CS 333: Operating Systems Lab
Autumn 2018

Lab Quiz # 1, 20 marks

1. **beegees** (10 marks)
   Write a simple command line shell interface (`beegees.c`) for the following,

   - A list of (one or more) simple commands separated by "`&&`" should all be executed in the **background** and in **parallel**. For example,
     `sleep 100 && echo hi && ls -l` (three commands execute in the background)
     `sleep 50 &&` (one command executes in the background)

     The shell should start the execution of all the commands in parallel by creating child processes for each, and come back to the shell prompt for the next user command(s) without waiting for all the commands to finish.
   - For each command that completes in the background at a future time, the shell should print a message about the background process being completed. `Child <pid> terminated.`
     If a command execution fails, an appropriate error message ("<pid> error: <command> not found") should be printed on the terminal and should not affect other processes.
   - An `exit` command should cause the shell to terminate immediately, along with all executing background processes. The `kill` system call will be required for this.
     Assume that the `exit` call is always standalone and not in a chain of background process invocations.
   - A `SIGINT` signal at the prompt should not terminate the shell as well any background processes, and should wait for command(s) as usual. Read the `NOTES` section of `man 2 waitpid` for related information on this.
     An `<Enter>` key press without any command should simply display the next prompt for commands.
   - **Note:** Implementation of `cd` is not required. Assume all commands are simple Linux programs (possibly with their arguments). No command chaining with redirection or pipe operators. The tokenizer to parse the command line is provided.
   - **Extra credit (5 marks):**
     A single command (without `&&`) should execute as a foreground process. The prompt for further commands is displayed after completion of the foreground process. A `SIGINT` signal for the foreground process should terminate only this process, not the shell, and not other background processes. This requirement needs to be handled on normal foreground process exit/termination as well.
     This functionality will require manipulating process groups using the `setpgid` system call and careful usage of the `waitpid` call.

2. **program plumbing** (10 marks)

   - A shell supports execution of several commands chained together in different ways. This question is related to the use of the *pipe* operator, which connects the output of one command to the input of the next command. For example,
     ```
     $> ls | grep a.out
     $> ls -al | grep a.out
     $> ls | grep a | wc -l
     ```
   - Write a simple command line shell interface (`plumb.c`) that implements the pipe functionality of chaining commands. The connection between two commands should be implemented using the `pipe` system call.
   - Once all commands in the chain are done, the prompt should be displayed for the next command. If a command execution fails, an appropriate error message ("<pid> error: <command> not found") should be printed on the terminal and the pipe should continue.

- On typing `exit` at the prompt, the shell program should exit. Assume that the `exit` call is always standalone and not in a chain of background process invocations.
  An `<Enter>` key press without any command should simply display the next prompt for commands.
- **Note:** The question does **not** ask you to implement the full shell functionality, you are expected to only implement the pipe functionality connecting a series commands. The tokenizer to parse the command line is provided. Assume that not more than two pipes (or three commands) will be issued in one go and the each command may have arguments of its own.

  The **plumbing** of commands should only use the following system calls: `fork`, `exec`, `pipe`, `dup`, `wait` and their variants.

  Implementation of `cd` and standalone commands is not required. Assume all commands are simple Linux programs.

3. Check the tar file for sample testcases. You can execute these commands with the sample shell program `turtle`.

Deadline: 6th August 5.00 p.m.
Submission via moodle.

Submission Guidelines

- All submissions via moodle. Name your submission as: <**rollno_labquiz1**>**.tar.gz**
- The tar should contain the following files in the following directory structure:

```
<roll_number_labquiz1>/
|__beegees.c
|__plumb.c
```