# Linux上机作业——“遍历目录”实验报告

## 实验题目

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编辑工具，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。

## 处理对象和选项

1. 处理对象可以有0个到多个
2. 0个：列出当前目录下所有文件
3. 普通文件：列出文件
4. 目录：列出目录下所有文件
5. 实现自定义选项r,a,l,h,m以及--
6. r：递归方式列出子目录（每一项需要包含路径）
7. a：列出文件名第一个字符为圆点的普通文件
8. l：后面跟一个整数，限定文件大小的最小值（字节）
9. h：后面跟一个整数，限定文件大小的最大值（字节）
10. m：后面跟一个整数，限定文件的最近修改时间必须在n天内
11. --：显式地终止命令选项分析

## 整体思路

1. dirent结构体和stat结构体。

次实验需要通过路径对文件及文件夹的一些属性进行访问，根据条件筛选，而获取这些信息我们需要使用到dirent结构体以及stat结构体，查询到这两个结构体内所包含的信息为：

|  |
| --- |
| struct dirent  {  　　long d\_ino; /\*索引节点号 \*/  off\_t d\_off; /\*在目录文件中的偏移 \*/  unsigned short d\_reclen; /\*文件名长 \*/  unsigned char d\_type; /\*文件类型 \*/  char d\_name [NAME\_MAX+1]; /\*文件名，最长255字符 \*/  } |

|  |
| --- |
| struct stat {  mode\_t st\_mode; //文件访问权限  ino\_t st\_ino; //索引节点号  dev\_t st\_dev; //文件使用的设备号  dev\_t st\_rdev; //设备文件的设备号  nlink\_t st\_nlink; //文件的硬连接数  uid\_t st\_uid; //所有者用户识别号  gid\_t st\_gid; //组识别号  off\_t st\_size; //以字节为单位的文件容量  time\_t st\_atime; //最后一次访问文件的时间  time\_t st\_mtime; //最后一次修改文件的时间  time\_t st\_ctime; //最后一次改变文件状态的时间  blksize\_t st\_blksize; //包含该文件的磁盘块的大小  blkcnt\_t st\_blocks; //该文件所占的磁盘块  }; |

红色为根据自定义选项功能发现主要需要查看的内容。

1. 逻辑结构

根据处理对象的数量，0个或多个，需要分别讨论，如果具有多个处理对象，需根据处理对象的不同（路径或文件）分别保存。其中对于文件来说只需要考虑这一个文件即可，但是对于路径而言，依次要进行打开目录，遍历文件，关闭目录这些步骤，而且如果需要递归查看文件目录就需要重复执行上述步骤，大致流程如下：

执行

获取文件名

获取选项

获取路径

判断文件是否满足条件

遍历路径下全部文件

若是文件夹判断是否递归

## 函数功能及实现

1. 全局变量：

|  |
| --- |
| int r = 0; //递归列出子目录，需要列出时为1  int a = 0; //列出隐藏文件，需要时为1  int l = -1; //限定文件最小字节，-1表示不限制  int h = -1; //限定文件最大字节，-1表示不限制  int m = -1; //限定文件最最近修改天数，-1表示不限制 |

1. 主函数需要分析指令，将指令、文件、路径分隔开：

|  |
| --- |
| int main(int argc, char \*argv[]){  if(argc == 1){  //没有参数，显示当前目录文件  }  else{  //多个参数  while(全部参数){  if(argv[i][0] == '-')  //分析指令  else if(argv[i][0] == '/')  //分析路径  else  //文件  }  }  } |

1. 以get\_all\_files\_under\_path函数为例，这个函数用于遍历路径中的文件，同时借由全局变量判断是否将文件信息打印或是递归文件夹：

|  |
| --- |
| void get\_all\_files\_under\_path(char path[]){  DIR \*dp;  struct dirent \*file;  dp = opendir(path);  if(!dp)打开文件夹失败  else{  while((file = readdir(dp)) != NULL){  if(file->d\_type != 4)  //普通文件  else{  //文件夹  if(r==1 && 文件夹不为本文件夹或上级文件夹)  //递归调用此函数  }  }  }  } |

