How To Manage or Kill a Process

Introduction

A Linux computer, like Apple or Windows machines you may be more familiar with, runs applications. To the computer, these are considered "processes". Each process is assigned a process ID, or PID. This is how the operating system identifies and keeps track of processes. In this guide, we will discuss some simple aspects of process management as well as how to prioritize which processes have first call on CPU resources.

This introduction assumes you have completed an introductory computer course like cs8 or cs16. If you have any questions, please talk to your TA.

Basic Command Line Signals

All processes in Linux respond to signals. Signals are an os-level way of telling programs to terminate or modify their behavior.

pgrep

A quick way of getting the PID of a process is with the `pgrep` command. The first process spawned at boot, called init, is given the PID of "1".

```
pgrep init
1
```

Init is responsible for spawning every other process on the system. The later processes are given larger PID numbers.

A process's parent is the process that was responsible for spawning it. If a process's parent is killed, then the child processes also die. The parent process's PID is referred to as the PPID.

You can see PID and PPID in the column headers in many process management applications, including top, htop and ps.

Any communication between the user and the operating system about processes involves translating between process names and PIDs at some point during the operation. This is why utilities tell you the PID.

Top also has a column called niceness or ni, This is process priority, more will be said on this later.

```
PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
4586  ipc-adm+  20   0 1303900 605152  92844 S  30,6 29,3   3:52.88 firefox
```

w

This command is similar to top, but shows you less statistics and does not update in real time.

Ps

Top provides a nice interface to view running processes similar to a graphical task manager.

However, this tool is not always flexible enough to adequately cover all scenarios. A powerful command called ps is often used.

Ps stands for process server. When called without arguments, the output can be less than helpful: ps

```
PID TTY TIME CMD
1017 pts/0  00:00:00 bash
1262 pts/0  00:00:00 ps
```
This output shows all processes associated with the current user and terminal session. To see a more complete picture of the processes on this system, run the following:

```
ps aux
```

<table>
<thead>
<tr>
<th>USER</th>
<th>PID</th>
<th>%CPU</th>
<th>%MEM</th>
<th>VSZ</th>
<th>RSS</th>
<th>TTY</th>
<th>STAT</th>
<th>START</th>
<th>TIME</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>root</td>
<td>1</td>
<td>0.0</td>
<td>0.2</td>
<td>24188</td>
<td>2120</td>
<td>?</td>
<td>Ss</td>
<td>14:28</td>
<td>0:00</td>
<td>/sbin/init</td>
</tr>
<tr>
<td>root</td>
<td>2</td>
<td>0.0</td>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>?</td>
<td>S</td>
<td>14:28</td>
<td>0:00</td>
<td>[bash]</td>
</tr>
<tr>
<td>root</td>
<td>3</td>
<td>0.0</td>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>?</td>
<td>S</td>
<td>14:28</td>
<td>0:00</td>
<td>[matlab]</td>
</tr>
<tr>
<td>root</td>
<td>6</td>
<td>0.0</td>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>?</td>
<td>S</td>
<td>14:28</td>
<td>0:00</td>
<td>[mozilla]</td>
</tr>
<tr>
<td>root</td>
<td>7</td>
<td>0.0</td>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>?</td>
<td>S</td>
<td>14:28</td>
<td>0:00</td>
<td>[chrome]</td>
</tr>
<tr>
<td>root</td>
<td>8</td>
<td>0.0</td>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>?&lt;</td>
<td>S&lt;</td>
<td>14:28</td>
<td>0:00</td>
<td>[cpuset]</td>
</tr>
</tbody>
</table>

```
ps aux | grep proccess_name
```

We sometimes have too many processes to see clearly, in those cases we can search ps aux by process name.

**Command Line Processes Signals**

The most common way of passing end process signals to a program is with the kill command.

As you might expect, the default functionality of this utility is to attempt to kill a process:

The command makes it possible to continue running a server without the need of reboot after a major change/update.

In order to kill a process, we need to know the Process ID of the process.

Init is the master process and usually can not be killed using the `kill` command, which insures that the master process don’t gets killed accidentally.

