ICS 53, Spring 2020

Assignment 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

I/O redirection
Most command line programs that display their results do so by sending their results to standard output (display). However, before a command is executed, its input and output may be redirected using a special notation interpreted by the shell.

> **To redirect standard output** to a file, the ">" character is used like this:
>
> ```
> prompt> ls > file_list.txt
> ```
>
> In this example, the ls command is executed and the results are written in a file named file_list.txt. Since the output of ls was redirected to the file, no results appear on the display. Each time the command above is repeated, file_list.txt is overwritten from the beginning with the output of the command ls.
>
> **To redirect standard input** from a file instead of the keyboard, the "<" character is used like this:
>
> ```
> prompt> sort < file_list.txt
> ```
>
> In the example above, we used the sort command to process the contents of file_list.txt. The results are output on the display since the standard output was not redirected.
>
> We could redirect standard output to another file like this:
>
> ```
> prompt> sort < file_list.txt > sorted_file_list.txt
> ```

Background commands
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.

Submission Instructions
Your source code should be a single c file named ‘hw2.c’. Submissions will be done through Gradescope. You have already been added on the Gradescope course for ICS 53. Please login to with your school (UCI) email to access it.