1. **实验内容**

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。

（对选项的处理，自行编程逐个分析命令行参数。不考虑多选项挤在一个命令行参数内的情况）

1. **实验步骤**
2. 依照题意分别按照不使用getopt.h和使用getopt.h的方式完成编写程序，并编译。
3. 尝试运行测试样例，观察输出结果是否符合预期。
4. 延伸学习借助getopt.h库中的getopt\_long设计长选项输入的识别。
5. **源代码（不使用getopt.h）**

#include <stdio.h>

#include <dirent.h>

#include <sys/stat.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

int rec = 0, all = 0; //指示递归、显示全部文件的选项状态

int lb = -1, ub = -1; //记录限制输出文件大小的字节数下限和上限

int day = -1; //记录限制输出文件修改时间的天数限制

void visit(const char\*);

int isDir(const char\* path) //判断该路径是否对应一个目录

{

struct stat st;

if(stat(path, &st) == 0)

if(S\_ISDIR(st.st\_mode)) return 1;

return 0;

}

int getFileSize(const char\* path) //获取文件大小

{

struct stat stFile;

if(stat(path, &stFile) == 0) return (int)stFile.st\_size;

}

int getFileModTime(const char\* path) //获取文件修改时间

{

struct stat stFile;

if(stat(path, &stFile) == 0) return (int)stFile.st\_mtime;

}

void visitFile(const char\* path) //访问一个文件

{

int size = getFileSize(path);

int able = 1;

if(lb != -1 && size <= lb) able = 0;

if(ub != -1 && size >= ub) able = 0;

if(day != -1)

{

int mtime = getFileModTime(path);

time\_t now;

time(&now);

if(now - day \* 86400 > mtime) able = 0;

}

if(able && size != -1) printf("%s\n", path);

}

void visitDir(const char\* path) //访问一个目录

{

struct stat st;

DIR\* dir;

struct dirent\* file;

char filepath[255];

dir = opendir(path);

if(dir == NULL) return;

while((file = readdir(dir)) != NULL)

{

memset(filepath, 0, 255);

strcat(filepath, path);

strcat(filepath, "/");

strcat(filepath, file->d\_name);

if(\*(char\*)(file->d\_name) == '.')

{

if(all && lb == -1 && ub == -1 && day == -1) visitFile(filepath);

}

else

{

if(isDir(filepath))

{

if(lb == -1 && ub == -1 && day == -1) printf("%s\n", filepath);

if(rec) visitDir(filepath); //递归访问子目录

}

else visitFile(filepath);

}

}

}

void visit(const char\* path)

{

if(isDir(path)) visitDir(path);

else visitFile(path);

}

char optarg[255]; //用于记录参数的缓冲区

int mygetopt(int argc, char\*\* argv, int cnt)

{

if(cnt >= argc) return -1;

if(strcmp(argv[cnt], "-r") == 0) return 'r';

else if(strcmp(argv[cnt], "-a") == 0) return 'a';

else if(strcmp(argv[cnt], "-l") == 0)

{

strcpy(optarg, argv[cnt + 1]); //将参数读入缓冲区中

return 'l';

}

else if(strcmp(argv[cnt], "-h") == 0)

{

strcpy(optarg, argv[cnt + 1]);

return 'h';

}

else if(strcmp(argv[cnt], "-m") == 0)

{

strcpy(optarg, argv[cnt + 1]);

return 'm';

}

else if(strcmp(argv[cnt], "--") == 0) return -1;

else return -1;

}

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

{

int argv\_cnt = 1;

while(1)

{

int opt\_idx = 0;

int c = mygetopt(argc, argv, argv\_cnt);

if(c == -1) break;

++argv\_cnt;

switch(c)

{

case 'r':

rec = 1;

break;

case 'a':

all = 1;

break;

case 'l':

lb = atoi(optarg);

++argv\_cnt;

break;

case 'h':

ub = atoi(optarg);

