CS 411 Operating System Fundamentals

Project 1: Mini-Shell (*msh*)
Due Date: 9/21/98

Introduction:
This project is intended to give you insight into implementing a mini command shell interpreter (*msh*). The shell interpreter will be similar to the Bourne shell or the C shell/bash of Unix. You will learn process creation, implementation of pipes, input/output redirection, background processes, signals, interrupt handling, and the use of some system calls.

Description:
Your shell should interpret the following commands:

quit
help
run <cmd> <arg1> <arg2> ...
bg <cmd> <arg1> <arg2> ...
alias <name> <replacement-command>
ps
kill <pid> [<pid>] ...

The shell should issue the prompt "#", read one of the above commands from the user and execute them. When the user issues the "quit" command, it should terminate the shell.

The above commands are explained below in detail.

quit: should exit from the shell.
help: should display the above message listing usage of all commands.
run: should execute the command following it. That command may not be present in the current directory, in which case your shell should prepend the appropriate path. The arguments following the command should be passed to the command.
E.g. #**run** cat cat.c
should execute "cat" with one argument "cat.c".

- *Input and Output redirection* should be implemented.
  E.g. #**run** cat < cat.c > myfile.c
  should cause "cat" program to read form the file "cat.c" and write output to the file "myfile.c".
- *Pipes* should be implemented.
  E.g. #**run** mycat mycat.c | newcat > new.c
  should cause the output of "mycat mycat.c" to be the input of "newcat > new.c".
  There may be more than one pipe in a single command line.
- <Control-C> should abort the command being run, but **not** cause exit from the *msh*.

bg: This is similar to run, except that it executes the command in the background.

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

E.g. #**bg** sleep 55 ; invoke sleep in background
#**run** cat < hello.c ; give other commands
... ; some more commands
.... ; some more commands
[2] "sleep" terminated ; sleep command is done

- All the background commands issued should be maintained in a process table in the *msh*, so that the program name is displayed when it terminates and for use in the "ps" and "kill" commands.

alias: Should add the name and replacement-command in an alias table. When **run** or **bg** are invoked with a name in the alias table, then replacement-command should be executed. Some pre-existing aliases can be stored in a .alias file. Each line in the .alias file has the same meaning as the alias command. Before your mish accepts any commands it should load the information in the .alias file into an alias table. Any new entries added using the **alias** command should be checked for consistency (like possible duplicates). When **alias** is typed without any arguments, it should display the alias table

ps: Displays all the active background programs, along with their *ids*. (This id need not be the same as the actual process id, but could be the index into your process table).

E.g. # **ps**
PID NAME
[0] mycat < myfile.c > newfile.c
[1] idle 20
[4] grep mish < doc | wc
#

kill: Without any arguments, prints the usage:
kill <pid> [<pid> ...]
Otherwise it kills the processes with the ids specified as the arguments and display the process killed.

E.g. # kill 4
[4] "grep" Killed
kill 3 7
[3] "cat" Killed
[7] "wc" Killed

- Errors should be detected and reported meaningfully.

Implementation Notes:

- *"system"* system call should NOT be used.
- The system calls that you need to use are fork, execv, signal, kill, open, dup, close, pipe. (The system calls are explained in the man pages).
- 400 lines is a good upper estimate for the size of the program (assuming an elegant

design). Plan your work accordingly.

- You will be provided with test scripts shortly. Your program should execute the script correctly and completely. You are advised to write your own (extensive) test scripts too.
- A good order strategy for implementing the program would be
  - quit command
  - help command
  - run command (with no argument passing, no redirection, no pipes)
  - run with argument passing
  - run with input and output redirection
  - bg command
  - alias checking and expansion
  - ps command
  - kill command
  - <Control-C> trapping
  - Trapping termination of background processes
  - Pipes for run/bg command

Submission:
You should submit the following:

- Lucid implementation documentation. It should be such that a person who knows the problem but not your program should be able to follow your program easily and be able to change (or enhance) it without any problems (and not get bored reading it). This documentation should list your important design decisions, any assumptions that you made (if any), and any other relevant information. Please note that your program will be normally tested on the SUN Sparcs.

Grading Policy:
All programs will be tested on the RCS systems. Points break up:

Input redirection	10
Output redirections	10 (2 for good file permissions)
Pipes(both run and bg)	10
run	10 (5 : argument passing, 2 : correct waiting for compeletion in pipes)
bg	10 (5 : message when child terminates)
ps	5
kill	5
alias	5
<Control-C> trapping	5 (2 for run + 3 for bg)
quit+help	5
Error Handling	5
Documentation	20

Penalties:

Too few/Bad/many comments	-10
Convoluted logic	-10
Non-modular programming	-10

Early submission by 5 days would get 5 extra points. Extremely elegant and concise programs will get bonus points (depending on the elegance).