程序设计及规格说明文档

需求说明

请使用程序设计语言(如C语言)，编程实现“ls”和“wc”命令，要求实现如下参数与功能。

ls –l(-d, -R, -a, -i)

wc [filename]

补充说明：一些细节可以不实现，例如参数解析时-al的连写、ls -l在遇到链接文件时对链接文件的打印等等。

设计说明

“ls”命令：

不输出具体文件属性时文件名的对齐说明：
1.
```
#define MAXROWLEN  80   //取命令行默认宽度80作为行宽
int g_leave_len = MAXROWLEN;    //一行剩余长度，用于输出对齐
int g_maxlen;   //存放某目录下最长的文件名的长度
```
2.输出排序：采用冒泡排序对输出的文件名按字典顺序进行排序
```
char* temp;
for(int i=0;i<count-1;i++)
{
  for(int j=0;j<count-1-i;j++)
  {
    if(strcmp(fileNames[j],fileNames[j+1])>0)
    {
      temp=fileNames[j];
      fileNames[j]=fileNames[j+1];
      fileNames[j+1]=temp;
     }
  }
}
```
3.输出格式：inode号（占据8个输出位）、inode号与文件名间的空格、文件名（以最长文件名长度占位，保证对齐）、不同文件之间的分隔用空格，其中inode部分可选（左对齐方式，即多余空间向右补充空格）

代码如下：
```
if(_i){
  if(g_leave_len<g_maxlen+9+1)
  {
    printf("\n");
    g_leave_len=MAXROWLEN;
  }
}
else{
  if(g_leave_len<g_maxlen+1)
  {
```

```
    printf("\n");
    g_leave_len=MAXROWLEN;
  }
}
if(_i){
  printf("%*ld ",-8,inodeNum);
  g_leave_len-=9;
}
printf("%*s ",-g_maxlen,name);
g_leave_len=g_leave_len-(g_maxlen+1);
```

函数介绍：

```
void display_single(long inodeNum,char* name) ;
//输出单个文件的文件名（inodeNum在指定-i时输出）
void display_attribute(struct stat buf,char* name);
//以longtext格式对单个文件进行输出（指定-l时的输出方式）
void normal_display(char* path);
//确定指定目录下最长的文件名长度，并采用display_single对目录下文件进行输出
void longtext_display(char* path);
//确定指定目录下最长的文件名长度，同时计算总用量并输出，最后采用display_attribute输出目录下文件
void recursive(char* path);
//递归输出目录及其子目录下所有的文件信息，采用深度遍历方式进行递归，首先输出指定目录下文件信息，然后循环目录项，对其下的子目录递归调用recursive
int main(int argc,char *argv[]);
//解析命令参数并调用相应的函数对命令进行处理
```

实现设计：

1.考虑到输入命令可能含有多个选项，故在参数解析时通过全局的bool布尔变量_l、_d、_R、_a及_i保存对应的选项-l(-d, -R, -a, -i)，而遇到非法选项则输出错误信息printf("Unknown argument type:%s\n",argv[index]);

2.区别于以-开头的选项，遇到路径参数时开始命令路径参数的保存：

```
int pathNum=argc-index;
char* paths[pathNum];
for(int i=0;index<argc;index++,i++){
  paths[i]=argv[index];
}
//若命令不指定路径，则以当前路径"."作为命令执行路径
```

3.根据_d分为两种命令处理模式：

命令输入含有-d选项的话，当遇到目录时列出目录本身而非目录内的文件，直接调用函数display_single / display_attribute进行处理，而输入不含有-d选项时，则解析指定目录下文件，调用normal_display / longtext_display进行处理；

4.关于-a选项的处理在normal_display / longtext_display函数中进行：

```
char* fileNames[80];
int count=0;
while((ptr =readdir(dir)) != NULL)
{
  if(!_a&&ptr->d_name[0]=='.'){
    continue;
  }
  fileNames[count]=ptr->d_name;
  count++;
}
```

5.关于-i选项的处理在display_single / display_attribute函数中进行:

```
if(_i){
  printf("%*ld ",-8,inodeNum);
  g_leave_len-=9;
}//display_single
if(_i) printf("%*ld ",-8,buf.st_ino);//display_attribute
```

6.关于-R选项的处理，直接调用recursive函数(recursive函数内部包含对-l参数的解析处理)

```
if(_R){
  if(pathNum==0){
  recursive(".");
}
else{
  for(int i=0;i<pathNum;i++){
    recursive(paths[i]);
  }
}
```

与命令源代码不同之处:

1.选项参数的获取/解析:

我是以选项参数起始的-为特征，直接从main函数的命令行参数获取-l(-d, -R, -a, -i);

而源代码则是getopt_long函数来获取选项参数:

```
int c = getopt_long (argc, argv,
                           "abcdfghiklmnopqrstuvw:xABCDFGHI:LNQRST:UXZ1",
                           long_options, &oi);
```

解析部分的话，都是使用switch case结构，不过由于实现参数较少，程序较为简单，我直接使用bool变量_l、_d、_R、_a及_i进行保存，而源代码除了ignore_mode、immediate_dirs、print_inode、recursive等相应bool变量，还使用enum枚举保存解析结构（如format = long_format）：

```
enum format
{
  long_format,      /* -l and other options that imply -l */
  one_per_line,     /* -1 */
  many_per_line,        /* -C */
  horizontal,           /* -x */
  with_commas           /* -m */
};
```

2.

由于功能的复杂性，源代码使用struct fileinfo保存文件信息以便进行后续处理，而我则是直接通过系统调用获取指定属性并输出，并未使用相应结构体;

3.

我直接将文件名是否以'.'开头作为隐藏标准，内置-a选项的ignore_pattern;

而源代码使用相应结构体指定文件输出时的ignore_pattern,

```
struct ignore_pattern
{
  const char *pattern;
  struct ignore_pattern *next;
};
```

4.