++argv\_cnt;

break;

case 'm':

day = atoi(optarg);

++argv\_cnt;

break;

}

}

if(argv\_cnt == argc) //default local addr

{

visitDir(".");

}

else //typed addr

{

for(int i = argv\_cnt; i < argc; ++i)

{

if(strcmp(argv[i], "--") == 0) continue;

visit(argv[i]);

}

}

}

1. **源代码（使用getopt.h）**

#include <stdio.h>

#include <dirent.h>

#include <sys/stat.h>

#include <getopt.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

int rec = 0, all = 0; //指示递归、显示全部文件的选项状态

int lb = -1, ub = -1; //记录限制输出文件大小的字节数下限和上限

int day = -1; //记录限制输出文件修改时间的天数限制

void visit(const char\*);

int isDir(const char\* path) //判断该路径是否对应一个目录

{

struct stat st;

if(stat(path, &st) == 0)

if(S\_ISDIR(st.st\_mode)) return 1;

return 0;

}

int getFileSize(const char\* path) //获取文件大小

{

struct stat stFile;

if(stat(path, &stFile) == 0) return (int)stFile.st\_size;

}

int getFileModTime(const char\* path) //获取文件修改时间

{

struct stat stFile;

if(stat(path, &stFile) == 0) return (int)stFile.st\_mtime;

}

void visitFile(const char\* path) //访问一个文件

{

int size = getFileSize(path);

int able = 1;

if(lb != -1 && size <= lb) able = 0;

if(ub != -1 && size >= ub) able = 0;

if(day != -1)

{

int mtime = getFileModTime(path);

time\_t now;

time(&now);

if(now - day \* 86400 > mtime) able = 0;

}

if(able && size != -1) printf("%s\n", path);

}

void visitDir(const char\* path) //访问一个目录

{

struct stat st;

DIR\* dir;

struct dirent\* file;

char filepath[255];

dir = opendir(path);

if(dir == NULL) return;

while((file = readdir(dir)) != NULL)

{

memset(filepath, 0, 255);

strcat(filepath, path);

strcat(filepath, "/");

strcat(filepath, file->d\_name);

if(\*(char\*)(file->d\_name) == '.') //第一个char为'.'

{

if(all && lb == -1 && ub == -1 && day == -1) visitFile(filepath);

}

else

{

if(isDir(filepath))

{

if(lb == -1 && ub == -1 && day == -1) printf("%s\n", filepath);

if(rec) visitDir(filepath); //递归访问子目录

}

else visitFile(filepath);

}

}

}

void visit(const char\* path)

{

if(isDir(path)) visitDir(path);

else visitFile(path);

}

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

{

static struct option long\_options[] =

{

{"recursive", no\_argument, 0, 'r'},

{"all", no\_argument, 0, 'a'},

{"low", required\_argument, 0, 'l'},

{"high", required\_argument, 0, 'h'},

{"modtime", required\_argument, 0, 'm'},

{NULL, 0, NULL, 0}

};

int argv\_cnt = 1;

while(1)

{

int opt\_idx = 0;

int c = getopt\_long(argc, argv, "ral:h:m:", long\_options, &opt\_idx);

if(c == -1) break;

++argv\_cnt;

switch(c)

{

case 0:

printf("long option %s", long\_options[opt\_idx].name);

if(optarg) printf(" with arg %s", optarg);

printf("\n");

break;

case 'r':

rec = 1;

break;

case 'a':

all = 1;

break;

case 'l':

lb = atoi(optarg);

++argv\_cnt;

break;

case 'h':

ub = atoi(optarg);

++argv\_cnt;

break;

case 'm':

day = atoi(optarg);

++argv\_cnt;

break;

default:

printf("default %s\n", long\_options[opt\_idx].name);

break;

}

}

if(argv\_cnt == argc) //默认路径为

{

visitDir(".");

}

else //遍历指定的目录

{

for(int i = argv\_cnt; i < argc; ++i)

