Lab 1 – A Simple Shell

IUPUI CSCI 503 – Fall 2013
Assigned: August 26, 2013
Due time: 11:59PM, September 9, 2013

1. Objectives

This lab is designed to achieve the following goals:

a) To warm up your C programming skills as quickly as possible;
b) To learn to use a collection of system calls: fork, wait, execvp, open, close, dup, dup2, and pipe;
c) To better understand how a Command Line Interpreter (CLI) actually works.

2. Background

A "shell" is also known as command line interpreter. It is an application that "wraps" around OSes, and provides a text-based human-computer interface. When logging onto a *terminal* or a *console*, 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 "shell" application. There are many types of shells out there (e.g., bourne, csh, tcsh, bash). In this lab you will write your own shell. How exciting it is! No?

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

```
UNIX> cat inpu.txt
```

```
UNIX>  ls –l
```

In the above example, "UNIX>" is the prompt string, "ls" is a command name, and "-l" is an argument of the command. The command name is usually the filename of the executable to run.

2.1 Command Language Grammar

Here is the grammar to compose a valid shell command. Your shell should be able to parse it and decide if a user's input is valid or not. In the grammar, [sth.] denotes sth. is optional, | denotes OR, * denotes 0 or >=1 occurrences.

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

```
cmd → cn [ar]*
```

```
cn → <a string> //command name
```

```
fn → <a string> //file name
```

```
ar → <a string> //argument
```

&: The shell must wait until the program completes unless the user runs it in the background (with &).

3. Problem

As the grammar indicates, your own shell should support the following 4 features:

- A command parser to process your text input, print an error if the command line is not valid.
- File redirection: first cmd **<** file, and last cmd **>** file
- Many pipes: cmd1 **|** cmd2 **|** cmd3 **|** … **|** cmdn
- Background: cmdn **&**

Your 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;
  Execute the array of parsed commands if the input is a valid command line; //system calls are used.
}
```

3.1 Score distribution

- Coding style: 10% (as described in the "Labs Grading Policy")
- Command line parser: 30%
- Support of < and >: 20%
- Support many pipes (|): 30%
- Support background process (&): 10%

Remember one "Segmentation Fault" will lead to an automatic deduction of 20%.

The total score is 100 points.

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

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

Hint 1: 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 simple shell should do the same thing as the shell on your Linux machine.

Hint 2: you need to understand pipe4.c thoroughly in order to complete Lab 1. It is posted on the course's website.

Hint 3: Test your program on Linux first before you submit it.

4. Deliverables

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

Make a tar ball of the above files and send it to the TA. The tar ball name may be "Lab1_name.tar".