我通过MAXROWLEN宏及int g_maxlen记录目录下最长文件名长度，并以此为基础进行输出文件名的对齐处理;

而源代码则有着相应的结构体实现输出的对齐:

```
struct column_info
{
  bool valid_len;
  size_t line_len;
```

```
  size_t *col_arr;
};
```

5.在输出时，我统一对文件名进行基于字典序的冒泡排序，而源代码则有着各种类型的排序（而且默认的也和我不一样。。。），对应结构体如下：

```
enum sort_type
{
  sort_none = -1,       /* -U */
  sort_name,            /* default */
  sort_extension,       /* -X */
  sort_size,            /* -S */
  sort_version,     /* -v */
  sort_time,            /* -t */
  sort_numtypes     /* the number of elements of this enum */
};
```

6.源代码和我的实现均使用递归实现-R选项功能，不过源码将子目录的解析亦封装成了函数

```
if (recursive)
    extract_dirs_from_files (name, false);
```

7.相比源代码，我没有针对文件类型实现不同颜色的输出；

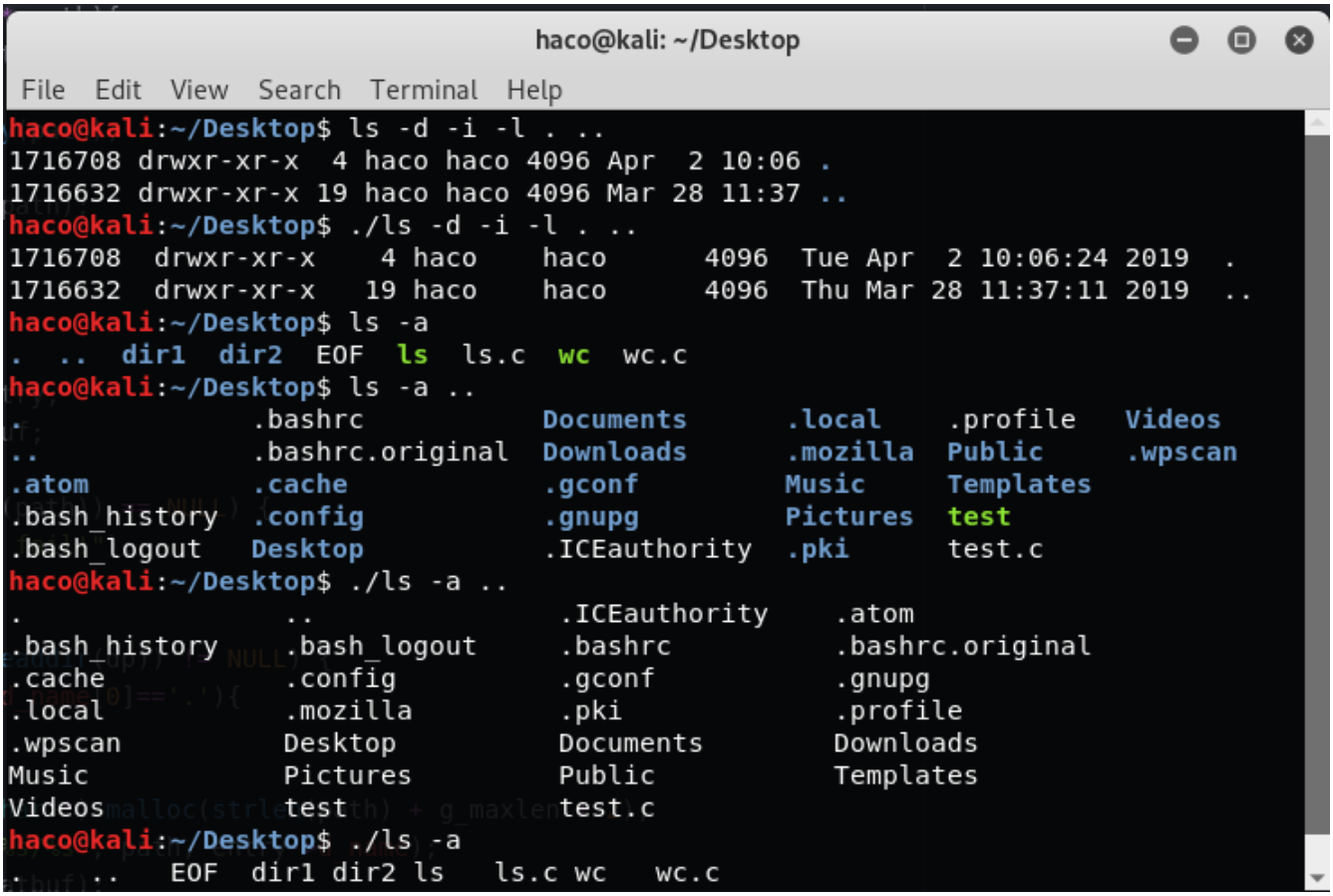
源代码中颜色相关的结构体：

```
static struct bin_str color_indicator[] =
  {
    { LEN_STR_PAIR ("\033[") },     /* lc: Left of color sequence */
    { LEN_STR_PAIR ("m") },     /* rc: Right of color sequence */
    { 0, NULL },            /* ec: End color (replaces lc+rs+rc) */
    { LEN_STR_PAIR ("0") },     /* rs: Reset to ordinary colors */
    { 0, NULL },            /* no: Normal */
    { 0, NULL },            /* fi: File: default */
    { LEN_STR_PAIR ("01;34") },     /* di: Directory: bright blue */
    { LEN_STR_PAIR ("01;36") },     /* ln: Symlink: bright cyan */
    { LEN_STR_PAIR ("33") },        /* pi: Pipe: yellow/brown */
    { LEN_STR_PAIR ("01;35") },     /* so: Socket: bright magenta */
    { LEN_STR_PAIR ("01;33") },     /* bd: Block device: bright yellow */
    { LEN_STR_PAIR ("01;33") },     /* cd: Char device: bright yellow */
    { 0, NULL },            /* mi: Missing file: undefined */
    { 0, NULL },            /* or: Orphaned symlink: undefined */
    { LEN_STR_PAIR ("01;32") },     /* ex: Executable: bright green */
    { LEN_STR_PAIR ("01;35") },     /* do: Door: bright magenta */
    { LEN_STR_PAIR ("37;41") },     /* su: setuid: white on red */
    { LEN_STR_PAIR ("30;43") },     /* sg: setgid: black on yellow */
    { LEN_STR_PAIR ("37;44") },     /* st: sticky: black on blue */
    { LEN_STR_PAIR ("34;42") },     /* ow: other-writable: blue on green */
    { LEN_STR_PAIR ("30;42") },     /* tw: ow w/ sticky: black on green */
    { LEN_STR_PAIR ("30;41") },     /* ca: black on red */
    { 0, NULL },            /* mh: disabled by default */
    { LEN_STR_PAIR ("\033[K") },    /* cl: clear to end of line */
  };
```

