Project 0: Linux Warm Up

项目简介

这个项目包含两个部分，第一部分主要有 `shell` 脚本完成一些基本的操作，而第二部分由 `linux` 下 `C` 语言编程完成一些基本的操作。在 `bonus` 部分，使用一些高级的文本处理工具完成人物。

学习 `linux shell` 非常简单，网上有大量博客讲解。我个人比较喜欢的是使用 `man` 指令和查阅 `gnu` 网站上的非常详细的文档 http://www.gnu.org/software/coreutils/manual/coreutils

0a

Part 1

- `./s1.sh`

1. 使用 `mkdir` 指令创建文件夹 `foo` 在当前文件夹
2. 使用 `touch` 指令创建新的文件
3. 使用 `echo` 指令和重定向将特定内容写入文件

mkdir

- 用法

```
mkdir [OPTION]... DIRECTORY...
```

创建目录，这里使用的选项为

```
-p, --parents
            no error if existing, make parent directories as needed
```

仅当未存在目录时创建，且自动创建相应的父级目录

touch

- 用法

```
touch [OPTION]... FILE...
```

`touch` 本来是改变文件的时间戳，意思就是更新文件的时间信息。但是如果不存在，则会新创建这个文件。

```
A FILE argument that does not exist is created empty, unless -c or
-h is supplied.
```

echo

- 用法

```
echo [SHORT-OPTION]... [STRING]...
echo LONG-OPTION
```

`echo` 指令用于输出一串文字。我们需要这些文字被写入文件，于是使用输出重定向，使用方法为 `> [file name]`

cp

- 用法

```
cp [OPTION]... [-T] SOURCE DEST
cp [OPTION]... SOURCE... DIRECTORY
cp [OPTION]... -t DIRECTORY SOURCE...
```

这里我们使用第二种用法，选项为 `-f`，即强制替换已有文件。

```
  -f, --force
         if an existing destination file cannot be opened, remove it
         and try again (this option is ignored when the -n option is
         also used)
```

- `./s2.sh`

大致思路

1. 使用 `find` 查找 `b` 开头文件
2. 使用 `ls -l` 列出完整信息
3. 使用 `awk` 输出特定的列
4. 使用 `pipeline` 组合这些程序的输入输出

5. 使用 `sed` 对调整格式

rm

- 用法

```
rm [OPTION]... [FILE]...
```

删除文件，这里使用 `-f` 选项。这里目的是删除之前已有的文件，所以使用该选项，如果没有要删除的文件也忽略警告。

```
-f, --force
            ignore nonexistent files and arguments, never prompt
```

find

- 用法

```
 find  [-H]  [-L] [-P] [-D debugopts] [-Olevel] [starting-point...]
    [expression]
```

使用的第一个选项为 `-iname`，用来匹配 `b` 字母文件名开头的文件

```
-iname pattern
            Like  -name,  but the match is case insensitive.  For exam-
            ple, the patterns `fo*' and  `F??'  match  the  file  names
            `Foo', `FOO', `foo', `fOo', etc.   The pattern `*foo*` will
            also match a file called '.foobar'.
```

awk

`awk` 可以以分隔符为界，逐步处理部分文本，默认为行，即逐行处理。

`awk` 可以使用 `%[0..9]` 指定选取哪些列，列由列分隔符隔开，默认为空格。 `%0` 为整行，而其他为指定列

这里我们需要打印第 1, 3, 9 列，并且没有其他条件，于是使用最基本的用法，最后输出到 `./output` 文件

```
awk '{print $9" "$3" "$1}' >> output
```

| (管道) 和 xargs

管道用来把一个指令的输出加入到另一个指令的标准输入，而 `xargs` 还可以将输入改为下一个命令的参数。我们需要将 `find` 指令找到的文件路径当作参数输入到 `ls` 指令中，而根据 `awk` 的用法， `ls` 输出到标准输出的内容，则需要成为 `awk` 的标准输入内容，故 `find` 和 `ls` 中间使用 `xargs` 而 `ls` 和 `awk` 之间使用 `|` 。

chmod

- 用法

```
chmod [OPTION]... MODE[,MODE]... FILE...
chmod [OPTION]... OCTAL-MODE FILE...
chmod [OPTION]... --reference=RFILE FILE...
```

linux 使用字母缩写来表示三个对文件操作的权限

- r 代表 read 读取
- w 代表 write 写入
- x 代表 execute 执行

又使用 `bitmask` 表示这三个字母，分别用 4,2,1 表示 r,w,x 那么所有的权限拥有情况都能用 0..7 表示。

linux 的文件分别有对三个对象的文件权限说明，分别是 `user` , `group` 和 `other` ，分别表示用户，用户所在组的其他用户，和其他组的用户。

