LAB1 编写Linux下的简单Shell

一、实验题目：

编写一个Linux下的简单shell。

二、实验目的：

1）熟悉Unix系统调用接口：fork，exec，open，close，pipe，dup，dup2等

2）熟悉Unix系统如何管理打开的文件

三、实验要求：

下载代码sh.c，在该代码基础上，编写满足下列要求的简单shell。首先结合代码弄明白execcmd等结构体以及结构体成员的用途；完成实验要求所需的代码主要加在“your code here”处。

要求能读写用户键盘输入的命令，解析命令，并创建子进程执行用户命令。

1、要求支持以下命令要求

1）输出重定向： >

2）输入重定向：<

3）管道：|

更详细的题目要求，请参考后面的英文描述。

2、编写实验报告

依据实验报告模板，编写实验报告。实验报告内容包括：

1）设计思路说明；实现过程若依赖系统某些特性，请增加说明性描述文字。

2）请给出编译代码的具体命令，简单说明如何编译、运行、测试你提交的代码。如果程序由多个源程序构成，建议编写Makefile，或者给出编译脚本。

3）请给出带运行测试结果的截图

3、作业提交要求

请将所有程序文件、实验报告、编译脚本（如果有）整体打包到单个zip文件，提交zip文件。zip文件名称：学号+姓名.zip，学号与姓名间不用添加任何符号，且学号在前。

四、补充材料

Homework: shell

This assignment will make you more familiar with the Unix **system call** interface and the shell by implementing several features in a small shell. You can do this assignment on any

operating system that supports the Unix API.

Download the 6.828 shell, and look it over. The 6.828 shell contains two main parts: parsing shell commands and implementing them. The parser recognizes only simple shell commands such as the following:

```
ls > y
cat < y | sort | uniq | wc > y1
cat y1
rm y1
ls | sort | uniq | wc
rm y
```

Cut and paste these commands into a file `t.sh`

If you are using your own computer, you may have to install gcc.

Once you have gcc, you can compile the skeleton shell as follows:

```
$ gcc sh.c
```

which produces an `a.out` file, which you can run:

```
$ ./a.out < t.sh
```

This execution will print error messages because you have not implemented several features. In the rest of this assignment you will implement those features.

Executing simple commands

Implement simple commands, such as:

```
$ ls
```

The parser already builds an `execcmd` for you, so the only code you have to write is for the ' ' case in `runcmd`. You might find it useful to look at the manual page for exec; type "man 3 exec", and read about `execv`. Print an error message when exec fails.

To test your program, compile and run the resulting a.out:

```
6.828$./a.out
```

This prints a prompt and waits for input. `sh.c` prints as prompt `6.828$` so that you don't get confused with your computer's shell. Now type to your shell:

```
6.828$ ls
```

Your shell may print an error message (unless there is a program named `ls` in your working directory or you are using a version of `exec` that searches `PATH`). Now type to your shell:

```
6.828$ /bin/ls
```

This should execute the program `/bin/ls`, which should print out the file names in your working directory. You can stop the 6.828 shell by typing ctrl-d, which should put you back in your computer's shell.

You may want to change the 6.828 shell to always try `/bin`, if the program doesn't exist in the current working directory, so that below you don't have to type "/bin" for each program. If

you are ambitious you can implement support for a `PATH` variable.

I/O redirection

Implement I/O redirection commands so that you can run:

echo "6.828 is cool" > x.txt

cat < x.txt

The parser already recognizes ">" and "<", and builds a `redircmd` for you, so your job is just filling out the missing code in `runcmd` for those symbols. You might find the man pages for open and close useful.

Note that the `mode` field in `redircmd` contains access modes (e.g., `O_RDONLY`), which you should pass in the `flags` argument to `open`; see `parseredirs` for the mode values that the shell is using and the manual page for `open` for the `flags` argument.

Make sure you print an error message if one of the system calls you are using fails.

Make sure your implementation runs correctly with the above test input. A common error is to forget to specify the permission with which the file must be created (i.e., the 3rd argument to open).

Implement pipes

Implement pipes so that you can run command pipelines such as:

$ ls | sort | uniq | wc

The parser already recognizes "|", and builds a `pipecmd` for you, so the only code you must write is for the '|' case in `runcmd`. You might find the man pages for pipe, fork, close, and dup useful.

Test that you can run the above pipeline. The `sort` program may be in the directory `/usr/bin/` and in that case you can type the absolute pathname `/usr/bin/sort` to run sort. (In your computer's shell you can type `which sort` to find out which directory in the shell's search path has an executable named "sort".)

Now you should be able to run the following command correctly:

6.828$ a.out < t.sh

Make sure you use the right absolute pathnames for the programs.