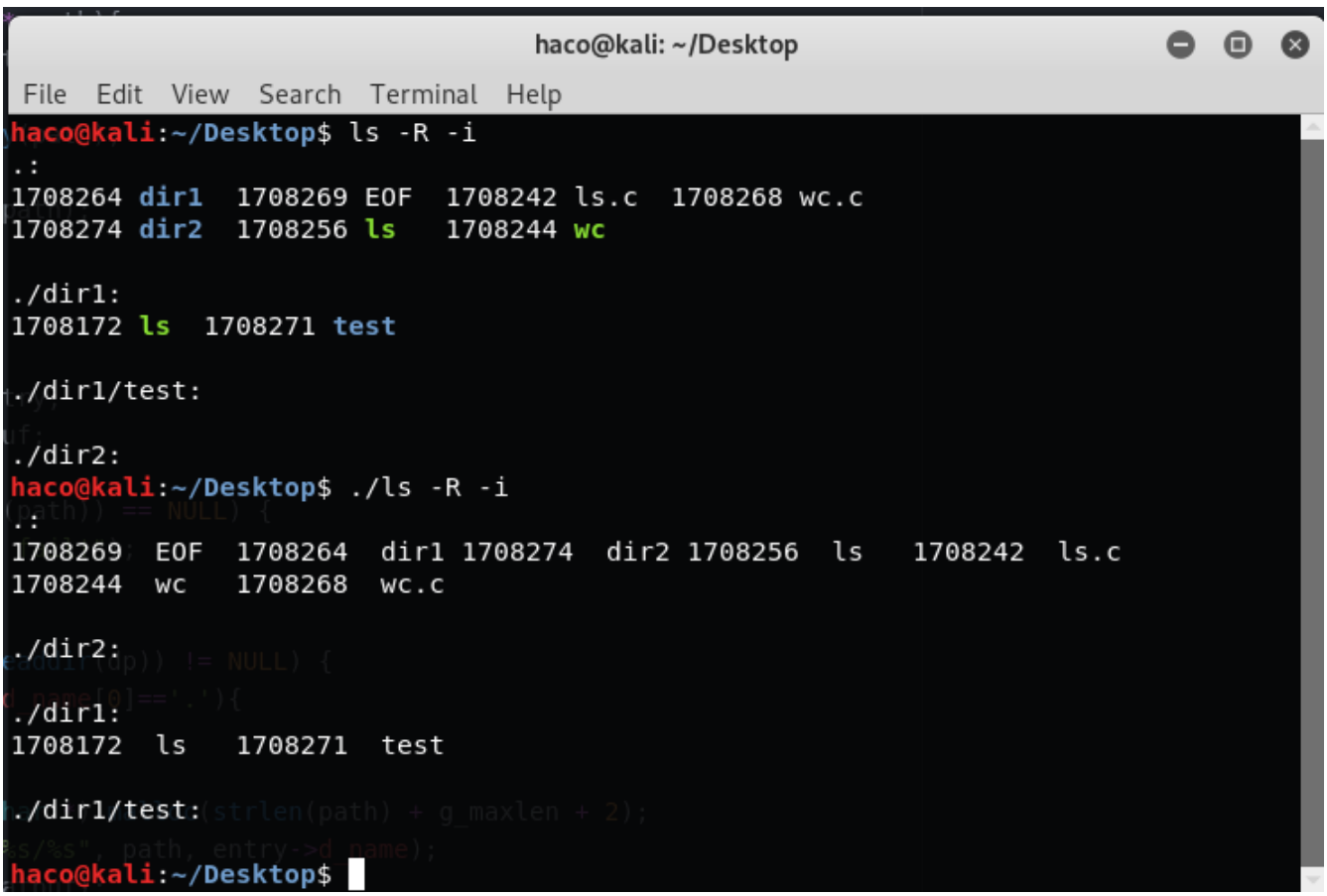
规格说明：

“ls”命令：（支持–l,-d, -R, -a, -i及其组合，不支持类似-al的连写、ls -l在遇到链接文件时对链接文件的打印）

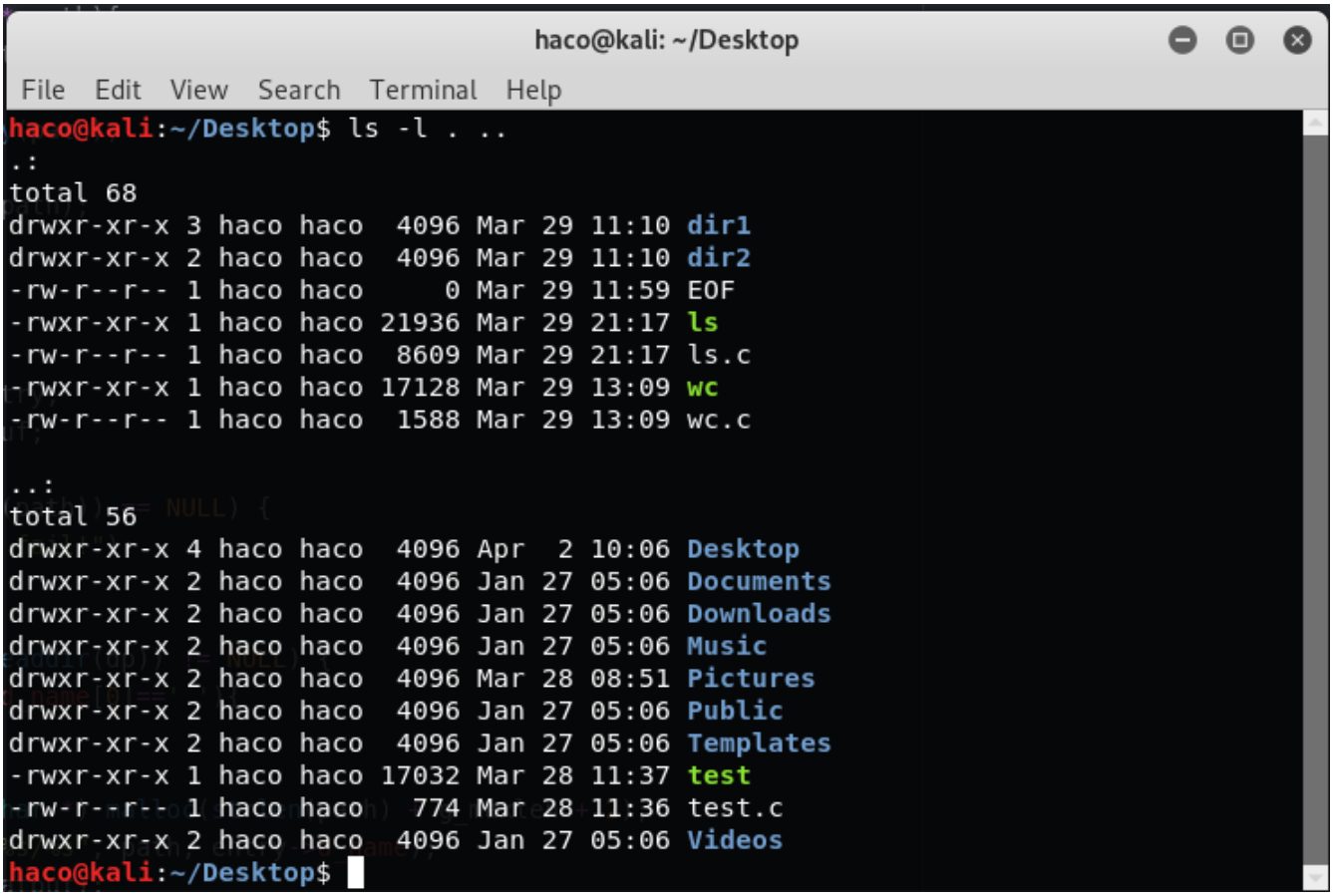
ls -d 及 -a 对比：

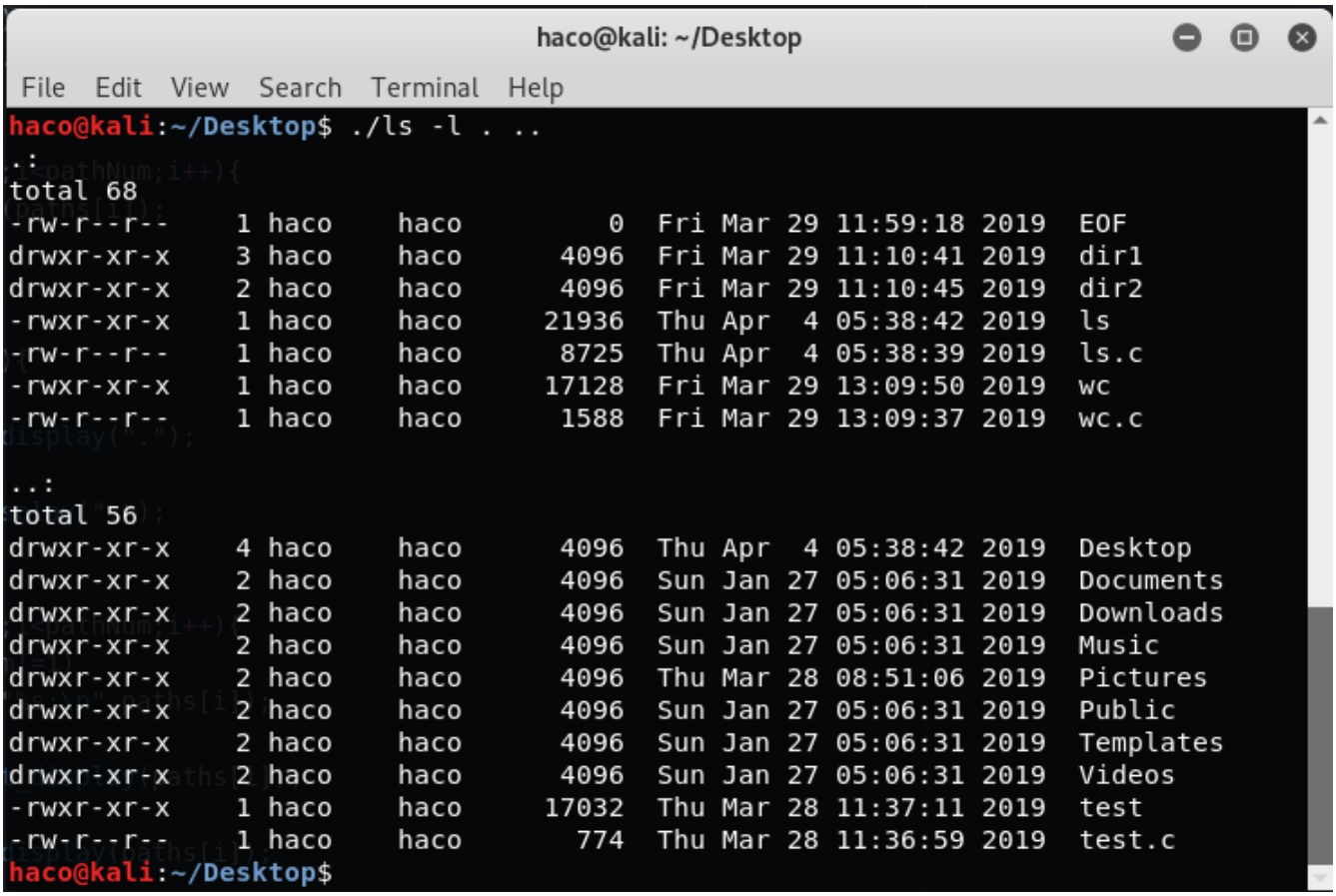


ls -R -i 对比：

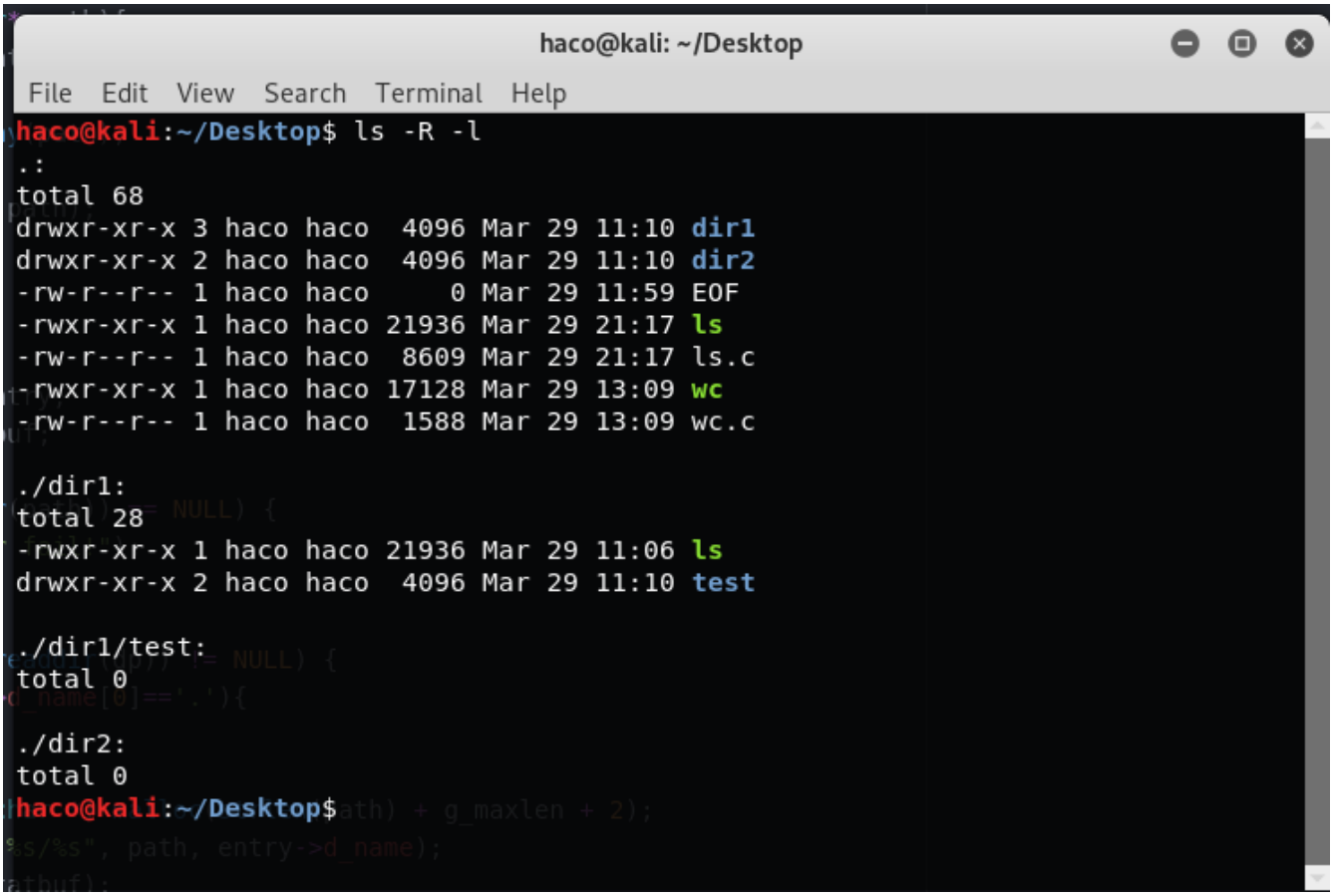


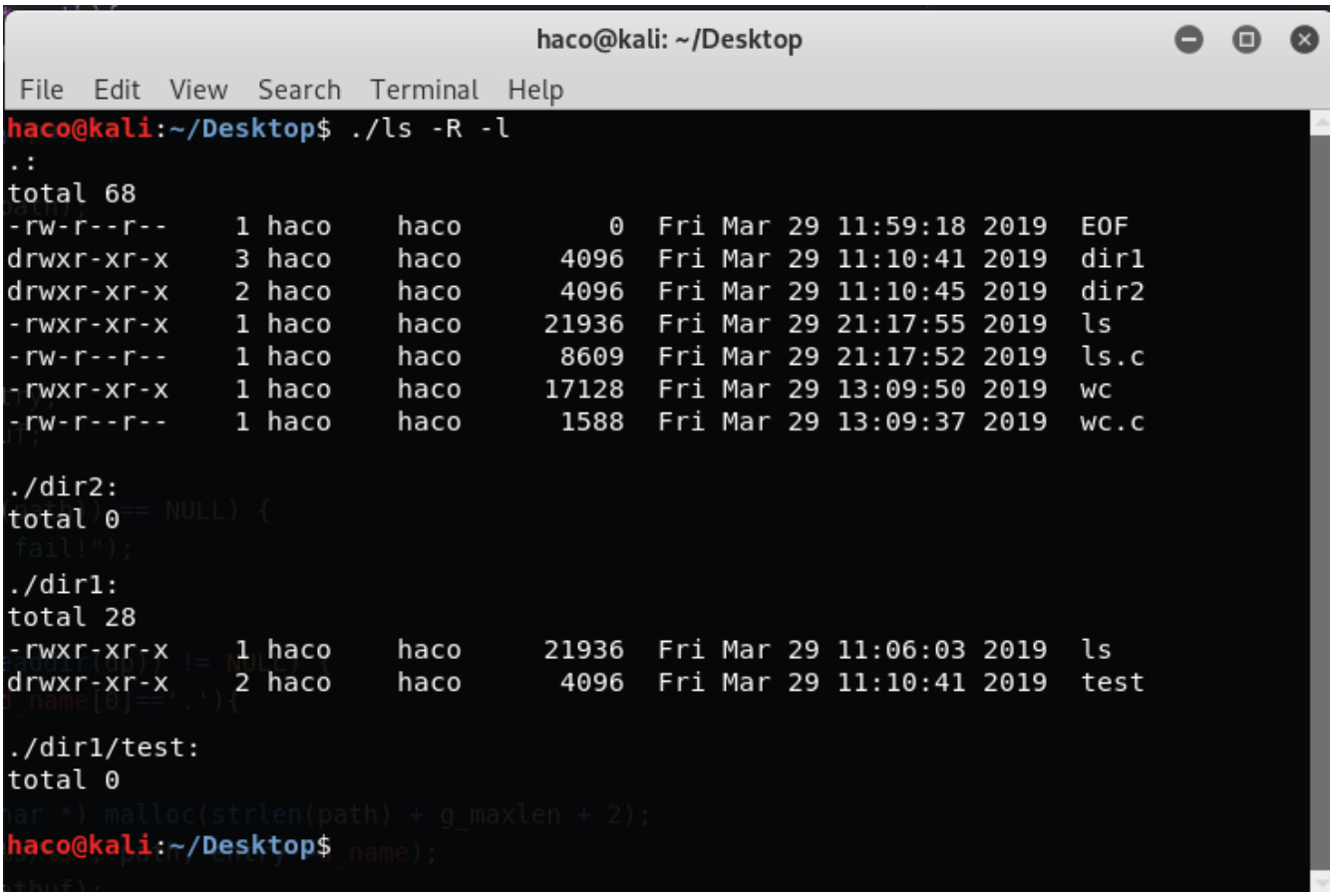
