CS 111: Operating System Principles
Lab 1

# Pipe Up 1.0.1

Jon Eyolfson
April 12, 2021
Due: April 23, 2021 at 8 PM PST

In this lab you'll be writing the low level code performed by the pipe (`|`) operator in shells. Users will pass in executable names as command line arguments and you will execute each one in a new process (again similar to a shell). You're expected to understand your own implementation and test it yourself. Lecture 5 and 6 cover the function calls you should need (except as noted below).

**Additional APIs.** Given you are just using the executable names, you may use the C library helper functions for `execve`, such as `execlp`. `execlp` will let you skip using string arrays (using C varargs), and it will also search for executables using the PATH environment variable. You may notice your program hanging waiting for input. You must call `close` on any file descriptors not explictly used in your process. Failure to call `close` will inform the operating system you are not done with it, and it will never return end-of-file from a `read` system call.

**Starting the lab.** You'll receieve all future labs through my repository (Lab 0 had you set it up as `upstream`). Run the following command to get the skeleton for Lab 1: `git pull upstream main`. You should be able to run `make` in the `lab-01` directory to create a `pipe` executable, and then `make clean` to remove all binary files.

**Files to modify.** You should only be modifying `pipe.c` and `README.md` in the `lab-01` directory.

**Your task.** You should execute the programs in `argv[1]`, ..., `argv[argc - 1]` as new processes. You also need to create a pipe between two subsequent processes. For example, a pipe should connect `argv[1]`'s standard output to `argv[2]`'s standard input (if there are at least two procsses). The standard input of first new process must be the same as the standard input of the parent process (`pipe`). Also, the standard output of your last new process must be the same as the standard output of the parent process. You should be able to handle between 1 to at least 8 programs (more is okay). All standard errors should be the same as the parent's standard error. You do not need to handle passing additional command line arguments to every individual new process. Finally, fill in your `README.md` so that you could use your program without having to use this document.

**Errors.** Your program should (of course) handle errors from all function calls you make. Your program should exit with the proper `errno` of the failing call. It is okay to exit as soon as you find an error, without any error recovery. If there are no programs as command line arguments, your program should exit with errno `EINVAL` (invalid argument). Your program should work with a single program as a command line argument. Your program should not create any orphan processes (you must `wait`).

**Tips.** You should come up with smaller subtasks yourself. One approach may be to execute one program from the command line, then multiple programs independently. Afterwards set up your pipe between two processes, then multiple processes. Start small then work big.

**Example output.** Your output should be the same as if you were to use `|` between each program in your shell.

```
> ./pipe ls
Makefile  pipe  pipe.c  pipe.o  README.md
> ./pipe ls cat wc
      5       5      38
```

The last command should have the same output of: `ls | cat | wc`.

**Submission.** Simply push your code using `git push origin main` (or simply `git push`). For late days will we look at the timestamp on our server. We will never use your commit times as proof of submission, only when you push your code to the course Git server.