br><b>3</b> |
| .1                 | Makefile . . . . . | 3            |
| .2                 | myfind.c . . . . . | 3            |

## 1 实验内容

编写程序myfind

### 1.1 命令语法

```
myfind <pathname> [-comp <filename> | -name <str>...]
```

### 1.2 命令语义

1. myfind <pathname>的功能：除了具有与程序4-7相同的功能外，还要输出在<pathname>目录子树之下，文件长度不小于4096字节的常规文件的数量。程序不允许打印出任何路径名。

## 2 程序设计与实现

### 2.1 程序设计

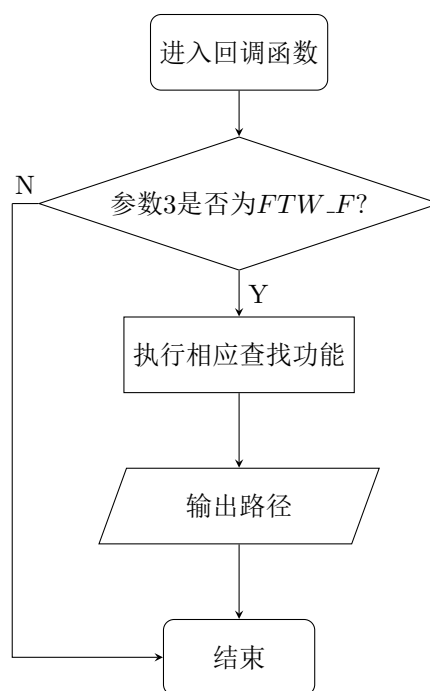
首先简单地分析课本程序4-7。myftw接受一个路径和一个回调函数作为参数。它为路径分配内存，然后调用dopath完成路径遍历的实际功能。dopath中通过lstat获取文件的信息，然后调用回调函数。其中调用的第三个参数将遍历到的文件系统节点分为了文件、目录、不可读的目录、不能执行stat的文件4类。然后dopath对可读的目录进行递归调用。这样实现了对目录树的先序遍历。

由于myftw提供了基本可用的接口来在遍历目录树的时候进行自定义的操作，为了完成实验要求，就需要对应的回调函数。只需要微调4-7中的myfunc，当文件尺寸比4096大时进行计数，就能完成所需要的功能。

### 2.2 程序实现

代码见附录：代码清单中的代码 2。任务1的实现较为简单，只需要在回调函数的参数3为FTW\_F且文件类型是S\_IFREG时，进一步判断文件的大小是否大于4096即可。

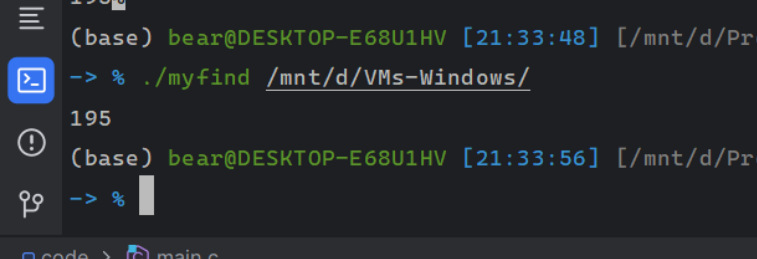
图 1: 流程图



## 3 程序测试

运行结果如图 2。该结果和目录中文件的实际大小相符。可见实现是正确的。

图 2: 运行结果



```
(base) bear@DESKTOP-E68U1HV [21:33:48] [/mnt/d/Pr  
-> % ./myfind /mnt/d/VMs-Windows/  
195  
(base) bear@DESKTOP-E68U1HV [21:33:56] [/mnt/d/Pr  
-> %
```

The image shows a terminal window with a dark background. On the left side, there is a vertical toolbar with icons for a menu, a terminal, an error/warning, and a search. The terminal text shows a user named 'bear' on a machine named 'DESKTOP-E68U1HV'. The prompt is '(base)'. The user has executed the command './myfind /mnt/d/VMs-Windows/'. The output of the command is '195'. Below this, the prompt is again '(base)' and the user has entered a new command starting with '-> %'. At the bottom of the terminal, there is a status bar showing 'code' and 'main.c'.

