



Unix 实验报告

实验:	实验 2 目录树的遍历
专业:	计算机科学与技术
班级:	1 班
姓名:	姚怀聿
学号:	22920202204632

2022 年 10 月 21 日

目 录

一、	实验内容描述	2
二、	设计、实验构思	3
	构思:	3
	功能 1 的构思	3
	功能 2 的构思	4
	功能 3 的构思	5
	实现细节:	6
	功能 1 实现细节	8
	功能 2 实现细节	8
	功能 3 实现细节	10
三、	实验结果	11
	测试功能 1	12
	测试功能 2	12
	测试功能 3	13
四、	体会和建议	13
五、	完成人姓名及完成时间	14

一、 实验内容描述

本实验的目的是掌握与文件和目录树有关的系统调用和库函数。。

实验要求编写程序 `myfind`;

命令语法为:

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

具体要实现的功能如下:

1. `myfind <pathname>` 的功能:

除了具有如程序 4-22 运行后所打印出的内容以外, 还要输出在<pathname>目录子树下, 文件长度不大于 4096 字节的常规文件在所有允许访问的普通文件中所占的百分比。程序不允许打印出任何路径名。

2. `myfind <pathname> -comp <filename>`的功能:

命令仅仅输出在<pathname>目录子树之下, 所有与<filename>文件内容一致的文件的绝对路径名。不允许输出任何其它的路径名, 包括不可访问的路径名。其中, <filename>是常规文件的路径名(非目录名, 但是其路径可以包含目录)。

3. `myfind <pathname> -name <str>...`的功能:

命令输出在<pathname>目录子树之下, 所有与<str>...序列中文件名相同的文件的绝对路径名。不允许输出不可访问的或无关的路径名。其中, <str>...是一个以空格分隔的文件名序列(不带路径)。

程序在实现时的要求如下：

1. 在以上三个功能实现时，<pathname>和<filename>均既可以是绝对路径名，也可以是相对路径名。<pathname>既可以是目录，也可以是文件，此时，目录为当前工作目录。
2. 尽可能地提高程序的效率
3. 避免因打开太多文件而产生的错误
4. 遍历目录树时，访问结点（目录项）的具体操作应当由遍历函数 dopath 携带的函数指针参数决定


二、 设计、实验构思

构思：

功能 1 的构思

功能一是本次实验三个功能中实现起来最简单的一个功能，只需要在程序 4-22 的基础上增加一个统计<pathname>目录子树下，文件长度不大于 4096 字节的常规文件的个数占有所有允许访问的普通文件的百分比即可，具体实现时，我在全局增加了一个 int 类型的变量 nlit，用于记录遍历到的所有文件长度不大于 4096 字节的常规文件的个数。然后在 main 函数中用该个数除以所有允许访问的常规文件的总数量，得到功能要求的百分比，最后在 main 函数中输出。程序具体实现如下：

```
35 typedef int Myfunc(const char*, const struct stat*, int);
36 static long nreg, ndir, nblk, nchr, nfifo, nlink, nsock, ntot, nlit; // 分别表示常规文件、目录文件...
37
38 static int path_noloop(const char* pathname) {
39     char* pos;
```



```

46 static int myfunc1(const char* pathname, const struct stat* statptr, int type) {
47     switch(type) {
48         case FIND_F: // 文件
49             switch(statptr->st_mode & S_IFMT) {
50                 case S_IFREG: nreg++; break;
51                 case S_IFBLK: nblk++; break;
52                 case S_IFCHR: nchr++; break;
53                 case S_IFIFO: nfifo++; break;
54                 case S_IFLNK: nslink++; break;
55                 case S_IFSOCK: nsock++; break;
56                 case S_IFDIR: fprintf(stderr, "for S_IFDIR for %s", pathname);
57             }
58             if(statptr->st_size <= 4096) nlit++; ← 如果是文件，且该文件的大小不大于4096字节，让nlit++
59             break;
60         case FIND_D:
61             ndir++; break;
62         case FIND_DNR: cur_path = getcwd(abs_path, PATHSIZE);
63             fprintf(stderr, "%s: ", fullpath); perror(""); break;
64         case FIND_NS:
65             fprintf(stderr, "%s: ", fullpath); perror(""); break;
66         default:
67             fprintf(stderr, "unknown type %d for pathname %s\n", type, pathname);
68     }
69     return 0;
70 }

