CREATING YOUR OWN SHELL

OVERVIEW

This assignment will expand the functionality of the first assignment to create a simplified shell as described below. In completing this assignment, you will explore the use of processes in a programming environment. Your shell will not contain all the functionality of a production shell, but will be sufficient to demonstrate the basics in process creation, destruction, and synchronization.

THE PROGRAM

You are to create a program in ANSI C named *myshell* that implements a basic shell program. The functionality of your shell program is similar to that of a production shell such as a c-shell or a bourne shell found in many operating systems. Input from a command line is read, interpreted, and executed. Input/Output redirection as well as foreground/background execution must also be implemented. However, interprocess-communication will not need to be implemented for this project.

Your program is not required to be "bulletproof". You can assume that input data will follow the description given in class. You may NOT assume that input data is valid. That is, the command may be misspelled. For example, I could test your code by entering the command *cqt file* instead of *cat file*. Your shell should provide a short, descriptive and interesting message explaining the error that has occurred. Your shell must continue processing commands until the exit command is typed. Before terminating execution, your shell must make certain that all background child processes have exited. In other words, the parent cannot terminate until <u>all</u> children have terminated.

For testing your shell, you need to download the *slow* program from *eLearning* in the Content area for Processes. The slow process will allow you to test if you handled the execution of background processes properly.

SAMPLE TEST CASES

The following describes a list of test cases as a starting point for testing your work. Your program must pass at least these tests for you to receive full credit. Additional test cases may be used during grading.

Command	**Explanation**
`ls –l`	shows a listing of files in the current directory
`ls –l >testfile.txt`	writes a listing of files into the text file *testfile.txt*
`grep –i shell testfile.txt`	list many lines containing the word shell in the previous file
`cat <myshell.c`	displays the source code of the program on the screen
`cat <myshell.c &`	as above except the output will be displayed in the background causing the prompt of the shell to be mixed with the output of the file
`cat testfile.txt &`	displays the content of the text file *testfile.txt* on the screen in the background
`slow &`	runs the program slow introduced in class in the background
`exit`	terminates the shell; all child processes must be terminated for the shell to close avoiding creation of zombie processes

Suggested System Calls

Most implementation details are left for you to decide. Below is a list of system calls that you need to use in the implementation of your program. You may use additional system calls as necessary. You <u>may not</u> use the `system(3)` function, nor use assistance from a production shell program to do any of the work for you.

- `exec(2)` – (`execl()` or `execlp()` or `execv()`, …) execute a file
- `fork(2)` – create a new process (exactly like the parent process)
- `freopen(3C)` – opens a stream (change an open stream to some other file)
- `waitpid(2)` or `wait(2)` – wait for a child process to change state (or terminate)

Deliverables

Your project submission should follow the instructions below. Any submissions that do not follow the stated requirements will not be graded.

1. Follow the submission requirements of your instructor as published on *eLearning* under the Content area.
2. You should have at a minimum the following files for this assignment:
   a. myshell.c
   b. parse.c (from the previous project)
   c. parse.h (defines the functionality you want to expose)
   d. Makefile
   e. README

The README file describes purpose and usage of your program. The file `parse.c` provides the parse functionality implemented for the previous project to parse the input string into the given struture. It is up to you to refactor your code so that the main function is in `myshell.c`. Keep in mind that documentation of source code is an essential part of programming. If you do not include comments in your source code, points will be deducted.

Your program will be evaluated according to the steps shown below. Notice that warnings and errors are not permitted and will make grading quick!

1. Program compilation with Makefile. The options –*g* and –*Wall* must be enabled in the Makefile. See the sample Makefile that I uploaded in *eLearning*.
   - If errors or warnings occur, no points will be given for the assignment.
2. Perform several evaluation runs with input of the grader's own choosing. At a minimum, the test runs address the following questions.
   - Are commands created correctly using `fork()` and `exec()`?
   - Is input/output redirection correctly implemented?
   - Is foreground/background processing correctly implemented?
   - Does the program terminate correctly when exit is entered?
   - Does the shell program wait for all children to terminate before it terminates?

GRADING

This project is worth 100 points total. The points will be given based on the following criteria

- 60 points for correct implementation of code (runs and all functionalities are implemented),
- 10 points for README and Makefile,
- 10 points for appropriate documentation of source code,
- 15 points for program runs,
- 5 points for correct submission format.

DUE DATE

The project is due September 25th by 3:00 pm to the Dropbox for project 2 in *eLearning*. I will not accept submissions emailed to me or the grader. Upload ahead of time, as last minute uploads may fail.

COMMENTS

It would be wise to start working on this project right away to leave enough time for handling unexpected problems. Insert lots of output statements to help debug your program. You should consider using debugging techniques from the first programming practice. Compile with the `–DDEBUG=1` option until you are confident your program works as expected. Then recompile without using the `–D` option to “turn off” the debugging output statements.