目录树的遍历

一、实验名称

实验二： 目录树的遍历

二、实验内容和要求

主要以课本 99 — 102 页程序为框架，在此基础上进行扩展。

- **正确理解程序 4-7 递归降序遍历目录层次结构，并按照文件类型进行计数，各函数的含义如下：**
  - `myftw(char* pathname, Myfunc* func)`
    - `pathname` 给出指要遍历开始的目录；
    - `func` 是 `Myfunc` 类型的函数指针，定义访问的实际操作；
  - `dopath(Myfunc* func)`
    - `func` 是 `Myfunc` 类型的函数指针，定义访问的实际操作；
  - `int myfunc(const char * pathname,const struct stat *statptr,int type)`
    - `pathname` 指向当前访问节点的路径名；
    - `statptr` 指向当前访问节点的 `i` 节点的结构，该结构保存有许多该文件的信息;
    - `type` 给出当前访问节点的类型，在实验中可以自己定义它的涵义;
    - `myfunc()` 在程序4-7中它的值总是0，但是在 `dopath()` 函数中，返回值非0意味着终止遍历。
    - `dopath()` 函数中，`myfunc()` 返回值非0意味着终止遍历;
- **根据用户输入的命令行选项的不同，来实现三种功能：**
  - `argc = 2` 时，命令格式为：`myfind <pathname>`，要求：
    - 实现程序 4-7 的功能，程序也不允许打印出任何路径名；
    - 在上述基础上，统计出，在常规文件中，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；
  - `argc = 4 && argv[2] == "-comp"` 时，命令格式为：`myfind <pathname> -comp <filename>`，要求：
    - 输出在 `pathname` 目录子树之下，所有与 `filename` 文件内容一致的文件绝对路径；
    - 为提高程序效率，在比较文件是否相同时，可先比较两个文件的大小；
    - 当参数 `pathname` 不是绝对路径时，调用 `getcwd()` 等函数取得文件的绝对路径名；
  - `argc >= 4 && argv[2] == "-name"` 时，命令格式为：`myfind <pathname> -name <str>...`，要求：
    - 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；

三、实验设计与实现

1、异常情况判断

- `argc, *argv[]` 输入参数检查

```
/* check argcs */
if (!(argc == 2 || (argc == 4 && !strcmp(argv[2], "-comp")) || (argc >= 4 && !strcmp(argv[2],
{
    puts("usage:  myfind <pathname> [-comp <filename> | -name <str>...]");
    exit(0);
}
```

- `lstat()` 函数异常情况处理

```
    if (lstat(argv[3], &statbuf) < 0) err_quit("lstat error: %s\n", argv[3]);
    if (!S_ISREG(statbuf.st_mode)) err_quit("that is not a regular file: %s\n", argv[3]);
```

- `read()` 函数异常情况处理

```
    if (read(fd, filebuf, filesize) != filesize)
        err_sys("read error '%s'\n", argv[3]);
```

- `open()` 函数异常情况处理

```
    if ((fd = open(argv[3], O_RDONLY)) == -1)
        err_sys("can't open the file '%s'\n", argv[3]);
```

- `malloc()` 函数异常情况处理

```
    /* malloc for <filename> */
    if ((filebuf = (char *)malloc(sizeof(char) * filesize)) == NULL)
        err_sys("malloc error\n");

    /* malloc for pathname */
    if ((comparebuf = (char *)malloc(sizeof(char) * filesize)) == NULL)
        err_sys("malloc error\n");
```

2、static 静态全局变量

- static 声明的全局变量、函数，作用域被限定在该文件内。当多个文件一起编译时，如本程序，各个文件的全局变量、函数是可以互相访问的，通过将只用于本文件的函数和变量声明为静态的，避免多文件编译时的命名冲突；

```
/* static functions */
typedef int Myfunc(const char *, const struct stat *, int);

static int dopath(Myfunc *);
static int myftw(char *, Myfunc *);
static Myfunc myfunc, myfunc1, myfunc2;
static void getfilename(const char* pathname, char* filename);
static void getRealpath(const char *pathname, char *realpath, size_t len);

static long n4096, nreg, ndir, nblk, nchr, nfifo, nslink, nsock, ntot;
```

