Welcome
Linux 2022

www.adafrolabs.com

Process Management

A running program/command is referred to as a **process**. One of the major roles of an operating system is to manage them, so each process gets an appropriate share of the resources such as CPU cycles, memory etc. While it appears to the human eyes that a computer is running multiple processes at once, generally that's not the case. The OS simply switches (multitasks) between processes very quickly to make it appear as if all processes run at the same time . Without multitasking in place, it would be impossible to run multiple programs at once or have more than one user using the same computer simultaneously.

Every user has control of the processes they own. User control ranges from viewing process status, monitoring resource use through pausing/resuming processes to terminating them. Each running shell allows running of multiple processes at once where one process is designated to be running in the foreground, while the others run in the background. This is similar to a GUI environment when multiple applications are running, but the user may only be directly interacting with one of them.

In this course we will limit process management to viewing process status and terminating processes from the command line. Process status is generally obtained in one of two ways:

1. A live feed of all running processes using the **top** command.This is a good way to get a general overview how resources are used by the entire system.You get a general CPU and memory summary as well as per-process details.

- Try the **top** command in the terminal.

- Examine the summary section at the top. You can view the processor core usage by pressing the 1 key on the keyboard. Can you determine how many processor cores there are? How about the amount or RAM? Notice that each process has an id and owner.
- Use the **q** key to exit.

2. A snapshot list of processes using ps command. The ps command has a long history and unfortunately due to this, it has inconsistent option syntax. For example, options can be specified with or without preceding dash (for example as: **ps a** or **ps -a** (where each syntax results with different output) as well as a double-dash syntax, for example: **ps --user bob**. Each syntax offers certain advantages, but for the purposes of this course we will be using options without any preceding dashes. Please see the manual for the **ps** command if you wish to obtain more detail about this.

The **ps** command without any options displays processes associated with the current shell, for example:

PID	TTY	TIME	CMD
28304	pts/2	00:00:00	bash
28415	pts/2	00:00:00	ps

The PID specifies the process ID and TTY notes the terminal ID for each command. Notice that the **ps** command "sees" itself. Information about other terminals and other processes is obtained using options such as **a** and **x**.

- Use the following commands in the terminal. Observe the output and discuss it with a classmate. What do you think option f does? Use the manual to verify your theory.

ps
ps a
ps x
ps ax

Most of the time we use the ps command to look up the ID of a process which we wish to terminate (kill). We resort to killing processes when they cannot be terminated using normal methods - for example when a program becomes unresponsive. Please note that killing a process may lead to lost data as it is an equivalent to pulling a plug on a working machine.

Processes are killed using the **kill** command. In the simplest case, a process is terminated using their numerical ID. For example, to kill process 1234, the command would be: **kill 1234**. Keep in mind that unless you are the system administrator (root) you can only kill processes you own. Multiple processes can be killed by listing multiple process IDs with the kill command.

Some programs are designed to trap (ignore) simple kill signals, in such cases the kill command may need an option, specified as a number. Most of the time a **kill -9** syntax is sufficient to terminate any process.

- Open two terminals on your system and run the **top** command in one of them.
- In the other terminal use the **ps a** command to find out the ID of the top command and a **kill** command to terminate **top** in the other terminal.
- Now find out the PID of the current shell and try to use the kill command to terminate it.You will find out that the shell protects itself from self-termination. Use the **kill -9** command (with the appropriate PID) to terminate the current shell.

BASH is one of those programs which require the -9 option to be killed. This is meant to protect processes running inside, as killing the shell terminates all processes running within.

In some cases it may be easier to bypass the process ID search before killing a process. The **pkill (or pkill -9)** command is capable of terminating processes by name. This approach is convenient, as long as you truly want to terminate all processes with a matching name.

- Open a couple of terminals and issue the following command: **pkill -9 bash**
- You should see all terminal windows disappear.