CS0330: SHELL 1 GEAR UP

Project Overview

You'll be writing a partial-functionality C shell, just like the terminal you use everyday on your machines.

This shell will be able to do a few things like run built-in commands, execute programs, and redirect input and output.

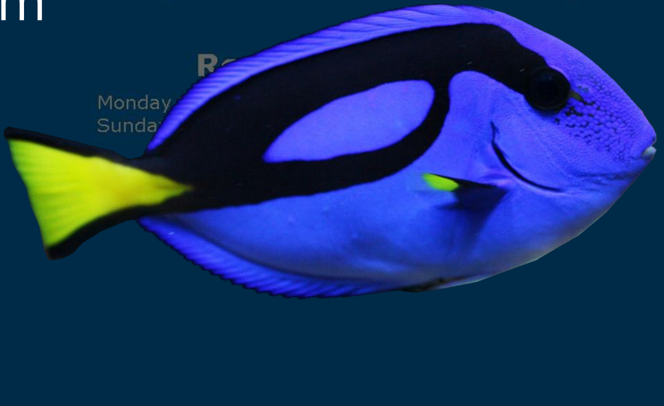
Topics to Review

- Parsing strings
  - Look at string-parsing library functions listed on the handout!
- File descriptors
  - Stdin, stdout, stderr
- REPL
- Terminal I/O
- UNIX commands
- Child processes

Roadmap

`cs0330_install shell_1`

- Start by filling in your **Makefile.**
  - Review Makefile lab for guidance
  - This is crucial to being able to test your code as you proceed!
- Write your **Read-Eval-Print Loop** (REPL)
  - Try typing and make sure it prints your prompt again
  - Which system calls should you use for the REPL?
    - Hint: read the handout!
- Implement command line **parsing**
  - Parse out redirection symbols and files from arguments to be passed to `execv()`
  - How does execv() take its arguments?



Roadmap, continued

- Implement forking & command execution
    - Using **`fork()`** & **`execv()`**
        - Look at the stencil code in the handout
    - Which commands will you need to use `fork()` for?
    - Test on the commands in the handout,
      using bin and usr directories appropriately.
    - Note: this is for all **non** built-in commands
- Implement **built-in commands**.
    - Which UNIX system calls do you need to
      use for these built-ins? (hint: read the handout!)

Roadmap, continued

- Implement input and output **redirection.**
  - Review file descriptors!
  - Test your redirection against the demo's behavior.
  - Account for cases like multiple redirection characters in the user input.
  - Redirection symbols can be **anywhere.**
- **Error handling**!
  - All system calls should be checked for their return values.
  - You should always print errors on bad input
  - Refer to demo to check which errors are handled and how

Make sure each roadmap step works before moving on to the next

(Try out edge cases)

(super) Simple REPL Example

~* this is pseudocode!! *~

```
while(1) {
    input = readUserInput(stdin);
    if (input != EOF) {
        // parse & eval input
    } else {
        // what are the other cases?
        return;
    }
}
```

File Descriptors

- 3 file descriptors:
  - Standard input (stdin)
  - Standard output (stdout)
  - Standard error (stderr)
- Important for input/output redirection
- Handout has details on how to manage them

Demo!

Run `cs0330_shell_1_demo` **from ~anywhere~!**

Testers!

Run all tests with `cs0330_shell_1_test -s 33noprompt -u /course/cs0330/pub/shell_1`

You can also run a single test by providing `-t /course/cs0330/pub/shell_1/<test>` instead of the `-u` option.

Feel free to look at the `setup`, `input`, `output`, and `error` files in each `/course/cs0330/pub/shell_1/<test>` directory to see the commands used in each test case.

It runs 99 tests, but unchecked sys calls ain't one.

Tips

- Only use allowed non-syscall functions from the handout.
- Parsing user input is an important part of shell. ~*Make use of the allowed string-parsing functions listed on the handout*~
- Your code will be used for the next assignment, so it is important to get the basic functionality correct.
    - Functioning REPL, child processes, and **cd**

More Tips

- This will be the largest assignment you've had so far, so give yourself enough time to think through your approach and debug.
- Plan out where everything will be handled before you start coding.
- Come to hours early in the week!
- Please don't write everything in main()!
- Style is important! Make your code clean, concise, readable, well-abstracted, and well-commented.

So Why Am I Doing This?

You will learn:

- To use the command line better.
- How to make a REPL! Wheee!
- About input, output, and file redirection.

You'll become a master of your terminal! All (most) of the things that are magical about your UNIX shell today will make sense to you after these two projects.

Additional Questions?