目录遍历

目录

目录遍历.....1
一、基本要求.....2
二、基础知识.....2
（1）整体思想.....2
（2）DIR 结构体.....2
（3）dirent 结构体.....3
（4）stat 结构体.....3
三、整体框架.....4
四、具体实现.....4
（1）limit_print 函数.....4
（2）printdir 函数.....5
（3）手工处理 option.....5
（4）getopt_long 处理 option.....6
五、总结.....9
（1）手工处理的优点与不足.....9
（2）getopt_long 处理的优点与不足.....10
附录 10

一、基本要求

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。
（1）使用 vi 编辑文件，熟悉工具 vi。
（2）使用 Linux 的系统调用和库函数。
（3）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。

二、基础知识

（1）整体思想

Linux 像 ls 一样遍历目录的方法就是：打开目录——>读取——>关闭目录，相关函数是 opendir->readdir->closedir

```
#include <dirent.h>
DIR *opendir(const char *dirname);
struct dirent *readdir(DIR *dirp);
int closedir(DIR *dirp);
```

（2）DIR 结构体

```
struct __dirstream
  {
   void *__fd;
   char *__data;
   int __entry_data;
   char *__ptr;
   int __entry_ptr;
   size_t __allocation;
   size_t __size;
   __libc_lock_define (, __lock)
  };

typedef struct __dirstream DIR;
```

DIR 结构体类似于 FILE，是一个内部结构，。函数 DIR *opendir(const char *pathname)，即打开文件目录，返回的就是指向 DIR 结构体的指针，而该指针可以被像 readdir、rewinddir、closedir、telldir 和 seekdir 等一些函数利用，这个内部结构保存当前正在被读取的目录的有关信息。

（3） dirent 结构体

```
struct dirent
{
    long d_ino; /* inode number 索引节点号 */

    off_t d_off; /* offset to this dirent 在目录文件中的偏移 */

    unsigned short d_reclen; /* length of this d_name 文件名长 */

    unsigned char d_type; /* the type of d_name 文件类型 */

    char d_name [NAME_MAX+1]; /* file name (null-terminated) 文件名，最长255字符 */
}
```

