

Linux Process Management

Introduction

A Linux system can have hundreds or even thousands of processes running at the same time. A **process** is simply a program or task that is currently running and using system resources.

A process could be something as simple as a terminal command, a text editor, a web server, a database, or even part of the graphical interface.

Understanding processes is an important Linux skill, especially when working in system administration and cybersecurity. Knowing how to identify, monitor, prioritize, stop, and schedule processes helps you understand what is happening on a system and how its resources are being used.

In this article, we will learn how to:

- View running processes
- Understand Process IDs (PIDs)
- Find specific processes
- Monitor resource usage
- Change process priority
- Stop or terminate processes
- Run processes in the background
- Move processes between the background and foreground
- Schedule processes to run at a specific time

Note: The commands in this article will be demonstrated practically on Kali Linux.

1. What Is a Process?

Whenever you run a program or command on Linux, the operating system creates a **process** for it.

For example, when you open a terminal and run:

ls

Linux creates a process to execute that command.

The same idea applies to larger applications such as:

- Web servers
- Databases
- Text editors
- Network tools
- Security tools
- Graphical applications

Each process is assigned a unique number called a **Process ID**, or **PID**.

The PID is important because Linux uses it to identify individual processes. When we want to inspect, modify, or terminate a particular process, we will often work with its PID.

2. Viewing Processes with **ps**

One of the most useful commands for viewing processes is:

`ps`

Running **ps** without additional options gives us a basic snapshot of processes associated with the current terminal/session.

A typical output may look similar to:

```
PID  TTY  TIME  CMD
39659 pts/0 00:00:01 bash
39665 pts/0 00:00:00 ps
```

The important thing to notice here is the **PID**.

Understanding the columns

Column	Meaning
PID	Process ID
TTY	Terminal associated with the process
TIME	CPU time used by the process
CMD	Command that started the process

The basic `ps` command is useful, but it does not show everything running on the system.

To see a much broader list, we can use:

```
ps aux
```

This displays processes belonging to different users as well as processes running in the background.

Some important columns include:

Column	Meaning
USER	User who started the process
PID	Process ID
%CPU	Percentage of CPU being used
%MEM	Percentage of memory being used
COMMAND	Command or program responsible for the process

Because a Linux system can have many processes running at once, the output from `ps aux` can be quite large.

3. Finding a Specific Process

Sometimes we are not interested in every process on the system. We may only want to find one particular process.

This is where `grep` becomes useful.

We can combine `ps aux` with `grep` using a pipe:

```
ps aux | grep <process-name>
```

For example, if we wanted to search for a process containing the word `msfconsole`, we could use:

```
ps aux | grep msfconsole
```

The `|` symbol sends the output of the first command to the second command.

In this case:

```
ps aux
```

lists the processes, while:

```
grep msfconsole
```

filters that output and displays only lines containing `msfconsole`.

This technique is useful when the process list is large and we need to quickly locate a particular process.

4. Monitoring Processes with `top`

The `ps` command gives us a snapshot of running processes. But what if we want to continuously monitor them?

This is where `top` is useful.

Run:

```
top
```

Unlike `ps`, `top` continuously refreshes the process list.

It provides information about:

- CPU usage
- Memory usage
- Running processes
- Process IDs
- Process priorities
- System load

Processes using significant system resources can therefore be identified more easily.

The `top` interface is interactive. While it is running, you can use keyboard shortcuts to perform different actions.

For example:

Q

can be used to exit `top`.

You can also press:

?

or:

H

to view available interactive commands.

5. Understanding Process Priority with `nice`

Linux may have many processes competing for CPU resources.

Sometimes we want a process to receive more or less priority compared with other processes. Linux provides the `nice` command for this purpose.

The **nice value** ranges from:

-20 → 0 → +19

The important rule to remember is:

Lower nice value = higher priority

Higher nice value = lower priority

A default nice value is normally:

0

For example, we can start a process with a higher nice value:

```
nice -n 10 <command>
```

This starts the command with a lower priority.

A negative nice value increases priority:

```
nice -n -10 <command>
```

However, increasing a process's priority generally requires elevated privileges.

6. Changing the Priority of a Running Process with **renice**

The **nice** command is mainly used when starting a process.

But what if the process is already running?

That's where **renice** comes in.

First, we need the process's PID.

For example:

```
renice 19 <PID>
```

Unlike **nice**, **renice** sets the process to a specific nice value rather than simply adding an increment.

For example:

```
renice 19 6996
```

This changes the process with PID **6996** to a nice value of **19**, giving it lower priority.

You can also change process priority directly from **top** using its interactive commands.

7. Stopping Processes with **kill**

Sometimes a process may stop responding, consume too many resources, or simply need to be terminated.

Linux provides the `kill` command for sending signals to processes.

The basic syntax is:

```
kill <PID>
```

For example:

```
kill 6996
```

By default, `kill` sends the `SIGTERM` signal.

There are several signals available, but some of the most commonly encountered are:

Signal	Number	Purpose
SIGHUP	1	Hangup/reload-related signal
SIGINT	2	Interrupt a process
SIGQUIT	3	Quit and generate a core dump
SIGTERM	15	Request that a process terminate
SIGKILL	9	Forcefully terminate a process

SIGTERM

This is the default signal used by:

```
kill <PID>
```

It politely asks the process to terminate and gives the program an opportunity to clean up.

SIGKILL

If a process refuses to terminate, we can use:

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

`SIGKILL` forcefully terminates the process.

Because it does not give the program an opportunity to clean up, it should generally be used when a normal termination does not work.

