Assignment # 1: A Simple Shell[1]

V 1.1 10/07/2013

COMP3301/COMP7308: Operating Systems Architecture

Due Friday 17 August, 2013, 8pm. (End of week 4)

This assignment asks you to write your own shell, using the fork, exec, wait, dup2, and pipe system calls. The purpose is to get you familiar with Unix processes, creating them, passing arguments, inheriting open files, waiting for them to terminate, background processes, and so on. You will also be introduced to pipes as well. You will write your shell in C.
A test script will be provided by end of week 2. Instructions on how to run the test will be included at the top of the file.

Submission will be via a tarball that includes a makefile.

1 Overview

In this assignment you will write your own shell, called qshell. This will work like the Bash shell you are used to using, prompting for a command line and running commands, but it will not have many of the special features of the Bash shell.

- Your shell will support:
  - the redirection of standard input
  - the redirection of standard output
  - use of | (an unnamed pipe)
  - foreground processes
  - background processes
  - comments (lines beginning with the # character).
- and the following two built in commands:
  - exit
  - cd

2 Specifications

Here are the specifications for your assignment. There should be no guess work on what is required for this assignment. If you have any questions, check the Discussion Board on Blackboard. If your questions aren't answered there, post your question there.

[1] This assignment is based on a similar assignment offered in the course CS311 at Oregan State University, USA: classes.engr.oregonstate.edu/eecs/winter2011/cs311/Winter11Projects/Proj4/assign4.doc
and on many other similar "write a shell" assignments.

2.1 Command Line

- The general syntax of a command line is

```
command [arg1 arg2 ...] [< input_file] [> output_file] [&]
      or
command [arg1 arg2 ...] [> output_file] [< input_file] [&]
      or
command [arg1 arg2 ...] [< input_file] | command [arg1 arg2 ...] [> output_file]
```

  where items in square brackets are optional.
- You can assume that a command is made up of words separated by spaces.
- The special symbols |, <, >, and & are recognized, but they must be surrounded by spaces like other words.
- You can assume that < will appear at most 1 time in the command. You can assume the > will appear at most 1 time in the command. You can assume that | will appear at most 1 time in the command. However, they may appear together in the same command. For example,
  cat myFile | wc > otherFile
  wc -l < myFile > otherFile
  sort < myFile | uniq > otherFile
- The command on the left side of a pipe will not also have output redirection.
- The command on the right side of a pipe will not also have input redirection.
- If the command is to be executed in the background, the last word must be &. As the above general syntax indicates, you do not have to worry about piped commands running in the background.
- If standard input or output is to be redirected, the < or > words followed by a filename word must appear after all the arguments.
- Input redirection can appear before or after output redirection.
- Your shell does not need to support any quoting; so arguments with spaces inside them are not required to be supported.
- Valid command lines may be up to 128 characters, including newline, with at most 20 arguments.
- Use the current working directory followed by a question mark as a prompt for each command line.

2.2 Command Execution

- For executing commands use:
  - fork()
    to start new processes
  - dup2()
    to duplicate file descriptors
  - pipe()
    to create a unnamed pipe
  - exec() family
    to execute new programs
  - waitpid()

You will need to use waitpid to check for completed background processes. Read about the WNOHANG option of waitpid. It might be useful.
    - Your shell should use the PATH variable to look for commands, and it should allow shell scripts to be executed.
      The right version of the exec function will do this for you automatically.
    - If a command fails because the shell could not find the command to run, then the shell will print an error message. This is done for you automatically with the right exec command.
- Background and foreground commands
    - Your shell will wait for the completion of foreground commands (commands without the & in the final position) before prompting for a new command.
    - Your shell will NOT wait for commands run in the background. Instead it will prompt for a new command without waiting for background commands to complete.
    - Background commands should have their standard input redirected to be from /dev/null if the user did not specify some other file to take standard input from.
- Input redirection
  After the fork but before the exec you must do any input/output redirection.
    - A redirected input file should be opened for reading only
    - If your shell cannot open the file for reading it should print an error message.
- Output redirection
  After the fork but before the exec you must do any input/output redirection.
    - A redirected output file should be opened for write only,
    - The output file should be truncated if it already exists or created if it does not exist.
    - If your shell cannot open the output file it should print an error message.
- Piping
  Create the pipe before the fork, then do input / output redirection, then exec.
    - A redirected input file should be opened for reading only
    - If your shell cannot open the file for reading it should print an error message.
    - A redirected output file should be opened for write only,
    - It should be truncated if it already exists or created if it does not exist.
    - If your shell cannot open the output file it should print an error message.