The exception is that Init decides and allows itself to be killed, where `kill` is merely a request for a shutdown.

**kill PID_of_target_process**

This sends the TERM signal to the process. The TERM signal tells the process to please terminate. This allows the program to perform clean-up operations and exit smoothly.

If the program is misbehaving and does not exit when given the TERM signal, we can escalate the signal by passing the KILL signal:

You can list all of the signals that are possible to send with kill by typing:

```
kill -l
```

**kill -KILL PID_of_target_process**

This is a special signal that is not sent to the program.

Instead, it is given to the operating system kernel, which shuts down the process. This is used to bypass programs that ignore the signals sent to them.

Each signal has an associated number that can be passed instead of the name. For instance, You can pass `-15` instead of `-TERM`, and `-9` instead of `-KILL`.

**kill [signal or option] PID(s)_of_target_process(s)**

Additionally you can kill multiple processes at once, or use a host of special signals

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Signal Value</th>
<th>Behaviour</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGHUP</td>
<td>1</td>
<td>Hangup</td>
</tr>
<tr>
<td>SIGKILL</td>
<td>9</td>
<td>Kill Signal</td>
</tr>
<tr>
<td>SIGTERM</td>
<td>15</td>
<td>Terminate</td>
</tr>
</tbody>
</table>

SIGTERM is the default and safest way to kill a process. SIGHUP is less secure way of killing a process as SIGTERM.

SIGKILL is the most unsafe way among the above three, to kill a process which terminates a process without saving.
for example to kill multiple processes at one time:

```
kill PID1 PID2 PID3
```

or

```
kill -9 PID1 PID2 PID3
```

**Process Signals by Name**

The conventional way of sending signals is through the use of PIDs, there are also methods of doing this with process names.

**pkill**

The `pkill` command works in almost exactly the same way as `kill`, but it operates on a process name instead. The following examples are involve terminating the `firefox` process as an example:

```
pkill -9 firefox
```

The above command is the equivalent of:

```
kill -9 `pgrep firefox`
```

If you would like to send a signal to every instance of a certain process, you can use the `killall` command:

```
killaall firefox
```

**How To Adjust Process Priorities**

Often, you will want to adjust which processes are given priority to conserve CPU resources. Some processes might be considered critical for your situation, while others may be executed whenever there is leftover resources.

Linux controls priority through a value called niceness. High priority tasks are considered less nice, because they don't share resources as well. Low priority processes, are nice because they take minimal resources.

When we ran `top` at the beginning of the article, there was a column marked "NI". This is the nice value of the process:

Nice values can range between "-19/-20" (highest priority) and "19/20" (lowest priority) depending on the system.

Zero in this field simply means priority will not be adjusted in determining a task's dispatch-ability

```
PID USER       PR  NI  VIRT  RES  SHR S %CPU %MEM     TIME+ COMMAND
4586 ipc-adm+  20   0  1303900 605152  92844 S  30,6 29,3   3:52.88 firefox
```

**PR - Priority** - The scheduling priority of the task. If you see 'rt' in this field, it means the task is running under 'real time' scheduling priority.

PR is the process's actual priority that use by Linux kernel. In Linux system priorities are 0 to 139. As Mentioned above nice value range is -20 to +19 where -20 is highest, 0 default and +19 is lowest. relation between nice value and priority is:

\[
PR = 20 + NI
\]

To run a program with a certain nice value, we can use the `nice` command. To run a program a low priority or n=15:

```
nice -n 15 command_to_execute
```
This only works when beginning a new program. To alter the nice value of a program that is already executing, we use a tool called renice:

```
renice -n 15 -p PID_to_prioritize
```

for the above example Firefox's PID is 4586, however this value will likely be different every time you run firefox depending on what else is running. Always check PID before using the renice command

Note: While nice operates with a command name by necessity, renice operates by calling the process PID

**Related articles**

- Securing Linux Computer
- How To Manage or Kill a Process
- How to do X11 Forwarding on a Mac
- Google chrome not opening?
- Firefox not saving/loading up bookmarks.