3、通用函数实现

- `myftw()` `dopath()` 函数；在实现时，可以直接参考课本，无需进行更改；
- 获得绝对路径函数：`static void getRealpath(const char *pathname, char *realpath, size_t len)`
    - 若 `pathname` 以 `/` 开头，则本身为绝对路径，无需修改；

```
    /* realpath, don't need modify */
    if (pathname[0] == '/')
    {
        strcpy(realpath, pathname);
        return;
    }
```

- 若需要进行修改，进行拼接 `当前工作路径 + 相对路径`；

```
    /* need modify, joint cwd and filename */
    getcwd(realpath, len);

    len = strlen(realpath);
    if (realpath[len - 1] != '/')
        realpath[len++] = '/';

    if (pathname[0] == '.')
    {
        path = pathname + 2;
        strcat(&realpath[len], path);
        return;
    }
    else
    {
        strcat(&realpath[len], pathname);
        return;
    }
```

- 实现结果如下：

```
/* get realpath according to pathname */
static void getrealpath(const char* pathname, char* realpath, size_t len)
{
    const char* path;

    /* realpath, don't need modify */
    if (pathname[0] == '/')
    {
        strcpy(realpath, pathname);
        return;
    }

    /* need modify, joint cwd and filename */
    getcwd(realpath, len);

    len = strlen(realpath);
    if (realpath[len - 1] != '/')
        realpath[len++] = '/';

    if (pathname[0] == '.')
    {
        path = pathname + 2;
        strcat(&realpath[len], path);
        return;
    }
    else
    {
        strcat(&realpath[len], pathname);
        return;
    }
}
```

- 获得文件名的函数：`static void getfilename(const char* pathname, char* filename)`

- 从 `pathname` 末尾开始遍历，直到第一个 `/` 结束；若不含 `/` 则直接返回原始文件名；

```
/* get filename according to pathname */
static void getfilename(const char* pathname, char* filename)
{
    for (int i = strlen(pathname) - 1; i >= 0; i--)
    {
        if (pathname[i] == '/')
        {
            strcpy(filename, pathname + i + 1);
            return;
        }
    }

    strcpy(filename, pathname);

    return;
}
```

- 实现结果如下：

```
/* get filename according to pathname */
static void getfilename(const char* pathname, char* filename)
{
    for (int i = strlen(pathname) - 1; i >= 0; i--)
    {
        if (pathname[i] == '/')
        {
            strcpy(filename, pathname + i + 1);
            return;
        }
    }

    strcpy(filename, pathname);

    return;
}
```

4、实验流程

- **功能一**
  - `argc = 2`，命令格式为：`myfind <pathname>`；

```
/* argc == 2: myfind <pathname> */
if (argc == 2)
{
    myftw(argv[1], myfunc);
```

  - 在程序 4-7 的基础上，需要多进行一次判断：文件大小是否不大于4KB；

```
switch (statptr->st_mode & S_IFMT)
{
case S_IFREG:
    nreg++;
    if (statptr->st_size <= 4096)
        n4096++;
    break;
```

  - 打印结果时，因需要求 `size <= 4KB` 所占普通文件中的比例，所以需要判断普通文件数是否为零；

```
ntot = nreg + ndir + nblk + nchr + nfifo + nslink + nsock;
if (!ntot) ntot = 1;

if (!nreg) nreg = 1;
```

