

C/Unix 程序设计大作业

目录

Contents

0 简介	1
1 实现细节	1
1.1 命令结构设计	1
1.2 输入命令处理	2
2 实验结果	2
2.1 执行步骤	2
2.2 命令执行	3
3 总结	5

0 简介

简介

本次大作业实现了一个命令行解释器 yaush, 能够实现如下功能:

- 用户输入命令与参数, 能够正常执行命令;
- 输入、输出重定向到文件;
- 管道;
- 后台执行程序;
- 作业控制 (jobs,bg,fg);
- 历史命令 (history);
- 文件名 tab 补全, 各种快捷键;
- 环境变量、简单脚本;

1 实现细节

1.1 命令结构设计

命令结构设计

在实验中, 设置了一个结构体来保存从输入解析到的命令结构:

```
struct cmd {  
    struct cmd* next; // 下一个命令  
    int begin, end; // 命令的开始位置和结束位置  
    int argc; // 命令和参数的总个数  
    char lredir, rredir; // 输入、输出重定向的标识
```

```

char toFile[MAX_PATH_LENGTH]; // 输出文件
char fromFile[MAX_PATH_LENGTH]; // 输入文件
char *args[MAX_ARG_NUM]; // 命令的参数
int bgExec; // 是否后台执行
};

```

1.2 输入命令处理

命令读取

在此，我们将命令行的输入以字符串的形式读取，读取后保存在一个字符串中为下一步的命令分割做准备。

- 使用 `gets()` 函数读取字符串；
- 支持多行字符串的读取：如果在输入中遇到‘\’之后紧接着‘\n’的情况时，将其看做多行字符串，不会立即执行命令，而是将下一行的输入也读取到字符串中；

命令分割

在命令读取完成后，由于可能包含多条命令，因此我们首先将其进行分割转换成单个命令并保存到数组之中为下一步命令解析做准备。

- 设置一个 `bool` 变量 `beginCmd`，初始为 0；如果 `beginCmd` 是 0，那么将结构体中的 `begin` 变量置为此时的下标并把 `beginCmd` 置 1；
- 遇到‘&’字符并且之后的字符为‘\n’或‘;’，将结构体中的 `bgExec` 置为 1；
- 遇到‘\n’或‘;’时将结构体中的 `end` 变量置为此时的下标；

命令解析

将上一步分割之后的单个命令在此进行解析，分割成命令 + 参数的形式。以链表的形式保存。

- 首先使用上一步骤确定的 `begin` 和 `end` 参数将字符串中命令的部分取出；
- 然后逐字符的考察来构造命令结构体；
- 如果遇到 `$`，那么他后面跟着的是一个变量；
- 如果遇到 `>`, `<`，那么他后面跟的是重定向的文件标识符；
- 如果遇到 `|`，那么后面跟着一个命令，初始化一个新的命令结构体，并且将此结构体链接到上一个结构体的后面形成链表。

命令执行

将输入的字符串成功分割成命令之后，之后执行命令的操作就比较简单了。

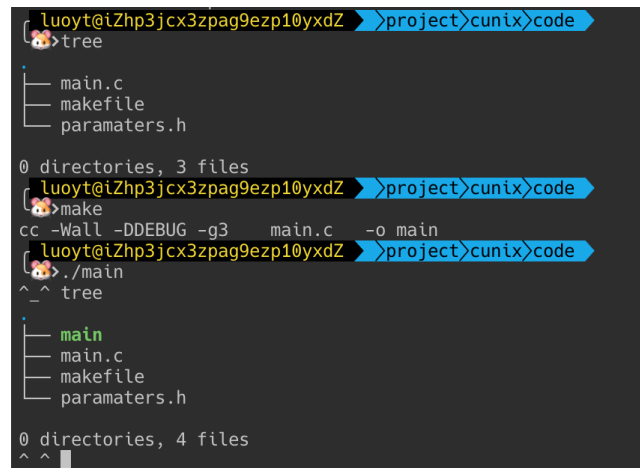
- 对于 `cd, pwd, unset, export, exit` 这些命令，直接进行了代码实现；
- 对于复杂的命令如 `cal, boxes` 等，fork 了一个子进程，直接使用 `execvp` 函数执行命令，父进程等待子进程结束即可；
- 对于后台执行的命令，直接输出“exec in bg”，父进程不用等待子进程结束即可。

2 实验结果

2.1 执行步骤

执行步骤

进入 code/文件夹下，输入”make” 进行编译，然后输入”./main” 执行进入 yaush 模式，如图 2.1所示：



```
luoyt@iZhp3jcx3zpag9ezp10yxdZ >project>cunix>code
^_^>tree
.
├── main.c
├── makefile
└── paramaters.h

0 directories, 3 files
luoyt@iZhp3jcx3zpag9ezp10yxdZ >project>cunix>code
^_^>make
cc -Wall -DDEBUG -g3 main.c -o main
luoyt@iZhp3jcx3zpag9ezp10yxdZ >project>cunix>code
^_^>./main
^_^tree
.
├── main
├── main.c
├── makefile
└── paramaters.h

0 directories, 4 files
^_^
```

