Shell: Signals, Filters and I/O redirection

Description

Extend the basic shell that you wrote in Assignment 1 to implement the following additional features.

- **I/O Redirection:** Redirect the input of a command from a file. For example:

```
sh550> Command < input_file
```

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

```
sh550> Command > output_file
```

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

```
sh550> ls -l | wc -l

sh550> cat somefile | grep somestring | less
```

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

- **Foreground Process Termination:** 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.
- **Background Process Termination:** Terminate a process in background using the `kill` command.

```
sh550> kill give_process_id_here
```

- **Combinations:** Be able to execute any feasible combinations of filters and file I/O redirection.

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.

Hints:

- Here is a sample code to get you started. It reads a line of input, breaks it up into individual tokens, and then waits for next line of input. Typing "exit" terminates the shell.
- Build and test one functionality at a time.
- 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:
  - fork()

- execv(), execl(), execlp(), execvp() (which one should you use?)
- waitpid()
- dup2() (for stdin/stdout redirection)
- pipe()
- open()
- close()
- fileno()
- kill()
- killpg()
- setsid()
- getgrp()
- getpgid()
- tcsetpgrp()
- sigaction()
- signal()

Grading Guidelines

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

- 10 - README, Makefile, Compilation without errors or warnings
- 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: Most important part here is to make sure that you check and handle the 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's irrelevant to the assignment's goals.
- Total score = 110
- During the demo, the TA will ask questions about your code. To get full credit, you need to be able to fully understand and explain your code, besides showing that the code works.