Creating a Shell Interface in C/C++

In this project you are going to create a simple shell interface in C/C++. This project is intended to teach you the basics of process creation, waiting for processes and process termination. Your program will continually read inputs from the keyboard, parse the input, check whether the input is a valid Unix command and execute the command in foreground or background (as specified by the user – with our without '&' at the end), and print out the results to standard output until the user enters "exit".

Specifically, your program should to the following.

1. Display the prompt toysh >
2. Read user input
3. Parse user input to obtain the command, options and arguments. You should also detect whether the user wants the command to be executed in foreground or background.
4. Check whether it is a valid command (i.e., it exists in the PATH). If not an error message needs to be printed (e.g., 'Unknown Command').
5. Irrespective of whether the command is executed in foreground or background, the command should be executed in a separate process.
6. If the command is to be executed in background, the program should immediately print the prompt and be ready to accept more commands.
7. If the command is executed in foreground, the program should wait for the command to complete and then print the prompt.
8. The above steps should be continually executed until the user enters "exit" (without quotes).

Points to be noted:

1. You can assume that user provides valid options and arguments. However, your program should still flag non-valid Unix commands. In other words, if the command is valid, you can assume that the arguments and options are valid.
2. You may also assume that the user inputs only one command at a time (i.e., you need not support commands such as "cat text.txt|grep dog").
3. Your program should be able to execute any number of commands in the background and at most one command in the foreground simultaneously.
4. You need not support changing directories.
5. You need not support input/output redirection.
6. Make sure your program works on Linux machines.

7. You are not required to handle signals (ctrl-c, ctrl-z, etc.).
8. Your program does not need to support the “kill”, “fg” or the “bg” commands.
9. Your program should support the “jobs” command. This command should print out the process ids of the processes that are *currently* running in the background.

What to Submit:

A Zip folder containing:

1. All source files (including any .h files).
2. makefile
3. Readme file containing: (1) Project partner information; (2) Any special program execution instructions

Submissions to be done via ELC