```

```

if(argc == 2) {
    ret = myfind(argv[1], myfunc1, TYPE_P);
    ntot = nreg + ndir + nblk + nchr + nfifo + nslink + nsock;
    if(ntot == 0) ntot = 1; // 避免除数为0 所有文件所占的百分比都为0
    fprintf(stdout, "regular files    = %5d %9.2f %%\n", nreg, nreg * 100.0 / ntot);
    fprintf(stdout, "directories      = %5d %9.2f %%\n", ndir, ndir * 100.0 / ntot);
    fprintf(stdout, "block special   = %5d %9.2f %%\n", nblk, nblk * 100.0 / ntot);
    fprintf(stdout, "char special    = %5d %9.2f %%\n", nchr, nchr * 100.0 / ntot);
    fprintf(stdout, "FIFOs           = %5d %9.2f %%\n", nfifo, nfifo * 100.0 / ntot);
    fprintf(stdout, "symbolic links  = %5d %9.2f %%\n", nslink, nslink * 100.0 / ntot);
    fprintf(stdout, "sockets         = %5d %9.2f %%\n", nsock, nsock * 100.0 / ntot);
    fprintf(stdout, "<=4096Bytes     = %5d %9.2f %%\n", nlit, nlit * 100.0 / ntot); ← 在main函数中打印
} else {

```

功能 2 的构思

功能 2 要求输出所有在<pathname>目录子树下，与<filename>文件中的内容相同的所有文件的绝对路径。本功能我认为是本次实验中，三个功能中最难以实现的一个功能，对于该功能的实现，综合了上一次实验中的读、写文件。具体思路如下：

新写一个 myfunc2 函数，在该函数中处理。函数 myfunc2 的函数原型和整体架构与 myfunc1 相同，其不同之处在于对文件和目录的处理上的不同。进入 myfunc2 函数，首先判断传入的是什么类型，这里只会有 3 个类型，分别是 FIND_F、FIND_DNR、FIND_NS，分别代表“文件”、“目录不能打开”、“无法 stat”，这里不会有目录类型，如果是目录类型，则会在 dopath 函数中递归处理。