{

if(strcmp(argv[i], "--") == 0) continue;

visit(argv[i]);

}

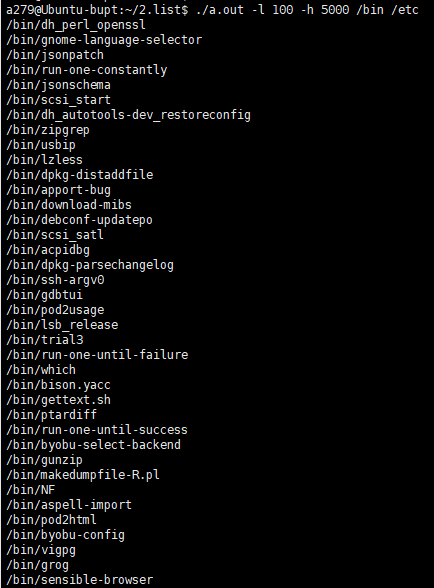
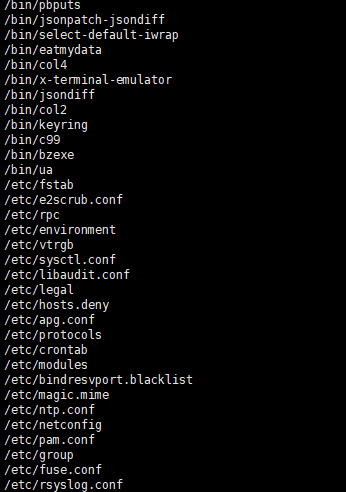
}

}

1. **实验结果**

./a.out -l 100 -h 5000 /bin /etc

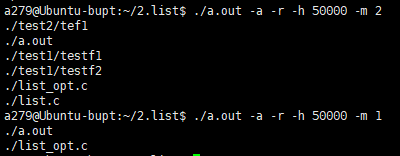
列出大小在100~5000字节之间的文件

（结果较长，截图为部分结果）

./a.out -a -r -h 50000 -m 2

递归式列出当前目录树下大小不超过50000字节且2天内修改过的文件

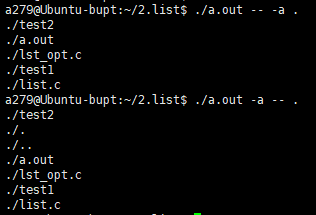


./a.out -- -a

在-a选项前显式地结束命令选项识别，观察到没有输出以‘.’开头的文件。

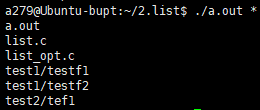
./a.out –a –

将-a选项放在—前，观察到-a选项被正确识别。



./a.out \*

该场景下相当于./a.out a.out list.c list\_opt.c test1 test2



延伸学习——长选项：

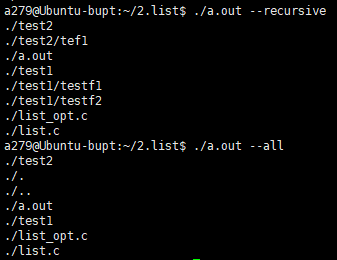
用于处理命令选项的库函数 getopt\_long 用这个函数重新设计选项处理部分，设计长短格式选项。体会这个库函数功能的设计思想。

./a.out –recursive

递归式列出当前目录树下的文件

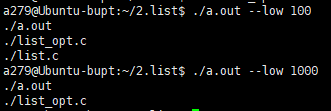
./a.out –all

列出开头为’.’的隐藏文件



./a.out –low 1000

列出当前目录树下大小至少为1000字节的文件



1. **实验总结**

通过本次实验的编写，我学会了如何获取程序运行时的命令输入，不仅采用了现成的工具库getopt.h来处理，还尝试了基于文本比较的命令输入判断。体会到了shell文件通配符的处理方式以及命令对选项的处理方式。最后还拓展学习了如何使用getopt\_long来处理长选项的方法。

非常感谢老师的实验设计！