1. 在limit函数中，需要对文件夹信息进行判断从而决定需不需要打印：

|  |
| --- |
| void limit(struct dirent \*file, char path[]){  struct stat statbuf;  stat(file->d\_name, &statbuf);  if(a==0)  //文件名开头为’.’则不显示  if(l!=-1)  //文件大小小于l则不显示  if(h!=-1)  //文件大小大于h则不显示  if(m!=-1)  //文件距离最近修改时间天数大于m则不显示  } |

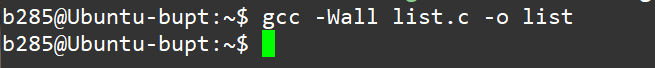
1. folder\_size函数用于计算文件夹的大小，包括其中的隐藏文件夹：

|  |
| --- |
| void folder\_size(char path[]){  while(遍历文件)  if(普通文件)  size+=普通文件大小  else(文件夹)  folder\_size(char path[])  } |

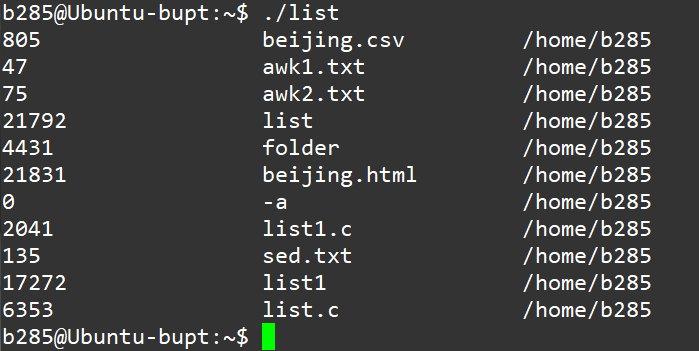
1. 除此之外，还有get\_file(char file[]), string\_to\_number(char str[])等函数用于其他功能。

## 用例测试

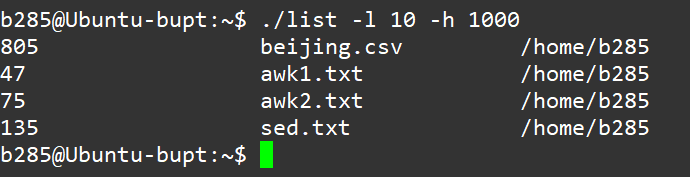
首先编译list.c文件:



