Linux Process States

Last modified: October 28, 2021

by Chin Ming Jun
(https://www.baeldung.com/linux/author/chinmingjun)

**Administration
(https://www.baeldung.com/linux/category/administration)**

**Processes
(https://www.baeldung.com/linux/category/processes)**

top (https://www.baeldung.com/linux/tag/top)

If you have a few years of experience in the Linux ecosystem, and you're interested in sharing that experience with the community, have a look at our **Contribution Guidelines** (/linux/contribution-guidelines).

1. Overview

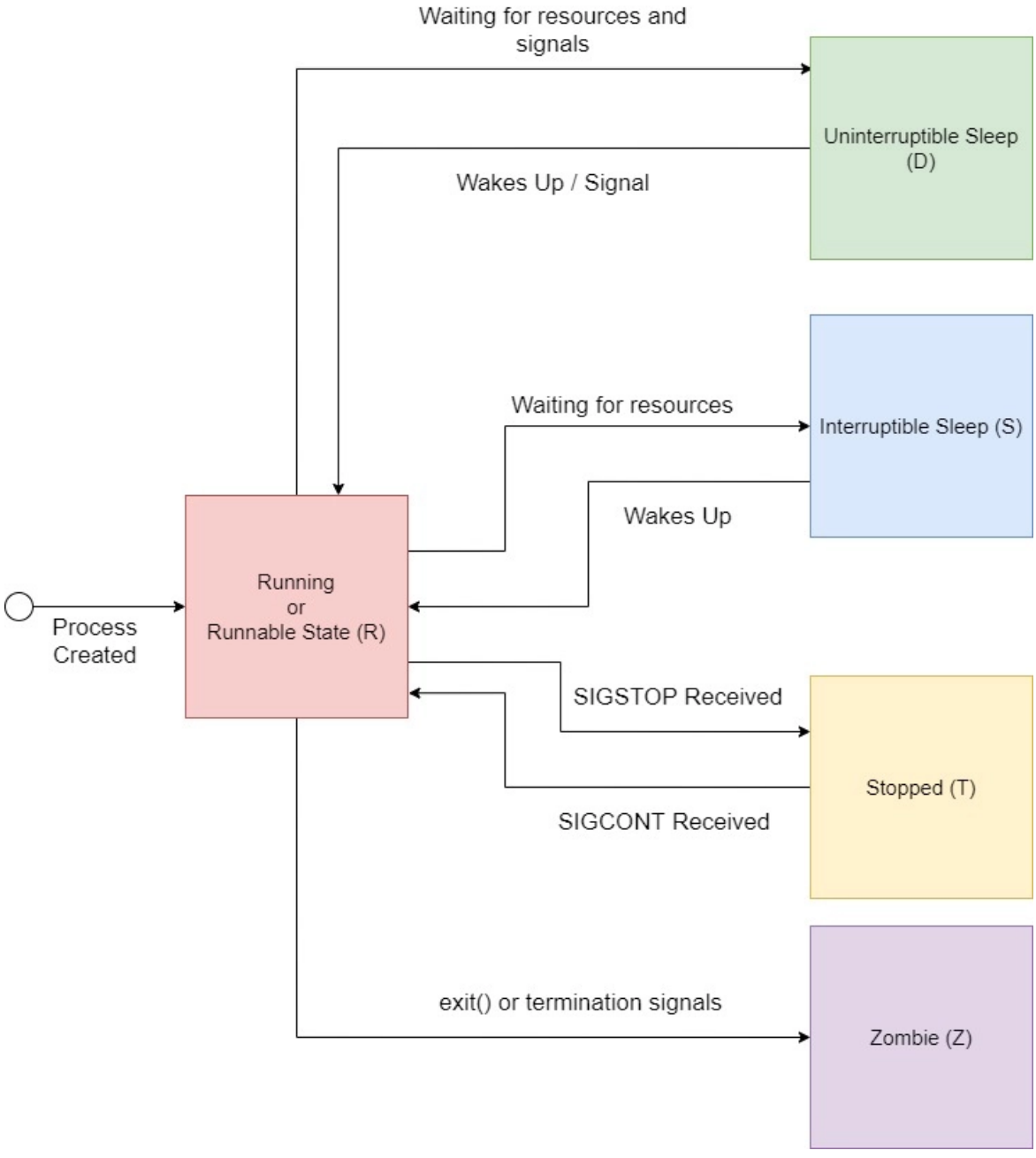
In this article, we'll learn about process states in Linux. Particularly, we'll learn about each of the five different states of a Linux process during the various parts of the lifecycle. Furthermore, we'll also look into different tools we can use to read the state of each process in our system.

