Lab 5 – The Cow Shell

A mini command interpreter
Due: 4:50 PM, Friday, March 19, 2005

For this week's lab, you are to write a C program called the Cow Shell (`cowsh`), a mini command shell interpreter. `cowsh` is similar to familiar Unix shells such as the Bourne shell (`sh`) the Bourne-Again shell (`bash`), and C shell (`csh, tcsh`). You will learn about process creation, implementation of pipes, input/output redirection, background processes, signals, interrupt handling, and the use of some system calls.

This lab is considered a "team program" (groups of size 2 or 3 permitted) for honor code purposes. Collaboration within a group is, of course, unrestricted. You may discuss the program with members of other groups, but what you turn in must be your own group's work. Groups must be formed no later than 4:00 PM, Friday, March 12, 2005, and be confirmed by all group members by electronic mail to *terescoj@cs.williams.edu*. All group members will be assigned the same grade for the lab. There are many subtasks that can be carved off and assigned to group members, so everyone is encouraged to join a group.

Requirements

Like the familiar Unix shells, `cowsh` should issue a prompt (perhaps "`cowsh#`"), at which it reads commands from the user and executes them.

Your shell should interpret the following commands and provide the following functionality:

- `exit`: exit from the shell
- `help`: display a message listing usage of all commands
- Execute a command (program on disk). That command may not be present in the current directory, in which case the directories in the `PATH` environment variable should be searched. Appropriate choice of `exec` function will help here. The arguments following the command should be passed to the command.

  For example,

  ```
  cowsh# cat cat.c
  ```

  should execute `cat` with one argument, `cat.c`

  – *Input and output redirection* should be implemented.
    For example,

    ```
    cowsh# cat < cat.c > myfile.c
    ```

    should cause the `cat` program to read from `cat.c` and write to the file `myfile.c`.

  - *Pipes* should be implemented.
    For example,

    ```
    cowsh# cat cat.c | wc > count.txt
    ```

    should cause the output of `cat cat.c` to be the input of `wc > count.txt`.
  - <ctrl-c> should abort a command being run, but not cause `cowsh` to terminate.

- As with the familiar Unix shells, appending an `&` to the end of your command line will execute the command in the background.

  - Typing <ctrl-c> should not kill commands running in the background.
  - When any background command terminates, it should be reported.

    ```
    cowsh# sleep 55 &          ; invoke sleep in background
    cowsh# cat < hello.c       ; give other commands
    cowsh# ...                 ; other commands
    cowsh# ...                 ; other commands
    [2] "sleep" terminated     ; sleep command is done
    ```

  - All backgrounded processes should be maintained in a process table by `cowsh`, so that the program name is displayed when it terminates, and for use in the `jobs` and `kill` commands.

- `jobs:` Displays all the active background programs that have been started from this shell, along with their *id*s. (This id need not be that same as the actual process id, but it could be the index into your process table).

  ```
  cowsh# jobs
  PID     Name
  [0]    mycat < myfile.c > newfile.c
  [1]    idle 20
  [4]    grep cowsh < doc | wc
  ```

- `kill:` Without any arguments, prints the usage statement:

  ```
  cowsh# kill
  kill <pid> [<pid> ...]
  ```

  Otherwise it kills the process with the specified ids and displays the process killed.

  ```
  cowsh# kill 4
  [4] "grep" Killed
  cowsh# kill 3 7
  [3] "cat" Killed
  [7] "wc" Killed
  ```

- Errors should be reported meaningfully.
- Additional functionality of your group's choosing should also be implemented (see also the Submission and Evaluation section below).

  Ideas for extra functionality include:

  - a built-in `cd` command
  - command history: `history` command, `!command`, `!!`, `^` modification of previous command
  - control structures in the shell (`for`, `while`, `if`)
  - aliases
  - user-specified prompts (as in `bash`, `tcsh`)
  - More advanced redirection and pipes: `>>`, `>&`, `|&`
  - `;`-separated commands
  - <ctrl-z> trapping and corresponding job control

  See `builtin(1)` for more ideas.

Notes

- A similar project is described at the end of Chapter 3 of SG&G. You may find some useful information and examples there.
- The `system(3)` system call is **not** to be used.
- The system calls that you should use are `fork(2)`, a variant of `exec(3)`, `signal(3)`, `kill(2)`, `open(2)`, `dup2(2)`, `close(2)`, and `pipe(2)`.
- You may find the `readline(3)` library function to be useful.
- Use `cowsh` scripts to test your shell.
- The following might be a good order to tackle the required functionality.
  - `exit` and `help` commands
  - run a command (with no argument passing, no redirection, no pipes)
  - run a command with argument passing
  - run a command with input and output redirection
  - run a command in the background
  - `jobs` command
  - `kill` command

  - <ctrl-c> trapping
  - trapping termination of background processes
  - pipes
  - extra functionality

- `/home/faculty/terescoj/shared/cs432/lab5/cowsh` contains my version of `cowsh`.

Submission and Evaluation

All necessary files should be submitted using `turnin` as a single "tar" file, `lab5.tar`. Include a `Makefile` to allow easy compilation of the Cow Shell program.

I will compile and test your shell programs on CSLab FreeBSD systems.

Your program will be graded based on a total of 40 points.

- 6 points for documentation. Your code should be commented appropriately throughout. Please also include a longer comment at the top of your program describing your implementation. This documentation should list your important design decisions, the assumptions that you made (if any), the additional functionality implemented, and any other relevant information.
- 1 point for a functional `Makefile`.
- 1 point for implementing the `exit` and `help` commands.
- 4 points for running a command with no arguments.
- 4 points for running a command with arguments.
- 4 points for running a command with input/output redirection.
- 4 points for launching a command in the background.
- 2 points for the `jobs` command.
- 2 points for the `kill` command.
- 2 points for correctly trapping the <ctrl-c> keystroke.
- 2 points for reporting termination of backgrounded jobs.
- 3 points for pipes.
- Up to 5 points for extra functionality.
  - 1-point enhancements: `cd` command, aliases, more advanced redirection and pipes: `>>`, `>&`, `|&`, user-specified prompts, `;`-separated commands.

- 2-point enhancements: command history, control structures in the shell (`for`, `while`, `if`), <ctrl-z> trapping and corresponding job control.
- Please ask about other possible enhancements.