**实验二：遍历文件目录**

**一、实验目的**

* 编程实现程序list.c，列表普通磁盘文件，包括文件名和文件大小。
* 使用vi编辑文件，熟悉工具vi
* 使用Linux的系统调用和库函数
* 体会Shell文件通配符的处理方式以及命令对选项的处理方式

**二、实验要求**

(一) ls基础命令

根据ls后接的处理对象可以分为：

* 0个：列出当前目录下的所有文件
* 普通文件：列出文件
* 目录：列出目录下所有文件

(二) ls自定义选项

* r 递归方式列出子目录(类似find的-print风格)，设计递归程序
* a 列出文件名第一个字符为圆点的普通(默认情况下不列出文件名首字符为圆点的文件)
* l 后跟一整数，限定文件大小的最小值(字节)
* h 后跟一整数，限定文件大小的最大值(字节)
* m 后跟一整数n，限定文件的最近修改时间必须在n天内
* -- 显式地终止命令选项分析

**三、实验环境**

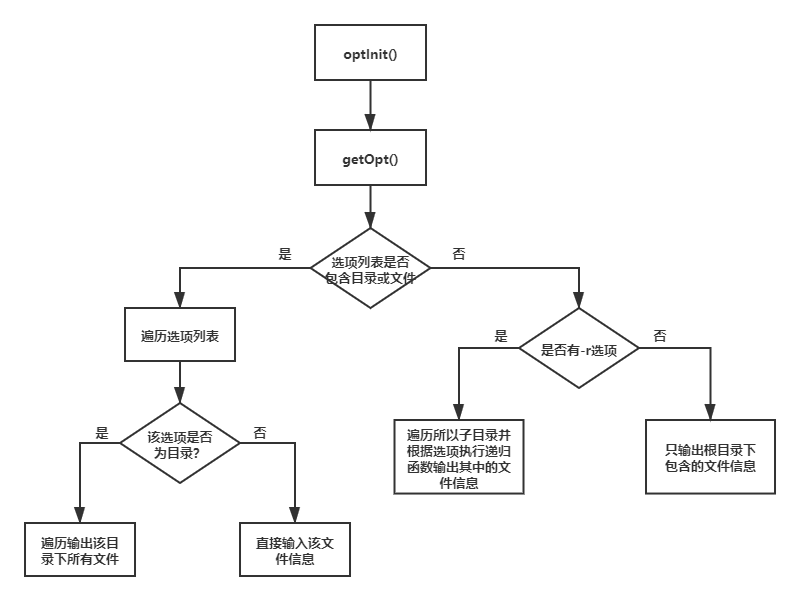
Ubuntu 20.04.2 LTS(GNU/Linux 5.4.0-58-generic x86\_64)

vim

gcc

**四、实验流程**

首先程序会通过自行定义的getOpt()函数进行参数读入，并初始化结构体Option，给opt的各项值赋值，之后判断当前程序是针对某个目录进行列表或列表根目录的文件，对于不同的情况调用scanDir()函数进行目录扫描，调用printS()函数进行文件信息输出，格式为“文件大小 文件路径”，风格与find -print一致，整个程序的流程图如下：



**图一 程序逻辑流图**

**五、数据结构定义**

**1. 选项结构体 Option**

struct Option

{

bool r;//代表是否有r命令

bool a;//代表是否有a命令

bool l;//代表是否有l命令

bool h;//代表是否有h命令

bool m;//代表是否有m命令

int llength;//如存在l命令 则l命令指定最小长度

int hlength;//如存在h命令 则h命令指定最大长度

int mdays;//如存在m命令 则m命令指定时间

int fileAndDirSum;//代表选项中包含的 目录/文件名 的个数

int fileAndDir[MAX\_LENGTH];//将选项中的目录/文件名储存在该数组

};