这其中最主要的是处理 FIND_F 类型，一旦发现该类型，说明递归到达叶子结点，我们需要对这个文件进行读写操作，然

后与之前在 main 函数中读入的<filename>中的内容做比较，如果相同，就输出该文件的绝对路径。

这其中，最麻烦的就是输出该文件的绝对路径这一功能；要得到该文件的绝对路径，在递归到该叶子结点之前，我们需要先记录下当前递归处理的文件所在的目录名称。每当递归到叶子结点的时候，就调用 chdir 函数将工作目录更改为`fullpath`除去最后的文件名的路径；如果叶子结点的文件内容与<filename>文件内容相同，就调用 getcwd 函数得到当前工作目录的绝对路径，在回溯的时候重新将工作路径改变回之前的。实现的时候还要注意处理字符串，将当前文件的名称附加在绝对路径的最后输出。程序具体实现如下：

```
72 static int myfunc2(const char* pathname, const struct stat* statptr, int type) { // pathname是传入的绝对路径或相对路径
73     if(type == FIND_F) {
74         int fd;
75         char* fullbuf = (char*)malloc(sizeof(char) * c_filelen); // 开辟一块和c_filelen一样大小的空间 用于存放从文件读入的数据
76         memset(fullbuf, 0, sizeof(fullbuf));
77         if((fd = open(pathname, O_RDONLY)) < 0) { // 将文件打开 并将文件描述符放在fd中
78             fprintf(stderr, "Error: ", pathname);
79             perror("");
80         }
81         if(read(fd, fullbuf, c_filelen) < 0) {
82             fprintf(stderr, "Error: ", pathname);
83             perror("");
84         }
85         //printf("%s\n", fullbuf); // fullbuf没问题
86         //printf("c_filebuf: %s\n", c_filebuf); // c_filebuf没问题
87         if(memcmp(fullbuf, c_filebuf, statptr->st_size) == 0) { // memcmp没问题
88             char* tmpopath = (char*)malloc(sizeof(char) * PATHSIZE);
89             char* curpath = (char*)malloc(sizeof(char) * PATHSIZE);
90             const char* pos = strrchr(pathname, '/');
91             if(pos == NULL) { // 如果没有 '/'
92                 pos = pathname;
93             }
94             if((tmpopath = getcwd(NULL, 0)) == NULL) perror("getcwd()");
95             strcpy(curpath, fullpath);
96             char* pos2;
97             pos2 = strrchr(curpath, '/');
98             *pos2 = 0;
99             chdir(curpath);
100             if((curpath = getcwd(NULL, 0)) == NULL) perror("getcwd()");
101             fprintf(stdout, "%s\n", curpath, pos);
102             if(chdir(tmpopath) < 0) perror("chdir()");
103             free(curpath);
104             return 1;
105         }
106     } else if(type == FIND_DNR) {
107         fprintf(stderr, "Error: ", fullpath);
108         perror("read()");
109     } else if(type == FIND_NS) {
110         fprintf(stderr, "Error: ", fullpath);
111         perror("stat()");
112     }
113     return 0;
114 }
```

判断类型

如果两文件内容相同

将最右边 '/' 及后面的文件名删除

改变工作路径

获得改变后的绝对路径

当前工作路径的绝对路径附加文件名输出

最后将工作路径改回原本的路径

功能 3 的构思

功能 3 的实现相比功能 2 要更加简单一些，只需要在遍历到叶子结点文件的时候判断这个文件名与一开始输入的 str 序列中有没有名称相同的即可，一旦找到一个名称相同的文件，就直接输出这个文件的绝对路径（得到绝对路径方法与实现功能 2 大致相同），并直接退出循环。myfunc3 具体程序如下：

```

118 static int myfunc3(const char* pathname, const struct stat* statptr, int type) {
119     if(type == FIND_F) { // 是文件
120         int fd, i;
121         char* namebuf = (char*)malloc(sizeof(char) * NAMESIZE); // 开辟一块和NAMESIZE一样大小的空间 用于存放文件名称
122         char* tmpopath = (char*)malloc(sizeof(char) * PATHSIZE);
123         char* curpath = (char*)malloc(sizeof(char) * PATHSIZE);
124         const char* pos;
125         pos = strrchr(pathname, '/');
126         strcpy(namebuf, pos + 1);
127
128         for(i = 0; i < str_num; i++) {
129             if(strcmp(namebuf, name_str[i]) == 0) {
130                 char* pos2;
131                 strcpy(curpath, fullpath);
132                 pos2 = strrchr(curpath, '/');
133                 *pos2 = 0;
134                 if(tmpopath = getcwd(NULL, 0)) == NULL perror("getcwd()");
135                 if(chdir(curpath) < 0) perror("chdir()");
136                 if(curpath = getcwd(NULL, 0)) == NULL perror("getcwd()");
137                 if(chdir(tmpopath) < 0) perror("chdir()");
138                 fprintf(stdout, "%s%s\n", curpath, pos);
139                 break;
140             }
141         }
142         free(namebuf);
143         free(tmpopath);
144         free(curpath);
145     } else if(type == FIND_DNR) {
146         fprintf(stderr, "%s ", fullpath);
147         perror("readlink()");
148     } else if(type == FIND_NS) {
149         fprintf(stderr, "%s ", fullpath);
150         perror("stat()");
151     }
152     return 0;
153 }

```

判断类型

pos+1存储的是文件名称，拷贝到namebuf中去

遍历str中的名称序列

如果找到相同的名称

得到当前的相对路径

得到当前的绝对路径

改变当前工作路径

得到改变后的路径

最后将路径改回来

直接退出循环

实现细节:

