Assignment #1: Building an OS Shell

Due: February 14, 2022 at 23:59 on myCourses

1. Assignment Description

Welcome to the first OS assignment where we will build an OS Shell!

This is the first of a series of three assignments that build upon each other. By the end of the semester, you will have a running simulation of a simple Operating System and you will get a good idea of how the basic OS modules work.

You will use the **C programming language** in your implementation, since most of the practical operating systems kernels are written in the C/C++ (e.g., including Windows, Linux, MacOS, and many others).

The assignment is presented from a Linux point of view using the SOCS server `mimi.cs.mcgill.ca`, which you can reach remotely using `ssh` or `putty`. Please make sure your assignment runs on `mimi.cs.mcgill.ca` since these are the computers the TAs will be using for testing your code.

1.1 Starter files description:

We provided you with a simple shell to start with, which you will enhance in this assignment, in the **A1-StarterCode folder**. Take a moment to get familiar with the code.

Compiling your starter shell

- Use the following command to compile: `make mysh`
- Re-compiling your shell after making modifications: `make clean; make mysh`
- *Note: The starter code compiles and runs on mimi. If you'd like to run the code in your own Linux virtual machine, you may need to install build essentials to be able to compile C code:* `sudo apt-get install build-essential`

Running your starter shell

- **Interactive mode:** From the command line prompt type: `./mysh`
- **Batch mode:** You can also use input files to run your shell. To use an input file, from the command line prompt type: `./mysh < testfile.txt`

Starter shell interface. The starter shell supports the following commands:

COMMAND	DESCRIPTION
`help`	*Displays all the commands*
`quit`	*Exits / terminates the shell with "Bye!"*
`set VAR STRING`	*Assigns a value to shell memory*
`print VAR`	*Displays the STRING assigned to VAR*
`run SCRIPT`	*Executes the commands in the file SCRIPT*

More details on command behavior:

- The commands are case sensitive.
- If the user inputs an unsupported command the shell displays *"Unknown command"*.
- `set VAR STRING` first checks to see if VAR already exists. If it does exist, STRING overwrites the previous value assigned to VAR. If VAR does not exist, then a new entry is added to the shell memory where the variable name is VAR and the contents of the variable is STRING. For now, each value assigned to a variable is a single alphanumeric token (i.e., no special characters, no spaces, etc.). For example:
    - `set x 10` creates a new variable x and assigns to it the string 10.
    - `set name Bob` creates a new variable called name with string value Bob.
    - `set x Mary`, replaced the value 10 with Mary.

- `print VAR` first checks to see if VAR exists. If it does not exist, then it displays the error *"Variable does not exist"*. If VAR does exist, then it displays the STRING. For example: `print x` from the above example will display Mary.

- `run SCRIPT` assumes that a file exists with the provided file name, *in the current directory*. It opens that text file and then sends each line one at a time to the interpreter. The interpreter treats each line of text as a command. At the end of the script, the file is closed, and the command line prompt is displayed once more. While the script executes the command line prompt is not displayed. If an error occurs while executing the script due a command syntax error, then the error is displayed, and the script continues executing.

1.2 Your tasks:

Your task is to add the following functionality to the starter shell.

1.2.1. Enhance the `set` command.

The `set` command in your starter shell assumes that the STRING variable is a single alphanumeric token. Extend the `set` command to support values of at most 5 alphanumeric tokens. If the command has more than 5 tokens *for the value*, the shell will not set the new value and will return the error: *"Bad command: Too many tokens"*.

Assumptions:

- You can assume the tokens are separated by a single space.
- You can also assume that the length limit for the tokens is < 100 characters each. We will not test with larger values.
- Only STRING can have multiple alphanumeric tokens. VAR remains unchanged (i.e., a single alphanumeric token).

Example execution:

```
$ set x 10
$ print x
```

```
10
$ set x 20 bob alice toto xyz
$ print x
20 bob alice toto xyz
$ set x 12345 20 bob alice toto xyz
Bad command: Too many tokens
$ print x
20 bob alice toto xyz
```

1.2.2. Add the `echo` command.

The `echo` command is used for displaying strings which are passed as arguments on the command line. This simple version of `echo` only takes **one token string** as input. The token can be:

- **An alphanumeric string.** In this case, `echo` simply displays the string on a new line and then returns the command prompt to the user.

