CSCI 274 - Intro to Linux OS

Week 10 - Process Control

Max Gawason (maxgawason@mines.edu) [A]
Rocco Marchitto (rmarchitto@mines.edu) [B/C/D]

TODO :

- Common Pipeline Utilities (wk 11)
- Stream Editing
- Identity, Ownership, and Permissions (wk 12, 13)
- Inodes and Linking (wk 15)
- Regular Expressions

Overview

1. Process control
   a. Background Jobs
   b. top
   c. ps
   d. pgrep
   e. kill
   f. pkill
   g. killall

Process

Processes carry out tasks within the operating system.

A program is a set of machine code instructions and data stored in an executable image on disk and is, as such, a passive entity; a process can be thought of as a computer program in action.

A program/command when executed, a special instance is provided by the system to the process. This instance consists of all the services/resources that may be utilized by the process under execution.

A Process can be run in two ways ...

- **Foreground Process :** Every process when started runs in foreground by default, receives input from the keyboard and sends output to the screen. When a command/process is running in the foreground and is taking a lot of time, no other processes can be run or started because the prompt would not be available until the program finishes processing and comes out.
- **Background Process :** It runs in the background without keyboard input and waits till keyboard input is required. Thus, other processes can be done in parallel with the process running in background since they do not have to wait for the previous process to be completed.

Background

When you execute a shell-script or command that takes a long time, you can run it as a background job.

```
$ ./myscript
```

Then I can't use my shell anymore while myscript is running because it will be busy making myscript happen in the foreground. If I do

```
$ ./myscript   >   ~/myscript.log   2>&1    &
```

Then myscript runs as a job in the background and I can continue to use my shell. In general, appending an ampersand (**&**) to the command runs the job in the background. The shell does not wait for the command to finish, and the return status is 0. Note that background jobs will close if/when you close the shell that started them.

Commands …

top - displays processor activity of your Linux box and also displays tasks managed by the kernel in real-time

PID: Shows task's unique process id.
PR: Stands for priority of the task.
SHR: Represents the amount of shared memory used by a task.
VIRT: Total virtual memory used by the task.
USER: User name of owner of task.
%CPU: Represents the CPU usage.
TIME+: CPU Time, the same as 'TIME', but reflecting more granularity through hundredths of a second.
SHR: Represents the Shared Memory size (kb) used by a task.
NI: Represents a Nice Value of task. A Negative nice value implies higher priority, and positive Nice value means lower priority.
%MEM: Shows the Memory usage of task.

Commands ...

ps - By default, the ps command produces a list of all processes associated with the current user and terminal session.

PID – the unique process ID
TTY – terminal type that the user is logged into
TIME – amount of CPU in minutes and seconds that the process has been running
CMD – name of the command that launched the process.

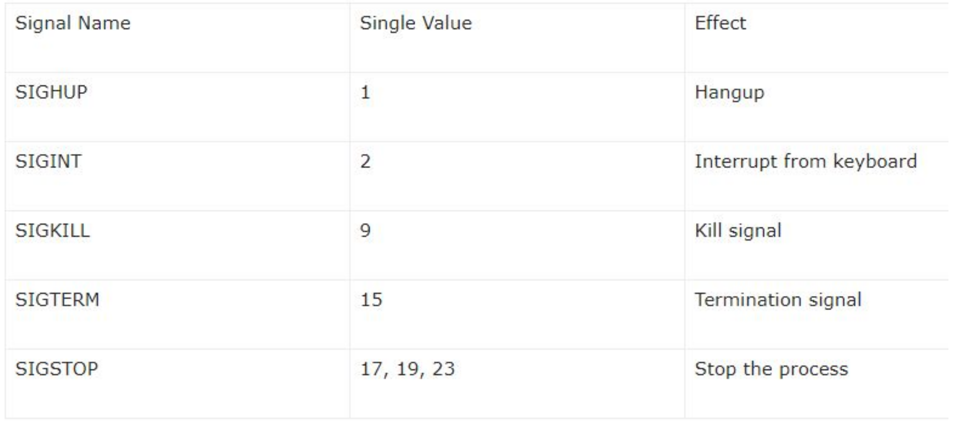
pgrep - quick way of getting the PID of a process. Lets users look up processes based on name and other attributes. There will be no output if the input is not associated with an actual process.

Commands ...

kill - to attempt to kill a process

Signals can be specified in three ways:

- By number (e.g. -5)
- With SIG prefix (e.g. -SIGkill)
- Without SIG prefix (e.g. -kill)



Signal Name	Single Value	Effect
SIGHUP	1	Hangup
SIGINT	2	Interrupt from keyboard
SIGKILL	9	Kill signal
SIGTERM	15	Termination signal
SIGSTOP	17, 19, 23	Stop the process

pkill - works in almost exactly the same way as kill, but it operates on a process name instead

User Input

read - asks for the user’s input and exit once the user provides some input.