  - 为了使最终结果更加清晰，最终选择以表格形式打印结构；

```
printf(" ----------------------------------------\n");
printf("|      TYPES       |  NUMNERS  |  PERCENT  |\n");
printf(" ----------------------------------------\n");
printf("|  REGULAR FILES   |  %-7ld  |  %-5.2f %%  |\n", nreg, nreg * 100.0 / ntot);
printf(" ----------------------------------------\n");
printf("|  <=  4KB FILES   |  %-7ld  |  %5.2f %%  |\n", n4096, n4096 * 100.0 / nreg);
printf(" ----------------------------------------\n");
printf("|   DIRECTORIES    |  %-7ld  |  %5.2f %%  |\n", ndir, ndir * 100.0 / ntot);
printf(" ----------------------------------------\n");
printf("|  BLOCK SPECIAL   |  %-7ld  |  %5.2f %%  |\n", nblk, nblk * 100.0 / ntot);
printf(" ----------------------------------------\n");
printf("|  CHAR  SPECIAL   |  %-7ld  |  %5.2f %%  |\n", nchr, nchr * 100.0 / ntot);
printf(" ----------------------------------------\n");
printf("|      FIFO        |  %-7ld  |  %5.2f %%  |\n", nfifo, nfifo * 100.0 / ntot);
printf(" ----------------------------------------\n");
printf("|  SYMBOLIC LINK   |  %-7ld  |  %5.2f %%  |\n", nsock, nsock * 100.0 / ntot);
printf(" ----------------------------------------\n");
```

- **功能二**
  - `argc = 4 && argv[2] == "-comp"`，命令格式为：`myfind <pathname> -comp <filename>`；

```
/* argc == 4 && argv[2] == "-comp": myfind  <pathname>  -comp  <filename> */
if (argc == 4 && strcmp(argv[2], "-comp") == 0)
{
```

- `malloc` 出两处空间，分别用于将来存储 `<pathname>` 和 `<filename>` 中的文件内容；

```
    /* malloc for <filename> */
    if ((filebuf = (char *)malloc(sizeof(char) * filesize)) == NULL)
        err_sys("malloc error\n");

    /* malloc for pathname */
    if ((comparebuf = (char *)malloc(sizeof(char) * filesize)) == NULL)
        err_sys("malloc error\n");
```

- 读取出 `filename` 中文件的内容；

```
    /* read <filename> content */
    filesize = statbuf.st_size;

    if ((fd = open(argv[3], O_RDONLY)) == -1)
        err_sys("can't open the file '%s'\n", argv[3]);

    /* malloc for <filename> */
    if ((filebuf = (char *)malloc(sizeof(char) * filesize)) == NULL)
        err_sys("malloc error\n");

    /* malloc for pathname */
    if ((comparebuf = (char *)malloc(sizeof(char) * filesize)) == NULL)
        err_sys("malloc error\n");

    if (read(fd, filebuf, filesize) != filesize)
        err_sys("read error '%s'\n", argv[3]);

    close(fd);
```

- 从 `<pathname>` 处起始，递归遍历目录子树；

```
    myftw(argv[1], myfunc1);
```

- 对于遍历到的每一个目录子树，首先检查类型和大小是否相同；

```
if (type == FTW_F && (statptr->st_mode & S_IFMT) == S_IFREG && statptr->st_size == filesize)
```

- 相同类型和大小的文件，将其内容读入缓冲区；

```
int fd;
char realpath[1024] = {0};

if ((fd = open(pathname, O_RDONLY)) == -1)
{
    puts("open error!");
    exit(0);
}

if (read(fd, comparebuf, filesize) != filesize)
{
    puts("read error!");
    exit(0);
}

close(fd);
```

- 当内容相同时，输出该文件对应的绝对路径；

```
    /* check content */
    if (strcmp(filebuf, comparebuf) == 0)
    {
        getrealpath(pathname, realpath, sizeof(realpath));
        printf("%s\n", realpath);
    }
```

- **功能三**
  - `argc >= 4 && argv[2] == "-name"`，命令格式为：`myfind <pathname> -name <str>...`；

```
/* argc >= 4 && argv[2] == "-name": myfind  <pathname>  -name str...      */
if (argc >= 4 && strcmp(argv[2], "-name") == 0)
{
```