  Example execution:
  ```
  $ echo mary
  mary
  $
  ```
- **An alphanumeric string preceded by $.** In this case, `echo` checks the shell memory for a variable that has the name of the alphanumeric string following the $ symbol.
  - If the variable is found, `echo` displays the value associated to that variable, similar to the `print` command and then returns the command prompt to the user.
  - If the variable is not found, `echo` displays an empty line and then returns the command prompt to the user.

  Example execution:
  ```
  $ echo $mary
                    // blank line
  $                 // return to prompt
  ```

  Example execution:
  ```
  $ set mary 123
  $ echo $mary
  123
  $
  ```

Assumptions:

- You can assume that the token string is <100 characters.

1.2.3. Enhance batch mode execution.

Batch mode execution in your starter shell enters an infinite loop if the last command in the input file is not `quit.` Fix this issue so the shell does not enter an infinite loop. Instead, the shell should display the `mysh` command prompt (i.e., entering interactive mode) after running all the instructions in the input file.

1.2.4. Add the ls command.

Add a new `my_ls` command to your shell that lists all the files present in the *current directory*.

- If the current directory contains other directories, `my_ls` displays only the name (not the contents) of the directory.
- Each file or directory name needs to be displayed on a separate line.
- The file/directory names are shown in alphabetical order.

Assumptions:

- You can assume that file/directory names are <100 characters.
- It is ok for your `my_ls` result to be different from the one we provide in the expected output in Section 2, depending on your file names and structure.

1.2.5. One-liners.

The starter shell only supports a single command per line. This is not the case for regular shells where multiple commands can be chained. Your task is to implement a simple chaining of instructions, where the shell can take as input multiple commands separated by semicolons (the ; symbol).

Assumptions:

- The instructions separated by semicolons are executed one after the other.
- The total length of the combined instructions does not exceed MAX_USER_INPUT (see starter code).
- There will be at most 10 chained instructions.
- Semicolon is the only accepted separator.

Example execution:

```
$ set x abc; set y 123; print y; print x
123
abc
$
```

2. TESTCASES

We provide you with 5 testcases and expected outputs for your solution in the **A1-testcases_public folder**. Please run the testcases to ensure your code runs as expected.

You can assume that the TAs will run one test at a time in batch mode i.e., the equivalent of:

```
make clean; make mysh; ./mysh < test
```

The expected outputs are the ones you get when the test is run in batch mode, i.e. `./mysh < testfile.txt`

IMPORTANT: The TAs will use batch mode when testing your code, so make sure that your program produces the expected outputs when testcases run in batch mode.

3. WHAT TO HAND IN

The assignment is **due on February 14, 2022 at 23:59, no extensions.** You need to submit the following:

- All the files in your project, including the testcases and a README, in one zipped file, named as follows: **lastname_firstname_mcgillid.zip**
- If you are working in a team, the zipped file name is **lastname1_firstname1_mcgillid1_ lastname2_firstname2_mcgillid2.zip** If the name is too long, only include the last names and McGill IDs.
- The README should mention the author name(s) and McGill ID(s), any comments the author(s) would like the TA to see, and mention whether the code uses the starter code provided by the OS team or not.
- The project must compile on mimi by running `make clean; make mysh`
- The project must run in batch mode, i.e. `./mysh < testfile.txt`

Note: You must submit your own work. You can speak to each other for help but copied code will be handled as to McGill regulations. Submissions are automatically checked via plagiarism detection tools.

4. HOW IT WILL BE GRADED

Your assignment is graded out of **20 points**.

- **IMPORTANT: Your program must compile and run on mimi to be graded.** If the code does not compile/run using the commands in Section 3, you will receive **0 points** for the entire assignment.

- **Known testcases: 10 points.** You were provided 5 testcases, with expected outputs. If your code matches the expected output, you will receive 2 points for each testcase. You will receive 0 points for each testcase where your output does not match the expected output. Formatting issues such as tabs instead of spaces, new lines, extra command line prompts, etc. **will not be penalized.**

- **Surprise testcases: 10 points.** The OS team also has 5 surprise testcases, similar to the ones already provided to you. The surprise testcases are graded the same as the known testcases. Note that the surprise testcases do not depend on each other, same as the public testcases.

- The TA will look at your source code only if the program runs (correctly or not). The TA looks at your code to verify that you implemented the requirement as requested. Specifically:
  - **Hardcoded solutions will receive 0 points for the hardcoded testcase**, even if the output is correct.
  - **Programming expectations.** Your code needs to meet the programming style posted on myCourses. **If not, your TA may remove up to 5 points, as they see fit.**
  - **You must write this assignment in the C Programming language**, otherwise the assignment will receive 0 points.