整体上实现时，适当定义全局变量，由于要实现三个功能，故定义了三个全局 int 变量，分别表示一个功能，基于程序 4-22 修改了 dopath 的函数原型，需要向 dopath 函数中传入两个参数，一个是使用的函数，另一个是当前的类型。程序截图如下：

```

17 #define TYPE_P 1
18 #define TYPE_C 2
19 #define TYPE_N 3

```

分别表示功能1、2、3

```

155 static int dopath(Myfunc* func, int type) {
156     struct stat statbuf;
157     DIR* dp;
158     struct dirent* dirp;
159     int ret, n;
160     if(type == TYPE_P) {
161         struct stat statbuf;
162         DIR* dp;
163         struct dirent* dirp;
164         int ret, n;
165         if(lstat(fullpath, &statbuf) < 0) // stat error
166             return (func(fullpath, &statbuf, FIND_NS));
167
168         if(S_ISDIR(statbuf.st_mode) == 0) // 不是目录
169             return (func(fullpath, &statbuf, FIND_F));
170         // 下面是目录
171         if((ret = func(fullpath, &statbuf, FIND_D)) != 0) {
172             fprintf(stderr, "FIND_D error\n");
173             return ret; // 如果不是0表示出错 这里直接返回
174         }
175     }
176 }

```

dopath函数需要传入的参数

在 main 函数中判断应该调用哪种函数，程序截图如下：

```

264     if(argc == 2) {
265         ret = myfind(argv[1], myfunc1, TYPE_P); ← 如果只有两个参数，说明是功能1
266         ntot = nreg + ndir + nblk + nchr + nfifo + nslink + nsock;
267         if(ntot == 0) ntot = 1; // 避免除数为0 所有文件所占的百分比都为0
268         fprintf(stdout, "regular files    = %5ld %9.2F %%\n", nreg, nreg * 100.0 / ntot);
269         fprintf(stdout, "directories  = %5ld %9.2F %%\n", ndir, ndir * 100.0 / ntot);
270         fprintf(stdout, "block special = %5ld %9.2F %%\n", nblk, nblk * 100.0 / ntot);
271         fprintf(stdout, "char special  = %5ld %9.2F %%\n", nchr, nchr * 100.0 / ntot);
272         fprintf(stdout, "FIFOs         = %5ld %9.2F %%\n", nfifo, nfifo * 100.0 / ntot);
273         fprintf(stdout, "symbolic links = %5ld %9.2F %%\n", nslink, nslink * 100.0 / ntot);
274         fprintf(stdout, "sockets       = %5ld %9.2F %%\n", nsock, nsock * 100.0 / ntot);
275         fprintf(stdout, "<=4096Bytes    = %5ld %9.2F %%\n", nlit, nlit * 100.0 / ntot);
276     } else {
277         if(strcmp(argv[2], "-comp") == 0) {
278             if(argc > 4) {
279                 fprintf(stderr, "Usage: ./myfind <pathname> [-comp filename]\n");
280                 exit(1);
281             }
282             struct stat statres;
283             if(lstat(argv[1], &statres) < 0) {
284                 fprintf(stderr, "%s: ", argv[1]);
285                 perror("");
286                 exit(1);
287             }
288             strcat(c_filename, argv[3]);
289             //printf("203G c_filename: %s\n", c_filename);
290             int fd = open(argv[3], O_RDONLY);
291             if(lstat(c_filename, &c_filestat) < 0) perror("stat()"), exit(1);
292             if(S_ISDIR(c_filestat.st_mode)) {
293                 fprintf(stderr, "%s is a directory, but it should be a filename\n", c_filename);
294                 exit(1);
295             }
296             if((c_filelen = read(fd, c_filebuf, BUFSIZE)) < 0) {
297                 perror("read()");
298                 exit(1);
299             }
300             ret = myfind(argv[1], myfunc2, TYPE_C);
301         } else if(strcmp(argv[2], "-name") == 0) {
302             str_num = argc - 3;
303             // str都是文件名
304             int i;
305             for(i = 0; i < str_num; i++) { // 为每一个str开辟NAMESIZE大小的空间
306                 strcpy(name_str[i], argv[i + 3]);
307             }
308             ret = myfind(argv[1], myfunc3, TYPE_N);
309         } else {
310             fprintf(stderr, "Usage: ./myfind <pathname> [-comp filename] [-name str]\n");
311             exit(1);
312         }
313     }
314     exit(ret);
315 }