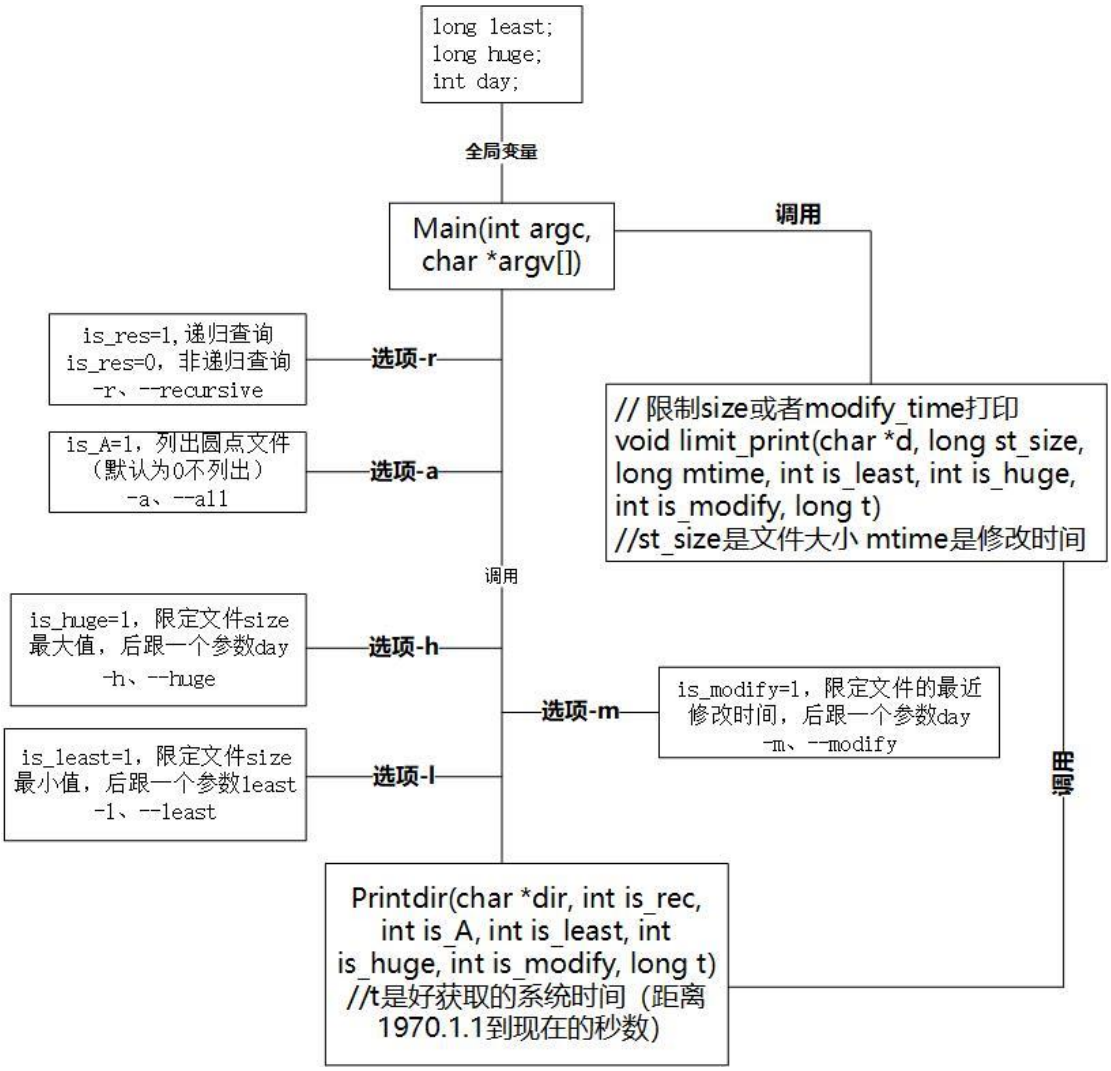
目录文件包含了其他文件的名字以及指向与这些文件有关的信息的指针，关于 dirent 结构体具体信息如下：dirent 结构体存储的关于文件的信息很少，所以 dirent 同样也是起着一个索引的作用，如果想获得类似 ls -l 那种效果的文件信息，必须要靠 stat 函数了

（4）stat 结构体

```
struct stat {
    dev_t     st_dev;         /* 存储该文件的块设备的设备号ID */
    ino_t     st_ino;         /* inode号 */
    mode_t    st_mode;        /* 访问权限及文件类型 */
    nlink_t   st_nlink;       /* link数 */
    uid_t     st_uid;         /* 文件主ID */
    gid_t     st_gid;         /* 组ID */
    dev_t     st_rdev;        /* device ID (if special file) */
    off_t     st_size;        /* 文件大小（字节数）*/
    blksize_t st_blksize;     /* blocksize for filesystem I/O */
    blkcnt_t  st_blocks;      /* 分配的512字节尺寸块个数 */
    struct timespec st_atim;  /* access时间 */
    struct timespec st_mtim;  /* modification时间 */
    struct timespec st_ctim;  /* change时间 */
};
```

- ✓ st_dev:文件的块设备的设备号，包括主设备号和次设备号
  821h/2081d（主设备号：8，此设备号：21H（33），表示/dev/sdc1）
- ✓ st_mode：16bit（文件的基本存取权限和 SUID/SGID 权限 11b）以及问价的类型（若干比特）文件类型 st_mode&S_IFMT（S_IFREG 普通磁盘文件、S_IFDIR 目录文件、S_IFCHR 字符设备、S_IFIFO 管道文件、S_IFLNK 符号链接文件）
- ✓ st_size 和 st_blocks
- ✓ st_atim（访问时间）、st_mtim（修改时间）、st_ctim（信息改变时间）域

三、整体框架



四、具体实现

（1）limit_print 函数

这个函数就是上图提到的，main 函数或者 printdir 函数仅仅需要给其传递参数：文件路径、文件大小 st_size、文件修改时间 mtime、是否限定文件大小选项 is_least 和 is_huge、是否限制文件修改时间选项 is_modify。然后该函数根据相应的参数条件判断打印结果即可，很简单，不再赘述。

