CS 143A
HW 3

In this assignment, you will learn to use **fork** and **exec** and practice coding with them.

Part 1 - Lecture Review (10 pts)

Briefly give 2 examples of why concurrency can be so useful. Why is it useful even on a single-CPU machine? Write your answers in a **plaintext** file called **part1.txt**

Part 2 - Unix Process Creation (30 pts)

Write a program, called **my_fork**, and an associated Makefile, named **Makefile_my_fork,** that uses fork() to create a total of 4 processes (3 calls to fork()) - each printing one letter of the alphabet from A to D K times. K will be a command line argument, but should default to 10. Have each process *flush the output* after printing each character with fflush(stdout).

Read about fork() in man or google. Be sure you do not leave any processes running. You can check this with the command, ps -ef or ps aux. Note a parent process must wait() for each child after it has terminated, or that child will become a zombie.

Here is an example call (note the letters should be a jumbled mess and should be different order every time you call your program).

```
$ my_fork
AAAAAABBBBBBBCDCDCDCDCDCDCBBBDCDCAAAADCD
$ my_fork 2
ABCDBCDA
```

NOTE: You must use fork() in this program to create new processes in order to receive credit for it.

NOTE: The only thing your program should print are capital letters! Printing lower-case letters or *any other extraneous output* will result in a grade of 0.

NOTE: If your shell creates extraneous processes (e.g. zombies) that persist after it exits, you will receive a grade of 0.

Part 3 - A Basic Shell (60 pts)

Write a C program called **my_shell** and an associated Makefile called **Makefile_my_shell**, which is a step towards a simple command shell to process

commands as follows: print a prompt ("$ "), read a command on one line, and then evaluate it. A command can have the following form:

commandName [argumentList]

where argumentList is a list of 0 or more words (don't worry about handling special characters like & or $) to be executed as a command just like your command line shell that you use on openlab (e.g. Bash). To evaluate a command, you'll need to **fork** a new process and then call some version of **exec** to execute the given command. If the command is not recognized by the Operating System, print the appropriate error on **one line and do not exit!** If the command is just a blank line (possibly containing spaces), you should not print anything and simply print another prompt and read another command as shown below. Your shell should exit properly only when it reads an **EOF** character at the start of a command line. Consider the following example commands, but keep in mind that we'll use different ones during testing:

```
$(BASH) my_shell
$ mkdir foo
$ touch foo/bar
$
$ mv foo baz
$ ls -al baz
bar
$ ^D
$(BASH)
```

We will also put **several commands in one input script** during testing:

```
$(BASH) cat inputScript.txt
mkdir foo
touch foo/bar

mv foo baz
ls -al baz
$(BASH) ./my_shell < inputScript.txt
bar
$(BASH) ls baz
bar
```

NOTE: Probably the hardest part of this assignment is parsing the commands into words: do this carefully! **DO NOT USE gets!!!** This function is unsafe and can crash your program when we give it a VERY LONG COMMAND, which you should be able to handle!

NOTE: You'll probably want to turn off buffering to ensure that your prompt is printed at the right time in relation to the command output.

NOTE: You should be using the **fork** and **exec** family of functions in this assignment. Failure to do so will result in a grade of 0.

NOTE: If your shell crashes at any point during testing, hangs after reading an EOF character or if your shell creates extraneous processes that persist after it exits, you will receive a grade of 0.

NOTE: Be very careful not to allow runaway calls to fork in your command loop: don't fork bomb yourself and other people working on the same machine!

NOTE: don't forget to trim the newline character from the user's input commands!

Submitting your work

Upload the following files to the EEE DropBox for this assignment:
part1.txt
my_fork.c
my_shell.c
Makefile_my_fork
Makefile_my_shell

Please **carefully name these files correctly and do not place them inside further folders**. Failure to follow this submission protocol may result in your files being skipped over or not compiling correctly, which will earn you a grade of 0.

Before submitting your programs, you may want to validate that their output will parse okay with our grading script by running the following test script (run them in the same directory as source code): test_my_fork.py

To test the shell, simply put a series of commands in a script file and redirect the file into your program's stdin as shown above.

Note: please don't zip the files you upload!

Upload your files to the EEE dropbox named "**CS143A HW3"** under the folder **"AssignmentSubmission"**.