CS-550 Section 01 Spring 2020 Assignment 01 : I/O Redirection and Processes in Shell

Goals

This assignment teaches about managing processes, inter-process communication using pipelines and I/O redirection, and how a UNIX shell works.

[Description] [Restrictions] [Hints] Grading Guidelines

Description

Write a program in C called **mysh** which acts like a shell. Your shell should contain a loop that:

1. Writes a prompt to standard output of the form **mysh>**
2. Reads a command from standard input, terminated by a newline (enter on the keyboard)
3. Execute the command it just read.

Since these commands may not function cleanly, mysh must run each command in its own process, and wait for that process to end.

The loop should be terminated when end-of-file is reached on standard input, or when the single word "exit" is typed on the command line. (You can type an "end-of-file" character to stdin using Ctrl-D.)

For instance, here is an example of the output from **mysh**:

```
~>mysh
mysh> cat input.txt
This is text inside input.txt
It may go on for several lines
mysh>
```

The commands themselves consist of any valid UNIX command, including parameters to that command. These commands must also support any valid combination of the following special features:

1. **Input Redirection** - If the command is followed by a less than sign (<) and a path and/or file name, such as **< input.txt**, then the file input.txt should become the command's standard input stream. For example:

   ```
   mysh> grep text < input.txt
   This is text inside input.txt
   mysh>
   ```

2. **Output Redirection** - If the command is followed by a greather than sign (>) and a path and/or file name, such as **> output.txt**, then the command's output stream should be redirected to the file output.txt. For example:

   ```
   mysh> grep text input.txt > output.txt
   mysh>
   ```

   After this command, output.txt will contain "This is text inside input.txt".

3. **Pipes** - If a command is followed by a vertical bar (|), and a second command, then the standard output stream of the first command should be connected to the standard input stream of the second command. **mysh** must be able to support an arbitrary number of pipes. For example:

```
mysh> cat input.txt | grep text | wc -w
5
mysh>
```

Any valid combination of input redirection, output redirection, or pipelining should be supported. Not all combinations are valid. For instance, a command on the left side of a pipe cannot use output redirection. **mysh** should detect these invalid combinations, and write error messages to indicate that there is a problem.

Restrictions

- **Do not** use any special wrapper libraries or classes to borrow the basic functionality required in this assignment. If in doubt, ask the instructor **before** using any libraries other than the C standard libraries.
- You may **not** use the **system(...)** syscall or library function to execute your commands.
- **Do not** write five or six different programs, one for each feature. Write one **single** program which supports all of the required features.

Hints

- Implement one feature at a time. Start simple - test it, get it working, then add the next feature.
- When you have an intermediate version that is working, make a back up copy of your code so if you irreperably screw up the next feature, you can go back to the previous working version of the code and start again.
- Before you start coding, figure out what data structures you will need, and design the overall structure of your code. Try to anticipate problems you will face.
- Research the following C standard library functions... these will be useful in implementing mysh:
  - fork()
  - execv(), execl(), execlp(), execvp() [which one should you use?]
  - waitpid() or wait()
  - dup2() [for standard input and output redirection]
  - pipe()
  - open()
  - close()
- I have provided functions to read the command line from standard input, and to parse that command line into white-space delimited tokens. You may use this code in your shell program, or may modify this code to meet your needs. The header file for the code I am providing is in cmdlLine.h, and the code itself is in cmdLine.c.
- One way of minimizing overlapping standard output messages from different processes is to turn off output buffering for stnard output. You can do this by putting the C code "setbuf(stdout,0);" at the top of your code.
- One way of handling an arbitrary number of pipeline stages is to use recursion.

Grading Guidelines

- We will download and build mysh using your Makefile on a CS LDAP machine. If it doesn't build, 0 points.
- If mysh runs simple commands without redirection or pipes, 60 points.

- If mysh also supports redirection, but not pipes; or supports pipes but not redirection, 75 points.
- If mysh supports redirection and pipes, but fails to identify incorrect combinations of redirection and pipes, 90 points.
- If mysh supports redirection and pipes, and correctly identifies invalid combinations, 100 points.
- If you are not able to demonstrate your code or if you are not able to explain how any of its features work, 0 points.