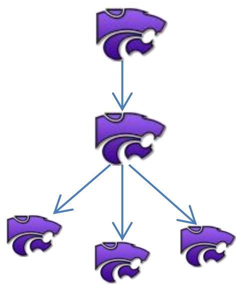

CIS 450 - Computer Organization and Architecture - Spring 2018

Programming Lab Assignment 5: Shell Lab
Lab Date: Apr. 13, Apr. 20, Apr. 27, Due Date: Mon., Apr. 30, 2018, 11:59 pm

(50 points)

Problem Statement

This assignment deals with developing a modified Unix shell called *mysh*. Instead of prompting the user with a shell prompt such as $ or %, the user should be presented with the prompt: *mysh*>. The shell should only accept commands beginning with the characters: a, A, p, P, w, W, f, F, g, G, l, L, n, N, s, S, e, E, i, I, c, C, d, D, k, K, h, H, q, or Q for the commands:

1. A <sec> = Alarm – set alarm to go off in given number of seconds (A 0 – turns the alarm off)
2. P [user] = Process status tree (**`ps --forest [-u [user]]`**)
3. W = Who is logged in and how many are logged in (**`who -q`**)
4. F [key] = Fortune cookie (**`fortune [-m [key]]`**)
5. G <num> = Game – execute wildcat shootout game with <num> players
6. L [dir] = Long directory listing (**`ls –al [dir]`**), directory **`dir`** is optional
7. N [filename] = Number lines in file or stdin (**`cat -n [filename]`**), output to stdout
8. S = Start or restart timer (internally computed using assembly instruction rdtsc)
9. E = Elapsed time in seconds to the nearest millisecond; e.g., xxxxx.xxxxxx sec.
10. I <signal number> = Ignore signal number <signal number>
11. C <signal number> = Catch signal number <signal number> using the signal handler
12. D <signal number> = Go back to default action for signal number <signal number>
13. K <signo> <pid> = Send signal <signo> to process with pid <pid>
14. H = Help (display this list of commands)
15. Q = Quit (exit the shell)

In addition, the user should be allowed to interrupt the shell by typing **`<ctrl>-c`** to display the current elapsed time (as in command E), if the SIGINT = 2 signal is not being ignored, and then reset/restart the timer (as in command 'S'). To expedite this part of the assignment, a template for the shell code is provided and some commands have already been completely implemented; e.g., **`<ctrl>-c`**, L, D, E, H and Q.

Instructions:
The code for this lab is available in the public directory **`/pub/cis450/programs/`** as **`Lab5.tgz.`** Copy this gzipped tar file to your own working directory where you plan to work, and unzip and untar it: **`tar xvzf Lab5.tgz.`** This will cause a number of files to be unpacked into the working directory. The only files that you will modify are **`myShell.c`** and **`Makefile`**.

Modify the Shell

For this assignment, you must augment the shell by modifying the existing shell code in myShell.c. In particular, add the additional functionality to implement the missing commands. Note that some of the commands are already implemented. The solution should be submitted in a single compressed folder containing Lab5 files, your updated **`Lab5.tgz`**, via K-State OnLine. In addition, as you modify the code, store your answers to the questions below in a text file, called **answers.txt**, to be submitted with your Lab5 files.

Build the shell using the command: `make mysh`, execute using: `./mysh`

(a) Before modifying the shell code, just build and run the shell using **`$ make`**, followed by **`$./mysh`**, if a user interrupts the shell by typing <ctrl>-c, the elapsed time since the shell started or was reset is displayed:

```
mysh>^C
*** Caught SIGINT signal ***
Absolute counter value = 9171556900610504.000000
Elapsed Time: 0.716308 seconds.
```

And the timer is reset. Hold the control key down and hit c several times. How quickly can you generate user input from the keyboard? _____. Don't just hold both keys down, that's cheating.

(b) Approximately how many cycles have elapsed between your two keystrokes assuming the machine is running at about 2.5 GHz (2.5 billion cycles/sec.)? ______.

Note that lots of processing can be completed between two keystrokes of an interactive process such as when a user is using an editor such as MS Word. Thus, we can easily give interactive users the illusion that they have exclusive access to the processor.

(c) Set the alarm to go off in 5 seconds using the command: mysh> **a 5.** What happens? _________. The default action for the alarm(5) system call is to set the alarm to go off in 5 seconds, and when it goes off, send the process a SIGALRM signal. The default action for what a process should do when it receives a SIGALRM signal is what happened; e.g., the process is terminated.

(d) Add the code to main( ) to catch the SIGALRM signal using the signal handler. Now, enter the command mysh> **a 5** several times in a row within 5 seconds of each other; how many signals are generated and caught? _______.

(e) Note that alarm(0) will turn off the alarm, so enter several **a 5's** again followed by an **a 0.** What happens? ________.

