Lab_02_1: Process Management

Otago Polytechnic, IN616 Operating Systems Concepts,

Semester 1 – 2020

1 Objectives

- Managing Processes & Priorities
- Working with Background Tasks
- Killing Processes
- Shutting down and restarting the System

2 Preparation

For today's lab please use your *TrainingVM* in **vRealize**. You can login to vRealize using the following link:

- `https://fthvra01.op.ac.nz/vcac/`

Login using your OP username and password. From the *Items* tab, select the *Machines* entry (on the left panel). Find the *TrainingVM* on your list of virtual machines (on the right panel). Determine the IP address of your virtual machine - it should start with `10.25.X.X`.

Open `PuTTY` in your Windows 10 system. Enter the IP address in the PuTTY *Hostname* field. Leave the default port number of 22. Finally, click *Open* to start an SSH connection to your vRealize virtual machine. You can log in with the default username and password:

- Username: `student`
- Password: `P@ssw0rd`

3 Viewing and Managing Processes

Today in the lecture we briefly talked about processes and managing processing in Linux. This section of the lab introduces you to the primary process management tools.

3.1 ps

The most basic command for inspecting processes is `ps`. By default it shows all processes running under your user handle. That is, processes you have privilege to access which does not usually include system processes. By default, the `ps` command shows you the following process properties:

- `PID` – the Process ID
- `tty` – the terminal you are running the process from
- `TIME` – the processor time consumed
- `CMD` – the name of the command

A visual example of output from the `ps` command is provided below for reference:

```
  PID TTY          TIME CMD
10345 pts/0    00:00:00 bash
10357 pts/0    00:00:00 ps
```

To show extended information, use the parameter `-F`, that is `ps -F`. This should give you extended information on process properties including:

- `UID` – the username or User ID (UID) of a process
- `STIME` – the start time of a process
- `PPID` – the Parent Process ID (PPID)
- `SZ` – the total memory occupied by a given process
- `RSS` – the amount of memory that cannot be moved into swap memory
- `PSR` – the processor a process is running on (starting with 0)

A visual example of output from the `ps -F` command is provided below for reference:

```
UID        PID  PPID  C    SZ   RSS PSR STIME TTY          TIME CMD
user     10345 10344  0  5568  4620   0 13:56 pts/0    00:00:00 -bash
user     10361 10345  0  9342  3160   0 14:04 pts/0    00:00:00 ps -F
```

To show all processes, use the parameter `-e` in addition (or combine both parameters using `-Fe`). This should give you the complete output for all processes, not just the processes from

the user you are logged in as. Note the username in the first column will be from a variety of users, including the `root` account.

You will note that most commands are run using the root user. In fact, all Linux kernel daemon processes are executed under the root handle. All long-running background processes end with 'd' by convention, which indicates that they are 'daemon' processes and generally do not directly interact with the user but provide system services. An example for this is `systemd`, which we will talk about in a later session.

Note that `ps` has a large number of further options described in its manpage. Remember this command, since it is the most powerful process management tool.

Q1. Filter the output of `ps -Fe` to show only processes from the `bash` command.

Included in the previous command output you should have noticed a process called: `grep --color=auto bash`, or similar. Look at the CMD column.

Q2. How can you filter all processes that belong to the user `root`? Document the command below:

fhfghgfhgfh

HINT: Remember to list all processes and use `grep` to filter the output.

3.2 Showing the process hierarchy using pstree

Processes can have a parent process (or PPID) that launches a process. The `pstree` command gives you an overview of the process hierarchy, that is, showing which process has been started another process.

Q3. What is the name of the top-level process in your Linux system?

HINT: Use the `pstree` command.

Q4. Using the information from the previous section and your knowledge of processes, can you figure out the process id (or PID) of the answer to Question 4?

HINT: Use the `ps` command coupled with the grep command to find the process name you identified in Question 4.

3.3 top

The `ps` command gives you a snapshot of the current process state and exits, therefore, the content is not dynamic – it is a snapshot in time. For a comparison, think of the Windows Task Manager, which provides constantly changing information about processes – this is not how the `ps` command works.

A dynamic process viewer that allows you to inspect processes continuously is the `top` command – this is similar to the Windows Task Manager output. It will constantly update runtime information and shows the processes that occupy most processor time, or volatile memory (RAM) on your system.

However, in addition the `top` command also provides general system information, including overall system runtime, system load, running tasks as well as overall CPU usage. The following columns provide some useful information:

- `%CPU` – Percentage of CPU usage
- `%MEM` – Percentage of RAM usage
- `TIME` – the processor time consumed

In addition, the header information provided by `top` provides very useful information on the entire system, not just individual processes. The following properties are available in the header of the `top` command:

- `Tasks:` – General information on the processes running
- `%Cpu(s):` – General information on CPU usage
- `KiB Mem:` – Total, free, used values for RAM usage
- `KiB Swap:` – Total, free, used values for SWAP memory usage

The `top` command provides the ability to filter and sort process information, again, very similar to how Windows Task Manager operates. However, you have to use keyboard shortcuts instead of clicking!