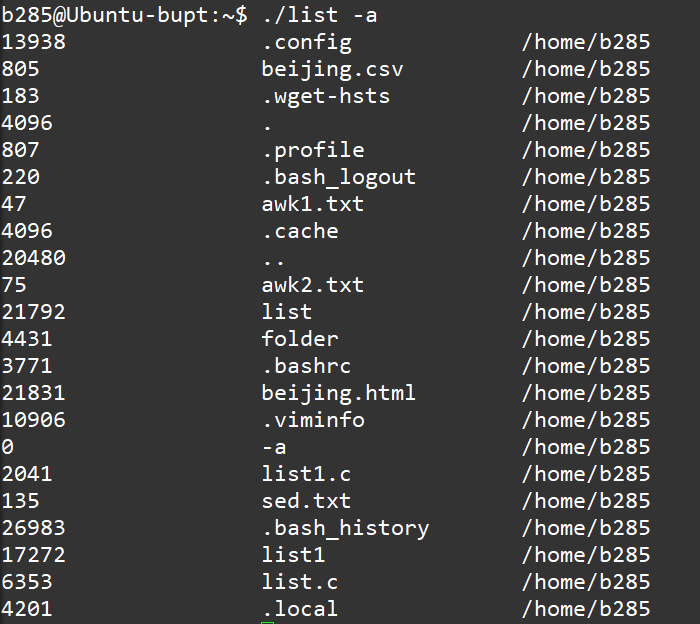
1. ./list:列出当前工作目录下全部文件。



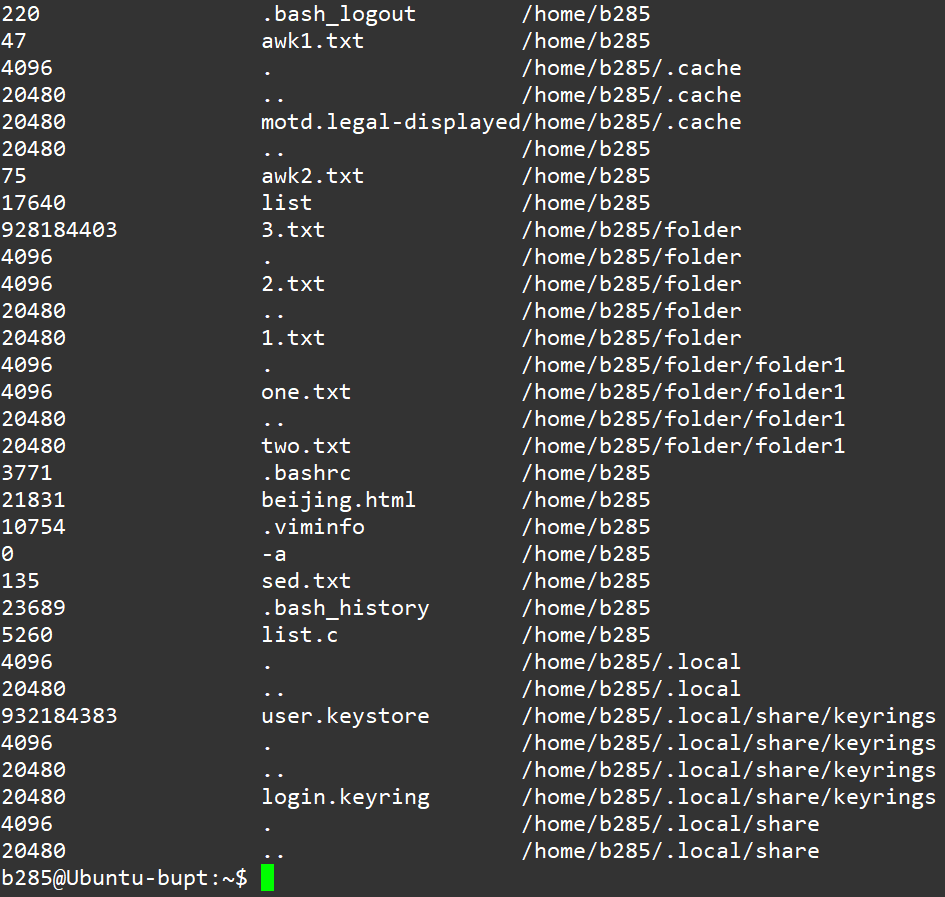
1. ./list -l 10 -h 1000：列出目录下大小在10-100字节的文件。



