Network Administration Lab
Experiment#1
Linux Basic & Practice

Dr. Ahmed Awad

September 15, 2020

1 Objectives

This experiment aims to practice on the basic Linux commands and scripting to perform basic operations that form the substrate for further networking tasks. Thereafter, process management administrative tasks are handled.

2 Overview

Linux is an operating system that is based on Unix, an earlier multi-user operating system that has been developed in 1970's and 1980's. Some of the Internet protocols have been developed on Unix, therefore, Unix-based computer systems have a strong link with computer networks.
In 2000's, Linux has become popular in both servers and workstations. Besides, it has been well-growing in desktop applications. Several Linux distributions (variants) have been released by several companies (such as RedHat, Debian, Fedora, and others). Ubuntu is a free open-source operating system that has been developed to provide a user-friendly Linux distribution. Ubuntu has become so popular to perform computer network operations.
To automate Linux administrative tasks, scripting is intensively used in Linux. The standard language for administrative scripts for a long time have been defined by the Linux shell. However, other high-level scripting languages are also exploited.
One of the most crucial tasks of a network administrator is to manage processes. A Network application is a processes running on some machine with which a process running on another machine is communicating through TCP/IP protocol suite. Therefore, managing processes and their local communication on a single machine, forms the basis to learn managing processes between multiple processes running on different machines across a network.

3 Procedure

3.1 Basic Linux Operations

a. Login to the Ubuntu with the user-name and password that will be given to you by the instructor. **Is this user the root?**

b. Create an account named **Network**. Assign a password for that account and Login again using it. This account can temporarily gain the root privileges to do some configurations. **What command is used for that purpose?**

c. Logout from the user you have created and remove it. Show all the steps necessary to remove that account.

3.2 Command Line Shell

Start a new terminal. State the purpose for each of the following commands and then execute it (with passing proper arguments when needed):

- cd (Explain the difference between '.', '..' , and ~)
- ls
- cp
- mv
- rm
- mkdir
- rmdir
- touch
- less
- cat
- echo
- pwd
- wc
- sort
- ><
- ps
- &

- Ctrl-c
- Ctrl-z
- bg
- fg

3.3 Aliases, Variables, and Environmental Variables

a. Create an alias to the command ls and show that it is working.

b. Create a variable whose value is the absolute path to your current directory.

c. Print the value of the variable you have created.

d. Define an environmental named **MYENVVAR**. Set the value of this variable to be the date of today.

e. Show that your variable has been successfully created.

f. Print all the environmental variables defined in the system.

g. What is the difference between an environmental variable and other variables?

h. Open another terminal and try to show the value of the environmental variable you have created. **Can you see its value? Why?**

i. Create an environmental variable and make sure that it can be seen by all terminals? Show your steps in details.

3.4 File and Directory Validation

Write a bash shell script that receives two arguments: The first argument is the absolute path of a file you have created. The second argument is the absolute path of a directory you have created. Your script should validate both inputs by making sure that both of them exist. Then, if the validation succeeds, your script should copy the file into that directory. Otherwise, you have to print an error message stating which argument causes validation failure.

3.5 Flow Control

Write a bash shell script that works as follows:

a. Receives the values of two integers (X and Y) from the command line.

b. Compare both variables and print the smaller one.

c. Checks whether the file **.bashrc** exists.

d. If the file exists, the script checks whether it is a symbolic link or a regular file. Based on that check, the script has to be print its type.

e. If the file does not exist, the script should state that.

3.6 Process Parameters

a. Show all the running processes in your machine.

b. Write the PIDs for all processes whose owner is only the root on a file named **rootPIDs.txt**

c. Write the PIDs for all processes whose parent is the init process on a file named **initPIDs.txt**

d. Manually run the Firefox and pick the process associated with it. Explain all the attributes of that process.

e. Get the PID of the terminal process you are running.

f. Write a bash shell script that receives a process PID as an argument through the command line. Your script should return the **nice** value of that process.

g. What is a nice value?

h. Write a command that maximizes the nice value of a randomly chosen running process.

3.7 Process Monitoring and Signaling

a. Write a command that dynamically monitors all the running processes.

b. Write a C code for a process that forks a child process every minute. Trace the running execution of that process and capture when the fork system call is invoked.

c. Besides your running terminal (main terminal), open two other terminals. Trace the execution of your main using one of the other terminals. Then, send a STOP signal to the original terminal using the third terminal. Make sure to capture that signal in the tracing terminal.

d. Send a CONT signal to the original terminal.

e. send a KILL signal to the other two terminals using the original terminal. Show all your steps in details.

f. What is the difference between kill and terminate signals?

g. Write a bash shell script that gets a process PID as an argument from the command line. Then, your script should find the number of threads this process is running using the information provided under **/proc** directory.

3.8 Processes Communication with Socket

a. From moodle, download the files **client.c** and **server.c**.

b. Briefly explain both codes.

c. Compile the server file.

d. Compile the client file.

e. Run the server file.

f. Run the client file.

g. Modify the client code to send its process PID to the server to be printed with the hello message.

h. What interprocess communication technique is being used?

i. Use (Ctrl+Z) to stop the server process.

j. Try to run the server again. **What did you get? Why**

3.9 To Do

Modify the code you have used in the previous section to work as follows:

a. The client process sends its PID to the server process.

b. The server process sends its PID back to the client process.

c. The client process sends USR1 and USR2 signals to the server process.

d. When the server process receives USR1 it prints the statement **Hello, I received SIG1 from the client.**.

e. When the server process receives USR2, it prints the statement **Hello, I received SIG2 from the client.**.

f. The client process sends kill signal to the server process.

g. The client process terminates.