Name: Bishoy Nader Fathy
ID: 22

Lab 1: Shell and System Calls

12th October 2017

OVERVIEW

You are required to implement a command line interpreter (i.e., shell). The shell should display a user prompt, for example: Shell>, at which a user can enter for example, ls -l command, as follows: Shell> ls -l. Next, your shell creates a child process to execute this command. Finally, when its execution is finished, it prompts for the next command from the user.
A Unix shell is a command-line interpreter that provides a traditional user interface for the Unix operating system and for Unix-like systems. The shell can run in two modes: interactive and batch.
In the shell interactive mode, you start the shell program, which displays a prompt (e.g. Shell>) and the user of the shell types commands at the prompt. Your shell program executes each of these commands and terminates when the user enters the exit command at the prompt. In the shell batch mode, you start the shell by calling your shell program and specifying a batch file to execute the commands included in it. This batch file contains the list of commands (on separate lines) that the user wants to execute. In batch mode, you should not display a prompt, however, you will need to echo each line you read from the batch file (print it) before executing it. This feature in your program is to help debugging and testing your code. Your shell terminates when the end of the batch file is reached or the user types Ctrl-D.
Commands submitted by the user may be executed in either the foreground or the background.
User appends an & character at the end of the command to indicate that your shell should execute it in the background. For example, if the user of your shell program enters Shell> myCommand, your shell program should execute ”myCommand” in the foreground, which means that your shell program waits until the execution of this command completes before it proceeds and displays the next prompt. However, if the user of your shell program enters Shell> myCommand &, your shell 1 of 4 program should execute ”myCommand” in background, which means that your shell program starts executing the command, and then immediately returns and displays the next prompt while the command is still running in the background.

GOALS

1. To get familiar with Linux and its programming environment.
2. To understand the relationship between OS command interpreters (shells), system calls, and the kernel.
3. To learn how processes are handled (i.e., starting and waiting for their termination).
4. To learn robust programming and modular programming.

How the code is organized

The project is divided to 7 files:

1. Main: the entry point to the project.
2. Handler: handles the variables and expressions.
3. Variables: contains the session variables.
4. Environment: containing the environment data and sources to apps.
5. Commands: handles some special commands like (cd, printenv ..)
6. File Processing: handles history, log and batch files.
7. Command Parser: responsible for parsing the commands.

Main Functions

```
/**
* entry point to the application
* @param argc the number of arguments when running the app.
* @param argv the arguments when running the app.
*/
int main (int argc, char *argv[]);

/**
* sets up the paths of bins from the current $PATH variable
*/
void setUpSources();
```

```
/**
* Used to initialize the terminal environment
*/
void setupEnvironment();


/**
* checks if a key is a name of an environment variable or not
* @param key the variable name
* @return true if key is an environment variable else false
*/
bool isEnvironmentVariable(char *key);

/**
* parses the given command to a list of arguments
* which can be returned using getParsedArguments()
* @param command the string to be parsed.
* @return 0 in case of invalid command,
* 1 in case of parsing success, 3 in case the command is a comment
*/
int parseCommand(const char *command);


/**
* replaces the $var with its in the command
*/
char* handleVariablesBeforeParsing(char* command);


/**
* handles assigning values to variables
*/
void handleExpression(char *sub, char *expression, int argumentSize);


/**
* check if a variable is available in the session variables or not.
* @param key the name of the variable
* @return true if present else false.
*/
char* lookupVariable(char *key);
```

```
/**
* sets a new session variable or override its value
* @param key the name of the variable
* @param value the value of the variable
*/
void setVariable(char *key, char *value);


/**
* changes the current directory,
* @param path the new path to go to.
* @param argumentSize the number of arguments of the command
*/
void cd(const char* path, int argumentSize);

/**
* prints all environment variables with their current values
*/
void printenv();
```

How To Compile & Run it

Interactive mode

1. Open the terminal
2. CD to the project directory
3. Write “make clean”
4. Write “make”
5. Write “./shell”

Batch File mode

1. Open the terminal
2. CD to the project directory
3. Write “make clean”
4. Write “make”
5. Write “./shell ” then you file name , e.g. “ ./shell /home/Test.txt