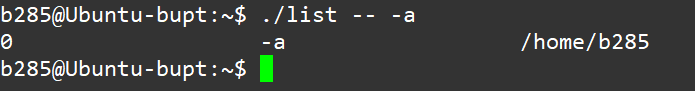
1. ./list -a:列出隐藏文件。



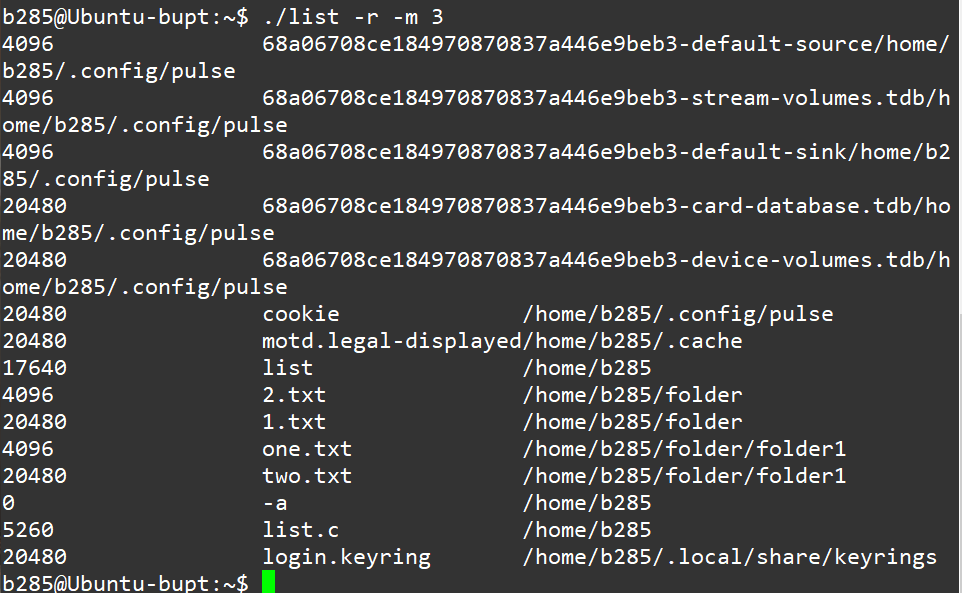
1. ./list -r -a:递归列出全部文件。



1. ./list -- -a：显示终止命令分析



1. ./list -r -m 3：修改时间必须在三天内



## 实验总结

本次实验通过对一条ls命令的简单实现，让我体会到了程序员与使用者是如何进行信息交换的，每一条看似简单的命令背后都有大量代码作为支撑，从而让我们方便快捷使用计算机。其中我认识到了dirent和stat结构体，两者相互配合可以达到遍历文件夹的目的，归根到底这两者背后也得有大量代码才能实现，其实我在这里既扮演程序员也扮演使用者的角色。

