CIS 3110: Operating Systems

Assignment 1: Writing a Shell

Due Date: Jan. 30, 2015

The requirement

In this assignment you are to implement a simple UNIX shell program. A shell is simply a program that conveniently allows you to run other programs. Read up on your favorite shell to see what it does.

The requirements for completing this assignment successfully are described under **specification**.

Specification

(1) Your shell must support the following:

1. **The internal shell command "exit" which terminates the shell**.
   Concepts: shell commands, exiting the shell
   System calls: exit()
2. **A command with no arguments**
   Example: ls
   Details: Your shell must block until the command completes and, if the return code is abnormal, print out a message to that effect.
   Concepts: Forking a child process, waiting for it to complete, synchronous execution
   System calls: fork(), execvp(), exit(), wait()
3. **A command with arguments**
   Example: ls -l
   Details: Argument 0 is the name of the command
   Concepts: Command-line parameters
4. **A command, with or without arguments, executed in the background using &.**
   For simplicity, assume that if present the & is always the last thing on the line.
   Example: xemacs &
   Details: In this case, your shell must execute the command and return immediately, not blocking until the command finishes.
   Concepts: Background execution, signals, signal handlers, processes, asynchronous execution
   System calls: sigset()
5. **A command, with or without arguments, whose output is redirected to a file**
   Example: ls -l > foo
   Details: This takes the output of the command and put it in the named file
   Concepts: File operations, output redirection
   System calls: freopen()

6. **A command, with or without arguments, whose input is redirected from a file**
   Example: sort < testfile
   Details: This takes the named file as input to the command
   Concepts: Input redirection, more file operations
   System calls: freopen()
7. **A command, with or without arguments, whose output is piped to the input of another command.**
   Example: ls -l | more
   Details: This takes the output of the first command and makes it the input to the second command. You are required to implement one level of pipe. However, multi-level pipe implementation is welcome.
   Concepts: Pipes, synchronous operation
   System calls: pipe()

(2) Your shell should implement the following new commands:

1. **A command "add" that adds all numbers in command line (decimal or hexadecimal).**
   Example: add 100 210 0xa
   Details: add all numbers and output the sum
   Output: 100 + 210 + 0xa = 320
2. **A command "arg" that counts and lists command line arguments.**
   Example: args 1 2 3 4 5 "six, seven"
   Details: count and list arguments
   Output: argc = 6, args = 1, 2, 3, 4, 5, "six, seven"
3. **A command "???" that is named and designed by yourself.**
   Example: you make an example
   Details: you specify the function
   Output: you give a sample output

Note: You must check and correctly handle all return values. This means that you need to read the man pages for each function to figure out what the possible return values are, what errors they indicate, and what you must do when you get that error.

Submission

- If you have any problems in the development of your programs, contact the teaching assistant (TA) assigned to this course.
- You are encouraged to discuss this project with fellow students. However, you are **not** allowed to share code with any student.
- Each student should submit a brief report (2-3 paragraphs) that explains your algorithm, any assumptions you made, and the way to run your program.
- If your TA is unable to run/test your program, you should present a demo arranged by your TA’s request.