Seminar 1: Shells and Processes

Week 1 - (2017/01/19)

Reminder

- Assignment 1 Due: 2017/01/30 @ 23:55
- Less than 2 weeks away!
- Get started early

Windows Users

- You **MUST** use a POSIX/Unix environment
- Either by
    - 1) Installing a Linux distribution on your computer (dual-booting or virtualization are fine)
    - 2) Installing Cygwin (GNU/Unix tools on Windows)
- If you're not sure, test it on the lab computers
- You can also use your RaspberryPi if you still have it

Submitting A1

- You need:
  - A readme file
  - All source code (including lex file)
  - A Makefile
- If it doesn't compile, it will not be marked!

Lex

- Lex is a tool that makes string parsing easier
- Allows definition of formal rules for strings and how to split them
- It has already been set-up for you (files on Moodle)

Note: On MacOS

- If you're using a Mac, you may get this error:
- **`ld: library not found for -lfl`**
- In the Makefile, change **`-lfl`** to **`-ll`**
- Why? It's just arbitrarily different **¯_(ツ)_/¯**

Shells

- A shell is just a program that **executes other programs**
- You use a shell (bash) every time you open your terminal
- The terminal **IS NOT** the shell!
- The terminal sends input to the shell, and displays its output

Process Management - fork

- `fork()` is an operation where a process creates a copy of itself
- Process is identical in every way **EXCEPT FOR** the process ID
- `fork()` is the primary method of process creation on POSIX operating systems
- `fork()` returns 0 if it is the **CHILD** process, a positive integer (the child process PID) if it is the **PARENT** process, and a negative number if fork failed

Process Management - fork

- Parent processes are responsible for ensuring that their child processes exit before they do
- If a parent process quits with child processes still running, they become zombie processes
- The operating system will clean these up if they exit, but they may continue to run in the background consuming resources

Process Management - exec

- **exec( )** is an operation where a process **replaces** its program code with that of a new program
- Note replace: all original code after exec is GONE

Process Management - wait

- **`wait()`** is an operation that pauses a parent process until a child process has finished
- Child process can exit due to errors (how will you handle this?)
- You'll have to figure out how to handle background processes (the & symbol in the assignment)

Final Words

- Remember to read the man pages
- http://www.gnu.org/software/libc/manual/html_node/Implementing-a-Shell.html is a very in-depth resource for building a shell (for goodness sake do not copy the code here)
- Talk to the TAs/ask questions on Moodle
- Don’t share your code!