## 附录：代码清单

### .1 Makefile

代码 1: Makefile

```
1 OBJS = main.o \  
2     error.o \  
3     pathalloc.o \  
4  
5 CC = gcc  
6 CFLAGS = -Wall -g -std=c99  
7  
8 %.o: %.c  
9     $(CC) $(CFLAGS) -c $< -o $@  
10  
11 myfind: $(OBJS)  
12     $(CC) $(CFLAGS) $(OBJS) -o myfind  
13  
14 all: myfind  
15  
16 clean:  
17     rm -f *.o myfind  
18  
19 .PHONY: clean all
```

### .2 myfind.c

代码 2: 程序实现

```
1 #include "apue.h"  
2 #include <dirent.h>  
3 #include <limits.h>  
4  
5 #include <libgen.h> // basename  
6  
7 typedef int (Callback)(const char *, const struct stat *, int);  
8  
9 static Callback simple_statistic;  
10  
11 static int myftw(char *, Callback *);  
12  
13 static int dopath(Callback *);  
14  
15 static long nreg, ndir, nblk, nchr, nfifo, nlink, nsock, ntot, nless4k;  
16  
17 #define NARGS 16  
18  
19 int main(int argc, char *argv[])  
20 {  
21     int ret;  
22     if (!(argc == 2))  
23     {  
24         err_quit("usage: myfind <params>");  
25     }  
26  
27     if (argc == 2)  
28     {  
29         ret = myftw(argv[1], simple_statistic);
```

```
30         ntot = nreg + ndir + nblk + nchr + nfifo + nslink + nsock;
31         if (ntot == 0)
32         {
33             ntot = 1;
34         }
35
36         printf("%ld", nless4k);
37     }
38
39     return ret;
40 }
41
42 #define FTW_F 1
43 #define FTW_D 2
44 #define FTW_DNR 3
45 #define FTW_NS 4
46
47 static char *fullpath;
48 static size_t pathlen;
49
50 static int myftw(char *pathname, Callback *func)
51 {
52     fullpath = path_alloc(&pathlen);
53
54     if (pathlen <= strlen(pathname))
55     {
56         pathlen = strlen(pathname) * 2;
57         if ((fullpath = realloc(fullpath, pathlen)) == NULL)
58         {
59             err_sys("realloc failed");
60         }
61     }
62     strcpy(fullpath, pathname);
63     return (dopath(func));
64 }
65
66 static int dopath(Callback *func)
67 {
68     struct stat statbuf;
69     struct dirent *dirp;
70     DIR *dp;
71     int ret, n;
72     if (lstat(fullpath, &statbuf) < 0)
73     {
74         return (func(fullpath, &statbuf, FTW_NS));
75     }
76     if (S_ISDIR(statbuf.st_mode) == 0)
77     {
78         return (func(fullpath, &statbuf, FTW_F));
79     }
80
81     if ((ret = func(fullpath, &statbuf, FTW_D)) != 0)
82     {
83         return (ret);
84     }
```

```
85     n = strlen(fullpath);
86     if (n + NAME_MAX + 2 > pathlen)
87     {
88         pathlen *= 2;
89         if ((fullpath = realloc(fullpath, pathlen)) == NULL)
90         {
91             err_sys("realloc failed");
92         }
93     }
94     fullpath[n++] = '/';
95     fullpath[n] = 0;
96     if ((dp = opendir(fullpath)) == NULL)
97     {
98         return (func(fullpath, &statbuf, FTW_DNR));
99     }
100    while ((dirp = readdir(dp)) != NULL)
101    {
102        if (strcmp(dirp->d_name, ".") == 0 || strcmp(dirp->d_name, "..") == 0)
103        {
104            continue;
105        }
106        strcpy(&fullpath[n], dirp->d_name);
107        if ((ret = dopath(func)) != 0)
108        {
109            break;
110        }
111    }
112    fullpath[n - 1] = 0;
113    if (closedir(dp) < 0)
114    {
115        err_ret("can't close directory %s", fullpath);
116    }
117    return (ret);
118 }
119
120 static int simple_statistic(const char *pathname, const struct stat *statptr, int type)
121 {
122     switch (type)
123     {
124     case FTW_F:
125         switch (statptr->st_mode & S_IFMT)
126         {
127         case S_IFREG:
128             nreg++;
129             if (statptr->st_size > 4096)
130             {
131                 nless4k++;
132             }
133             break;
134         case S_IFBLK:
135             nblk++;
136             break;
137         case S_IFCHR:
138             nchr++;
139             break;
```

```
140         case S_IFIFO:
141             nfifo++;
142             break;
143         case S_IFLNK:
144             nmlink++;
145             break;
146         case S_IFSOCK:
147             nsock++;
148             break;
149         case S_IFDIR:
150             err_dump("for S_IFDIR for %s", pathname);
151     }
152     break;
153 case FTW_D:
154     ndir++;
155     break;
156 case FTW_DNR:
157     err_ret("can't read directory %s", pathname);
158     break;
159 case FTW_NS:
160     err_ret("stat error for %s", pathname);
161     break;
162 default:
163     err_dump("unknown type %d for pathname %s", type, pathname);
164 }
165 return 0;
166 }
```