2. The Linux Process States

In Linux, a process is an instance of executing a program or command. While these processes exist, they'll be in one of the five possible states:

- Running or Runnable (R)
- Uninterruptible Sleep (D)
- Interruptable Sleep (S)
- Stopped (T)
- Zombie (Z)

To visualize the lifecycle of the process, we can model it in a finite state machine:



For any Linux process, their starting point is the moment they are created. For example, a parent process can initiate a child process using the *fork()* (https://man7.org/linux/man-pages/man2/fork.2.html) system call. **Once it starts, the process goes into the running or runnable state.** While the process is running, it could come into a code path that requires it to wait for particular resources or signals before proceeding. While waiting for the resources, the process would voluntarily give up the CPU cycles by going into one of the two sleeping states.

Additionally, we could suspend a running process and put it into the stopped state. Usually, this is done by sending the *SIGSTOP* signal (https://man7.org/linux/man-pages/man7/signal.7.html) to the process. A process in this state will continue to exist until it is killed or resumed with *SIGCONT*. Finally, **the process completes its lifecycle when it's terminated and placed into a zombie state until its parent process clears it off the process table.**

2.1. Running or Runnable State (R)

When a new process is started, it'll be placed into the running or runnable state. **In the running state, the process takes up a CPU core to execute its code and logic.** However, the thread scheduling algorithm might force a running process to give up its execution right. This is to ensure each process can have a fair share of CPU resources. In this case, the process will be placed into a run queue, and its state is now a runnable state waiting for its turn to execute.

Although the running and runnable states are distinct, they are collectively grouped into a single state denoted by the character *R*.

2.2. Sleeping State: Interruptible (S) and Uninterruptible (D)

During process execution, it might come across a portion of its code where it needs to request external resources. Mainly, the request for these resources is IO-based such as to read a file from disk or make a network request. Since the process couldn't proceed without the resources, it would stall and do nothing. In events like these, they should give up their CPU cycles to other tasks that are ready to run, and hence they go into a sleeping state.

There are two different sleeping states: the uninterruptible sleeping state (D) and the interruptible sleeping state (S). **The uninterruptible sleeping state will only wait for the resources to be available before it transit into a runnable state, and it doesn't react to any signals**. On the other hand, **the interruptible sleeping state (s) will react to signals and the availability of resources.**

2.3. Stopped State (T)

From a running or runnable state, we could put a process into the stopped state (T) using the *SIGSTOP* or *SIGTSTP* signals. The difference between both signals is that we send the *SIGSTOP* is programmatic, such as running *kill -STOP [pid]*. Additionally, the process cannot ignore this signal and will go into the stopped state. On the other hand, we send the *SIGTSTP* signal using the keyboard *CTRL + Z*. **Unlike *SIGSTOP*, the process can optionally ignore this signal and continue to execute upon receiving *SIGTSTP*.**

While in this state, we could bring back the process into a running or runnable state by sending the *SIGCONT* signal.

2.4. Zombie State (Z)

When a process has completed its execution or is terminated, it'll send the *SIGCHLD* signal to the parent process and go into the zombie state. The zombie process, also known as a defunct process, will remain in this state until the parent process clears it off from the process table. To clear the terminated child process off the process table, the parent process must read the exit value of the child process using the *wait()* (https://man7.org/linux/man-pages/man2/wait.2.html) or *waitpid()* (https://linux.die.net/man/2/waitpid) system calls.

3. Checking Process State

There are multiple ways to check the state of a process is in Linux. For example, we can use command-line tools like *ps* and *top* to check the state of processes. Alternatively, we can consult the pseudo

status file for a particular PID.

3.1. Displaying Process State Using *ps*

To display process state using *ps* (/linux/ps-command), let's run the *ps* command to include a column that tells us the state of the process:

```
$ ps a
    PID TTY      STAT   TIME COMMAND
   2234 tty2     Ssl+   0:00 /usr/lib/gdm3/gdm-x-session --run-
script env GNOME_SHELL_SESSION_MODE=ubuntu /usr/bin/gnome-session
--systemd --session=ubuntu
   2237 tty2     Rl+    0:07 /usr/lib/xorg/Xorg vt2 -displayfd 3
-auth /run/user/1000/gdm/Xauthority -background none -noreset -
keeptty -verbose 3
   2287 tty2     Sl+    0:00 /usr/libexec/gnome-session-binary --
systemd --systemd --session=ubuntu
   2982 pts/0    Ss     0:00 bash
   3467 pts/0    R+     0:00 ps a
```

The first letter of the value under the *STAT* column indicates the state that the process is in. For example, the process with PID 2234 is currently in an interruptible sleeping state, as denoted by the character S. Besides that, we can also observe that process 2237 is currently in the running or runnable state.