根据题目要求，只有本用户有读取权限，所以应该为 `rwx-wx-wx` ，即指令为 `chmod 644 [file]`

Part 2

对 `set_operation.c` 进行调试

编译选项

我们可以通过如下方法获取 `gcc` 的使用帮助

```
$ gcc --help
```

- `-o`

一般来说，直接运行程序的话，我们直接运行 `$ gcc [sourcefile] -o [outputfile]` ，然后运行 `$ ./outputfile [arg]` 即可。若没有 `-o` 选项，则输出文件为 `a.out`

- `-O`

之后比较常用的选项是 `-O` ，表示为 `Opitmization` 进行编译优化。很多情况开 `-O2` 编译优化可以让程序的运行速度快上四倍左右

- `-g`

若要进行 `gdb` 调试，则需要加上该选项。不过一般不和 `-O` 选项一起用。自己以前这么用到的时候，会发现程序会自动过滤一些可以优化掉的代码部分，比如空循环。

- `-Wall`

输出额外的编译警告，如 `scanf` 用错格式符，或者是函数没加返回值，变量没有初始化等。

- `-std`

在老的 `gcc` 编译器上，C语言用的是老的语法标准，甚至没有 `for` 循环的语法。一般习惯使用 `-std=c11` 。这里使用的是 `gcc-7` ，故不会出现此问题。

大致过程

个人来说，我比较喜欢在使用 `gdb` 调试之前，先手动阅览一边代码，即肉眼查错。 `gdb` 虽然简单，在有错误数据的情况下一般一定能找出错误，但是万一某些错误在这个数据中和正确的写法会得到同样输出，这时就需要更多的数据，或者干脆来说，直接检查代码找出错误。（根据个人的经验上在 online judge 上练习题目，测试数据是看不到的，往往手动检查代码，比自己再去大规模生成随机数据并写对拍程序更可靠

bugs

1. `set_operation.c:26`

   ```
   if((p->->next)->number==num)
   ```

   从函数的作用可知，这段代码需要检查一个链表中是否包含特定的元素。 `while(p != NULL)` 保证了当前 `p` 不为 `NULL` ，结合后面 `p = p->next` 也是遍历整个链表的手段。所以循环体内检查的当前结点是 `p` 而不是 `p->next` 。

2. `set_operation.c:89`

   ```
   for(i=0;i<=A_size;i++){
   ```

   这个是常见的错误，循环一共执行了 `A_size+1` 次，而一共只有 `A_size` 个

3. `set_operation.c:107, 133`

```
if(!check(p1->number,B_head)){ //if this element is in B
if(!check(p1->number,A2_head)){
```

`check()` 在包含元素返回真，那么这里代码搞反了真假的条件

gdb

以下是使用 `gdb` 的方法调试程序。

首先第一个错误的地方可以通过编译程序的返回错误信息看到。

通过编译选项 `-g`，可以生成调试信息。

```
$ gcc set_operation.c -o set_operation -g
```

从 gdb documentatin 上我们可以看到 gdb 的 user manual

```
$ break location
```

其中 location 可以是行号，函数名或者是一条指令的地址。

```
$gdb set_operation
(gdb) run
```

使用 `gdb` 运行程序

```
Starting program: /root/ECNU-OSLab/lab1/0a/set
---------------------------------
----Computing (A-B)union(B-A)----
---------------------------------
----input the number of elements of A: 3
1-th element: 3
2-th element: 4
3-th element: 5
----input the number of elements of B: 2
1-th element: 1
2-th element: 4

Program received signal SIGSEGV, Segmentation fault.
0x0000000000400970 in main () at set_operation_original.c:91
91          p2->number=p3->number;
```

