BCIT

COMP 4736 **Lab 04: Building a Shell (Part B & C)**

Part B: Background Execution

Now, we will extend the shell to support background execution of processes. Extend your shell program of part A in the following manner: if a Linux command is followed by `&`, the command must be executed in the background. That is, the shell must start the execution of the command, and return to prompt the user for the next input, without waiting for the previous command to complete. The output of the command can get printed to the shell as and when it appears. A command not followed by `&` must simply execute in the foreground as before.

You can assume that the commands running in the background are simple Linux commands without pipes or redirections or any other special case handling like `cd`. You can assume that the user will enter only one foreground or background command at a time on the command prompt, and the command and `&` are separated by a space. You may assume that there are no more than 64 background commands executing at any given time. A helpful tip for testing: use long running commands like `sleep` to test your foreground and background implementations, as such commands will give you enough time to run `ps` in another window to check that the commands are executing as specified.

Across both background and foreground execution, ensure that the shell reaps all its children that have terminated. Unlike in the case of foreground execution, the background processes can be reaped with a time delay. For example, the shell may check for dead children periodically, say, when it obtains a new user input from the terminal. When the shell reaps a terminated background process, it must print a message `Shell: Background process finished` to let the user know that a background process has finished.

You must test your implementation for the cases where background and foreground processes are running together, and ensure that dead children are being reaped correctly in such cases. Recall that a generic `wait` system call can reap and return any dead child. So if you are waiting for a foreground process to terminate and invoke wait, it may reap and return a terminated background process. In that case, you must not erroneously return to the command prompt for the next command, but you must wait for the foreground command to terminate as well. To avoid such confusions, you may choose to use the `waitpid` variant of this system call, to be sure that you are reaping the correct foreground child. Once again, use long running commands like `sleep`, run `ps` in another window, and monitor the execution of your processes, to thoroughly test your shell with a combination of background and foreground processes. In particular, test that a background process finishing up in the middle of a foreground command execution will not cause your shell to incorrectly return to the command prompt before the foreground command finishes.

Part C: The `exit` command

Up until now, your shell executes in an infinite loop, and only the signal SIGINT (`Ctrl+C`) would have caused it to terminate. Now, you must implement the `exit` command that will cause the shell to terminate its infinite loop and exit. When the shell receives the `exit` command, it must terminate all background processes, say, by sending them a signal via the `kill` system call. Obviously, if the shell is receiving the command to exit, it goes without saying that it will not have any active foreground process running. Before exiting, the shell must also clean up any internal state (e.g., free dynamically allocated memory), and terminate in a clean manner.

Sample Testing Criteria

- Handles a single background process properly.
  - Runs process in background if command ends with "`&`".
  - Prints message when background process completes (can be at next user input).
  - Reaps the completed background process (can be at next user input).
- Handles multiple background processes properly.
  - Prints messages when background processes complete (can be at next user input).
  - Reaps completed background processes properly (can be at next user input).
- Handles concurrent foreground and background processes properly.
  - Reaps completed background processes.
  - If foreground process is running and a background process completes, the prompt is NOT returned until foreground process completes.
- Handles "`exit`" command properly.
  - Terminates all background processes.
  - Reaps all background processes.
  - Exits the shell.

Submission Instructions

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

— End of Lab 04 —