Lab 1 – A Simple Shell

Part 1: A Command Line Interface and Parser

IUPUI CSCI 503 – Fall 2019
Assigned: September 3, 2019
Due time: 11:59PM, September 17, 2019

1. Background

A "shell" is also known as Command Line Interpreter (CLI). It is a program that "wraps" (i.e., a shell) around OSes, and provides a text-based human-computer interface. When logging onto a Unix *terminal*, you type in various commands to change directories, copy files around, and list files. The commands you entered will be parsed and processed by the so-called "shell" program. There are different types of shells out there (e.g., bourne, csh, tcsh, bash shells). In this lab, you will write your own IUPUI shell. How exciting!

First, your IUPUI shell must have a prompt string. It may look like the following:

```
MyPrompt> cat inpu.txt
```

```
MyPrompt>  ls –l
```

In the above example, "MyPrompt>" is the prompt string, "ls" is a command name, and "-l" is an argument of the *ls* command. The command name is usually the file name of an executable.

2. Shell Language Grammar

The following shows the grammar for users to compose a valid shell command. Lab 1 should be able to parse an input line, and decide if the user's input is valid or not. If it is not valid, print out the corresponding meaningful error information.

In the grammar below, [*sth*] denotes *sth* inside [ ] is optional; "|" denotes Unix pipe; "*" denotes 0 or >=1 occurrences. The symbol ">" could be ">>" too.

```
command line → cmd  [< fn]  [| cmd]*  [>  fn] [&] EOL
```

```
cmd → cn [ar]*
```

cn → <a string> //command name

fn → <a string> //file name

ar → <a string> //argument

About "&": The input command line will be running as a background process. Note that a shell must wait until the program completes unless the user runs a background process (i.e., using &).

Note: If you are not familiar with shell, please take a look at the following tutorial:
http://linuxcommand.org/lc3_lts0070.php

3. Problem

To support the above grammar, your own IUPUI shell should provide the following 4 features:

- A command line parser to process your text input, print an error message if the command line is not valid.
- Input may include file redirection: **<**, **>**, and/or >>
- Input may have many pipes: cmd1 **|** cmd2 **|** cmd3 **|** … **|** cmdN
- Input may support the background: &

Your IUPUI shell program may be structured as below:

```
while (1) {
  printf("your prompt");
  Read one line from the user;
  Parse the line into an array of commands;
  Prints out the array of commands: If the input is not valid, print a meaningful error message. /* The required print out format is shown in the red color in next page. */
}
```

4. Score distribution

- Coding style: 10% (as described in the "Labs Grading Policy")
- Command line interface and parser: 90% (depending on how many test cases your program can pass). The TA will test if your program can correctly handle input lines consisting of commands and <, >, >>, |, &. Some test cases are invalid and some are valid.

The total score is 100 points.

Our TA will prepare a number of tests (from simple to complex) to test the correctness of your shell. For example, you may want to test the following command line.

```
cat < aaa | more | more | grep 2 | sort | head | wc > bbb &
```

Your shell should recognize this is a valid input, and prints those command names and their arguments from left to right. Information about input/output/background shall also be printed, which are defined as in the following 4-line output format:

> Commands: cat, more, more, grep 2, sort, head, wc
>
> Input file: aaa (or “None” if “< ” is not used)
>
> Output file: bbb (or “None” if “>” and “>>” are not used)
>
> Background or not: Yes (or No if "&" is not used)

Hint: you can enter any command line on **Linux** to see what the expected behavior should be. It may print an error message, or execute and produce an output. Your own shell should print the same message as the shell on your Linux machine if it is not valid (if it is valid, just print the above 4-line information, you don’t need to execute those commands).

Note: You will need gdb. Please look at the gdb reference card on Canvas.

5. Deliverables

Source codes in C, a Makefile, and a README.

README shall be written in a PLAIN text file (instead of MS doc), and include information such as:

1. How to build your shell (normally, TA will run “make” to build everything).
2. How to launch, then quit your shell correctly (normally, users will type in “exit” to quit a shell).
3. Several example inputs and your expected outputs based on your own tests.

Make a tar ball of the above files and send it to the TA via Canvas. The tar ball name should be “Lab1_name.tar”.