

# CS 213, Spring 2003

## Lab Assignment L5: Extra Credit (10 points)

Dave Koes (dkoes@andrew.cmu.edu) is the lead person for this portion (and only this portion) of the assignment.

### Pipes (8 points)

Pipes allow the output of one process to be directly mapped to the input of another process. For example:

```
tsh> /bin/cat foo | /bin/sort
1
2
3
4
```

Add support for pipes (|) to your shell. For the purposes of this assignment, each program in the commandline should be a distinct job. Instead of changing the provided code to support having multiple jobs in the foreground, you may find it easier to mark just the last job in the commandline as being in the foreground. However, as with a real shell, suspending (Ctrl-Z), interrupting (Ctrl-C), or continuing (bg or fg) a single process should affect all related processes (processes are related if they are joined by pipes). You'll find process groups useful for achieving this behavior.

You will find the system calls `pipe`, `dup2`, `setpgid`, and `_getpgid` useful. It wouldn't hurt to read the relevant portions of the book, either.

A reference shell, `tshpipe` is provided in `/afs/cs.cmu.edu/academic/class/15213-s03/labs/L5`. You should do your best to match the reference shell *exactly*. If it exhibits behavior that you find so abhorant you refuse to duplicate it in your own shell, please contact me (dkoes@andrew) and we'll discuss it.

Since this is for extra credit, a certain amount of pain and suffering is expected. Therefore, we're not providing the four trace files we'll use to test your pipes implementation. You're expected to do all testing yourself. Each trace file is worth 2 points and performs the following types of tests:

- `pipe01.txt`: Basic pipe support.
- `pipe02.txt`: More complicated pipe support, pipes with i/o redirection, simple error checking.

- `pipe03.txt`: Make sure `signals`, `bg`, and `fg` work correctly with pipes.
- `pipe04.txt`: Major stress testing of all of the above. Excessive error checking.

## A Better Shell (2 points)

You probably shouldn't do this part of the extra credit. It definitely isn't worth the measly 2 points you get. However, if you feel like a challenge, and have bountiful amounts of free time (which presumably only happens at CMU after you've given up sleeping and socializing), you can earn up to 2 points for making significant improvements to your shell.

Your improvements should not in anyway change the output of your shell when running the trace files (unless you want to lose 4 points per a trace file). If you so desire, you may create a commandline switch that can be passed to your shell to enabled different (and presumably better) output.

Some possible improvements:

- Make it so your shell can run inside itself. There's a **lot** involved in doing this correctly (and you do have to do it correctly). In particular you have to correctly control access to the terminal (see past 15-412, project 1, writeups for more info).
- Add spiffy user-friendly features like a history and tab completion.
- Rewrite the shell framework to be efficient (just say no to linear time algorithms) and get rid as many hardwired limits (like `MAXARGS`) as possible.
- Anything else you can think of that requires a fair amount of work or an inordinate amount of cleverness and results in a better shell.

In order to receive your 2 points, you'll have to demo and defend your work sometime after the deadline. Depending how impressed I am, you'll receive 0, 1, or 2 points.

Good luck!