Write your own shell in C

The **shell** is a command-line interpreter that reads user input and execute commands. Write a shell-like program . This program will provide its own prompt to the user, read the command from keyboard input, execute the command, and then loop back to get another command.

Note: You may *not* use the `system` library calls. Rather, you must use system calls such as `fork`, and the `exec`-family.

Part 1:

You can use the following scheme as a work plan:

1. If a command is not found, you should give an error message complaining that shell cannot understands your command.
2. For the first part, it is sufficient to only handle ``argument-less'' command invocations, such as ``ls'' and ``date''.
3. Make the mini-shell (from the previous part) more powerful by allowing (an arbitrary number, zero or more) arguments to the commands. For example, it should be able to execute commands such as ``more filename'', ``ls -l /tmp'', ``gcc -o testing hello.c'' etc.
4. The shell should scan the input line for a greater-than-sign (>) and *if one exists,* treat the single word after it as a filename. Any output that the command (the part before the >) generates to standard output, must be written to the file, rather than displayed on the screen. If the file already exists, it should be truncated and overwritten. If the file cannot be created, an error must be generated, and the command not executed. For example: ``ls>outputfile''.

   (Note: any string after the filename is illegal, and the shell should detect that, and print an error message instead of executing anything. For example: ``ls>outputfile foo'')
5. Extend your mini-shell to allow it to provide some common built in commands:
   - `cd` *`path`* : Change directory to specified *path*. Implement using `chdir(2)`.
   - `pwd` : Print the current directory.
   - `echo` *`rest of line`* : Display the rest of the line.
   - `exit` : Exit the mini-shell.

A working implementation of the above sub-assignments will earn 20/30. Additional 5 marks can be earned by:

Part 2:

Implementing extra features:

- Command history
- File completion
- Command completion

Rest of the 5 marks can be earned by doing the following:

PART 3:

Implement **closh** (Clone Shell), a simple shell-like program designed to run multiple copies of a program at once.
Like any other shell, closh takes as input the name of the program to run (e.g., hostname, pwd, echo, etc.). However, closh also takes a additional input: The number of copies (processes) of the program to run. This is an integer from 1 to 9.

Closh executes the given program the specified number of times, then returns to the prompt once all processes have completed. Here is a simple example of using closh:

rishi@rishi26$ ./closh
closh> hostname
count> 3
rishi26
rishi26
rishi26
closh>

Lab Instructions:

1. You are required to submit two .c codes:
    1. **myshell.c:** Part 1 and Part 2
    2. **closh.c:** Part 3
2. One .pdf file containing documentation about the basic functionality of your shell. It should include what all features your shell provide and a little description how to operate your shell.
3. In the above .pdf file, make a section **Work completed.** This section should tell, which parts, you have implemented. Also make a note in this section, which part or subpart, you have not implemented.
4. Put all files in a folder named as Name_EntryNumber. Compress the folder and upload compressed file on the moodle before deadline.