Exercise 03

In class, we talked about how many shell commands are implemented as separate, modular programs. The shell lets us run them one at a time or, if we want, we can start up multiple commands at once and have them work together. That's what's going on in the following:

$ ps -ef | grep bash

This command runs **ps** with the **-ef** option. When run like this, ps will print a report of all processes running on the system. The vertical bar asks the shell to start up the grep command at the same time and send the output of ps as input to grep. Grep is a pattern matching program. Here, we're asking grep to filter out all processes except those started with the word "bash" on the command line. So this command will (mostly) print a report of all copies of bash currently running on the system.

For this exercise, we're going to write our own code to do what the shell is doing here. We already know how to start two child processes, get one of them to run ps and one to run grep and then wait for them both to terminate. The new thing we need to figure out is how to get them to communicate. We'll do that with an (anonymous) pipe.

I'm giving you a partial implementation, pipe.c, to help get you started. It does the work of creating two child processes and waiting for them to finish. You just need to add the communication code and the code to actually start up the ps and grep programs. You'll need to do the following. I've left little comments like "// ..." in the code, where you need to add something.

- You'll need to create a pipe before you start creating children. That way, the children will inherit the file descriptors for reading and writing through the pipe.
- Before starting ps in the first child, you'll need to redirect its standard output so it writes into the writing end of the pipe, rather than to the terminal. This is a two-step process, a little like what we did to redirect to a file on the previous exercise.
    - First, use dup2() to close the child's old standard output file descriptor and replace it with a copy of the writing end of the pipe. Henceforth, any printing the child does will automatically get written into the pipe. If you need to print out any messages for debugging, you may want to use standard error since it will still go to the terminal. Kind of convenient to have it around.
    - Now, the child has two file descriptors that both write to the pipe, the one it inherited from its parent and the copy it just made as standard output. Close the original one (the one inherited from the parent), since the child doesn't need it now.
- In the first child, use execl to start ps with the "-ef" option.
- Put similar code in the second child, to replace its standard input with the reading end of the pipe. Then, have

this child run grep with the word “bash” as a command-line option.

- In general, you'll need to make sure you close any unneeded file descriptors for reading or writing to the pipe. Grep will keep reading from its standard input until it reaches EOF. For that to happen, all buffered data in the pipe needs to be read and all file descriptors for writing into the pipe need to be closed. So, for example, the parent should close its copies of both the reading and writing file descriptors for the pipe after it creates its children. The parent no longer needs either of them (and keeping them open would cause the grep child to hang). Likewise, the children don't need to keep both ends of the pipe open, just the one they need to use.

Once your program is working, it should print a list of running commands with the word “bash” in them. This will mostly be all the shells running on the system, maybe along with a copy of the “grep bash” your own program started.

When you're done, turn in your source code for your completed pipe.c program. Submit the file to the Gradescope assignment named EX03.