ls -l 对比：


haco@kali: ~/Desktop
File Edit View Search Terminal Help
haco@kali:~/Desktop$ ls -l . ..
.:
total 68
drwxr-xr-x 3 haco haco 4096 Mar 29 11:10 dir1
drwxr-xr-x 2 haco haco 4096 Mar 29 11:10 dir2
-rw-r--r-- 1 haco haco   0 Mar 29 11:59 EOF
-rwxr-xr-x 1 haco haco 21936 Mar 29 21:17 ls
-rw-r--r-- 1 haco haco 8609 Mar 29 21:17 ls.c
-rwxr-xr-x 1 haco haco 17128 Mar 29 13:09 wc
-rw-r--r-- 1 haco haco 1588 Mar 29 13:09 wc.c
..:
total 56
drwxr-xr-x 4 haco haco 4096 Apr 2 10:06 Desktop
drwxr-xr-x 2 haco haco 4096 Jan 27 05:06 Documents
drwxr-xr-x 2 haco haco 4096 Jan 27 05:06 Downloads
drwxr-xr-x 2 haco haco 4096 Jan 27 05:06 Music
drwxr-xr-x 2 haco haco 4096 Mar 28 08:51 Pictures
drwxr-xr-x 2 haco haco 4096 Jan 27 05:06 Public
drwxr-xr-x 2 haco haco 4096 Jan 27 05:06 Templates
-rwxr-xr-x 1 haco haco 17032 Mar 28 11:37 test
-rw-r--r-- 1 haco haco   774 Mar 28 11:36 test.c
drwxr-xr-x 2 haco haco 4096 Jan 27 05:06 Videos
haco@kali:~/Desktop$