**2. 函数 struct Option optInit()**

用于初始化Option结构体，返回初始化后的结构体变量opt

**3. 函数 void getOpt(int argc, char \*argv[])**

用于根据用户输入，从数组argv[]中读出字符串，给opt赋值

**4. 函数 void scanDir(char \*dir, int depth)**

用于扫描当前路径\*dir目录下的所有文件和目录，并根据深度depth的值做不同的处理，scanDir()是主要的递归函数，可以用于递归遍历整个目录及子目录

**5. 函数 void printS(char \*path)**

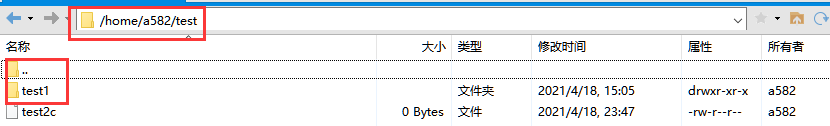
用于打印path路径的文件信息，根据opt的内容，进行筛选和处理后，打印文件信息。格式为“文件大小 文件路径/文件名”

**六、测试案例对比**

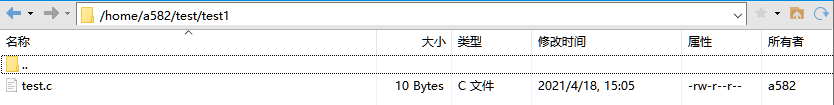
使用gcc编译生成可执行程序exe后，在根目录下运行，首先可以看一下该目录下的文件和目录结构：



**图二 根目录结构**



**图三 test子目录**

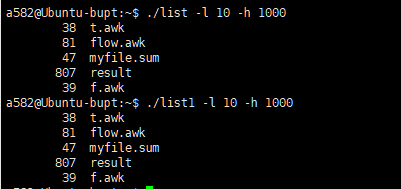


**图四 test1子目录**

下面测试各项命令的执行情况对比，其中list为本程序，list1为示例程序

* **./list –l 10 –h 1000**

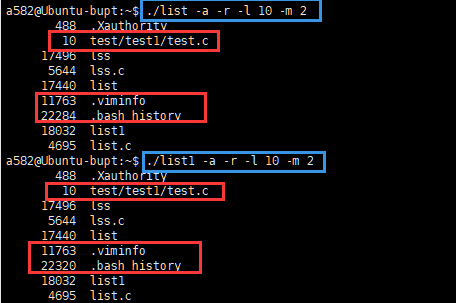
列出文件大小在10~1000之间的文件



**图五 ./list和./list1对比图(-)**

* **./list –a -r -l 50000 –m 2**

递归列出当前目录树下大小超过10且2天内未修改的文件

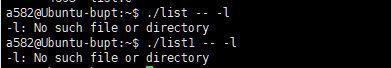


**图六 ./list和./list1对比图(二)**

如图所示，列出的文件包含子目录中的test.c和以圆点开头的文件名的文件.viminfo等

* **./list -- -l**

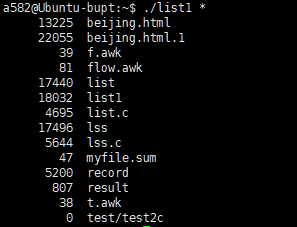
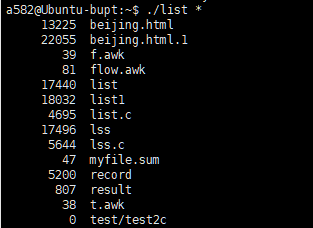
显示取消-l选项，将-l当前文件/目录名处理



**图七 ./list和./list1对比图(三)**

* **./list \***

\*可替换任意字符串，匹配成功则输出文件信息



**图八 ./list和./list1对比图(四)**

* **./list -r test**





**图九 ./list和./list1对比图(五)**

**六、实验心得**

本次实验通过自行编写list程序并执行，实现了对于目录下普通磁盘文件的信息输出，学习了vim编辑工具的使用和Shell下文件通配符的处理方式，以及对于不同的ls选项的处理方式，对于Linux下文件的组织方式和ls命令有了更加深刻的理解。首次使用Linux的一些库函数和系统调用，对于部分相关函数的使用有了全新的体会。

**附 录**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <stdbool.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <unistd.h>

#include <dirent.h>

#include <time.h>

#define MAX\_LENGTH 255

struct Option

{

bool r;

bool a;

bool l;

bool h;

bool m;

int llength;

int hlength;

int mdays;

int fileAndDirSum;

int fileAndDir[MAX\_LENGTH];

};

struct Option opt;

struct Option optInit()