8. Using `killall`

What if we know the name of a process but don't know its PID?

We can use:

```
killall <process-name>
```

For example:

```
killall rogueprocess
```

The command works with the process name instead of requiring us to provide a PID.

A signal can also be specified:

```
killall -9 rogueprocess
```

This sends `SIGKILL` to matching processes.

Be careful with `killall`, especially when running it with elevated privileges, because it can affect multiple processes with the same name.

9. Running Processes in the Background

Normally, when we execute a command from the terminal, the shell waits for that command to finish.

For example:

```
leafpad newscript
```

The terminal remains occupied while the application is running.

But sometimes we want the application to continue running while we use the same terminal for other commands.

Linux allows us to run a process in the background by adding `&` to the end of the command:

```
leafpad newscript &
```

The `&` tells the shell to start the process in the background.

The terminal then becomes available again for other commands.

This is particularly useful when working with multiple tools or long-running tasks.

10. Moving a Process to the Background

A process that is currently running in the foreground can also be moved to the background.

The general workflow involves stopping the foreground job and then using:

`bg`

The `bg` command tells the shell to continue a stopped job in the background.

We will demonstrate this practically later in the article.

11. Bringing a Process Back to the Foreground

If a process is running in the background and we want to interact with it again, we can use:

`fg`

For example:

`fg`

The command brings a background job back into the foreground.

This is useful when a program requires interaction through the terminal.

12. Scheduling Processes

Linux can also schedule commands or scripts to run at a particular time.

This is useful for tasks such as:

- Backups
- Maintenance scripts
- Automated jobs
- Recurring administrative tasks

Two important tools for scheduling tasks are:

`at`

and:

`cron`

The `at` command is designed for jobs that need to run **once in the future**, while `cron` is commonly used for **recurring scheduled tasks**.

13. Scheduling a One-Time Task with `at`

The basic structure is:

`at <time>`

For example:

`at 7:20am`

After entering the command, Linux provides an interactive prompt where we can enter the command we want to execute.

For example:

`at 7:20am`

`at> /root/myscanningscript`

The scheduled command will then execute at the specified time.

To finish entering commands into the `at` prompt, press:

Ctrl + D

The `at` command can accept different time formats, including relative times such as:

at now + 20 minutes

or:

at now + 5 days

This makes it useful when we want to schedule a command without manually calculating an exact clock time.

14. Why Process Management Matters in Cybersecurity

Process management is not just a Linux administration skill. It is also relevant to cybersecurity.

When investigating a Linux system, processes can provide useful information about what is currently running.

For example, you may need to identify:

- Unexpected applications
- Resource-heavy processes
- Security tools
- Network services
- Suspicious processes
- Processes belonging to particular users

In a security investigation, understanding PIDs, process ownership, resource consumption, and process behavior can help you determine what is happening on a system.

For offensive security work, process management is also important when working with multiple tools and long-running tasks.

The key is to understand the commands and use them only on systems where you have permission to perform the actions.

15. Practical Lab

Now that we understand the concepts, let's put them into practice on Kali Linux.

Practical 1 View Running Processes

Run:

```
ps
```

Then:

```
ps aux
```

Record:

- The PID of a few processes
 - The users running them
 - Their CPU usage
 - Their memory usage
-

Practical 2 Find a Process

Choose a running process and search for it using:

```
ps aux | grep <process-name>
```

Observe how `grep` filters the output.

Practical 3 Monitor Processes

Run:

```
top
```

Identify the processes currently consuming the most CPU and memory.

Exit `top` using:

```
Q
```

Practical 4 Experiment with Process Priority

Start a harmless process with a custom nice value:

```
nice -n 10 <command>
```

Then use `ps` to inspect its priority.

We will also practice changing the priority of an existing process using:

```
renice
```

Practical 5 Terminate a Process

Start a harmless test process and identify its PID.

Then terminate it using:

```
kill <PID>
```

If necessary, we will compare the behavior of:

```
kill -15 <PID>
```

and:

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

Practical 6 Background and Foreground Processes

Start a command normally and observe how it occupies the terminal.

Then try running a command with:

```
<command> &
```

We will also practice using:

```
bg
```

and:

```
fg
```

to move jobs between the background and foreground.

Practical 7 Schedule a Command

Finally, we will use `at` to schedule a harmless command to execute at a future time.

For example:

at now + 5 minutes

Then enter a simple command and press:

Ctrl + D

We will verify that the scheduled job was created successfully.

Key Commands to Remember

Command	Purpose
<code>ps</code>	View processes
<code>ps aux</code>	View processes from all users
<code>grep</code>	Filter process output
<code>top</code>	Monitor processes in real time
<code>nice</code>	Start a process with a chosen priority
<code>renice</code>	Change the priority of a running process
<code>kill</code>	Send a signal to a process
<code>killall</code>	Send a signal using a process name
<code>&</code>	Start a process in the background
<code>bg</code>	Continue a job in the background
<code>fg</code>	Bring a job to the foreground

at

Schedule a one-time task

Conclusion

Process management is one of the fundamental skills every Linux user should develop.

By understanding how to view processes, identify their PIDs, monitor resource usage, change priorities, terminate processes, work with background jobs, and schedule tasks, we gain much better visibility and control over a Linux system.

For cybersecurity learners, these skills become especially useful when analyzing systems, troubleshooting problems, investigating suspicious activity, or working with multiple security tools.

The best way to understand process management, however, is not just to read about it.

Open the terminal, run the commands, observe the output, and learn what the system is actually doing.