- 递归遍历 `pathname` 下的目录子树，对于遍历到的每一个文件，均检查其是否和 `str...` 中的任一名称相同；

```
/* check file type */
if (type == FTW_F)
{
    getfilename(pathname, filename);

    for (int i = 0; i < arglen; i++)
    {
        if (strcmp(argvs[i], filename) == 0)
        {
            getrealpath(pathname, realpath, sizeof(realpath));
            printf("%s\n", realpath);
        }
    }
}
```

四、实验结果

- 编译，链接 `myfind.c` 文件；

```
echo -e "----------------------------"
echo "-> start compiling ..."
gcc -c myfind.c -o myfind.o
if [ $? -eq 0 ]
then
    echo "-> compile successfully !"
fi

echo "-> start  linking  ..."
gcc error.o pathalloc.o myfind.o -o myfind
if [ $? -eq 0 ]
then
    echo "-> linking successfully !"
fi
echo -e "----------------------------"
```

```
----------------------------
-> start compiling ...
-> compile successfully !
-> start  linking  ...
-> linking successfully !
----------------------------
```

- `argc = 2` ，命令格式为：`myfind <pathname>` 时，测试结果为：

```
$ TEST FOR QUESTION-1: ./myfind /usr
 ----------------------------------------
|      TYPES       |  NUMNERS  |  PERCENT  |
 ----------------------------------------
|  REGULAR FILES   |  36385    |  78.08 %  |
 ----------------------------------------
|  <=  4KB FILES   |  21622    |  59.43 %  |
 ----------------------------------------
|   DIRECTORIES    |  4508     |   9.67 %  |
 ----------------------------------------
|  BLOCK SPECIAL   |  0        |   0.00 %  |
 ----------------------------------------
|  CHAR  SPECIAL   |  0        |   0.00 %  |
 ----------------------------------------
|      FIFO        |  0        |   0.00 %  |
 ----------------------------------------
|  SYMBOLIC LINK   |  0        |   0.00 %  |
 ----------------------------------------
```

- `argc = 4 && argv[2] == "-comp"` ，命令格式为：`myfind <pathname> -comp <filename>` 时，测试结果为：

```
$ TEST FOR QUESTION-2: ./myfind /home -comp apue.h
/home/catcher/unix/homework2/lib/apue.h
/home/catcher/unix/homework2/apue.h
```

- `argc >= 4 && argv[2] == "-name"` ，命令格式为：`myfind <pathname> -name <str>...` 时，测试结果为：

```
$ TEST FOR QUESTION-3: ./myfind /home -name apue.h myfind.c
/home/catcher/unix/homework2/myfind.c
/home/catcher/unix/homework2/lib/apue.h
/home/catcher/unix/homework2/apue.h
```

五、实验体会和建议

- 上次实验中由于没有配置好 `apue.h` 相关的文件，对于一些作者封装好的函数无法使用，实验中也添加了很多莫名其妙的麻烦；
- 这次虽然并未完全配置好 `apue.h` 相关的头文件，但是，下载并解压好源代码后，通过编译相关需要的 c 文件（`pathalloc` `apue.h` `error.c`），在此次的实验中也已经完全够用；这样省去了查询需要的头文件的麻烦，还可直接使用作者封装好的函数；
- 对于 UNIX 系统中的有关文件，内存，目录等等函数的参数，返回值等了解不清楚，实验中也反复进行了查询；
- Unix系统的文件系统是树形结构。既能扩大文件存储空间，又有利于安全和保密；Unix系统把文件、文件目录和设备统一处理。它把文件作为不分任何记录的字符流进行顺序或随机存取，并使得文件、文件目录和设备具有相同的语法语义和相同的保护机制，这样既简化了系统设计，又便于用户使用。

实验者：张嘉庆 完成时间：2021/10/23