```

如果传入的第二个参数是"-comp" 则判断为功能2

其中，功能 2 的情况下，argc 应该等于 4，否则报错。

```

300     ret = myfind(argv[1], myfunc2, TYPE_C);
301 } else if(strcmp(argv[2], "-name") == 0) { ← 如果第二个参数是"-name"
302     str_num = argc - 3;
303     // str都是文件名
304     int i;
305     for(i = 0; i < str_num; i++) { // 为每一个str开辟NAMESIZE大小的空间
306         strcpy(name_str[i], argv[i + 3]);
307     }
308     ret = myfind(argv[1], myfunc3, TYPE_N);
309 } else {
310     fprintf(stderr, "Usage: ./myfind <pathname> [-comp filename] [-name str]\n");
311     exit(1);
312 }
313 }
314 exit(ret);
315 }

```

说明是功能3

实现三个功能时都需要判断 pathname 是否是一个路径名，如果不是，则报错。

```

243 static int myfind(const char* pathname, Myfunc* func, int type) {
244     strcpy(fullpath, pathname);
245     struct stat statres;
246     if(lstat(fullpath, &statres) < 0) {
247         fprintf(stderr, "%s ", fullpath);
248         perror("stat()");
249         exit(1);
250     }
251     if(S_ISDIR(statres.st_mode)) { // 如果是文件
252         fprintf(stderr, "error: %s is a filename, but it should be a pathname\n", fullpath);
253         exit(1);
254     }
255     return (dopath(func, type));
256 }
257

```

判断pathname是否是一个路径名
如果不是，报错

功能 1 实现细节

功能 1 的实现未涉及到太多细节，需要注意的是 `nlit` 不需要加到总数上。

功能 2 实现细节

功能 2 实现细节较多，分几点阐述：

(1) 判断<filename>是否是一个文件名，如果不是，则报错。

```
strcpy(c_filename, argv[3]);
//printf("203G c_filename: %s\n", c_filename);
int fd = open(argv[3], O_RDONLY);
if(lstat(c_filename, &c_filestat) < 0) perror("stat()"), exit(1);
if(S_ISDIR(c_filestat.st_mode)) {
    fprintf(stderr, "%s is a directory, but it should be a filename\n", c_filename);
    exit(1);
}
if((c_filelen = read(fd, c_filebuf, BUFSIZE)) < 0) {
    perror("read()");
    exit(1);
}
```

如果是目录，报错

(2) `dopath` 函数：

如果是 `FIND_D`、`FIND_NS`、`FIND_DNR` 表示到达叶子结点；剩下一种则是 `FIND_F`，表示是还是一个目录，递归处理。

```
191 } else if(type == TYPE_C) {
192     if(lstat(fullpath, &statbuf) < 0) { // stat error
193         char* curpath = getcwd(NULL, 0);
194         fprintf(stderr, "curpath : %s\n", curpath);
195         return func(fullpath, &statbuf, FIND_NS);
196     }
197     if(!S_ISDIR(statbuf.st_mode)) {
198         return func(fullpath, &statbuf, FIND_F);
199     }
200     if((dp = opendir(fullpath)) == NULL) {
201         return func(fullpath, &statbuf, FIND_DNR);
202     }
203     n = strlen(fullpath);
204     fullpath[n++] = '/';
205     fullpath[n] = 0;
206     while((dirp = readdir(dp)) != NULL) {
207         strcpy(fullpath + n, dirp->d_name);
208         if(path_noloop(fullpath)) {
209             if(ret = dopath(func, type) != 0) break;
210         }
211     }
212     fullpath[n - 1] = 0;
213     if(closedir(dp) < 0) perror("closedir()");
214     return ret;
215 } else { // TYPE_N
```

