PROCESS MANAGEMENT

At any given time, a Linux system typically has hundreds, or sometimes even thousands, of processes running simultaneously. A process is simply a program that's running and using resources. It includes a terminal, web server, any running commands, any databases, the GUI interface, and much more. Any good Linux administrator—and particularly a hacker—needs to understand how to manage their processes to optimize their systems.

VIEWING PROCESS

```
Kali> ps
```

The Linux kernel, the inner core of the operating system that controls nearly everything, assigns a unique process ID (PID) to each process sequentially, as the processes are created. When working with these processes in Linux, you often need to specify their PIDs, so it is far more important to note the PID of the process than the name of the process.

Running the ***ps*** command with the options aux will show all processes running on the system for all users

```
Kali> ps aux
```

USER	The user who invoked the process
PID	The process ID
%CPU	The percent of CPU this process is using
%MEM	The percent of memory this process is using
COMMAND	The name of the command that started the process

FILTERING BY PROCESS NAME

```
Kali> ps aux | grep msfconsole
```

FINDING THE GREEDIEST PROCESS

When you enter the ***ps*** command, the processes are displayed in the order they were started, and since the kernel assigns PIDs in the order they have started, what you see are processes ordered by PID number. In many cases, we want to know which processes are using the most resources. This is where the top command comes in handy because it displays the processes ordered by resources used, starting with the largest. Unlike the ***ps*** command, which gives us a one-time snapshot of the processes, top refreshes the list dynamically—by default, every 10 seconds.

```
Kali> top
```

System administrators often keep top running in a terminal to monitor use of process resources.

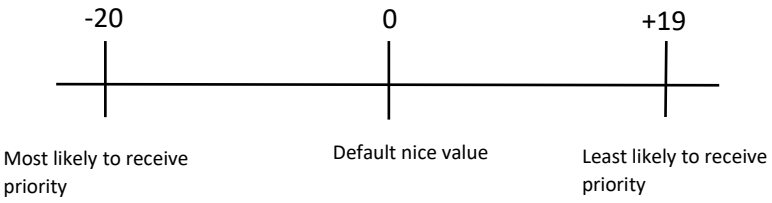
MANAGING PROCESS

Hackers often need to multi process, and an operating system like Kali is ideal for this. The hacker may have a port scanner running while running a vulnerability scanner and an exploit simultaneously. This requires that the hacker manage these processes efficiently to best use system resources and complete the task.

CHANGING PROCESS PRIORITY WITH NICE

The nice command is used to influence the priority of a process to the kernel. If our process is using most of the system resources, we are not being very nice.

The value for nice range from -20 to +19, with zero being the default value. A high nice value translates to a low priority, and a low nice value translates to a high priority. When a process is started, it inherits the nice value of its parent process. The owner of the process can lower the priority of the process but cannot increase its priority. Of course, the superuser or root user can arbitrarily set the nice value to whatever they please.



When you start a process, you can set the priority level with the nice command and then alter the priority after the process has started running with the renice command. The syntax for these two commands is slightly different and can be confusing. The nice command requires that you increment the nice value, whereas the renice command wants an absolute value for niceness.

SETTING THE PRIORITY WHEN STARTING PROCESS

```
Kali> nice -n -10 /bin/slowprocess
```

This command would increment the nice value by -10, increasing its priority and allocating it more resources.

On the other hand, if we want to be nice to our follow users and processes and give slowprocess a lower priority, we could increment its nice value positively by 10:

```
Kali> nice -n 10 /bin/slowprocess
```

CHANGING THE PRIORITY OF A RUNNIG PROCESS WITH RENICE:

The renice command takes absolute values between –20 and 19 and sets the priority to that particular level, rather than increasing or decreasing from the level at which it started. In addition, renice requires the PID of the process you are targeting rather than the name. So, if ***slowprocesses*** using an inordinate amount of resources on your system and you want to give it a lower priority, thus allowing other processes a higher priority and more resources, you could renice the ***slowprocess*** (which has a PID of 6996) and give it a much higher nice value,

```
Syntax> renice <particular nice value> <PID>
```

```
Kali> renice 20 1234
```

As with nice, only the root user can renice a process to a negative value to give it higher priority, but any user can be nice and reduce priority with renice.

We can also use the top utility to change the nice value. With the top utility running, simply press the R key and then supply the PID and the nice value.

KILLING PROCESS

At times, a process will consume way too many system resources, exhibit unusual behaviour, or – at worst – freeze. A process that exhibits these types of behaviour is often referred to as a Zombie process.

The kill command has 64 different kill signals, and each does something slightly different. Here, we focus on a few we will likely find most useful. The syntax for the kill command is kill-signal PID, where the signal switch is optional. If you don't provide a signal flag, it defaults to ***SIGTERM***.

COMMONLY USED KILL SIGNAL

SIGNAL NAME	NUMBER	DESCRIPTION
SIGHUP	1	This is known as the Hangup (HUP) signal. It stops the designated process and restarts it with the same PID.
SIGINT	2	This is the Interrupt (INT) signal. It is a weak kill signal that is not guaranteed to work, but it works in most cases.
SIGQUIT	3	This is known as the core dump. It terminates the process and saves the process info in memory, and then it saves this info in the current working directory to a file named core.
SIGTERM	15	This is the Termination (TERM) signal. It is the kill command's default signal.
SIGKILL	9	This is the absolute kill signal. It forces the process to stop by sending the process's resources to a special device, /dev/null.

```
Kali> kill -1 1234
```

```
Kali> kill -9 1234
```

If we don't know the process ID then we can use the ***killall*** command to kill the process. This command takes the name of the process, instead of PID as an argument.

```
Syntax> killall <option> <process name>
```

```
Kali> killall -9 zombieprocess
```

Finally, we can also terminate a process in the top command. Simply press the K key and enter the PID of the offending process.

RUNNING PROCESS IN THE BACKGROUND

```
Kali> nano script &
```

Now the nano opens and the terminal returns a new command prompt so we can enter other commands on our system while also editing our script.

MOVING A PROCESS TO THE FOREGROUND

```
Syntax> fg <PID>
```

```
Kali> fg 1234
```

SCHEDULING PROCESS

In Linux, you can accomplish this in at least two ways: with at and ***crond***. The at command is a daemon—a background process—useful for scheduling a job to run once at some point in the future. The ***crond*** is more suited for scheduling tasks to occur every day, week, or month. We use the at daemon to schedule the execution of a command or set of commands in the future. The time argument can be provided in various formats.

```
Syntax:
```

```
Kali> at 7:20am
```

```
at> /root/myscaningscript
```

TIME FORMATS ACCEPTED BY THE AT COMMAND

TIME FORMAT	MEANING
at 7:20pm	7:20 am at the current day
at 7:20pm June 25	7:20 pm on June 25
at noon	Noon at the current day
at noon June 25	Noon on June 25
at tomorrow	Run on tomorrow
at now + 20 minutes	Run in 20 mins from current time
at now + 10 hours	Run in 10 hours from now
at now + 5 days	Run in five days from the current date
at now + 3 weeks	Run in three weeks from the current date
at 7:20pm 05/25/2022	To run at 7:20 pm on June 25,2022