ESE 333: Real Time Operating System

Linux Shell
Project 2

1 Introduction

In this assignment you will build your custom shell that supports a subset of linux commands. To do so, you are expected to take input and place output of user commands, use Portable Operating System Interface for Unix (POSIX) and implement some of the standard libc functions. (To understand about any function, please refer to linux manual, e.g. if you want to understand how gets works, type *man gets* in your terminal and you will get standard documentation on how it works, or google the Linux manual page for that function)

2 Part 0: (Milestone) A Basic Shell

In this section, you are requested to execute the commands you just read from the user and feed them into the POSIX commands. You are to use the following Linux system calls to implement the requested methods: ***fork, execvp, exit, waitpid*** (remember to use *man* to know what each system call does. You can also find man pages online, e.g., https://linux.die.net/man/)

3 Part 1: Reading Input and Parsing

In this section, you are to parse the input from user . Your code should support multiple line commands and flags concurrently. You are to use the *fputs* to print output to user.

3.1 Understanding input

The input in linux standard commands take 4 forms:

- commands
- one character flag (identified by single -)
- multi characters flags (identified by double –)
- command symbols ($|, <, >, \&$)

For example the following command:

```
ls * -a --author
```

prints current directory files, authors, and all internal files too, etc.

Your code is supposed to take unknown number of flags commands concurrently and parse it.For each command, a new process should be created (via 'fork'); and pipes for connecting output/input among processes will be needed if necessary. To do so, you are to form one linked list with each entry in it is one argument/command

For example, if the user is to type

```
echo "hello world" | ls -a --author
```

Your linked list will be one command/argument in each link. For this example, it will be the following nodes:

- echo
- "hello world"
- |
- ls
- -a
- –author

3.2 Printing Output

For first part, we do not process the arguments, thus you are required just to print it back in the following order: list of commands user enters, and then each command and its flags. **Example: Following the same example above, your part1 code should output the following:**

```
Commands: echo  ls
echo: "hello world"
ls: -a --author
etc...
```

You are to use POSIX and libc functions only for this part (i.e. *malloc*, standard data types, and create your linked list struct). You may either use the string of strings data structure that the given parse function returns or parse the user input string from scratch but you must construct a new linked list data structure that stores both the string of user input as well as the type of string it represents (see section 3.1 for types of input).

Note: Your program should run like a proper shell (i.e. in a while loop that takes one line of arguments, processes, and waits for next, etc.)

4 Part 3: Executing Commands

In this section, you are to expand upon your milestone submission with your part 1 data structure. You are to use the following Linux system calls to implement the requested methods: ***fork, execvp, exit, waitpid, dup2, pipe, open, close, getcwd, chdir*** (remember to use *man* to know what each system call does. You can also find man pages online, e.g., https://linux.die.net/man/)

You are to support the following command types:

```
">": echo hello > file.txt    (i.e. output redirect)
"<": cat < file               (i.e. input redirect)
"|": man info | grep os       (i.e. pipe)
"&": xterm &                  (i.e.background execute)
```

5 Bonus

This section is optional (but we encourage taking a look and trying some of it).

5.1 ls and cd function (20% bonus)

In order to implement *cd* and *ls* in your shell you will need to keep track of the current folder. This is where the commands getcwd and chdir come in. If you implement this function please print the current process path in the part of your shell that takes user input.

Hint: Take a look at the following library functions: readdir link1, opendir link2, and scandir link3