Operating Systems

Programming Assignment 1

Part one of implementing our simple Shell.

Student: **Tarcisio Bruno Carneiro Oliveira**
Date: 2.4.2015

Report

1) Code functionality:

This is the first part of an assignment of three parts. The goal of these assignments is to build a simple Shell. Therefore, for the first part, the goal is to acquire a string typed by the user in the command prompt between double quotes.

First of all, the code verifies if the user has typed only two arguments in the command prompt.

```
if(argc == 2){
code
…
…
return 1;
}

return 0;
```

This condition guarantees that the program will not crash if the user does not use double quotes.

The code was divided in independent functions that have a specific hole. The functions are:

- char* getInput(char *argument[]);
- void getCommand(char *input, char *command);
- void getArguments(char *input, char *arguments);
- void printArguments(char *arguments);
- int hasPipe(char *input);
- int getRedirection(char *input, char *file);
- void splitPipeCommand(char *input, char *commands[]);

Moreover, the main function is based on conditional commands. Some loops are used in the functions. Two additional libraries are used: string.h and stdlib.h. The library string.h has functions to manipulate strings in C, and stdlib.h has functions to allocate memory dynamically.

Considering that the user has typed the string correct, the **getInput()** function is called and returns a character pointer for argv[1] that is where the string that contains the user commands and arguments typed is stored. The value is stored in the variable *input*.

After that, the code is divided by an if-else command. If the function **hasPipe()** returns 0, three variables are created. The command, arguments and file variables are character pointers allocated dynamically which represents strings in C language.

The next step is to get the command typed by the user using the function **getCommand()**. The command is stored in the variable *command*. In this code, all variables are passing by reference for the functions. Then, the **getArguments()** function is used to acquire the arguments typed by the user. The arguments are stored in the variable *arguments*. The **printArguments()** function print the arguments in the screen. The **getRedirection()** function returns 1 if there is any redirection symbol in the string. Moreover, the variable *file* stores the name of the file where the output will be written.

Now, if the function **hasPipe()** returns 1, the process is similar to the process described above. The difference is that the function **splitPipeCommand()** divides the *input* in two strings. In addition, the variables are duplicated, in other words, an array of character pointers with size two are used.

Difficulties:

The principal difficulty was the pointer manipulation. Passing by reference was chosen to facility the code construction. Moreover, another difficulty was processing the string to acquire the values desired. The library string.h was used to help with the string manipulation.

Finally, probably, the dynamic memory allocation is not used correct. It is possible to improve the efficiency of the allocation increasing the size of the array for iteration instead of use a fixed size.