haco@kali: ~/Desktop
File Edit View Search Terminal Help
haco@kali:~/Desktop$ ./ls -l . ..
.:
total 68
-rw-r--r-- 1 haco haco   0 Fri Mar 29 11:59:18 2019 EOF
drwxr-xr-x 3 haco haco 4096 Fri Mar 29 11:10:41 2019 dir1
drwxr-xr-x 2 haco haco 4096 Fri Mar 29 11:10:45 2019 dir2
-rwxr-xr-x 1 haco haco 21936 Thu Apr 4 05:38:42 2019 ls
-rw-r--r-- 1 haco haco 8725 Thu Apr 4 05:38:39 2019 ls.c
-rwxr-xr-x 1 haco haco 17128 Fri Mar 29 13:09:50 2019 wc
-rw-r--r-- 1 haco haco 1588 Fri Mar 29 13:09:37 2019 wc.c
..:
total 56
drwxr-xr-x 4 haco haco 4096 Thu Apr 4 05:38:42 2019 Desktop
drwxr-xr-x 2 haco haco 4096 Sun Jan 27 05:06:31 2019 Documents
drwxr-xr-x 2 haco haco 4096 Sun Jan 27 05:06:31 2019 Downloads
drwxr-xr-x 2 haco haco 4096 Sun Jan 27 05:06:31 2019 Music
drwxr-xr-x 2 haco haco 4096 Thu Mar 28 08:51:06 2019 Pictures
drwxr-xr-x 2 haco haco 4096 Sun Jan 27 05:06:31 2019 Public
drwxr-xr-x 2 haco haco 4096 Sun Jan 27 05:06:31 2019 Templates
drwxr-xr-x 2 haco haco 4096 Sun Jan 27 05:06:31 2019 Videos
-rwxr-xr-x 1 haco haco 17032 Thu Mar 28 11:37:11 2019 test
-rw-r--r-- 1 haco haco   774 Thu Mar 28 11:36:59 2019 test.c
haco@kali:~/Desktop$

ls -R -l 对比：


haco@kali: ~/Desktop
File Edit View Search Terminal Help
haco@kali:~/Desktop$ ls -R -l
.:
total 68
drwxr-xr-x 3 haco haco 4096 Mar 29 11:10 dir1
drwxr-xr-x 2 haco haco 4096 Mar 29 11:10 dir2
-rw-r--r-- 1 haco haco   0 Mar 29 11:59 EOF
-rwxr-xr-x 1 haco haco 21936 Mar 29 21:17 ls
-rw-r--r-- 1 haco haco 8609 Mar 29 21:17 ls.c
-rwxr-xr-x 1 haco haco 17128 Mar 29 13:09 wc
-rw-r--r-- 1 haco haco 1588 Mar 29 13:09 wc.c
./dir1:
total 28
-rwxr-xr-x 1 haco haco 21936 Mar 29 11:06 ls
drwxr-xr-x 2 haco haco 4096 Mar 29 11:10 test
./dir1/test:
total 0
./dir2:
total 0
haco@kali:~/Desktop$


haco@kali: ~/Desktop
File Edit View Search Terminal Help
haco@kali:~/Desktop$ ./ls -R -l
.:
total 68
-rw-r--r-- 1 haco haco   0 Fri Mar 29 11:59:18 2019 EOF
drwxr-xr-x 3 haco haco 4096 Fri Mar 29 11:10:41 2019 dir1
drwxr-xr-x 2 haco haco 4096 Fri Mar 29 11:10:45 2019 dir2
-rwxr-xr-x 1 haco haco 21936 Fri Mar 29 21:17:55 2019 ls
-rw-r--r-- 1 haco haco 8609 Fri Mar 29 21:17:52 2019 ls.c
-rwxr-xr-x 1 haco haco 17128 Fri Mar 29 13:09:50 2019 wc
-rw-r--r-- 1 haco haco 1588 Fri Mar 29 13:09:37 2019 wc.c
./dir2:
total 0
./dir1:
total 28
-rwxr-xr-x 1 haco haco 21936 Fri Mar 29 11:06:03 2019 ls
drwxr-xr-x 2 haco haco 4096 Fri Mar 29 11:10:41 2019 test
./dir1/test:
total 0
haco@kali:~/Desktop$

设计说明

“wc”命令：

函数介绍：

```
int lineCounter(char* fName); //行计数
int wordCounter(char* fName); //单词计数
int charCounter(char* fName); //字符计数
int main(int argc,char *argv[]); //参数解析&&流程控制
```

实现设计：

1.确保命令完整性（含有filename参数）

```
if(argc==1){
  perror("MISSING FILENAME");
  return -1;
}
```

2.参数解析及保存：

```
int fileNum=argc-1;
char* files[fileNum];
for(int i=0;i<fileNum;i++){
  files[i]=argv[i+1];
}
```

3.分别调用lineCounter、wordCounter、charCounter对文件行数、单词数、字符数进行统计，若文件参数有多个，则同时统计并输出总行数、单词数、字符数。代码如下：

```
int total_line=0;
int total_word=0;
int total_char=0;

for(int i=0;i<fileNum;i++){
  int lCount=lineCounter(files[i]);
  int wCount=wordCounter(files[i]);
  int cCount=charCounter(files[i]);
  printf("%d\t%d\t%d\t%s\n",lCount,wCount,cCount,files[i]);
  total_line+=lCount;
  total_word+=wCount;
  total_char+=cCount;
}

if(fileNum>1){
  printf("%d\t%d\t%d\ttotal\n",total_line,total_word,total_char);
}
```

与命令源代码不同之处：

1.由于作业只要求wc [filename]，源代码与我的实现相比，多了对选项参数的解析，如下所示：

```
    switch (optc)
      {
      case 'c':
        print_bytes = true;
        break;

      case 'm':
        print_chars = true;
        break;

      case 'l':
        print_lines = true;
```

```
        break;

      case 'w':
        print_words = true;
        break;

      case 'L':
        print_linelength = true;
        break;

      case FILES0_FROM_OPTION:
        files_from = optarg;
        break;

      case_GETOPT_HELP_CHAR;

      case_GETOPT_VERSION_CHAR (PROGRAM_NAME, AUTHORS);

      default:
        usage (EXIT_FAILURE);
      }
```

2.我的实现并没有使用缓冲机制，通过单个字符的不断读取进行统计，而源代码的读取则有着相应的缓冲区；

```
#define BUFFER_SIZE (16 * 1024)
(bytes_read = safe_read (fd, buf + prev, BUFFER_SIZE - prev)) > 0
```

