ICS 53, Spring 2016

Lab 2: A Simple Shell

A *shell* is a mechanism with which an interactive user can send commands to the OS and by which the OS can respond to the user. The OS assumes a simple character-oriented interface in which the user types a string of characters (terminated by pressing the Enter or Return key) and the OS responds by typing lines of characters back to the screen. The character-oriented shell assumes a screen display with a fixed number of lines (say 25) and a fixed number of characters (say 80) per line.

Typical Shell Interaction

The shell executes the following basic steps in a loop.

1. The shell prints a prompt to indicate that it is waiting for instructions.

```
prompt>
```

2. The user types a command, terminated with an <ENTER> character (‘\n’). All commands are of the form `COMMAND [arg1] [arg2] … [argn].`

```
prompt> ls
```

3. The shell executes the chosen command and passes the command the arguments. The command prints results to the screen. Typical printed output for an ls command is shown below.

```
hello.c hello testprog.c testprog
```

There are two types of commands, built-in commands which are performed directly by the shell, and general commands which indicate compiled programs which the shell should cause to be executed. You will support only one built-in command, quit, which ends the shell process. General commands can indicate any compiled executable. We will assume that any compiled executable used as a general command must exist in the current directory. The general command typed at the shell prompt is the name of the compiled executable, just like it would be for a normal shell. For example, to execute an executable called hello the user would type the following at the prompt:

```
prompt> hello
```

Built-in commands are to be executed directly by the shell process and general commands should be executed in a child process which is spawned by the shell process using a fork command. Be sure to reap all terminated child processes (discussed in Section 8.4.3 of the book).

General commands can be executed either in the foreground or in the background. When a user wants a command to be executed in the background, an “&” character is added to the end of the command line, before the <ENTER> character. The built-in command is always executed in the foreground. When a command is executed in the foreground, the shell process must wait for the child process to complete.

Your shell does not need to support I/O redirection.

Submission Instructions

There will be a dropbox on EEE which you will use to submit your code. Only one group member should submit a solution to the dropbox. You should submit a single C source code file. The code should compile and execute on the openlab machines. The name of your C source code file should be in the form “Lab2_ID1_ID2_ID3.c”, where “IDx” is replaced with the student ID of each group member. The first lines of the C source file should contain the names and student ID numbers of each group member. Each of the first lines should like this “lastname, firstname: studentid”. The name and student ID of each student should be on a separate line.

Handling Unusual Inputs

- Input and command line format errors should not cause your shell to crash
- Your shell should be able to handle the following inputs without errors: blank lines and extra whitespaces
- When in doubt, your code should produce the same output as the **tcsh** shell.