NIU CSCI 330 - Using Pipes

Purpose

The purpose of this assignment is to make sure that the students can work with the `fork`, `pipe`, and `dup` system calls. It should also grant some insight into how the shell makes input/output redirection work 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 of either one of the inputs is "quit", the program should immediately exit. If "quit" is not found, each of these lines of input will be treated as a command line to be executed. These two commands should be executed as if the user had typed `command1 | command2` at the shell prompt, meaning that the standard output of the first command gets redirected into the standard input of the second command. (It should be noted that using `cin >>` will not grab a whole line; you will need to use another function instead.)

You will need to be able to split the text entered by the user into the different command line arguments that comprise it. I do not care how this is accomplished, but some possibilities are to use the C function `strtok`, or a C++ `istringstream` object. You do not need to worry about escaping special characters when 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 process over and over again until the user gives "quit" as input.

Requirements

- You may not use the `system` function to do this. You must make your own child processes, do the appropriate changes to the file descriptors in each of them, and have them use one of the `exec` calls to run the commands given by the user.
- Use the `pipe` system call to create 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 are up to 127 characters long.
- Your program must be able to handle user-specified commands that contain up to five command-line arguments.
- Make sure to close unused pipe file descriptors in all process. If you do not, your program is likely to stall.
- 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.