While in the `top` window, press `u` and enter the username of choice (use the user account you are logged in as – probably `student`). This should filter all processes, and only display the processes from the username you entered. To check you did it correctly, inspect the **USER** column – you should only see the username of the account you enetered. To reset the view back to displaying all processes, press `u` and Enter. This resets it back to showing all processes for all users.

You can also customise many aspects of the `top` interface. Perform some research on the Internet to find way to customise the `top` interface. For example, try to sort the processes by the percentage of memory each process is using.

Q5. Document the method you used to sort the processes displayed by percentage of memory (%MEM):

HINT: Try using `Shift + f` in the `top` interface, then reading the instructions provided.

The `top` command is a good general-purpose tool to monitor the system dynamically. Note that there are a variety of further tools such as `htop`, which have additional functionality and features. However, such tools are not installed by default on most Linux distributions.

FYI - press `q` to exit `top`

3.4 Moving programs into the foreground

Similarly to moving programs into the background at runtime, we can retrieve commands and move them into the foreground. To do so, identify the command to be retrieved based on its job id listed in the `jobs` output (which is something like 1, 2, etc. It is not the process ID that we used before). You can then retrieve the process using `fg %id`. For example, to retrieve the second command you would type `fg %2`.

Q6. What would be an alternative way of running multiple commands at the same time?

4 Killing Processes

4.1 Signal Overview

As all operating systems, Linux interacts with processes using signals, or 'traps'. In Linux operating systems, the `kill` command can be used to terminate processes without having to restart the system – it is somewhat similar to *End Task* using Windows Task Managers, but with additional functionality. We can perform more than just terminating a process with the `kill` command, as there are a variety of signals available. To see a list of available signals, try the following command:

```
kill -l
```

Q7. After reviewing the output from the previous command, how many signals are available in our Linux system?

As you have hopefully guessed, the `-l` in `kill -l` stands for *list*, which *lists* all available signals. Most of them are relatively useless, at least from a user perspective.

Generally, signals can be identified by their **id** (the number) or **name**. All signals can be sent using the `kill` command. However, it is important to note that not all of them actually *kill* processes! For example, `SIGHUP` or signal hang up, was originally designed to notify a process of a serial line drop (aka a serial cable disconnection) – which is not specifically related to terminating a process. In comparison, the `SIGKILL` signal is specifically designed to terminate a process.

As an example of usage, to send the `SIGKILL` signal, we can execute any of the following commands, each with slightly different syntax, but the same result:

- `kill -s -KILL <PID>`
- `kill -KILL <PID>`
- `kill -9 <PID>`

In all of the above examples, `<PID>` stands for the Process ID of the targeted process. To look up PIDs you can you commands such as `top` or `ps` – both of which, we used in the previous exercises. A more specific tool is `pidof`, which allows you to identify the process by name, such as `pidof bash`, or `pidof script.sh`. The command `pgrep` serves a similar purpose, and can be used to *grep* a pattern of the command name; such as `pgrep bash`, or `pgrep script.sh`. We should try to find the PID of a command (or service) to practice.

Q8. Write a command to search output from the `ps` command to only print processes related to the keyword `sshd`. Document the command below:

HINT: Try using the `ps` command, and display processes from **all** users (or **-e**veryone!) Use a pipe and redirect the `ps` command command to `grep` with an appropriate keyword

Q9. How many instances of the `sshd` process are present?

As you have probably noticed, it is very difficult to differentiate between multiple processes when they all have the same name! Which leaves the question – how can we determine the difference between each of the `sshd` processes and what they are used for? Also, try the following commands:

- `pgrep sshd`
- `pidof sshd`

OK... that did not help! Both, `pgrep` and `pidof` only printed a list of PIDs matching the input keyword.

Q10. Using the `ps` command find more information about each of the `sshd` processes. For example, a more informative command name. Document your command below:

HINT: You need to perform a **-F**ull listing!

Review the output from the command you executed in Question 4. The output should be similar to that listed below:

```
UID          PID  PPID  C    SZ   RSS PSR STIME TTY          TIME CMD
root         746     1  0 16381  5856   0 Aug07 ?        00:00:00 /usr/sbin/sshd -D
root       13115   746  0 23852  6608   0 19:29 ?        00:00:00 sshd: user [priv]
user       13134 13115  0 23852  3332   0 19:29 ?        00:00:00 sshd: user@pts/0
user       13148 13135  0  3557  1020   0 19:29 pts/0    00:00:00 grep --color=auto ssh
```

Have a look at the **CMD** colum and locate the line that contains the following command name (CMD):

```
sshd: user@pts/0
```

Q11. What is the Process ID (PID) number of the process? Document the PID number below:

For interest sake... the `sshd: user@pts/0` command (or process) is the PuTTY SSH session you are using to connect to your Linux machine via vRealize. The `user` part indicates what user is connected, and the `@pts` stands for psuedo terminal slave, which is very similar to `tty`.