Additionally, we can see that there are additional characters besides each of the state characters. These characters indicate several attributes of the state of the process. For example, the lower capital letter *s* means the process is the session leader. For a comprehensive list of the meaning of each of the characters, we can find it on the official man page (https://man7.org/linux/man-pages/man1/ps.1.html#PROCESS_STATE_CODES).

3.2. Using the *top* Command

In Linux, the *top* (/linux/top-command) command-line tool displays the process details in a real-time fashion. It shows different aspects of the system, such as memory and CPU usage of individual processes. To see the process state, let's run *top* in the terminal:

```
Tasks: 183 total,   1 running, 182 sleeping,   0 stopped,   0
zombie
%Cpu(s):  0.7 us,  1.1 sy,  0.0 ni, 97.1 id,  0.4 wa,  0.0 hi,
0.7 si,  0.0 st
MiB Mem :   3936.4 total,   1925.0 free,    850.6 used,   1160.8
buff/cache
MiB Swap:   2048.0 total,   2048.0 free,      0.0 used.   2834.2
avail Mem

    PID USER  PR  NI    VIRT    RES    SHR S  %CPU  %MEM
TIME+ COMMAND
   2237 bob   20   0  252252  81740  49204 S   2.3   2.0
0:09.37 Xorg
   2519 bob   20   0 3428664 375256 125080 S   2.0   9.3
0:19.57 gnome-shell
   2909 bob   20   0  966852  49944  37308 S   1.0   1.2
0:02.28 gnome-terminal-
      1 root  20   0  103500  13312   8620 S   0.7   0.3
0:04.44 systemd
   3588 bob   20   0   20600   3936   3380 R   0.3   0.1
0:00.01 top
      2 root  20   0       0      0      0 S   0.0   0.0
0:00.00 kthreadd
      3 root   0 -20       0      0      0 I   0.0   0.0
0:00.00 rcu_gp
```

At the bottom section of the output of the *top* command, we can find the *S* column, which shows the state of each process. Contrary to the *ps* command, the *top* command displays the state of each process without additional process attributes.

3.3. The */proc* Pseudo File

The */proc* pseudo filesystem contains all the information about the processes in our system. Hence, we could directly read the state of a process through this pseudo filesystem. The downside of this approach is we'll first need to know the PID of the process before we can read its state.

To obtain the state of a process, we can extract the value from its pseudo *status* file under */proc/{pid}/status*. For example, we can get the state of the process with PID 2519 by reading the file */proc/2519/status*:

```
$ cat /proc/2519/status | grep State
State:  S (sleeping)
```

4. Summary

In this tutorial, we've looked at the lifecycle of a Linux process. Furthermore, we've learned how we can model a Linux process lifecycle as a finite state machine. Then, we've looked at the five different states as a Linux process undergoes the entire lifecycle. Finally, we ended the article with demonstrations on getting the Linux process state using various tools such as *ps*, *top,* and the */proc* pseudo-file.

If you have a few years of experience in the Linux ecosystem, and you're interested in sharing that experience with the community, have a look at our **Contribution Guidelines** (/linux/contribution-guidelines).

Comments are closed on this article!

CATEGORIES

ADMINISTRATION (/LINUX/CATEGORY/ADMINISTRATION)
FILES (/LINUX/CATEGORY/FILES)
FILESYSTEMS (/LINUX/CATEGORY/FILESYSTEMS)
INSTALLATION (/LINUX/CATEGORY/INSTALLATION)
NETWORKING (/LINUX/CATEGORY/NETWORKING)
PROCESSES (/LINUX/CATEGORY/PROCESSES)
SCRIPTING (/LINUX/CATEGORY/SCRIPTING)
SEARCH (/LINUX/CATEGORY/SEARCH)
SECURITY (/LINUX/CATEGORY/SECURITY)
WEB (/LINUX/CATEGORY/WEB)

SERIES

LINUX FILES (/LINUX/LINUX-FILES-SERIES)
LINUX SCRIPTING (/LINUX/LINUX-SCRIPTING-SERIES)

ABOUT

ABOUT BAELDUNG (/ABOUT)
THE FULL ARCHIVE (HTTPS://WWW.BAELDUNG.COM/LINUX/FULL_ARCHIVE)
WRITE FOR BAELDUNG (/LINUX/CONTRIBUTION-GUIDELINES)
EDITORS (HTTPS://WWW.BAELDUNG.COM/EDITORS)

TERMS OF SERVICE (HTTPS://WWW.BAELDUNG.COM/TERMS-OF-SERVICE)
PRIVACY POLICY (HTTPS://WWW.BAELDUNG.COM/PRIVACY-POLICY)
COMPANY INFO (HTTPS://WWW.BAELDUNG.COM/BAELDUNG-COMPANY-INFO)
CONTACT (/CONTACT)