EECS 678 - Lab 03

Theodore Lindsey

April 24, 2015

`http://people.eecs.ku.edu/~vvivekan/lab/ipc/ipc.html`

1. *Briefly describe the design of the program. How many processes and how many pipes are created with your solution? How does data flow from process to process?*

   First it initializes three pipes, then it creates four forks in succession. Within each fork, the program first assembles the text necessary to call the intended process image, then it links the necessary pipe ends to the in and out streams, then closes unused pipes, and then executes the intended bash program. Since the pipes map the output stream to the write end of the pipe and then from the read end of the pipe to the input stream, data from the output of one command is transferred to the input of the following command as if it were a commandline parameter.

2. *Did you have any problems writing or debugging this program? How did you test and debug your solution?*

   The slides for this lab suggested using `int pipe(int pipefd[2])` to set up the pipe. In practice, however, I found that method didn't work reliably and it was instead necessary to first issue `int pipe1[2];` and then, later, issue `pipe(pipe1);`. Generally, however, this lab didn't have any significant bugs to work out. The structure of each of the fork if statements were essentially identical and the only modifications needed for each fork was to change the string structure prior to replacing the current process image with a new one.

3. *When he was head of Bell Labs, Doug McIlroy summarized the 'Unix philosophy' as follows:* "Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface." *How do pipes contribute to the Unix philosophy? Also, do you think this is a "good" philosophy for software engineering? Why or why not?*

   Pipes provide a way for programs to communicate between each other with text streams. Because of the ability to communicate between programs, instead of having to have a multitude of features bundeled into a single program (eg putting the features of `ls`, `rm`, `cd`, `mkdir`, `mv`, `cp`, etc into the same program), each program is able to, as stated in McIlroy's quote, focus on a single task and make sure it works well with plenty of options. If these small, modular programs need to communicate to other programs, they can do so through a pipe stream.

   I think this is a good philosophy. A very useful result of designing software as described by McIlroy is that each component is very modular. As a result, it can be re-used over and over in many applications. And, as new features or options are needed, implementing them is much more simple and will usually not result in breaking pre-existing software. Additionally, when the individual programs focus on accomplishing only one task, it is easier to prevent bugs and also remove bugs that may crop up.

   Using text streams is a good idea as it allows communication between processes in an open manner. It encourages a standardized output format that takes no proprietary software to read. It also allows for unanticipated used further down the line. If one piece of software is no longer maintained, it doesn't require a patch to work with newer software.