## 源代码

|  |
| --- |
| #include <stdio.h>  #include <dirent.h>  #include <sys/stat.h>  #include <unistd.h>  #include <string.h>  #include <time.h>  int r = 0;  int a = 0;  int l = -1;  int h = -1;  int m = -1;  long size = 0;  int string\_to\_number(char str[]){  int i = 0;  int sum = 0;  while(i < strlen(str)){  sum += str[i] - '0';  sum \*= 10;  i++;  }  return sum/10;  }  int is\_num(char str[]){  int i=0;  while(i < strlen(str)){  if(str[i] >= '0' && str[i] <= '9'){  i++;  }  else{  return 0;  }  }  return 1;  }  void folder\_size(char path[]){  DIR \*dp;  struct dirent \*file;  dp = opendir(path);  if(!dp){  printf("%s 2open dir error\n", path);  }  else{  while((file = readdir(dp)) != NULL){  char old[1024];  strcpy(old, path);  strcat(path, "/");  strcat(path, file->d\_name);  if(file->d\_type != 4){  struct stat statbuf;  stat(path, &statbuf);  size += statbuf.st\_size;  }  else if(strcmp(file->d\_name, ".") != 0 && strcmp(file->d\_name, "..") != 0){  folder\_size(path);  }  strcpy(path, old);  }  }  closedir(dp);  }  void limit(struct dirent \*file, char path[]){  char old[1024];  struct stat statbuf;  stat(file->d\_name, &statbuf);  //不显示隐藏文件  if(a == 0){  if(file->d\_name[0] == '.'){  return;  }  }  //限制文件最小大小  if(l != -1){  if(statbuf.st\_size < l){  return;  }  }  //限制文件最大大小  if(h != -1){  if(statbuf.st\_size > h){  return;  }  }  //限制文件最近修改时间  if(m != -1){  time\_t now = time(NULL);  int interval = (now - statbuf.st\_mtime) / 86400;  if(interval > m){  return;  }  }  if(file->d\_type == 4 && strcmp(file->d\_name, ".") != 0 && strcmp(file->d\_name, "..") != 0){  size = 0;  strcpy(old, path);  strcat(path, "/");  strcat(path, file->d\_name);  folder\_size(path);  strcpy(path, old);  printf("%-20ld%-20s%s\n", size + statbuf.st\_size, file->d\_name, path);  }  else  printf("%-20ld%-20s%s\n", statbuf.st\_size, file->d\_name, path);  }  void get\_all\_files\_under\_path(char path[]){  DIR \*dp;  struct dirent \*file;  dp = opendir(path);  if(!dp){  printf("open dir error");  }  else{  while((file = readdir(dp)) != NULL){  /\*普通文件\*/  if(file->d\_type != 4){  limit(file, path);  }  /\*文件夹\*/  else{  if(r == 1 && strcmp(file->d\_name, ".") != 0 && strcmp(file->d\_name, "..") != 0){  char old\_path[1024];  strcpy(old\_path, path);  strcat(path, "/");  strcat(path, file->d\_name);  get\_all\_files\_under\_path(path);  strcpy(path, old\_path);  }  else{  limit(file, path);  }  }  }  closedir(dp);  }  }  void get\_file(char file[]){  char pwd[1024];  if(!getcwd(pwd, 1024)){  return;  }  struct stat statbuf;  if(stat(file, &statbuf) == -1){  return;  }  //不显示隐藏文件  if(a == 0){  if(file[0] == '.'){  return;  }  }  //限制文件最小大小  if(l != -1){  if(statbuf.st\_size < l){  return;  }  }  //限制文件最大大小  if(h != -1){  if(statbuf.st\_size > h){  return;  }  }  //限制文件最近修改时间  if(m != -1){  time\_t now = time(NULL);  int interval = (now - statbuf.st\_mtime) / 86400;  if(interval > m){  return;  }  }  printf("%-20ld%-20s%s\n", statbuf.st\_size, file, pwd);  }  void analyze(char \*argv[], int \*flag, int \*i){  \*i = \*i + 1;  if(is\_num(argv[\*i]) == 1){  \*flag = string\_to\_number(argv[\*i]);  }  else{  \*i = \*i - 1;  }  }  int main(int argc, char \*argv[]){  char path[1024][1024];  char file[128][128];  memset(path, '\0', sizeof(path));  memset(file, '\0', sizeof(file));  int pathindex = 0, fileindex = 0;  //0个参数  if(argc == 1){  if(!getcwd(path[0], 1024)){  return 0;  }  get\_all\_files\_under\_path(path[0]);  }  //多个参数  else{  int i = 1;  while(i < argc){  //指令  if(argv[i][0] == '-'){  if(argv[i][1] == 'r'){  r = 1;  }  else if(argv[i][1] == 'a'){  a = 1;  }  else if(argv[i][1] == 'l' && i < argc - 1){  analyze(argv, &l, &i);  }  else if(argv[i][1] == 'h' && i < argc - 1){  analyze(argv, &h, &i);  }  else if(argv[i][1] == 'm' && i < argc - 1){  analyze(argv, &m, &i);  }  else if(argv[i][1] == '-'){  i++;  strcpy(file[fileindex], argv[i]);  fileindex++;  }  }  //路径  else if(argv[i][0] == '/') {  strcpy(path[pathindex], argv[i]);  pathindex++;  }  //文件  else{  strcpy(file[fileindex], argv[i]);  fileindex++;  }  i++;  }  //输出结果  if((path[0][0] == '\0' && file[0][0] == '\0') || file[0][0] == '\*'){  if(!getcwd(path[0], 1024)){  return 0;  }  get\_all\_files\_under\_path(path[0]);  }  else{  int i = 0;  while(path[i][0] != '\0'){  get\_all\_files\_under\_path(path[i]);  i++;  }  i = 0;  while(file[i][0] != '\0'){  get\_file(file[i]);  i++;  }  }  }  } |