（2） printdir 函数

这个函数也是上图提到的，main 函数仅仅需要给其传递参数：目录信息 dir、是否递归选项 is_rec、是否打印全部文件选项 is_A、是否限定文件大小选项 is_least 和 is_huge、是否限制文件修改时间选项 is_modify。

然后具体实现起来也很简单，其中有一个关于递归的处理我觉得比较好，就是先将所有是目录的文件路径存在二维数组 char rec_fpath[][]中，等打印完成当前目录的所有文件之后，再去递归打印这些目录中的文件，这样打印出来的格式比较容易理解。

另外，每次打印文件信息的时候，我都是以“文件的大小，路径名”为格式打印的，简单的演示结果如下：

```
u104@LiSongYang:~$ ./list1 -r -a ch3
ch3:
          4096          ch3/d1
          4096          ch3/.
          4096          ch3/..
         12848          ch3/list
ch3/d1:
          4096          ch3/d1/.
            51          ch3/d1/1.c
            18          ch3/d1/2.txt
          4096          ch3/d1/..
```

源代码附在后面

（3） 手工处理 option

这个思路就是根据 argc 大小对 argv[]一个一个处理，实现起来分支比较多，但是没有比较难的地方。我在实现过程中总体分为四种情况：短选项、长选项、--选项、用户输入的文件路径。

短选项处理：就是根据 argv[i][0] == ‘-‘，然后再去判断 argv[i][1] == ‘r’/’a’/’l’/’h’/’m’这几种情况，另外在处理过程中，对于用户的输入情况作了简单的**容错处理**，用户如果将-r 输入成-p 或者在-l 后面忘了加参数都会抛出相应的错误提示，并且直接退出。另外对于-l/-h/-m 等需要后面跟一个参数的选项，在处理的时候我还是用来一个判断字符串中是否全是数字的子函数 is_digit（char *str）,具体如下：

```
// 判断字符串是否全是数字
int is_digit(char *str)
{
    return (strspn(str, "0123456789") == strlen(str));
}
```

（这类需要参数的 option 使用可以使用形如“-l 20”或者“-l20”这样的格式输入）

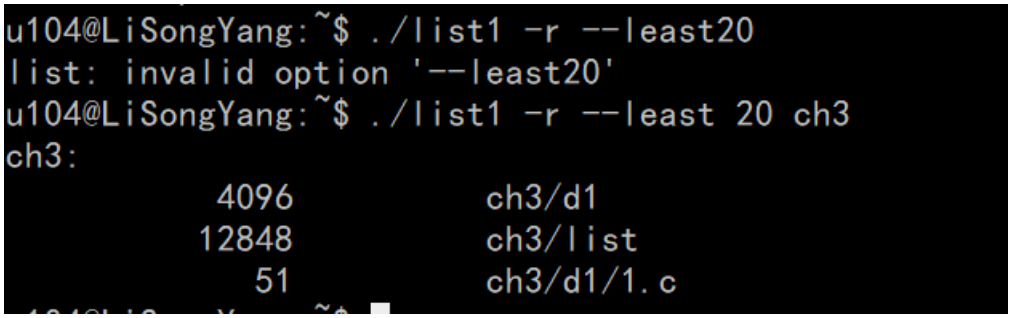
长选项：类似于短选项的处理，对于形如“—least 20”或者“—least=20”的 option 一个一