递归处理时注意不要处理 `../` 和 `./` 文件夹，这会产生回路，使用一个函数 `path_noloop` 判断是否会产生回路。

(3) path_noloop 函数实现细节:

```
38 static int path_noloop(const char* pathname) {
39     char* pos;
40     pos = strrchr(pathname, '/'); // 取到最右边的 '/' 的位置 保存在 pos 中
41     if(pos == NULL) exit(1); // 如果没有 '/'
42     if(strcmp(pos + 1, ".") == 0 || strcmp(pos + 1, "..") == 0) return 0;
43     else return 1;
44 }
45
```

表示出现回路

(4) 如果是 FIND_DNR 或者 FIND_NS

直接报错即可

```
105     } else if(type == FIND_DNR) {
106         fprintf(stderr, "%s ", fullpath);
107         perror("readdir()");
108     } else if(type == FIND_NS) {
109         fprintf(stderr, "%s ", fullpath);
110         perror("stat()");
111     }
```

(5) 如果是 FIND_F

表示是文件，需要进一步处理。

在 main 函数中先将 <filename> 文件中的内容读入 c_filebuf 中

```
295     if((c_filelen = read(fd, c_filebuf, BUFSIZE)) < 0) {
296         perror("read()");
297         exit(1);
298     }
```

在 myfunc2 函数中将遍历到的文件中的内容读入 fullbuf 中

```
if(type == FIND_F) {
    int fd;
    char* fullbuf = (char*)malloc(sizeof(char) * c_filelen); // 开辟一块和 c_filelen 一样大小的空间 用于存放从文件读入的数据
    char* tmp_path = (char*)malloc(sizeof(char) * PATHSIZE);
    char* cur_path = (char*)malloc(sizeof(char) * PATHSIZE);
    memset(fullbuf, 0, sizeof(fullbuf));
    if((fd = open(pathname, O_RDONLY)) < 0) { // 将文件打开 并将文件描述符放在 fd 中
        fprintf(stderr, "%s: ", pathname);
        perror("");
    }
    if(read(fd, fullbuf, c_filelen) < 0) {
        fprintf(stderr, "%s: ", pathname);
        perror("");
    }
}
```

读入 fullbuf 中

判断 c_filebuf 和 fullbuf 中的内容是否相同，若相同，则更改当前的工作目录为 fullpath 中除去最后一个文件名的目录，然后取得当前工作目录的绝对路径，最后改变回原样。

```

//printf( "filebuf: %s\n", c_filebuf); // c_filebuf没问题
if(memcmp(fullbuf, c_filebuf, statptr->st_size) == 0) { // memcmp没问题
    const char* pos = strrchr(pathname, '/');
    if((tmppath = getcwd(NULL, 0)) == NULL) perror("getcwd()");
    strcpy(curpath, fullpath);
    char* pos2;
    pos2 = strrchr(curpath, '/');
    *pos2 = 0;
    if(chdir(curpath) < 0) perror(chdir());
    if((curpath = getcwd(NULL, 0)) == NULL) perror("getcwd()");
    fprintf(stdout, "%s%s\n", curpath, pos);
    if(chdir(tmppath) < 0) perror("chdir()");
    free(fullbuf);
    free(tmppath);
    free(curpath);
    return 0;
}

```

功能 3 实现细节

功能 3 在处理绝对路径时的细节与功能 2 实现时大致相同，这里不再赘述。

功能 3 输入进来的 str 是一个序列，里面存放的是一系列的文件名，在 main 函数中处理该序列，将它们放入全局的一个字符串数组中去。

全局字符串数组：

```

32 static char name_str[STRNUM][NAMESIZE];

300 } else if(strcmp(argv[2], "-name") == 0) {
301     str_num = argc - 3;
302     // str都是文件名
303     int i;
304     for(i = 0; i < str_num; i++) {
305         strcpy(name_str[i], argv[i + 3]);
306     }
307     ret = myfind(argv[1], myfunc3, TYPE_N);
308 } else {
309     fprintf(stderr, "Usage: ./myfind <pathname> [-comp filename] [-name str]\n");
310     exit(1);
311 }