Figure 2.1: 编译运行文件执行结果

2.2 命令执行

正确执行简单命令

在此我们演示几个较为简单地命令 (ls, cd, cowsay 等)，如图 2.2所示：

输入输出重定向到文件

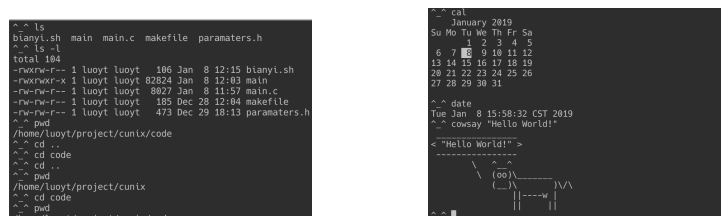
在此我们演示将命令输入重定向和输出重定向的功能，如图 2.3所示：

管道操作

在此我们通过管道操作符'|' 演示管道操作，如图 2.4所示：

后台执行程序

在此我们对比演示'sleep 10' 和'sleep 10 &' 两个命令的区别，如图所示：



```
^_^ ls
bianyi.sh main main.c makefile paramaters.h
^_^ [s -l
total 184
-rwxr-xr-x 1 luoyt luoyt 196 Jan 8 12:15 bianyi.sh
-rwxr-xr-x 1 luoyt luoyt 82824 Jan 8 12:03 main
-rw-rw-r-- 1 luoyt luoyt 8822 Jan 8 11:57 main.c
-rw-rw-r-- 1 luoyt luoyt 185 Dec 28 12:04 makefile
-rw-rw-r-- 1 luoyt luoyt 473 Dec 29 18:13 paramaters.h
^_^ pwd
/home/luoyt/project/cunix/code
^_^ cd ..
^_^ cd code
^_^ pwd
/home/luoyt/project/cunix
^_^ cd ..
^_^ cd code
^_^ pwd
/home/luoyt/project/cunix/code

^_^ cal
January 2019
Su Mo Tu We Th Fr Sa
1 2 3 4 5
6 7 8 9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31
^_^ date
Tue Jan 8 15:58:32 CST 2019
^_^ cowsay "Hello World!"
      ^__^
     (oo)\_______
        (__)\       )\/\
           ||----w |
           ||     ||

^_^
```

Figure 2.2: 简单命令执行示意图

[illegible]

Figure 2.3: 输入输出重定向运行结果

[illegible]

Figure 2.4: 管道操作结果示意图

```
luoyt@iZhp3jcx3zpag9ezp10yxdZ >project>cunix>code
^_^>./main
^_^ date
Sun Jan 13 16:13:36 CST 2019
^_^ sleep 10
^_^ date
Sun Jan 13 16:13:56 CST 2019
^_^ sleep 10 &
exec in bg!
^_^
```

Figure 2.5: 后台执行命令结果示意图

历史记录操作

在此我们演示了 'history' 命令的执行结果，如图 2.6 所示：

```
luoyt@iZhp3jcx3zpag9ezp10yxdZ >project>cunix>code
^_^ ./main
^_^ ls
main main.c makefile newfile.txt out.txt paramaters.h test.sh
^_^ pwd
/home/luoyt/project/cunix/code
^_^ cowsay "Hello World!"
< "Hello World!" >
-----
      \      ^__^
       (oo)\_______
          (__)\       )\/\
              ||----w |
              ||     ||
^_^ history
0:ls
1:pwd
2:cowsay "Hello World!"
3:history
^_^
```

Figure 2.6: history 结果示意图

环境变量设置

在此我们执行 export 设置环境变量并且用 echo 输出环境变量作为演示，如 2.7 所示：

```
^_^ export LUOYT=19970526
^_^ echo '$LUOYT'
19970526
^_^ echo '$PATH'
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games
```

Figure 2.7: 环境变量设置与输出示意图

简单脚本执行

在此我们首先使用 vim 建立一个脚本文件，然后在 yaush 中执行此脚本文件，如图 2.8 所示：

3 总结

总结

本次大作业简单地实现了一个命令执行程序 (shell)，但是在使用自己 shell 的过程中发现和自带的 bash、zsh 等成熟的 shell 相比较还是差很多。通过本次实验，充分复习了上课学到的知识也通过自己的查阅资料学习到了很多新的东西，希望在以后能够用到自己的科研以及工作之中。但是由于考试的原因，实现的 shell 并不完美，希望之后有时间能够在以下几个方面进行优化。

- 能够正确使用上下左右箭头进行调整；

```

^^ vim test.sh
^^ cat test.sh
#!/bin/zsh
ls -l;

for variable in {1,2,3,4,5}
do
    echo "Hello World, My name is Luoyt! $variable"
done
^^ ./test.sh
total 112
-rwxrwxr-x 1 luoyt luoyt 82824 Jan  8 12:03 main
-rw-rw-r-- 1 luoyt luoyt  8027 Jan  8 11:57 main.c
-rw-rw-r-- 1 luoyt luoyt   185 Dec 28 12:04 makefile
-r--w-r-- 1 luoyt luoyt   544 Jan  8 17:12 newfile.txt
-rw-rw-rw- 1 luoyt luoyt    24 Jan  8 18:16 out.txt
-rw-rw-r-- 1 luoyt luoyt   473 Dec 29 18:13 paramaters.h
-rwxrw-r-- 1 luoyt luoyt   106 Jan  8 12:15 test.sh
Hello World, My name is Luoyt! 1
Hello World, My name is Luoyt! 2
Hello World, My name is Luoyt! 3
Hello World, My name is Luoyt! 4
Hello World, My name is Luoyt! 5
^^

```

Figure 2.8: 简单脚本执行结果示意图

- 能够支持中文；
- 能够实现自动补全、快捷键等。

致谢

- 感谢老师一学期以来的辛勤讲述；
- 感谢助教一学期认真批改作业以及对作业的指导；