Programming a simple Linux shell

This is a mandatory lab. (Approx. 20 full hrs)

Write a shell similar to the skeleton in MOS figure 1-19 but consisting of enough code to actually work and thus making it possible for you to test it. – You might want to refer to the description of the Linux shell in MOS 10.2.3.

Part 1

Level 1:
Initially your shell should just be able to execute one command at a time before prompting the user for the next command (called executing in the foreground). Your shell should be able to accept commands with parameters or options such as **`ls –l /home`** (2 parameters in addition to the command name).

You will have to:

- write a prompt to the user
- read the command line
- parse the command line into the command name and parameters and convert it to a format suitable for the exec function you want to use. (There are several possible front ends to the ***exec*** system call – check the Linux Programmer's Manual manual).
- ***fork()*** and ***exec()*** as described on fig 1-19 – you may have to add additional error handling code, consider what would happen if ***exec*** failed and returned....! – How could we recover?
- have the system pick up and print the return code from the command by modifying the ***wait*** call.

Level 2:
Once your shell can execute simple commands in the foreground add the ability to execute the command in the background by having a "&" as the last character on the command line.

Test that your shell can execute simple commands both as foreground and background commands.

- you can use simple commands like ls, or graphical applications like ***kate***.
- study the process tree with the ***pstree*** command or ***top***.
- when you omit picking up the child return code with wait you should be able to see your finished child processes as "*zombies*".

Once you are able to use your shell to execute commands (foreground and background) you may consider other additions.
The ***cd*** command is built into the shell (can you explain why?).
Implementing your own use of the PATH environment variable

Part 2

Test the signal example on CampusNet (at this moment an exercise in Lesson 5 - IPC) . – Try modifying it to use the ***signal*** system call and not ***sigaction*** (this is simpler).

Also study the Wikipedia page on signals : http://en.wikipedia.org/wiki/Signal_(computin g) – notice the references at the end of this page especially the presentation.

You should implement

- A ***killbg*** command that sends the signal sent by ***ctrl-C*** (***SIGINT***) to terminate any background process running in your shell.
- You will have to keep the process-id of your child process and terminate it ... how ? – (hint: ***man 2 kill***)
- Implement a handler in your shell for picking up the signal sent by a background child process upon exit – collect and print out the child return code.

It is acceptable to allow only a fixed number of background jobs, which should simplify the necessary "housekeeping".

Useful reading:

Manual pages for fork system call (***man 2 fork***) , the family of ***exec*** functions (***man 3 exec***) and the ***wait*** system call (***man 2 wait***) also the ***exit*** system call (***man 2 exit***) and function (***man 3 exit***).

GNU C Library documentation and man pages on signals:

- http://www.gnu.org/software/libc/manual/html_mono/libc.html#Signal-Handling
- ***man 2 kill***
- ***man 7 signal***

Getting and manipulating the current working directory:

- ***man 3 chdir*** and
-

Wikipedia on environment variables – UNIX and Windows

- http://en.wikipedia.org/wiki/Environment_variables
-

Linux man page on environment variables etc.

- ***man 7 environ***

The assignment must be submitted in print as well as by e-mail to henhu@dtu.dk. Hand in no later than **October 8**.

- The journal should clearly state name and student-id for all team members.
- Source files and binary files should be handed in as attachments to the email.
- Operating system version, compiler version and compiler directives used to build the binaries must be documented.
- Test runs and/or test data used to verify the functionality of the program must be included in the documentation.