操作系统实验二
实验报告

57119101 王晨阳

2021年7月30日

实验目的

通过实验，了解Shell实现机制。实现具有管道、重定向功能的shell，能够执行一些简单的基本命令，如进程执行、列目录等。

实验过程

提示符

prompt 模仿了 Ubuntu 的格式，并在前面加了个 `SEUSH`。我们实现了一个彩色的 prompt

```
/// header [SEUSH]
printf("\033[38;5;202m\033[1m[SEUSH]\033[m"); /// @color #ff5f00

/// username@hostname
offset = 0;
getlogin_r(tmpStr + offset, MAX_PROMPT_LEN - offset);
offset += strlen(tmpStr + offset);
tmpStr[offset++] = '@';
gethostname(tmpStr + offset, MAX_PROMPT_LEN - offset);
offset += strlen(tmpStr + offset);
tmpStr[offset] = '\0';
```

```
12  printf("\033[38;5;30m\033[1m%s\033[m", tmpStr); /// @color #008787
13
14  printf(":");
15
16  /// current directory
17  offset = 0;
18  getcwd(tmpStr + offset, MAX_PROMPT_LEN - offset);
19  offset += strlen(tmpStr + offset);
20  tmpStr[offset] = '\0';
21  printf("\e[1;34m%s\033[m", tmpStr);
22
23  /// prompt
24  printf((!geteuid()) ? "# " : "$ ");
```

着色使用了 ANSI [1]，效果如下所示

输入处理

获取用户输入后，我们先将回车改为 `\0` 存储到字符串中，然后按照空格对命令进行分割。

```
1   while (1)
2   {
3       char *p = strtok(command, split); /// split by space
4
5       if (p == NULL) /// no arguments anymore
6           break;
7
8       argv[count++] = p;
9       command = NULL;
10  }
```

考虑到诸如 `|`、`<`、`>` 之类的符号前后可能会不加空格，我们再对这些符号进行分割。由此得到了一个完整的分割好的命令。

运行命令

用户输入一条命令后，父进程会 fork 一个子进程，在子进程中使用 `exec` 函数运行用户输入的命令，父进程等待子进程退出后，等待用户输入下一条命令，如此反复 [2] [3]。

```
1   pid = fork();
2
3   if (pid == 0) /// child process, run the command
4   {
5       execvp(argv[0], argv);
6       exit(0);
7   }
8   else /// parent process, wait for children to exit
9   {
10      while (wait(&status) != pid)
11          ;
12  }
```

内建命令

主要是 `cd` 、 `exit` 等命令。这个做个特判手动实现一下就好。

它们的命令分别为

```
/// cd
chdir(argv[1]);

/// exit
exit(0);
```

IO重定向

直接使用 `dup2` 函数。

对于输入重定向

```
int inFile = open(filename, O_RDONLY);
dup2(inFile, 0);
close(inFile);
```

将文件描述符0重定向到文件。

对于输出重定向

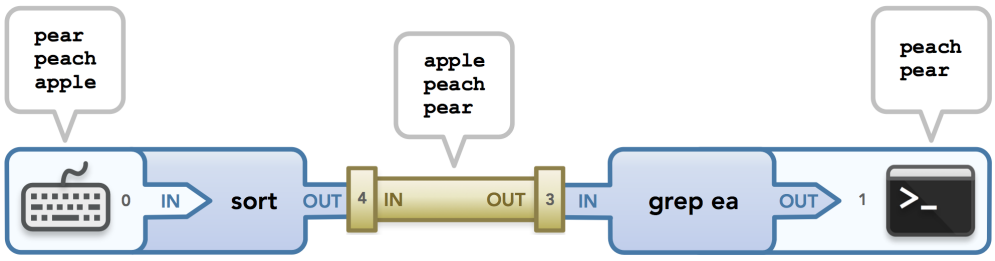
```
int outFile = open(filename, O_RDONLY);
dup2(outFile, 1);
close(outFile);
```

将文件描述符1重定向到文件。

然后正常处理命令即可。

管道

google 了一张大致的示意图[4]



要实现管道功能，只需将前一进程的标准输出重定向到管道的写数据端，将后一进程的标准输入重定向到管道的读数据端。同时整个进程 `fork` 两条子进程来实现这一目的[5]。这个就是上面的 运行命令 部分和 IO重定向 的合体 ， 不再赘述。

> 限于实验报告篇幅，以上涉及的代码均作了简化处理，实际编写时除了完善功能，还有对于程序健壮性的考量。

实验结果

我们实现了 shell 几乎所有常用功能。

基本命令 & 内建命令

```
[SEUSH]seu@localhost:/home/seu/Desktop$ pwd
/home/seu/Desktop
[SEUSH]seu@localhost:/home/seu/Desktop$ cd ..
[SEUSH]seu@localhost:/home$ ls -l
total 8
drwx------ 26 seu seu 4096 2021-07-30 08:50 seu
[SEUSH]seu@localhost:/home$ cd /home/seu/Desktop
[SEUSH]seu@localhost:/home/seu/Desktop$
```

管道

```
[SEUSH]seu@localhost:/home/seu/Desktop$ ls -al | wc
     19     146    1053
```

IO重定向

```
[SEUSH]seu@localhost:/home/seu/Desktop$ ls -al | wc >a.txt
[SEUSH]seu@localhost:/home/seu/Desktop$ cat <a.txt >b.txt
[SEUSH]seu@localhost:/home/seu/Desktop$ cat a.txt
     19     146    1053
[SEUSH]seu@localhost:/home/seu/Desktop$ cat b.txt
     19     146    1053
```

实验总结

这次实验代码量不小，前后 500 多行，边查资料边写，搞了我两天。最大的收获是对 C 语言的熟悉程度进一步提升，掌握了一些以前根本没见过的函数的用法。

通过实验，了解Shell实现机制。实现具有管道、重定向功能的shell，能够执行一些简单的基本命令，如进程执行、列目录等。

参考资料

1. ANSI escape code - Wikipedia ↩
2. purdue.edu ↩
3. Making your own Linux Shell in C - GeeksforGeeks ↩
4. Pipes, Forks, & Dups: Understanding Command Execution and Input/Output Data Flow | rozmichelle ↩
5. Tutorial - Write a Shell in C • Stephen Brennan ↩