Now we will try to terminate, or kill, a task. Be careful during this step, if you select the wrong PID to kill... you may break your SSH service! Using the PID you extracted and answered for Question 5, run the following command:

```
kill -9 <PID>
```

Q12. What happened to you PuTTY session, and why?

Reconnect to your vRealize virtual machine using PuTTY...

4.2 Practising Using Signals: Preparation

Go to your home directory and execute the following lines to create a script.

```
echo '#!/bin/bash' > loop.sh
echo 'while true ; do' >> loop.sh
echo '  sleep 10' >> loop.sh
echo 'done' >> loop.sh
echo 'exit 0' >> loop.sh
```

Q13. Any thoughts about the purpose of this script?

Let's run it. However, instead of adjusting the execution permissions and run the script by calling `./loop.sh`, pass it to the interpreter `sh` for execution using the following command:

```
sh loop.sh
```

This is an alternative way of running scripts, especially when you don't have the permissions to adjust permissions. Note that the process of this script will be shown as **sh** in `top` or `ps`. This is becuase `sh` is the interpreter that runs the script.

Let's try different ways of interacting with the script.

4.3 Signal SIGINT (2)

Once the script is running you can interrupt its execution using:

```
Ctrl-C
```

This sends the `SIGINT` signal to the process. This is equivalent to sending the following commands:

```
kill -2 <pid>
kill -INT <pid>
```

Open a second terminal (right click the PuTTY top bar, and select *Duplicate Session* and log in again). Now, try to stop the script using the `kill` command instead of `Ctrl-C`.

The `kill` command is the most general panic command for interrupting execution... and it works in most cases. However, it relies on the process being able to react to the signal and to terminate properly – saving data and closing files etc. A command that not only requests interruption but termination is `SIGTERM` (`15`). But similar to `SIGINT` the process has the choice whether to follow this request.

4.4 Signal SIGTSTP (20)

If you want to merely suspend process execution you can use the command `Ctrl-Z` (which is equivalent to the `SIGTSTP` signal (`20`)). To reactivate that interrupted process, look up the job id using `jobs` and recover the process using `fg <jobId>` (where `jobId` stands for the respective job id). This is similar to shifting processes in the foreground and background as done previously. Give it a go to see if you understand how it works.

You see that most interaction with processes can be handled using those basic signals.

4.5 Signal SIGKILL (9)

In cases when the process is not responsive (e.g. hanging), you will need to resort to the more extreme signals, such as `SIGKILL` (`9`).

Start the `loop.sh` script again. Now, open a second terminal and identify the PID of the running script (look for the process name `sh`). Finally kill the process using `kill -9 <PID>` and return to the first terminal to see if it worked. Remember: This should be your last resort, since you may lose data. Generally try to stop processes using `Ctrl-C` (which is the same as `kill -INT` or `kill -2`), or `kill -15` (`SIGTERM`).

4.6 Killing Processes by Name

If you are sure that only one instance of a process runs, you can use the command `pkill` along with the process name to send signals to the process. For example:

```
pkill vi
```

If you simply want to address all instances of a program (e.g. all instances of `vi`), you can use the command `killall`. For example:

```
killall vi
```

The best method to try this is to run multiple instances of the loop script. Try the following steps:

```
# Start the loop script 5 times
sh loop.sh &
sh loop.sh &
sh loop.sh &
sh loop.sh &
sh loop.sh &
# View using ps
ps -e | grep sh
# Kill all the scripts using killall
killall sh
# Check the processes have been killed
ps -e | grep sh
```

5 Managing System Runtime

Although we saw a command overview in an earlier lecture, we did not really talk about shutting down and rebooting the system properly.

The primary commands for this purpose include:

- `shutdown`
- `halt`
- `poweroff`
- `reboot`

The core command is `shutdown`, while the latter are essentially aliases for specific parameter combinations of the `shutdown` command (shutdown commands with arguments). Since Linux is primarily a server operating system, rebooting or shutting down *requires superuser privileges*.

An interesting feature is Linux is the ability to notify all logged on users about an upcoming reboot or shutdown. For this purpose the `shutdown` command has various parameters (check its help and lecture slides). For example to shut down the system in 2 minutes, you could run the following command:

```
shutdown -P 2
```

In addition you could send a message to all signed in users:

```
shutdown -P 2 Shutting down soon. Save your work.
```

Try shutting down your computer. Log onto a second user on a different terminal and send the shutdown signal with a user message. Once it has sent the message, press `Ctrl-C`. This should interrupt the shutdown process. So even after to set off the shutdown command you can still change your mind. (Alternatively, you can execute the command `shutdown -c` to cancel the shutdown.)

To shut down the system immediately you could use `shutdown -h now` or `poweroff`. To reboot your system you can use `shutdown -r now` or `reboot`.

Try to reboot your system to assure you are comfortable with it. Finally, shut down your machine properly. Note that you will need to use vRealize to switch it back on if you want to continue working on it (use the web interface).