实验报告（二）

李国楷 22  4126

1.实验名称：实验 2 目录树的遍历

2.实验内容描述

编写程序 myfind
命令语法：
myfind <pathname> [-comp <filename> | -name <str>...]
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：
<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）。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。
（3）myfind <pathname> -name <str>...的功能：
<str>...是一个以空格分隔的文件名序列(不带路径)。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。
<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、注意尽可能地提高程序的效率。注意避免因打开太多文件而产生的错误。

3、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。这样程序的结构清晰，可扩充性好。

3.设计原理

main 函数部分：

```
int main(int argc, char** argv){
    if(argc==2)
        exit(fun1(argc,argv));
    if(argc==4) {
        if (strcmp(argv[2], "-comp") == 0)
            exit(fun2(argc, argv));
        else if(strcmp(argv[2],"-name")==0)
            exit(fun3(argc, argv));
    }
    else if(argc>4)
        if(strcmp(argv[2],"-name")==0)
            exit(fun3(argc, argv));
    exit(-1);
}
```

根据输入变量的情况来取用对应的函数，fun1,fun2,fun3 分别对应要求中的(1),(2),(3)

对于要求 1:

我们直接在课本代码的基础上加以修改，多设置一个变量 nsmall，在 myfunc1 中对文件进行判断，如果文件大小满足要求则 nsmall++

如下：

```
case S_IFREG:nreg++;
    if (statptr->st_size <= 4096)
        nsmall++;
    break;
```

对于要求 2:

这里用到函数 fun2,myfunc2,myftw2,dopath2

从 fun2 开始，直接调用 myftw2，在 myftw2 中使用 realpath 函数将输入的两个路径都得到绝对路径，如果 pathname 原本就是绝对路径的话 realpath 不会把路径写入 fullname，所以在上面先用 strcpy 把 pathname 写到 fullname，这样保证无论如何 fullname 都有写入

然后检查一下输入格式是否合法，如果合法则调用 dopath2。

```
static int myftw2(char *pathname,char *pathname2,Myfunc *func){
    fullpath= path_alloc(&pathlen);
    if(pathlen<= strlen(pathname)){
        pathlen= strlen(pathname)*2;
        if((fullpath= realloc(fullpath,pathlen))==NULL)
            err_sys("");
    }
    strcpy(fullpath,pathname);
    realpath(pathname,fullpath);
    fullpath2= path_alloc(&pathlen2);
    if(pathlen2<= strlen(pathname2)){
        pathlen2= strlen(pathname2)*2;
        if((fullpath2= realloc(fullpath2,pathlen2))==NULL)
            err_sys("");
    }
    strcpy(fullpath2,pathname2);
    realpath(pathname2,fullpath2);
    struct stat statbuf;
    DIR *dp;
    if(lstat(fullpath2,&statbuf)<0)
        err_quit("");
    if(S_ISREG(statbuf.st_mode)==0)
        err_sys("<filename> must be regular!");
    return dopath2(func);
}
```

到了 dopath2 函数，在 dopath 基础上进行修改，路径如果是常规文件就调用 compare_files 函数进行检查文件是否一致，如果一直则输出当前路径。路径如果是文件夹则递归调用 dopath2 进一步搜索。在这里 myfunc2 只有报错作用，一般不会被调用。

对于要求 3:

这里用到函数 fun3,myftw3,dopath3，myfunc2(用于报错，故不用写 myfunc3，继续用 2 就行)

在 fun3 中需要将 argv 数组中的每个字符串收集起来放到一个字符串中，然后送给

myftw3

```
int fun3(int argc,char **argv){
    int ret;
    int n=0;
    int i=3;
    for (; i < argc; ++i) {
        n+= strlen(argv[i]);
        n++;
    }
    char str[n+1];
    char *p=str;
    for (i = 3; i < argc; ++i) {
        strcpy(p,argv[i]);
        p+= strlen(argv[i]);
        *p=' ';
        p++;
    }
    ret=myftw3(argv[1],str,myfunc2);
    return ret;
}
```

