Assignment: Mini Shell

Part A

This assignment helps you learn about processes and basic process management in a shell. You are asked to write a simple shell program called `minish`. This shell must work as follows. When you start your mini shell, it prints the following prompt and waits for a command from user.

```
minish>
```

From here onwards, you should be able to execute and control **any binary executable** just as you would in a normal shell. For instance

```
minish> Command

[ OUTPUT OF COMMAND SHOWN HERE ]

minish>
```

Additionally, your shell should be able to do the following:

1. Execute commands with multiple arguments. For example:

   ```
   minish> Command arg1 arg2 arg3
   ```

2. Execute commands in either foreground or background mode. In foreground mode, the shell just waits for the command to complete before displaying the shell prompt again (as in the above example). For background mode, a command (say Command 1) is executed with an ampersand `&` suffix. The shell prints the process ID of the backgrounded process and immediately prints the prompt to accept and execute the next command (say Command2), even as Command1 continues executing in the background. For example:

   ```
   minish> Command1 &
   Process print_process_id_1_here in background mode
   minish> Command2

   [OUTPUT OF Command1 AND Command2 MAY INTERLEAVE IN ARBITRARY ORDER]
   ```

3. Maintain multiple processes running in background mode simultaneously. For example:

   ```
   minish> Command1 &
   Process print_process_id_1_here in background mode
   minish> Command2 &
   Process print_process_id_2_here in background mode
   minish> Command3 &
   Process print_process_id_3_here in background mode
   minish>

   [OUTPUT OF Command1, Command2, AND Command3 MAY INTERLEAVE IN ARBITRARY ORDER]
   ```

4. Redirect the input of a command from a file. For example:

   ```
   minish> Command < input_file
   ```

   Redirect the output of a command to a file. For example:

   ```
   minish> Command > output_file
   ```

Part B

5. Implement command filters, i.e., redirect the stdout of one command to stdin of another using pipes. For example:

   ```
   minish> ls -l | wc -l


   minish> cat somefile | grep somestring | less
   ```

Ideally, your shell should be able to handle any number of filter components.

6. Terminate a foreground process by pressing `[Ctrl-C]`. Your shell must not get killed; only the process running in foreground mode must terminate. If executing a chain of filters (as in the above example), all processes in the filter chain must terminate.
7. Kill a process in background using the `kill` command.

```
minish> kill give_process_id_here
```

8. Be able to execute any feasible combinations of filters and file I/O redirection.
9. The `exit` command should terminate your shell. Clean up any orphan processes as your shell exits.

Do Nots:

- DO NOT use any special wrapper libraries or classes to borrow the basic functionality required in this assignment. If in doubt, ask the instructor first BEFORE doing so.
- DO NOT use the **system(...)** syscall to execute the programs in your shell directly.
- DO NOT write five or six different programs, one for each feature. Write **one single program** that includes all the above features.
- You do not need to implement any scripting functionalities.
- You do not need to implement wild character expansions such as *,?,[0-9], etc.

Hints:

- Build and test one featur at a time.
- Use Github to save and track changes to your code. Make backup copies of partially working versions of your code. This way, if you irreparably mess up your code, then you can at least roll back to your last backup copy.
- First design your data structures and code-structure before you begin coding each feature. Anticipate specific problems you will face.
- Check out man page for the following, which might be useful:
  - fork()
  - execv() and execl() (which one should you use?)
  - waitpid()
  - dup2() (for stdin/stdout redirection)
  - pipe()
  - open()
  - close()
  - fileno()
  - kill()
  - killpg()
  - setsid()
  - getgrp()
  - getpgid()
  - tcsetpgrp()

Grading Guidelines

```
This is how we will grade your assignment during the demo. So please prioritize your work accordingly.

 5 - README, Makefile, Compilation without errors

 5 - Executing a command with no arguments in foreground

 5 - Executing a command with multiple arguments in foreground

 5 - Executing a single command in background

10 - Executing multiple commands in background simultaneously

10 - Input redirection from file: Executing a single command that takes standard input (stdin) from a file

10 - Output redirection to file: Executing a single command that sends standard output (stdout) to a file

10 - Terminating a foreground process using Ctrl+C without killing the shell

10 - Terminate a background process using kill command

30 - Filters
        10 - Be able to execute filter chain with two components
        20 - Be able to execute filter chain of arbitrary length
```

```
20 - Combinations
        10 - Executing a filter chain while redirecting input of the first command and/or output of the last command
        10 - Terminating a chain of filters

10 - Error Handling
        For example, you should handle all errors returned by ALL systems calls used in your
        program. Also check for other common error conditions in your program.
        But don't go overboard with error checking.
        We will not try to intentionally break your code with bad
        input that may be irrelevant to the assignment's goals.

Total = 130
```