(f) Implement the code to determine who is logged in and how many users are logged into a given host by executing **`who -q`** when the command **w** is entered.

(g) Update the code to excute **`ps --forest [-u [user]]`** when the **p [user]** command is entered. This should display the process status tree. Note that "--forest" is one argument, and the user name is optional. If the user enters, just "p", then execute ps --forest, and if the user specifies a user name and enters "p username", then the command executed should be ps --forest -u username.

(h) Implement the code to send a signal via the kill function using the command: **k <signo> <pid>. This is done locally by the shell process without the need to fork off a child using the kill(pid,signo) statement.** Test the code by setting an alarm to go off in 60 seconds; display the process tree; and finally send an alarm signal to the mysh process to make the alarm go off before 60 seconds. Remember the signal numbers are all listed in /usr/include/asm/signal.h. What command did you use to send the process the SIGALRM signal? _________.

(i) Implement the code to display a fortune cookie and test it with the command **f "Software Engineering";** e.g., to execute **fortune -m "Software Engineering".** Note that the argument is optional, if the user just enters "f", then the command "fortune" without any arguments should be invoked to return a single fortune.

(j) Implement the code to play a wildcat process shoot'em up game when the command **g <n>** where <**n**> denotes the number of players is entered. The shell currently forks off a child process to act as the game manager and the child process calls shootout(**n**). The game manager should fork off **n** children (the wildcat processes) and they should each execute "wildcat" to shoot at each other by sending SIGUSR1 signals and SIGUSR2 signals at each other. Each child only has ten lives, a SIGUSR1 signal received diminishes one life (regular bullet), and a SIGUSR2 signal should diminish 2 lives. When ten or more lives have been lost, the child will exit. The manager will wait for the children to exit and declare the last child standing as the winner. Of course, then the winner is killed by the manager, and the manager also exits to return to the shell. Note that the player code is already implemented in wildcat.c. You can have players shoot at each other from the command line using $ **`./wildcat &`** to start up the first player in the background, and then start up a second player with the pid of the first player as a command line argument; e.g., if the pid of the first child is 1234, then start up the second player using $ **`./wildcat 1234`**. If the pid of the second player is 1240, then start up a third player using $ **`./wildcat 1234 1240.`** Note that any number of command line arguments can be used. Also, note that the last child has the best chance to win because the other players won't know of it's existence until they are shot at least once by the last process. You should set the shell to IGNORE all SIGUSR1 and SIGUSR2 signals. Test the code with 10 players; e.g. **G 10. Wildcats will shoot back at players who shoot at them.**

(k) The command **e** for obtaining elapsed time has already been implemented. Implement the command **s** to start or restart the timer. Note that <ctrl>-c does both e and s.

(l) Implement **i** to ignore a given signal; for example, if you ignore signal 2, then type <ctrl>-c to generate signal 2, it should be ignored by the process. Also, implement **c** to catch a signal by the signal handler; e.g., if you then **c 2** then signal 2 should get caught when the user types <ctrl>-c. The command **d is already implemented** to go back to the default action for a given signal.

(m) Finally, add code at the beginning of the case statement in myShell.c to redirect standard output to a file if the user include an argument of '>' for re-direct. The argument after '>' should be the filename that the output is redirected to. So, just open up a file with the given filename, and use dup2(fd, 1); to redirect the output from stdout (1) to the fd associated with the file. If the file already exists, just append the output to the end of the file. This can be accomplished by opening the file using: fd = open(cptr[i+1], O_WRONLY|O_CREAT|O_APPEND, S_IRUSR|S_IWUSR);

Modify the Makefile

Modify the Makefile so that **make mysh** causes myShell.o and clock.o to be linked with a user-created archive called libString.a. The archive is already created in the local directory using the command ar rcs libString.a split.o toUpper.o. Just change one line to compile and link myShell.c with libString.a using the link library argument; e.g., -l String at the end of the line. Remember to tell the compiler to look in the local directory for the library by adding -L . Rebuild mysh and make sure that it still works.

Hints:

- Arguments in square brackets [ ] are optional, but in angled brackets < > required.
- For the child process shootout game, there is a separate function called shootout( ) with some comments that provide hints to its implementation. Note that kill(pid,0) will return 0 if the process is still active (has not been killed and removed), and non-zero otherwise. This allows the children to only shoot at other living children. Also, the last child created has an advantage of already knowing the pids of all other children; e.g., the others only come to know of the last child after getting shot.

Submit contents of your Lab5 folder after saving answers.txt in the folder and creating a gzipped tar file containing the contents; e.g., $ tar cvzf Lab5.tgz Lab5.

Challenge: Modify wildcat.c to make the shoot-out game a "fair" game; that is, each wildcat should have an equal probability of winning the game. In the current version, the last child created clearly has an advantage over all other processes. Clearly indicate the modifications made to make the game fair.