Diagnose and Manage Processes

Whenever a program is launched on Linux, it lives as a process on the OS.  
That is, until it finishes its job or if it's closed.

For example, when I run the ls command, a very short-lived process is created. The process for ls is to display a directory's content and then as soon as it is done doing so, the process will end.

The commonly used command to inspect processes is the “ps” command.  
The ps command can be a bit weird because it supports two kinds of syntax for its options  
-a (ps –a) uses the UNIX syntax.  
A (ps a) uses the BSD syntax.  
And “ps a” has an entirely different effect than “ps -a” so the “a” and “–a” options are NOT equivalent.

By default, the ps command with no options will show the processes in my current terminal session.

Analyzing the ps aux output:  
I will only not the non-obvious ones (like %MEM & %CPU are too obvious)  
Kernel processes are between square brackets []. They run inside a privileged area inside the Linux kernel.

Processes that are not wrapped in square brackets are operating in the “user space”.

The ps aux only shows the state of processes at the exact time I ran the command. Use the “top” command to continuously observe ongoing processes, with the most CPU intensive processes at the top of the list.  
  
For the ps command:  
Use the “u” flag to show the result of a specific process (add the PID in the input) in user friendly mode.

Use the “-U” + the username to show the processes for another user.

To search for a specific process via a name, we can use the “pgrep” command:

$: pgrep –a syslog  
The –a command means all and it will even show the command that started the process.  
In Linux, there is a concept called process “niceness”  
That is, how nice a process is to other processes.  
The process value can be a number between –20 and 19.  
The lower the number, the less nice it is. **In other words, a lower number means a higher priority**. There are 2 ways to set a nice value for a process.  
1. $: nice –n [NICE VALUE] [COMMAND]  
  
You can use the “l” flag to view nice values when using the ps command:

$: ps

$: ps lax = You can use this to view All processes with their niceness values.  
All processes inherit the niceness value of their parent process.

For example:  
$: ps –n 11 bash = will launch a bash shell with the niceness value of 11. Now in that same bash shell I run the “ps lax” command.

The “ps lax” command will spawn a process with the inherited niceness value (11) of the earlier command that launched bash, which is its parent process.

$: ps f = will list the processes and their parent processes. It will display the relationship in a tree format.  
  
A regular user can only assign nice values from 0 to 19.  
To define lower values (higher priority), you would need to use root privileges.  
What if I wanted to start a process under a regular user rather than root?  
It can be done this way:   
(Context) = I want to start a process that I want to give large CPU priority to.  
Step 1: Launch the program like normal.  
Step 2: Run the “ps fax” command to view the processes and its parent(s)  
Step 3: Identify the PID of the process you want to give priority to.  
Step 4: Use the “renice” command to adjust the current niceness value for that PID = renice [NEW NICE VALUE] [PID]

Regular users also cannot renice a value lower than it currently has. A regular user can make the niceness value higher though without the need for root privileges.

Linux has a functionality that sends processes “signals”. These are like high priority messages that tell the process to stop what its doing, there is a special request.  
But, an application can only act on a specific signal if it was programmed to respond to that signal. The only exceptions are SIGSTOP & SIGKILL. These cannot be ignored by the process. When SIGSTOP is run, the process will pause until it gets the SIGCONT is received.

Use the kill -l command to get a list of SIG\* commands. I can use the number rather than the full name. Ex: kill -9 == kill -SIGKILL

Backgrounding and foregrounding processes

Some commands can take a long time to finish their job, so you can push that command to the background.  
For example, open vim and edit / create a file. While in the editor type:  
CTRL + Z on the keyboard, this will pause the application and return to the terminal.

The “fg” command can be used to return to the paused application. In my case, VIM.

CTRL-C & CTRL-Z will work for most programs, but not all to keep that in mind.   
When the program is paused, no progress is made, it is frozen in place. If you want it to continue with its work, you can use the & symbol.  
Ex: $: sleep 300 & 🡪 will send this command to run in the background.  
To check programs running in the background, you can run the “jobs” command.  
Backgrounded programs have ids, so you can specify which job you want to return to like this:

$: fg <job id here>

You can use the bg <job id here> to continue a paused job already in the background.  
  
Sometimes, I would want to see what files a process is currently using. I can use the lsof command after identifying the PID that I want to analyze using the pgrep command.

$: pgrep -a bash  
$: lsof -p <PID of bash here>

Locate and Analyze System Log Files

Linux systems are largely server oriented, so you always want to know what happened, who did what, when etc...  
Everything important on a Linux system is saved somewhere in a text file in the system.  
The Linux kernel and most programs on the OS generate status messages, error messages, warning messages and so on.

Logging Daemons 🡪 collect, organize, and store logs.  
The most popular daemon on Linux is rsyslog.

This Daemon stores all logs in the /var/log directory.  
Most of these files cannot be read by regular users.

Here’s an example, we can search for logs that mention ssh by using grep:  
$: grep -r ‘ssh’ /var/log  
And the output will be all files that mention ssh and what directory those logs are in.

Following log files

$: tail -F /var/log/<logfile.log>  
As a test, you can ssh into the same machine using another computer and you will see the successful (or unsuccessful) login attempt appear in the terminal.  
  
Here is another command that lets you analyze logs more efficiently.  
  
$: journalctl /usr/bin/sudo (You can use the -f flag for following mode)  
  
Journalctl lets us monitor logs for a specific command  
  
use the which command to locate binary location for a command  
$: which sudo  
  
Here is how to monitor a service ( -u flag tells the command to monitor the logs )  
$: journalctl -u ssh.service

When you just type:  
$: journalctl  
This will show all logs collected by the journal daemon.

$journalctl -e   
This will jump to the end of the journal showing the latest logs

info, warning, err, crit is logging statuses that can be found when analyzing logs.  
You can isolate logs with these flags like so:  
-p 🡪 is the flag for priority  
  
$: journalctl -p err  
$: journalctl -p TAB+TAB to view the applicable status modes.  
  
I can use the grep command with journalctl as well to add regex to my query:  
$: journalctl -p <status> -g ‘<regex here>’

The -g flag for grep allows me to add basic regex to my query.  
  
The -S flag (used with journalctl) stands for since. I can query logs “since” a specified time that I set:  
$: journalctl -S <time here 24hr format>  
$: journalctl -S 02:00 🡪 Shows logs generated since 2AM.  
  
There is also an “until” option (-U)  
$: journalctl -S 01:00 -U 02:00 🡪 Generates logs from 1AM to 2AM  
  
The -S and -U options also supports dates  
$: journalctl -S ‘2024-03-03 01:00:30’ 🡪 It is important to wrap the dates in single quotes  
  
By default, journalctl alone will show the oldest data first.   
The journal daemon groups logs based on when the system booted.  
So, to see logs generated for the current boot, we can use the -b flag.

$: journalctl -b 0 🡪 0 specifies my current boot. If you wanted to see the logs from previous boot you can use 1:  
$: journalctl -b -1  
You can keep going from here changing the number, be mindful that some Linux OSs are configured to keep logs in memory and not on disk. So if its in memory, it will wipe upon reboot.  
But the journal daemon is configured to decide whether to save a persistent journal depending on the existence of the directory /var/log/journal. If it doesn’t exist, it won’t preserve logs on disk, but if it does exist, it will start to preserve systemd journal logs in that directory. By simply creating this directory, we can instruct the journaling daemon to preserve these logs there.

Use the “last” command to see a history of who logged into the system.  
$: last  
An alternative to this is the “lastlog” command  
$: lastlog  
This shows more clearly remote logins in addition to the ip address they used to log in.