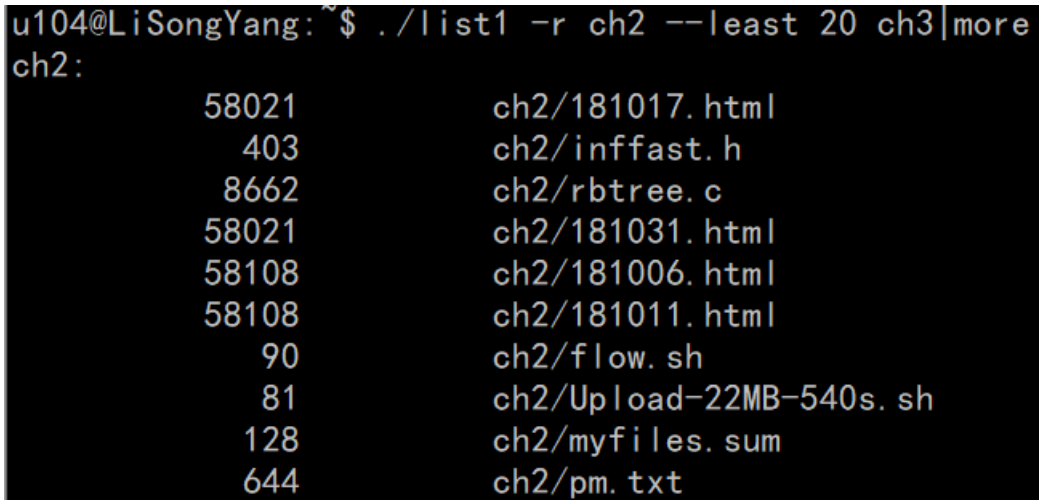
个判断就可以。

--选项：显示终止 option 的分析，这个具体实现就是“--”后面的字符串全部处理成路径名存放在二维数组 user_fpath[][]（**使用这个数组的好处就是可以提取 option 之间的路径名，比如用户在-r 和-a 选项之间输入了一个文件路径 ch3**）。

用户输入的文件路径：像上面说的一样，这个处理就是将所有的不是选项的字符串当成路径名存在数组中（在 option 正确的前提下），然后 option 分析完之后一并处理。

具体实现如下：





源代码附在后面

（4） getopt_long 处理 option

A. 首先具体介绍一下这个函数

```c
#include <unistd.h>

int getopt(int argc, char * const argv[],
           const char *optstring);

extern char *optarg;
extern int optind, opterr, optopt;

#include <getopt.h>

int getopt_long(int argc, char * const argv[],
           const char *optstring,
           const struct option *longopts, int *longindex);

int getopt_long_only(int argc, char * const argv[],
           const char *optstring,
           const struct option *longopts, int *longindex);
```

getopt_long 具体参数

- **argc** 和 **argv** 和 **main 函数**的两个参数一致。
- **optstring:** 表示**短选项**字符串。

  形式如“a:b::cd:“，分别表示程序支持的命令行**短选项**有-a、-b、-c、-d，冒号含义如下：

  (1)只有一个字符，不带冒号——只表示选项， 如-c

  (2)一个字符，后接一个冒号——表示选项后面带一个参数，如-a 100

  (3)一个字符，后接两个冒号——表示选项后面带一个可选参数，即参数可有可无，如果带参数，则选项与参数直接不能有空格形式应该如-b200
- **longopts：**表示**长选项**结构体。结构如下：

```c
struct option
{
    const char *name;
    int         has_arg;
    int        *flag;
    int         val;
};
```

(1) name:表示选项的名称,比如 daemon,dir,out 等。

(2)has_arg:表示选项后面是否携带参数。该参数有三个不同值，如下：

a: no_argument(或者是 0)时 ——参数后面不跟参数值，eg: --version,--help

b: required_argument(或者是 1)时 ——参数输入格式为：--参数 值 或者 --参数=值。eg:--dir=/home

c: optional_argument(或者是 2)时 ——参数输入格式只能为： --参数=值

(3)flag:这个参数有两个意思，空或者非空。

a:如果参数为空 NULL，那么当选中某个长选项的时候，getopt_long 将返回 val 值。eg，可执行程序 --help，getopt_long 的返回值为 h.

b:如果参数不为空，那么当选中某个长选项的时候，getopt_long 将返回 0，并且将 flag 指针参数指向 val 值。eg: 可执行程序 --http-proxy=127.0.0.1:80 那么 getopt_long 返回值为 0，并且 lopt 值为 1。

(4)val：表示指定函数找到该选项时的返回值，或者当 flag 非空时指定 flag 指向的数据的值 val。

- **longindex：longindex 非空**，它指向的变量将记录当前找到参数符合 longopts 里的第几个元素的描述，即是 longopts 的下标值。