3.源代码的行统计与我的实现一样，均通过统计'\n'个数实现

```
lines += *p++ == '\n';
```

4.我的实现与源代码的单词统计采用相同的单词分隔字符（'\n'、'\r'、'\f'、'\t'、' '、'\v'），判断代码如下

```
if(ch1!=' '&&ch1!='\t'&&ch1!='\v'&&ch1!='\r'&&ch1!='\n'&&ch1!='\f'&&ch1!=EOF&&
           (ch2==' '||ch2=='\t'||ch2=='\v'||ch2=='\r'||ch2=='\n'||ch2=='\f'))
        {
            wCount++;
        }
```

```
//ch1为当前字符，ch2为前一个字符；
```

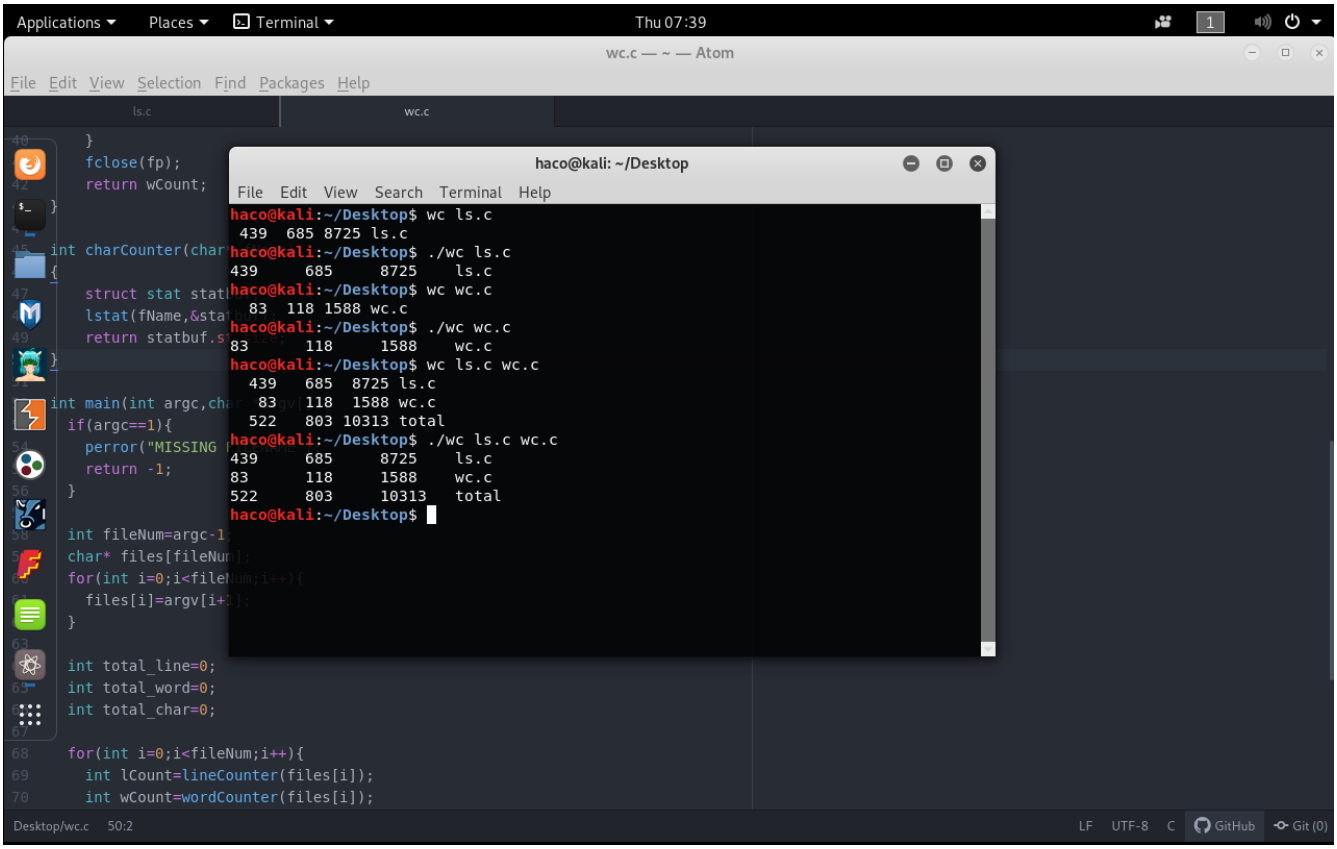
而源代码则有着bool变量in_word/in_shift来记录当前读取是在单词中还是处于单词间隔；

5.就字符个数的统计方面，源代码是对读取的字符进行累加（char++），而我则直接通过文件状态结构体stat获取文件字符个数st_size；

规格说明：

“wc”命令（统计行数、单词数、字符数，支持单文件/多文件）：

使用截图（wc [filename]）：


Applications
Places
Terminal
Thu 07:39
wc.c — ~ — Atom
File Edit View Selection Find Packages Help
ls.c
wc.c
haco@kali: ~/Desktop
File Edit View Search Terminal Help
haco@kali:~/Desktop$ wc ls.c
439 685 8725 ls.c
haco@kali:~/Desktop$ ./wc ls.c
439 685 8725 ls.c
haco@kali:~/Desktop$ wc wc.c
83 118 1588 wc.c
haco@kali:~/Desktop$ ./wc wc.c
83 118 1588 wc.c
haco@kali:~/Desktop$ wc ls.c wc.c
439 685 8725 ls.c
83 118 1588 wc.c
522 803 10313 total
haco@kali:~/Desktop$ ./wc ls.c wc.c
439 685 8725 ls.c
83 118 1588 wc.c
522 803 10313 total
haco@kali:~/Desktop$
Desktop/wc.c 50:2
LF UTF-8 C GitHub Git (0)