CSCI 330 Assignment 6 (Spring 2021)

Pipes (100 pts)

Purpose

The purpose of this assignment is to make sure that the students can work effectively with the `fork`, `pipe`, and `dup`, system calls. It should also grant some insight into how the shell enables input and output redirection behind the scenes.

Description

For this assignment, you will be writing a single program that enters a loop in which each iteration prompts the user for two, single-line inputs. If the text entered for either of these is `done`, then the program should immediately exit. If `done` is not found, then each of these lines of input will be treated as a command line to be executed. These two commands should be executed so they behave as if the user had typed `command1 | command2` at the shell prompt. This means that the standard output of the first command will be redirected into the standard output of the second command.

It should be noted that using `cin >>` will not grab a whole line; you will need another function to get the whole line.

You will need to be able to split the text entered by the user into the individual command line arguments that comprise it. I do not care how you accomplish this, as long as it's either part of the standard library or code that you wrote entirely on your own. Possibilities include the C function `strtok` or the C++ `istringstream` object. You do not need to worry about escaping special characters while splitting the string for the purposes of this assignment; the split can be performed based on spaces alone.

After these commands have both finished executing, your program should prompt for another pair of input commands to run, repeating the procedure over and over again until the program receives `done` as one of its inputs.

Example

```
% ls
a  b  c  def  example  typescript
% ls | wc
      6       6      29
% ls -a
. .. a  b  c  def  example  typescript
% ls -a | wc -w
8
% ./assign6
command 1? ls
command 2? wc
      6       6      29
command 1? ls -a
command 2? wc -w
8
command 1? done
%
```

Requirements

- You may not use the `system` function to do this. You must make your own child processes with `fork`, do the appropriate changes to the file descriptors in each of them, and have each of them use one of the `exec` calls to run the commands given by the user.
- Use the `pipe` system call to create the pipe and supply the file descriptors used to communicate between the two programs that are run.
- If an error occurs at any point, there should be an appropriate error message printed to the standard error stream.
- Your program must be able to handle user-specified commands that contain up to five command-line arguments past the command.
- Your program must be able to handle user-specified commands that are up to `255` characters long.
- Make sure to close unused pipe file descriptors in all processes. If you fail to do this, your program will likely stall. (deadlock)
- Do not forget that this program involves a loop. After both programs have finished running (but not until), the program should go back to the beginning and prompt for two more commands to run.
- Like all other programming assignments, this must be well-documented.
- Like all other programming assignments, your submission must compile and run on `turing` and/or `hopper`. It doesn't matter if it ran on your home computer if it won't compile when the TA tries to grade it on the department UNIX machine.