全局变量：

（1）optarg：表示当前选项对应的参数值。

（2）optind：表示的是下一个将被处理到的参数在 argv 中的下标值。

（3）opterr：如果 opterr = 0，在 getopt、getopt_long、getopt_long_only 遇到错误将不会输出错误信息到标准输出流。opterr 在非 0 时，向屏幕输出错误。

（4）optopt：表示没有被未标识的选项。

返回值：

（1）如果短选项找到，那么将返回短选项对应的字符。

（2）如果长选项找到，如果 flag 为 NULL，返回 val。如果 flag 不为空，返回 0

（3）如果遇到一个选项没有在短字符、长字符里面。或者在长字符里面存在二义性的，返回“？”

（4）如果解析完所有字符没有找到（一般是输入命令参数格式错误，eg： 连斜杠都没有加的选项），返回“-1”

（5）如果选项需要参数，忘了添加参数。返回值取决于 optstring，如果其第一个字符是“：”，则返回“：”，否则返回“？”。

注意：

（1）longopts 的最后一个元素必须是全 0 填充，否则会报段错误

（2）短选项中每个选项都是唯一的。而长选项如果简写，也需要保持唯一性。

B. 知道上面这些参数之后实现起来就很简单了，我程序中短选项和长选项的实现如下：

```
char *optstring = "ral:h:m:";
static struct option long_options[] =
{
    {"recursive", no_argument, NULL, 'r'},
    {"all", no_argument, NULL, 'a'},
    {"least", required_argument, NULL, 'l'},
    {"huge", required_argument, NULL, 'h'},
    {"modify", required_argument, NULL, 'm'},
    {0, 0, 0 ,0}
};
```

C. 整体框架就是使用 getopt_long 这个处理完所有的 option 之后，就提取用户输入的路径，然后一个一个处理（类似于手工处理中的处理用户输入路径名的操作）

```
while((opt = getopt_long(argc, argv, optstring, long_options, &option_id)) != -1)
{…
}

//printf("%d %d %d\n", optind, user_id, user_top);
while(optind < argc)    //处理用户输入路径名
{…
}
//printf("%d %d %d\n", optind, user_id, user_top);

if (strcmp(wd, ".") == 0 && user_top == 0)  // 默认输入
{…
}
```

简单演示如下：

```
u104@LiSongYang:~$ ./list_getopt -ar|20 ch2|more
ch2:
          58021          ch2/181017.html
            403          ch2/inffast.h
           8662          ch2/rbtree.c
          58021          ch2/181031.html
```

```
u104@LiSongYang:~$ ./list_getopt /etc -r -a --least=20 | more
/etc:
            111          /etc/magic
           4096          /etc/security
           4141          /etc/securetty
           4096          /etc/gss
           4096          /etc/xdg
           4096          /etc/kernel
           4096          /etc/binfmt.d
            711          /etc/hosts.deny
           4096          /etc/alternatives
           1260          /etc/ucf.conf
            554          /etc/localtime
           4096          /etc/.
             34          /etc/ld.so.conf
           4096          /etc/dbus-1
```

五、总结

（1）手工处理的优点与不足

- **优点**

  高适用：用户输入 option 时，可以在其中夹杂着路径名

  容错：对于用户输入的形如“-p”等不可解析、“-l -a”等缺少参数的 option，可以抛出异

常并退出查询

参数形式多样：用户可以输入形如“-l20”、“-l 20”、“—least 20”、“—least=20”这种形式的 option

- **缺点**

  没法处理多选项挤在一个命令行参数内的情况，必须将 option 分开输入

（2）getopt_long 处理的优点与不足

- **优点**

  简单高效：这个命令使用起来简单，不需要判断各种复杂的输入形式，仅仅需要给定 optstring 和 longopts 参数就可以帮我们提取所有符合条件的 option，另外这个命令特别对于夹杂在 option 之间的文件路径特别“友好”，可以在处理完所有 option 之后全部提取（真的很神奇）

  容错：同上

  参数形式多样：同上，另外可以处理**多选项挤在一个命令行参数内的情况**

  高适用：同上

- 缺点：目前没发现，手工处理 option 难以达到这个效果（可能是自己水平有限）

附录


list.c


list_getopt.c


list.exe


list_getopt.exe