2.3 General Behavior

- control-C interrupt from the keyboard does NOT terminate your shell,
- control-C interrupt from the keyboard does terminate the foreground command currently running.
- Background commands should not be terminated by a control-C signal.
- Your shell should allow blank lines and comments.
- Any line that begins with the # character is a comment line and should be ignored.
  #This is a comment.
  # Another comment.
- A blank line (one without any commands) should do nothing;
  your shell should just output a prompt for another command (like Bash does).

- The shell will print the process id of a background process when it begins.
- When a background process terminates, a message showing the process id and exit status will be printed.
- You should check to see if any background processes completed just before you prompt for a new command and print the message then. Note: In this way the messages about completed background processes will not appear during other running commands, though the user will have to wait until they complete some other command to see these messages (this is the way the C shell and Bourne shells work).

2.4 Built in Commands

- Your shell will support two built in commands: exit and cd.
- You do not have to support input/output redirection, running in the background, or pipes for these built in commands.
- Built-in command specifications
  - *exit*

    `exit`
    - The exit command exits the shell.
    - It takes no arguments.
    - If arguments are supplied, then they are ignored.
  - *cd*

    `cd [dir]`
    - Purpose: The cd command changes directories.
    - when called with no arguments, it changes to the directory specified in the *HOME* environment variable.
      You can get your process' value of HOME with getenv("HOME").
      You can change directories with chdir().
    - It can also take one argument, the path of the directory to change to.
    - If it is passed more than one argument, the extra arguments are ignored.
      For example:
      `cd dir1 dir2`
      Will have the same effect as:
      `cd dir1`

2.5 'man' page

As documentation for your qshell, you need to produce a "man" page for your shell, with a maximum of 10 kbytes of characters. Any lack of functionality can be explained in the "bugs" section.

3 Submission

The shell should be submitted as a gzipped tarball, <studentnumber>.tar.gz

It should include a makefile which produces a "qshell" executable in the current working directory. You should "make clean" before producing the tarball.

I still need to confirm the best way to submit a tarball using Blackboard.

You should use a version control system, such as subversion, to develop your program, but submission is only of the finished product.

You should confirm that your code runs on *moss.labs.eait.uq.edu.au*, because this is where it will be tested.

4 Marking Scheme

- WARNING: If your code doesn't work, you won't get credit for it.
  1. For this assignment, break the assignment into pieces that you can work on independently.
  2. Get one piece fully functional and thoroughly tested.
  3. Then go on to the next piece.
  4. If you've tried to implement everything but haven't completed any pieces, you won't get credit for anything.

Individual functions will be checked, tested and marked.
For each function, percentages of marks will be allocated as
0-20% Function not implemented, or no working code.
20-40% Some simple but incorrect functionality
40-60% Significant problems, such as frequent crashes or infinite loops
60-80% Moderate problems, fails relatively often.
80-100% Few or no problems.

Total marks out of 100:

Makefile works, program which addresses the specifications compiles [10]
Coding style, readability, organization [10]
Foreground Execution [10]
Background Execution [10]
Built-in Command exit [10]
Built-in Command cd [10]
Input redirection [10]
Output redirection [10]
Pipes [10]
Man page [10]

5 A Simple Test File

```
#!/bin/sh

# The following is a here document.
# It takes everything between the ___EOF___ as input to qshell.
# This should run just as if you typed each command in at the prompt.
./qshell <<___EOF___
echo No Arguments Command Test
echo ps
ps
echo Command with Arguments Test
echo ls -l
ls -l
echo Output Redirection Test
echo  date > myFile
date > myFile
echo Input Redirection Test
echo wc < myFile
wc < myFile
echo Comment Test
# Does myShell treat this as a comment?
echo Blank Line Test



exit
___EOF___
# here document done
```

6 Revision History

Version 1.1, 10/7/2013 – Draft for Comment.