```

放入该数组中去

对于递归遍历到的每一个文件名称，将它放入 namebuf 中去

```

if(type == FIND_F) { // 是文件
    int fd, i;
    char* namebuf = (char*)malloc(sizeof(char) * NAMESIZE); // 开辟一块和NAMESIZE一样大小的空间 用于存放文件名称
    char* tmppath = (char*)malloc(sizeof(char) * PATHSIZE);
    char* curpath = (char*)malloc(sizeof(char) * PATHSIZE);
    const char* pos;
    pos = strrchr(pathname, '/');
    strcpy(namebuf, pos + 1);
}

```

namebuf 与全局定义的 name_str 字符串数组中的每一项比较，一旦找到相同的，就将该文件的绝对路径输出，然后跳

出循环，继续递归到下一个结点。

```
124
125     for(i = 0; i < str_num; i++) {
126         if(strcmp(namebuf, name_str[i]) == 0) {
127             char* pos2;
128             strcpy(curpath, fullpath);
129             pos2 = strrchr(curpath, '/');
130             *pos2 = 0;
131             if((tmppath = getcwd(NULL, 0)) == NULL) perror("getcwd()");
132             if(chdir(curpath) < 0) perror("chdir()");
133             if((curpath = getcwd(NULL, 0)) == NULL) perror("getcwd()");
134             if(chdir(tmppath) < 0) perror("chdir()");
135             fprintf(stdout, "%s%s\n", curpath, pos);
136             break;
137         }
138     }
139     free(namebuf);
140     free(tmppath);
141     free(curpath);
```

三、 实验结果

源程序名	可执行程序名
myfind.c	myfind

编译生成可执行文件：

使用如下指令：

`make myfind`，即可得到可执行文件`myfind`

```
rongrong@rongrong-virtual-machine:~/Desktop/huaiyuyao/unix/homework/homework_2$ ls
myfind.c myfind_glob.c
rongrong@rongrong-virtual-machine:~/Desktop/huaiyuyao/unix/homework/homework_2$ make myfind
cc      myfind.c  -o myfind
rongrong@rongrong-virtual-machine:~/Desktop/huaiyuyao/unix/homework/homework_2$ ls
myfind myfind.c myfind_glob.c
rongrong@rongrong-virtual-machine:~/Desktop/huaiyuyao/unix/homework/homework_2$
```

运行程序：

测试功能 1

```
rongrong@rongrong-virtual-machine:~/Desktop/huaiyuyao/unix/homework/homework_2$ ./myfind ..
regular files = 12      80.00 %
directories   = 3       20.00 %
block special = 0      0.00 %
char special = 0      0.00 %
FIFOs       = 0      0.00 %
symbolic links = 0     0.00 %
sockets     = 0      0.00 %
<=4096Bytes = 5       33.33 %
rongrong@rongrong-virtual-machine:~/Desktop/huaiyuyao/unix/homework/homework_2$ ./myfind ../..
regular files = 12      75.00 %
directories   = 4       25.00 %
block special = 0      0.00 %
char special = 0      0.00 %
FIFOs       = 0      0.00 %
symbolic links = 0     0.00 %
sockets     = 0      0.00 %
<=4096Bytes = 5       31.25 %
rongrong@rongrong-virtual-machine:~/Desktop/huaiyuyao/unix/homework/homework_2$ ./myfind .
regular files = 3       75.00 %
directories   = 1       25.00 %
block special = 0      0.00 %
char special = 0      0.00 %
FIFOs       = 0      0.00 %
symbolic links = 0     0.00 %
sockets     = 0      0.00 %
<=4096Bytes = 0       0.00 %
```