程序自动在 `Segmentation fault` 的地方断点，接着我们在这个地方设置 `display` 和 `break point`，这里一共显示 `i,p2,p2->number,p3,p3->number`

```
(gdb) break 91
Breakpoint 1 at 0x40096c: file set_operation_original.c, line 91.
Breakpoint 1, main () at set_operation_original.c:91
91          p2->number=p3->number;
(gdb) display i
1: i = 0
(gdb) display p2
2: p2 = (struct node *) 0x6038d0
(gdb) display p2->number
3: p2->number = 0
(gdb) display p3
4: p3 = (struct node *) 0x603830
(gdb) display p3->number
5: p3->number = 1
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) n
Program not restarted.
(gdb) info display
Auto-display expressions now in effect:
Num Enb Expression
1:   y  i
2:   y  p2
3:   y  p2->number
4:   y  p3
5:   y  p3->number
(gdb) continue
Continuing.

Breakpoint 1, main () at set_operation_original.c:91
91          p2->number=p3->number;
1: i = 1
2: p2 = (struct node *) 0x6038f0
3: p2->number = 0
4: p3 = (struct node *) 0x603850
5: p3->number = 2
(gdb) continue
Continuing.

Breakpoint 1, main () at set_operation_original.c:91
91          p2->number=p3->number;
1: i = 2
2: p2 = (struct node *) 0x603910
3: p2->number = 0
4: p3 = (struct node *) 0x603870
5: p3->number = 3
(gdb) continue
Continuing.

Breakpoint 1, main () at set_operation_original.c:91
91          p2->number=p3->number;
1: i = 3
2: p2 = (struct node *) 0x603930
3: p2->number = 0
```

```
4: p3 = (struct node *) 0x0
5: p3->number = <error: Cannot access memory at address 0x0>
(gdb)
```

使用 `continue` 继续到下一个断点， `display` 信息会在每次遇到断点的时候跳出，也可以使用 `print [expression]` 打印信息。

我们可以发现，在第四次到断点的时候 `p3` 变成了 `NULL` ，于是发现循环多执行了一次，遂发现 89 行的错误。

接着我们可以发现 (A-B)(A – B) 和 (B-A)(B – A) 的计算结果有误，我们分别在第 107 和 第 118 行设置断点，分别是两种情况的地方。

```
(gdb) b 107
Breakpoint 1 at 0x4009ea: file set_operation_original.c, line 107.
(gdb) b 118
Breakpoint 2 at 0x400a44: file set_operation_original.c, line 118.
(gdb) continue
The program is not being run.
(gdb) run
Starting program: /root/ECNU-OSLab/lab1/0a/set
---------------------------------
----Computing (A-B)union(B-A)----
---------------------------------
----input the number of elements of A: 3
1-th element: 1
2-th element: 2
3-th element: 3
----input the number of elements of B: 2
1-th element: 1
2-th element: 4

Breakpoint 1, main () at set_operation_original.c:107
107         if(!check(p1->number,B_head)){ //if this element is in B
(gdb) display i
1: i = 0
(gdb) display p1->number
2: p1->number = 1
(gdb) continue
Continuing.

Breakpoint 2, main () at set_operation_original.c:118
118             if(sign==0){
1: i = 0
2: p1->number = 1
```

发现 `1` 本来在链表中，却跳到118行这里，和预期正好相反，肯定是 `check` 部分有误，发现布尔值弄反了。

Bonus 1

从 https://awk.readthedocs.io/en/latest/chapter-one.html 上能看到中文且比较简练的 `awk` 使用教程

```
pattern { action }
```

`awk` 程序组成比较简练，模式 + 行为，我们可以按照自己的行为处理文本信息。

这里我们需要用到的是内建变量 `$NR`，它能告诉我们当前处理的是第几行。

显然我们只需要加上判断语句 `$NR == x` 就能输出第 x 行了。

Bonus 2

这个问题用编程语言是很容易的，放入二维数组中，循环变量打印出就行。

`awk` 恰好可以“复制”我们想要的功能，使用类似 `c` 语言风格的循环和 `[,]` 风格的二维数组，我们可以和编程语言中一样完成任务。

```
awk '
  {
      for (i=1; i<=NF; i=i+1) {
          line[NR, i] = $i
      }
      # store the data in a 2d array
  }

  END {
      for (i=1; i<=NF; i=i+1) {
          for (j=1; j<=NR; j=j+1) {
              printf("%s ", line[j,i])
          }
          print ""
      }
      # print the elements just like we program in C
  }' input
```

