CS 0449 – A Shell

Due Date: Wednesday, December 13, 2017, 11:59 PM

The Basics

In this assignment you are to implement a Unix shell program. A shell is simply a program that conveniently allows you to run other programs. Read up on your favorite shell (such as "bash") to see what it does.

The requirements for completing this assignment successfully are described below.

The Input

A shell, at its simplest, is a program that reads input from the user and tries to do the commands. The commands for our shell will be of one of two categories:

1.) Internal commands the shell knows how to do itself, and

2.) Other programs on the system to execute.

Your shell should read in a line of input using the `fgets()` command. You then need to tokenize it and interpret it according to the features we are supporting described in the section below.

To tokenize the input, use the C standard library function `strtok()`. This function behaves oddly, so make sure you read about the right way to invoke it. Our delimiter set will be all whitespace characters (space, tab, newline) and the characters:

```
( ) < > | & ;
```

We won't be using them all in this assignment, but they are all useful in the real UNIX/Linux shell.

The Details

Your shell must support the following:

1. The internal shell command "`exit`" which terminates the shell.
   Example: `exit`
   Concepts: shell commands, exiting the shell
   System calls: `exit()`

2. The internal shell command "`cd`" which changes the present working directory
   Example: `cd private`
   Details: This command takes a relative or absolute path and changes the present working directory to that path
   Concepts: present working directory, absolute and relative paths
   System calls: `chdir()`

3. Any UNIX command, with or without arguments
   Example commands: `ls, pico, pwd, ls –l, wc –l, ps -a`
   Details: Your shell must block until the command completes and, if the return code is abnormal, print out a message to that effect. Argument 0 is the name of the command
   Concepts: Forking a child process, waiting for it to complete, synchronous execution, Command-line parameters
   System calls: `fork(), execvp(), exit(), wait()`

4. Any UNIX 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()`

5. Any UNIX command, with or without arguments, whose output is appended to a file
   Example: `ls -l >> foo`
   Details: This takes the output of the command and appends it to the named file
   Concepts: File operations, output redirection
   System calls: `freopen()`

6. Any UNIX command, with or without arguments, whose input is redirected from a file
   Example: `more < foo`
   Details: This takes the named file and makes it the standard input of the program
   Concepts: File operations, input redirection
   System calls: `freopen()`

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.

Please note that the UNIX commands in the list above are *examples*. Do not hard code support for them in. If it is not a built-in shell command, you should be attempting to run whatever program the user has typed as the first word on the commandline. For example, `ls –l` should work just as well as `ls –a`, `pico`, `cat`, `pwd`, etc.

Requirements and Hints

- You must read in input using `fgets()`.
- You must tokenize the input using `strtok()`.
- You do not need any special environment to run or execute this program (beyond using thoth as always). A shell can run another shell inside of it. Her is an example run of the shell within which we execute pwd, pwd with output redirection, and then exit:

```
thoth $ ./myshell
myshell $ pwd
/afs/pitt.edu/home/w/a/wahn
myshell $ pwd > pwd.txt
myshell $ exit
thoth $ cat pwd.txt
/afs/pitt.edu/home/w/a/wahn
```

- You must synchronize with the child process when running a command such that the shell waits for the command to complete before printing the next prompt. Use the wait() system call for this purpose. Try doing 'man 2 wait' on thoth for help (the '2' refers to the section of the manpages dedicated to system calls).

Submission

You need to submit:

- Your `myshell.c` file

Make a tar.gz file named `USERNAME-project5.tar.gz`

Copy it to `~wahn/submit/449/RECITATION_CLASS_NUMBER` by the deadline for credit.

Remember this is an extra credit project so completion is optional.