到了 myftw3 只需要用 realpath 获取绝对路径，然后调用 dopath3，与前面的 myftw 相差不大。

dopath3 的修改也不大，对于文件夹就递归调用读取，对于文件就调用 compare_names 函数进行检查，如果文件名一样就输出文件名。

```
else {
    n= strlen(fullpath);
    char *p=&fullpath[n-1];
    while(*p!='/')p--;
    p++;
    if(compare_names(p,files))
        fprintf(stderr, "%s\n", fullpath);
}
return ret;
```

这里在进入 compare_names 前先把指针放到最后的文件名的第一个字符上，方便后续进行核对

```
int compare_names(char *s1,char *s2){
    int flag=0;
    int i,j=0;
    int n1= strlen(s1),n2= strlen(s2);
    while(!flag){
        if(j>=n2)break;
        i=0;
```

```
        while(s1[i]==s2[j]){
            i++;
            j++;
        }
        if(i<n1) {
            while (s2[j] != ' ' && j < n2)j++;
            j++;
        }
        else if(i>=n1 && (s2[j]==' '|| s2[j]=='\0'))
            flag=1;
        else break;
    }
    return flag;
}
```

在 compare_names 中使用两重循环，分别比对 s1 和 s2 是否相同就行，当 i 到达 s1 数组长度时且 s2[j]是空格或者结束符号则表示找到同名文件，flag 置 1 然后 return 就行。

4.实验结果：

实验文件由 main.c, erro.c extend.c Makefile apue.h 组成

其中 erro.c 和 extend.c 中写的是 apue.h 中定义函数的具体内容 main.c 为主要实验函数

首先通过 Makefile 文件和 make 指令进行编译，得到可执行文件

```
#myfind
myfind:main.o extend.o erro.o
        gcc main.o extend.o erro.o -o myfind
main.o:main.c
        gcc -c main.c
extend.o:extend.c
        gcc -c extend.c
erro.o:erro.c
        gcc -c erro.c
clean:
        rm myfind *.c
```

```
[cs214126@mcore 2]$ make -f Makefile
gcc main.o extend.o erro.o -o myfind
[cs214126@mcore 2]$
```

编译好后按照要求进行调用

调用 1:eg.搜索用户目录文件 ./myfind <pathname>

```
[cs214126@mcore 2]$ ./myfind ~/
regular files  =     26, 76.47 %
directories    =      8, 23.53 %
block special  =      0,  0.00 %
char special   =      0,  0.00 %
FIFOs          =      0,  0.00 %
symbolic links =      0,  0.00 %
sockets        =      0,  0.00 %
<=4096 files   =     13, 38.24 %
[cs214126@mcore 2]$
```

调用 2:eg.查找内容相同文件 ./myfind <pathname> -comp <filename>

```
[cs214126@mcore 2]$ ./myfind ../../test/ -comp ../../test/file1.txt
/home/cs21/cs214126/test/file1.txt
/home/cs21/cs214126/test/file2.txt
```

注：file1 和 file2 的内容完全一致，这里仅对比文件内容不对比名称

调用 3:搜索同名文件 ./myfind <pathname> -name <str>

```
[cs214126@mcore 2]$ ./myfind ../../test/ -name file2.txt
/home/cs21/cs214126/test/file2.txt
```

这里只核对名称，所以只找出 file2, 虽然 file1 的内容和它一样。

```
[cs214126@mcore 2]$ ./myfind ../../test/ -name file2.txt file1.txt
/home/cs21/cs214126/test/file1.txt
/home/cs21/cs214126/test/file2.txt
```

加上 file1.txt 就都可以找到啦！

5.体会和建议

这个实验的程序有三个要求，还是很费精力的，main 函数写了 356 行，不过好在有书上的源代码，可以在其基础上进行修改缝补，如果期末项目也能像这样有个框架就太好了，有难度但是努努力确确实实可以做出来，做出来后很有成就感，尤其是成功跑通的那一刻！

真的建议后面的实验都采用这种给定框架加功能的方式考核，没有从头开始的迷茫，自始自终都有个方向！

6.实验人姓名：李国楷 完成时间 2023.10.28