0b

对二进制文件进行读写操作，使用系统调用。

使用 `man` 指令，可以得知各种系统调用的信息。

```
man 2 open
man 2 read
man 2 close
man 2 write
```

事实上，我们需要用到的行为和 `./dump.c` 与 `./generator.c` 中的用法非常类似，学习这两个程序中的用法也不失为一种好的方法。

这里实现了一个类似cpp中 `std::vector<>` 的可变长数组，读入结构体，调用 `qsort` ，排序后输出。

错误信息处理

```
void usage()
{
  fprintf(stderr, "Usage: fastsort inputfile outputfile\n");
  exit(1);
}

void err(char *msg) {
  fprintf(stderr, "%s\n", msg);
  exit(1);
}
```

`usage` 用来提醒用户正确的参数输入
`err` 用来输出错误信息并返回值 `1`

open

- 用法

```
int open(const char *pathname, int flags);
```

其中第一个参数为文件名，第二个参数为选项。

可以使用 `or` 运算添加多个选线，这里的选线为 `O_WRONLY` 只写， `O_RDONLY` 只读， `O_CREAT` 如果没有图像即创建， `O_TRUNC` 如果已有文件，则将之清空。 `S_IRWXU` 表示创建的文件用户拥有读取写入和执行的权限。

```
if (argc != 3) usage();
outFile = strdup(argv[2]);
inputFile = strdup(argv[1]);

int inputfd = open(inputFile, O_RDONLY);

if (inputfd < 0) {
    sprintf(msg, "Error: Cannot open file %s\n", inputFile);
    err("failed to open inputfile");
}

int outputfd = open(outFile, O_WRONLY | O_CREAT | O_TRUNC, S_IRWXU);

if (outputfd < 0) {
    sprintf(msg, "Error: Cannot open file %s\n", outFile);
    err(msg);
}
```

`open` 函数会返回一个 `file descriptor`，一个整数表示文件。一般来说返回值会从 3 开始，因为默认会打开两个文件流 `stdin,stdout,stderr`

read

```
ssize_t read(int fd, void *buf, size_t count);
```

返回读取的字节大小，如果遇到 `EOF` 返回0，将数据读入 `*buf` 中

我们可以认为如果读取到的结果不等于 `EOF` 或 `sizeof(rec_)`，那么输入的数据是不合法的

```
int rd;
while (1) {
    rd = read(inputfd, &r, sizeof(rec_t));
    if (rd == 0) break; // EOF
    else if (rd != sizeof(rec_t))
    err("Error: failed to read valid data");
    if (size == len) arr = makeUpNewRoom(arr, &len);
    arr[size++] = r;
}
```

write

```
ssize_t write(int fd, const void *buf, size_t count);
```

与 `read` 类似返回成功读取的字节数，当我们写入 `rec_t` 类型数据如果返回值不是 `sizeof(rec_t)`，我们应该认为写入文件失败。

```
for (int i=0; i<size; ++i) {
    int wd = write(outputfd, &arr[i], sizeof(rec_t));
    if (wd != sizeof(rec_t))
    err("Error: failed to write data");
}
```

vector

如果数组中元素已经到达最大元素，则重新分配一个大数组

当 `malloc` 返回 `NULL` 时，分配内存失败，这也是需要报出异常的情况。

```
rec_t* makeUpNewRoom(rec_t *arr, int *plen) {
  static rec_t *tmp;
  int len = *plen;
  if (len) {
      tmp = (rec_t*)malloc(sizeof(rec_t) * len);
      if (tmp == NULL) {
          err("Error: failed to allocate memory.");
      }
      memcpy(tmp, arr, sizeof(rec_t) * len); // move to tmp
      free(arr);
  }

  arr = (rec_t*)malloc(sizeof(rec_t) * (len + BLOCK)); // allocate a bigger array
  if (arr == NULL) {
      err("Error: failed to allocate memory.");
  }
  if (len) {
      memcpy(arr, tmp, sizeof(rec_t) * len);
      free(tmp); // clear the tmp
  }
  *plen = len + BLOCK;
  return arr;
}
```