SFWRENG 3SH3 Operating Systems

by Dr. Anwar M. Mirza

Lab Assignment 01

Due in Week starting Jan 31st, 2022

Shell Scripting and Signals

1. Background

1.1. Bourne Shell Scripting

Read the background material given in PracticeLab01 handout. To do the assignment you need to explore internet for

- `$$` - Process ID (PID) of the current process
- `$!` - PID of the last program that your shell ran in the background
- Foreground and Background processes
- `sleep` command
- `date` command
- variables
- `if .. then .. else .. fi` construct
- `while` and for `loops`

in the context of Bourne shell scripting.

1.2. Trapping Signals

Signals provide one of the main communication mechanisms between processes as well as between the kernel and the processes. When you are running your programs in UNIX/Linux, if you press `Ctrl-C` or `Ctrl-\`, your program terminates as soon as the signal arrives. There are times when you would rather not have the program terminate immediately after the signal arrives. You could arrange to ignore thee signal and keep running or perform some sort of cleanup operation before actually exiting the script. The `trap` command allows to to control the way a program behaves when it receives a signal.

A signal is defined as an asynchronous message that consists of a number that can be sent from one process to another, or by the operating system to a process if certain keys are pressed or if something exceptional happens.

The `trap` command tells the shell to terminate the command currently in execution upon the receipt of a signal. If the `trap` command is followed by commands within quotes, the command string will be executed upon the receipt of a specified signal. The shell reads the command string twice, once when the `trap` is set, and again when the signal arrivces. If the command string is surrounded by double quotes, all variables and command substitutions will be performed when the `trap` is set the first time. If single quotes enclose the command string, variables and command substitutions do not take place until the signal is detected and the `trap` is executed.

If an interrupt signal comes in while the script is running, the `trap` command lets you handle the signal in several ways. You can let the signal behave normally (default), ignore the signal, or create a handler function to be called when the signal arrives.

You can use the command `kill -l` to get a list of all signals on your version of UNIX/Linux (as the output of the `kill` command may differ depending on the OS and/or the shell being used). On macOS, using bash shell, you get the following output:

```
$ kill –l
 1) SIGHUP     2) SIGINT     3) SIGQUIT    4) SIGILL
 5) SIGTRAP    6) SIGABRT    7) SIGEMT     8) SIGFPE
 9) SIGKILL   10) SIGBUS    11) SIGSEGV   12) SIGSYS
13) SIGPIPE   14) SIGALRM   15) SIGTERM   16) SIGURG
17) SIGSTOP   18) SIGTSTP   19) SIGCONT   20) SIGCHLD
21) SIGTTIN   22) SIGTTOU   23) SIGIO 24) SIGXCPU
25) SIGXFSZ   26) SIGVTALRM 27) SIGPROF   28) SIGWINCH
29) SIGINFO   30) SIGUSR1   31) SIGUSR2
```

Example 1.1: Write the following script in a file names `myTrap`.

```
mkdir tmp
cd tmp
touch a.text b.text
ls
cd ..
trap ‘rm -r tmp; exit 1’ SIGHUP SIGINT SIGTERM
```

Save the file and execute it either using `sh myTrap` on the command prompt or first changing its mod to execute using `chmod a+x myTrap` and then running it using `./myTrap`. After running it, the `trap` for the signals listed in the script become activated. Now if any of these signals `SIGHUP` (handup), `SIGINT` (interrupt) or `SIGTERM` (software termination) arrives, the result will be the removal of the tmp directory that you created with all its contents. Press `Ctrl-C` and see the result.

Resetting Signals – To reset a signal to its default behavior, the `trap` command is followed by the signal name or number. For example, executing the command `trap 2`, resets the default action for signal 2, `SIGINT`, which is used to kill a process (i.e., `CTRL-C`)

Ignoring Signals – if the `trap` command is followed by a pair of empty quotes, the signals listed will be ignored by the process. For example, executing the command `trap “” 1 2` will result into signals 1 and 2 ignored by the shell.

Listing Traps – To list all traps and commands assigned to them, type `trap`.

2. Assignment

Write down a program in bash script that prompts a user for some text input. The text provided by the user is written back on screen and the script prompts again for a new text input from the user. This process goes on as long as the user keeps on entering text at program command prompt.

- If the user presses `Ctrl-C`, the program is not terminated and returns the message "`Ctrl-C will not terminate [name of your script]`".
- If the user presses `Ctrl-\`, the program is not terminated and returns the message "`Ctrl-\ will not terminate [name of your script]`".
- If the user presses `Ctrl-Z`, the program is not terminated and returns the message "`Ctrl-Z was pressed! [name of your script]`".
- Your program should gracefully exit if you input `stop` as a text string on its prompt.
- The script runs for a total execution time of 30 seconds. If you do not input any text and leave the program running unattended, the program should automatically exit back to bash shell command prompt after a total execution time of 30 seconds.

3. Guideline

You will

- Present you implementation and output to your lab TA.
- Work either individually or in a team of not more than 5 members
- Implement this assignment using Bourne Shell Script of Linux or macOS.

You may preseny your program in a different section in the same week **only once** throughout the term.