Assignment 1: Process API in Unix

Due: Monday, February 1 at 11:59pm on Canvas.

Learning Objectives

The objective of this assignment is to experiment with process API in Unix and to better understand how the Unix shell works by trying to mimic its behavior.

Problem 1

Write a C program that does the following:

- Takes an input from the command line. This input can be a sequence of characters without separators, such as "date" or "ls", or a sequence that contains separators (e.g., space or "-"), such as "ls -l". Let's refer to this input as *cmd* if only one word, or *cmd* and *params* if more than one word. If the input has multiple separators, *cmd* is the sequence of characters before the first separator, and *params* is the rest of the input. For example, if the input is:
  `ls -a -l`
  then *cmd* is '`ls`' and *params* is '`-a -l`'.
- Creates a new process (using `fork()`);
- Makes the new process execute *cmd* with *params* as parameters, if given.
- Waits for the new process to finish executing, and then prints `++++` on a new line.

Problem 2

This problem builds significantly on the previous problem. Specifically, it asks you to again use `fork()` to create processes and `exec()` (or one of the many variants) to assign the newly created processes what to do. In addition, however, it asks you to mimic the behavior of a shell command such as:

```
ls | wc
```

What happens in the case above is the output of the first command ('`ls`') becomes the input to the second command ('`wc`'). (Try it in a terminal on a Unix machine).

Thus, you are required to write another program that:

- Expects an input of the form: `cmd1 | cmd2`
- Creates two processes
- Makes the first process run `cmd1`
- Makes the second process run `cmd2`
- Makes sure that the output of the first process becomes the input of the second process (using the function `pipe()`).
- Waits for the two processes to finish before it ends by printing `++++` on a new line.

What to Submit

Name your C files as `problem1.c` and `problem2.c` and place them in a folder called `assignment1`. Compress the folder into a .tar file:

```
tar -cvf assignment1.tar assignment1
```

You will then need to download the tar file into your local machine. You may use FileZilla, PuTTY or a similar tool. If doing it from the CLI, issue the following command:

```
scp <usf_id>@cselxXX.csee.usf.edu:<abs_path_to_the_tar_file> .
```

This will download the tar file to the location this command was run from.
Submit the tar file on Canvas.

Other Instructions and Suggestions

- This is an independent assignment. You are expected to work by yourself. If you use external sources (such as code from the web), please cite them as comments in your code. We will use MOSS to identify plagiarism in code.
- The C functions (that, in fact, use system calls) you'll need include: fork(), wait(), pipe(), exec() (or variants). You might want to also experiment with and use for debugging the functions getpid() and getppid().
- I may have confused some of you during office hours by suggesting that you use the function call `system()`. No way. Ignore. What `system()` does it creates another process (by calling fork() internally) – you do not need that.
- For on-line reference manuals use the command `man` as below:

  ```
  man -s 2 pipe
  man man
  man getppid
  man -s 2 exec
  ```