Operating Systems 0512.4402

Homework 1: Build a shell under Linux

We will build a simple shell in C under Linux, called `hw1shell`, to execute user commands. The commands `exit`, `cd` and `jobs` are **internal commands** which we'll implement in our shell. Any other command is an **external command** which we'll execute using one of the `exec` system calls. Example external commands are `ls -l`, `cat file.txt`, `echo 123`, `sleep 10`. There is no need to support redirection or pipes, nor multiple commands on the same line.

Implementation guidelines

1. The shell runs an infinite loop that writes the prompt ``hw1shell$ `` (hw1shell, followed by $, followed by space) to stdout, and accepts a user command from stdin.

2. If the command was `exit`, we terminate and exit. Before exiting, we must make sure to wait and reap all the child processes possibly running in the background (as well as do cleanups such as free dynamically allocated memory if any).

3. If the command was `cd`, we implement it locally inside our shell using the `chdir` system call (think why `cd` has to be an internal and not external command). The command `cd <directoryname>` must cause the shell process to change its working directory, `cd ..` should change to the parent directory, and other variants should display `hw1shell: invalid command` error message.

4. If the command was `jobs`, we should display a list of the current background processes. Each line should contain the `pid`, then a `tab`, then the background command entered by the user.

5. If the command ends with `&`, it's a background command, which is to be executed in the background in parallel to other commands. Otherwise, it's a foreground command.

6. We support a total of 4 background commands running simultaneously, in addition to a possible foreground command (so a total of 5 commands overall). If a new background command is received while there are already 4 running, we should dispay `hw1shell: too many background commands running` error message.

7. We use the `fork` system call to create a child process, and then in the child use one of the `exec` system call variants to execute the user command. *NOTE: We don't use the "`system`" library call to run commands, we do it manually using fork/exec/wait.*

8. If the command was a foreground command, in the father process we use the `waitpid` system call to reap the child process so it doesn't stay as a zombie after it finishes execution, and only then proceed in the loop to display the prompt and accept the next command.

9. Otherwise if the command was a background process, we display a line `hw1shell: pid %d started`, remember it and continue in the loop to accept additional commands. In the message, `%d` is replaced by the pid number.

10. An empty command (pressing enter) is not an error, just print the $ prompt again and continue.

11. If there is an error executing a command (for example, a command which does not exist), display an error `hw1shell: invalid command` and continue.

12. At the end of each iteration of the loop, the shell should reap any of the child background commands which were running in case they finished and are now in zombie state (just check, without waiting on purpose if they are still running). For any background command that we detected as finished, we should display a line `hw1shell: pid %d finished`, where `%d` is replaced by the pid number. Any background command which finished should free space in one of the four supported background commands, so the user can now enter new background commands in those slots.

13. If any system call fails with an error, we display a message ``hw1shell: %s failed, errno is %d'' and continue, where `%s` is replaced with the system call name, and `%d` is replaced with the contents of the global variable errno.

Submission guidelines

The solution should be submitted in moodle in a gzipped tar file called hw1_id1_id2.tgz, where id1 and id2 are the "tehudat zehut" of the two students (or hw1_id.tgz if submitting alone).

The tgz file should contain a subdirectory hw1_id1_id2, and in the subdirectory there should be a Makefile, as well as the .c and .h source files. Running "make" should build the code into an executable called "hw1shell". Running "make clean" should clean up the executables and object files.

You can build the tar file by running the command `tar -zcvf hw1_id1_id2.tgz hw1_id1_id2` from the directory above your hw1_id1_id2 subdirectory.

Make sure to have comments in your source code.