CSE422: Operating Systems Organization Spring 2015

Lab 1

Assigned: Feb. 5th Due: Feb. 19th

Overview

You are to write and test a C program that implements a simple shell whose executable is called *xssh*. The features of the xssh shell are described below. It is a toy shell since we will assume that the program only operates correctly when given correct input; the behavior of xssh when given incorrect input is undefined. It is an incomplete shell since it is missing many features included in other shells. Some of the missing features are briefly described at the end of this document. It is a pedagogic shell since it produces output that exposes the workings of the shell and contains many of the fundamental features in common shells. The command line looks like:

```
xssh [-x] [-d <level>] [-f file [arg] ... ]
```

where the square brackets ([ ]) indicate an optional word(s), and "..." indicates 0 or more words. xssh normally reads commands from stdin. The options are:

- -x: The command line after variable substitution is displayed before the command is evaluated.
- -d <DebugLevel>: Output debug messages. DebugLevel=0 means don't output any debug messages. DebugLevel=1 outputs any debug messages you might need while implementing the lab. For your convenience, the default value will be DebugLevel=1, but your code should support the ability to turn off any extraneous debug messages using "-d 0".
- -f file [Arg] ...: Input is from a file instead of stdin; i.e., File is a shell script. If there are arguments, the strings are assigned to the shell variables $1, $2, etc. The location of File follows the rules described in Search Path below. For example "xssh -f cmds.txt foo bar" would read commands (one line in the file is equivalent to typing in a line through stdin) with $1="foo" and $2="bar"

External Versus Internal Commands

An internal or built-in command is one that the shell executes itself instead of running another program (via fork/exec). All commands that are not listed as internal commands below are treated as if they were external or non-built-in commands.

Search Path

The shell searches for external commands and shell scripts using these rules:

- If the command name is an absolute or relative pathname P, the file P should be an executable binary (e.g., /usr/bin/zip) or executable script (/usr/bin/spell).

- Otherwise, search the directories listed in the PATH environment variable.

Internal commands

In the description below, W stands for a single word. A non-whitespace character (SPACE or TAB) or a newline character terminates a word. ”...” indicates a continuation of 0 or more repetitions of the preceding symbol. For example, ”W ...” means one or more W’s. I stands for a particular type of word that is an integer. In cases where the ordinal position of a word is important, we will use W1, W2, ... to mean Word1, Word2. Also, a word can be a variable and should be substituted as noted in the “Variable Substitution” section below.

- **show *W ...*** : display the word(s) followed by a newline
- **set *W1 W2***: set the value of the local variable W1 to the value W2
- **unset *W1***: un set a previously set local variable W1
- **export *W1 W2***: export the global variable W1 with the value W2 to the environment
- **unexport *W***: unexport the global variable W from the environment
- **chdir *W***: change the current directory
- **exit I**: exit the shell and return status I
- **wait I**: the shell waits for process I (a pid) to complete. If I is -1, the shell waits for any children to complete. Note that you can't wait on grandchildren.

External Commands

An *internal* (or *built-in*) command is one that the shell executes itself instead of running another program (using fork/exec). All commands not listed in the Internal Commands section should be consider *external* (or *non-built-in*) commands.

Background Commands

A command that should be run in the background is ended with an ampersand "&" as in "C &" which means execute "C" in the background. Background commands return immediately and execute in the background. The xssh should return control immediately to the user so they may execute additional commands.

Local vs. Global Variables

Local variables are available only to the current shell and not any sub shells or processes. These are variables that are configured using the “set/unset” internal commands. Global variables are available to the current shell and any child processes including sub shells. Global variables are configured using “export/unexport” internal commands.

Variable Substitution

The dollar sign character "$" as the first character of a word "$XY" signifies that "XY" is a variable Single-level variable substitution is always done before command evaluation No recursive substitution is allowed (e.g. the "$Y" in "$X$Y" is not replaced since XY is a single unit of text, aka “word”).

There are three special shell variables

$$: PID of this shell

$?: Decimal value returned by the last foreground command

$!: PID of the last background command

Stdin/Stdout Redirection

Stdin and/or stdout can be redirected to/from a file using the following syntax "C < F1 > F2" which means redirect stdin from F1 and redirect stdout to F2 for program "C"

Terminal-Generated Signals

"CTRL-C" should terminate the foreground process but not xssh.

Other Features

1. Each word (token) is separated by whitespace.
2. Multiple spaces/tabs are reduced to a single space during the substitution and line scanning phase.
3. The command line prompt is the three character sequence '>> ' (i.e., >, >, space).
4. The # character signifies the beginning of a comment. All other characters following and including the # is ignored during interpretation.
5. Blank lines are ignored.

Assumptions

You make assume all of the following for this simple shell:

- invalid input is undefined
- no control structures (e.g. if, while, etc)
- no interactive features like auto-complete
- no filename expansion like "~" or "*"
- no command substitution
- no job control commands like "fg" or "bg"

Man pages

In addition to fork(2), waitpid(2), execvp(2), sh(1), exit(3), and fgets(3), you may want to look at getenv(3), putenv(3), chdir(2), dup2(2), open(2), read(2), write(2), close(2), setpgid(2), sigaction(2), sigsetjmp(3) and siglongjmp(3), strtok(3)

Implementation Notes

You are NOT allowed to use the ***system*** (e.g. "man system") system call to implement commands. The ***system*** system call refers to the function defined as "int system(const char *command);". But as a strategy, there is nothing wrong with using the ***system*** system call as a temporary implementation means while debugging.

Grading

- If your lab doesn’t compile on shell.cec.wustl.edu, you’ll lose 50 (out of 100) points
- Every compiler warning is a loss of 1 point (e.g. three warnings would be -3 points). Your code will be compiled using the “-Wall” flag
- Documentation is critical and if it’s missing, you’ll lose 25 (out of 100) points.
- You may discuss the lab concepts at a high with classmates but please don’t share any code.

Hints

Here are some helpful hints

- Start early. The hardest part of this lab is understanding how it’s supposed to work
- Don’t implement too much at once. Start at a high level and implement a little at a time. Hard code values and gradually replace them with the appropriate values
- Don’t spend more than an hour on any one issue. Take a break, post on Piazza or work on another feature
- Commit your code regularly to the SVN repository
- Auto-indentation is your friend and will help you spot mistakes