经检验，该功能无误。

测试功能 2

首先使用 tree 命令看一下当前目录的结构：

其中 backup.c 和 myfind_glob.c 两文件中的内容相同，
backup.c 是由 myfind_glob.c 文件拷贝到上一级目录中去的。

```
rongrong@rongrong-virtual-machine:~/Desktop/huaiyuyao/unix/homework$ tree .
.
├── backup.c
├── homework_1
│   ├── 22920202204632.pdf
│   ├── apue.h
│   ├── error.c
│   ├── gen_ran.c
│   ├── Makefile
│   └── timewrite.c
├── homework_2
│   ├── myfind
│   ├── myfind.c
│   └── myfind_glob.c
└── README.md
```

这两个文件的内容相同

使用`./myfind .. -comp ../backup.c`

```
rongrong@rongrong-virtual-machine:~/Desktop/huaiyuyao/unix/homework/homework_2$ ./myfind .. -comp ../backup.c
/home/rongrong/Desktop/huaiyuyao/unix/homework/backup.c
/home/rongrong/Desktop/huaiyuyao/unix/homework/homework_2/myfind_glob.c
```

结果无误。

使用`./myfind .. -comp myfind_glob.c`

```
rongrong@rongrong-virtual-machine:~/Desktop/huaiyuyao/unix/homework/homework_2$ ./myfind .. -comp myfind_glob.c
/home/rongrong/Desktop/huaiyuyao/unix/homework/backup.c
/home/rongrong/Desktop/huaiyuyao/unix/homework/homework_2/myfind_glob.c
```

结果无误。

```
rongrong@rongrong-virtual-machine:~/Desktop/huaiyuyao/unix/homework/homework_2$ ./myfind ../../ -comp myfind_glob.c
/home/rongrong/Desktop/huaiyuyao/unix/homework/backup.c
/home/rongrong/Desktop/huaiyuyao/unix/homework/homework_2/myfind_glob.c
rongrong@rongrong-virtual-machine:~/Desktop/huaiyuyao/unix/homework/homework_2$ ./myfind ../../ -comp ../homework_2/myfind_glob.c
/home/rongrong/Desktop/huaiyuyao/unix/homework/backup.c
/home/rongrong/Desktop/huaiyuyao/unix/homework/homework_2/myfind_glob.c
```

多次测试，结果均无误。

测试功能 3

```
rongrong@rongrong-virtual-machine:~/Desktop/huaiyuyao/unix/homework$ ./homework_2/myfind ../../ -name myfind_glob.c
/home/rongrong/Desktop/huaiyuyao/unix/homework/homework_2/myfind_glob.c
rongrong@rongrong-virtual-machine:~/Desktop/huaiyuyao/unix/homework$ ./homework_2/myfind ../../ -name backup.c
/home/rongrong/Desktop/huaiyuyao/unix/homework/backup.c
```

多次测试，结果均无误。

四、 体会和建议

体会：本次实验大量运用了 C 语言中字符串处理的知识，让我感受到打好 C 语言的基础是多么的重要！以及这次的实验使用到了一个新的函数 `getcwd`，这个函数一开始用的时候不知道该怎么传参，得到什么内容，我使用 man 手册去查看该函数的使用方法，才知道这个函数的正确使用姿势。本次实验基本上只更改了 `myfunc`，而对其他的函数没有做太大的修改，这让我意识到程序的可拓展性也是很重要的。以及动态

分配内存要记得及时释放，否则极易造成内存泄露，这都是以前知道的知识，但是在这次实验中才真正感受到这些知识在实践中的重要性。

建议：希望以后上机课之前，老师能提前说明实验课需要做什么，比如要用到哪些知识点，以及书上哪些程序对本次实验是比较有帮助的，我们可以通过看书本上的例程去更好的实现实验要求。以及可能要用到的不熟悉的函数能为我们做详细的解释。

五、 完成人姓名及完成时间

完成人姓名	完成时间
姚怀聿	2022 年 10 月 24 日