Part D: Handling the `Ctrl+C` signal

Up until now, the `Ctrl+C` command would have caused your shell (and all its children) to terminate. Now, you will modify your shell so that the signal SIGINT does not terminate the shell itself, but only terminates the foreground process it is running. Note that the background processes should remain unaffected by the SIGINT, and must only terminate on the `exit` command. You will accomplish this functionality by writing custom signal handling code in the shell, that catches the `Ctrl+C` signal and relays it to the relevant processes, without terminating itself.

Note that, by default, any signal like SIGINT will be delivered to the shell and all its children. To solve this part correctly, one of the approaches is to carefully place the various children of the shell in different process groups, say, using the `setpgid(pid, pgid)` system call. For example, `setpgid(0,0)` places a process in its own separate process group, that is different from the default process group of its parent. Your shell must do some such manipulation on the process group of its children to ensure that only the foreground child receives the `Ctrl+C` signal, and the background children in a separate process group do not get killed by the `Ctrl+C` signal immediately.

Once again, use long running commands like sleep to test your implementation of `Ctrl+C`. You may start multiple long running background processes, then start a foreground command, hit `Ctrl+C`, and check that only the foreground process is terminated and none of the background processes are terminated.

Part E: Serial and parallel foreground execution

Now, we will extend the shell to support the execution of multiple commands in the foreground, as described below.

- Multiple user commands separated by `&&` should be executed one after the other serially in the foreground. The shell must move on to the next command in the sequence only after the previous one has completed (successfully, or with errors) and the corresponding terminated child reaped by the parent. The shell should return to the command prompt after all the commands in the sequence have finished execution.
- Multiple commands separated by `&&&` should be executed in parallel in the foreground. That is, the shell should start execution of all commands simultaneously, and return to command prompt after all commands have finished execution and all terminated children reaped correctly.

Like in the previous parts of the assignment, you may assume that the commands entered for serial or parallel execution are simple Linux commands, and the user enters only one type of command (serial or parallel) at a time on the command prompt. You may also assume that there are spaces on either side of the special tokens `&&` and `&&&`. You may assume that there are no more than 64 foreground commands given at a time. Once again, use multiple long running commands like `sleep` to test your series and parallel implementations, as such commands will give you enough time to run `ps` in another window to check that the commands are executing as specified.

The handling of the `Ctrl+C` signal should terminate all foreground processes running in serial or parallel. When executing multiple commands in serial mode, the shell must terminate the current command, ignore all subsequent commands in the series, and return back to the command prompt. When in parallel mode, the shell must terminate all foreground commands and return to the command prompt.

Submission Instructions
You must submit a single shell code file as `my_shell.c`.

— End of Lab 05 —