{

struct Option opt;

opt.r = **false**;

opt.a = **false**;

opt.l = **false**;

opt.h = **false**;

opt.m = **false**;

opt.llength = 0;

opt.hlength = 0;

opt.mdays = 0;

opt.fileAndDirSum = 0;

return opt;

};

void getOpt(int argc, char \*argv[])

{

int i = 0;

bool optCancle = **false**;

for (i = 1; i < argc; i++)

{

if (argv[i][0] == '-' && argv[i][1] == '-')

{

optCancle = **true**;

continue;

}

*/\*如果有--选项 则所有自定义选项命令终止分析\*/*

if (argv[i][0] == '-' && !optCancle)

{

switch (argv[i][1])

{

case 'r':

opt.r = **true**;

**break**;

case 'a':

opt.a = **true**;

**break**;

case 'l':

opt.l = **true**;

i++;

opt.llength = atoi(argv[i]);

**break**;

case 'h':

opt.h = **true**;

i++;

opt.hlength = atoi(argv[i]);

**break**;

case 'm':

opt.m = **true**;

i++;

opt.mdays = atoi(argv[i]);

**break**;

default:

**break**;

}

}

else

{

opt.fileAndDir[opt.fileAndDirSum] = i;*//将目录路径存入数组*

opt.fileAndDirSum++;

}

}

};

void printS(char \*path)

{

struct stat st;

time\_t t\_now;

char \*fileName;

time(&t\_now);

int ret = stat(path, &st);

if (ret == -1)

{

printf("%s: No such file or directory**\n**", path);

}

else

{

*//printf("filename:%s\n",fileName);*

if (opt.l)

{

if (st.st\_size < opt.llength)

return;

}

if (opt.h)

{

if (st.st\_size > opt.hlength)

return;

}

if (opt.m)

{

if (t\_now - st.st\_mtime > opt.mdays \* 24 \* 60 \* 60)

return;

}

printf("%10ld %s**\n**", st.st\_size, path);

}

};

void scanDir(char \*dir, int depth) *// 定义目录扫描函数*

{

DIR \*dp; *// 定义子目录流指针*

struct dirent \*entry; *// 定义dirent结构指针保存后续目录*

struct stat statbuf; *// 定义statbuf结构保存文件属性*

char path[512] = {0};

if ((dp = opendir(dir)) == NULL) *// 打开目录，获取子目录流指针，判断操作是否成功*

{

printf("%s**\n**:No such file or directory**\n**", dir);

return;

}

while ((entry = readdir(dp)) != NULL) *// 获取下一级目录信息，如果未否则循环*

{

if (strcmp(".", entry->d\_name) == 0 || strcmp("..", entry->d\_name) == 0) *//如果遇到.文件夹..文件夹就跳过进入下一个循环*

continue;

if (entry->d\_name[0] == '.' && opt.a == **false**)*//如果目录或文件名以圆点开头且无a选项 则不输出信息*

continue;

if (depth == 0)*//若是第一次进入该函数，则将当前目录/文件名赋值给path*

{

sprintf(path, "%s", entry->d\_name);

}

else

{

sprintf(path, "%s/%s", dir, entry->d\_name);*//若是递归调用或确定该文件不是目录，则将dir/entry->d\_name赋值给path*

}

lstat(path, &statbuf); *// 获取下一级成员属性*

if (S\_ISDIR(statbuf.st\_mode))

{

if (opt.r)

{

scanDir(path, depth + 1); *// 若是目录且有-r选项，则递归调用自身，扫描下一级目录的内容*

}

}

else

printS(path);*//否则输出该文件信息即可*

} *// 回到上级目录*

closedir(dp); *// 关闭子目录流*

};

int main(int argc, char \*argv[])

{

opt = optInit();

getOpt(argc, argv);

*/\*不跟具体文件目录\*/*

if (opt.fileAndDirSum == 0)

{

char \_path[MAX\_LENGTH];

getcwd(\_path, MAX\_LENGTH);*//当前目录路径*

scanDir(\_path, 0);

}

else

{

for (int i = 0; i < opt.fileAndDirSum; i++)

{

struct stat st;

lstat(argv[opt.fileAndDir[i]], &st);

if (S\_ISDIR(st.st\_mode))

{

scanDir(argv[opt.fileAndDir[i]], 100);

}

else

{

printS(argv[opt.fileAndDir[i]]);

}

}

}

return 0;

}