Learn Linux, 101: Process execution priorities

Keeping your eye on what's going on

Ian Shields
Senior Programmer
IBM

02 February 2010

Learn how to set and change process priorities so that applications get as much processing time as they need. You can use this material in this article to study for the LPI® 101 exam for Linux system administrator certification, or just to learn for fun.

View more content in this series

Overview

This article grounds you in the basic Linux techniques for managing execution process priorities. Learn to:

- Understand process priorities
- Set process priorities
- Change process priorities

This article helps you prepare for Objective 103.6 in Topic 103 of the Linux Professional Institute's Junior Level Administration (LPIC-1) exam 101. The objective has a weight of 2.

Prerequisites

To get the most from the articles in this series, you should have a basic knowledge of Linux and a working Linux system on which you can practice the commands covered in this article. Sometimes different versions of a program will format output differently, so your results may not always look exactly like the listings and figures shown here. The results in the examples shown here were obtained on a Ubuntu 9.10 (Karmic Koala) distribution. This article builds on the concepts discussed in the previous article "Learn Linux 101: Create, monitor, and kill processes."

Knowing your priorities

Connect with Ian

Ian is one of our most popular and prolific authors. Browse all of Ian's articles on developerWorks. Check out Ian's profile and connect with him, other authors, and fellow readers in My developerWorks.

Linux, like most modern operating systems, can run multiple processes. It does this by sharing the CPU and other resources among the processes. If one process can use 100% of the CPU, then other processes may become unresponsive.

If you run the `top` command, its default is to display processes in decreasing order according to their CPU usage, as shown in Listing 1. In the previous article, "Learn Linux 101: Create, monitor, and kill processes," we showed a Poor Man's Clock script, which prints the time on the console every 30 seconds and does nothing for the rest of the time. If we ran that process, it probably wouldn't make it onto the output list from `top` because the process spends most of its time not using the CPU.

Listing 1. Typical output from top on a Linux workstation

```
top - 08:00:52 up 1 day, 10:20,  5 users,  load average: 0.04, 0.08, 0.04
Tasks: 172 total,   1 running, 171 sleeping,   0 stopped,   0 zombie
Cpu(s):  3.7%us,  0.3%sy,  0.0%ni, 95.6%id,  0.0%wa,  0.0%hi,  0.3%si,  0.0%st
Mem:   4057976k total,  1777976k used,  2280000k free,   225808k buffers
Swap: 10241428k total,        0k used, 10241428k free,   655796k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
11220 ian       20   0  555m 106m  27m S    8  2.7  36:06.16 firefox
    7 root      15  -5     0    0    0 S    1  0.0  10:59.36 ksoftirqd/1
10849 ian       20   0  212m  15m  10m S    0  0.4   0:08.11 gnome-terminal
    1 root      20   0 19584 1888 1196 S    0  0.0   0:00.83 init
    2 root      15  -5     0    0    0 S    0  0.0   0:00.01 kthreadd
    3 root      RT  -5     0    0    0 S    0  0.0   0:00.02 migration/0
    4 root      15  -5     0    0    0 S    0  0.0   0:01.08 ksoftirqd/0
    5 root      RT  -5     0    0    0 S    0  0.0   0:00.00 watchdog/0
    6 root      RT  -5     0    0    0 S    0  0.0   0:00.03 migration/1
```

Your system may have many commands that are capable of using lots of CPU. Examples include movie editing tools, and programs to convert between different image types or between different sound encoding, such as mp3 to ogg.

About this series

This series of articles helps you learn Linux system administration tasks. You can also use the material in these articles to prepare for Linux Professional Institute Certification level 1 (LPIC-1) exams.

See our series roadmap for a description of and link to each article in this series. The roadmap is in progress and reflects the latest (April 2009) objectives for the LPIC-1 exams: as we complete articles, we add them to the roadmap. In the meantime, though, you can find earlier versions of similar material, supporting previous LPIC-1 objectives prior to April 2009, in our LPI certification exam prep tutorials.

When you only have one or a limited number of CPUs, you need to decide how to share those limited CPU resources among several competing processes. This is generally done by selecting one process for execution and letting it run for a short period (called a *timeslice*), or until it needs to wait for some event, such as IO to complete. To ensure that important processes don't get starved out by CPU hogs, the selection is done based on a *scheduling priority*. The **NI** column in Listing 1 above, shows the scheduling priority or *niceness* of each process. Niceness generally ranges from -20 to 19, with -20 being the most favorable or highest priority for scheduling and 19 being the least favorable or lowest priority.

Using ps to find niceness

Develop skills on this topic

This content is part of a progressive knowledge path for advancing your skills. See Basics of Linux system administration: Working at the console

In addition to the `top` command, you can also display niceness values using the `ps` command. You can either customize the output as you saw in the article "Learn Linux 101: Create, monitor, and kill processes," or you can just use the `-l` option to get a long listing. The output of `ps -l` is shown in Listing 2. As with `top`, look for the niceness value in the **NI** column.

Listing 2. Using ps to find niceness

```
ian@attic4:~$ ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000 26502 26501  0  80   0 -  5368 wait   pts/4    00:00:00 bash
0 R  1000 27046 26502  0  80   0 -  1684 -      pts/4    00:00:00 ps
```

Default niceness

You may have guessed from Listing 1 or Listing 2 that the default niceness, at least for processes started by regular users, is 0. This is usually the case on current Linux systems. You can verify the value for your shell and system by running the `nice` command with no parameters as shown in Listing 3.

Listing 3. Checking default niceness

```
ian@attic4:~$ nice
0
```

Setting priorities

Before we look at how to set or change niceness values, let's build a little CPU-intensive script that will show how niceness really works.

A CPU-intensive script

We'll create a small script that just uses CPU and does little else. The script takes two inputs, a count and a label. It prints the label and the current date and time, then spins, decrementing the

count till it reaches 0, and finally prints the label and the date again. This script shown in Listing 4 has no error checking and is not very robust, but it illustrates our point.

Listing 4. CPU-intensive script

```
ian@attic4:~$ echo 'x="$1"'>count1.sh
ian@attic4:~$ echo 'echo "$2" $(date)'>>count1.sh
ian@attic4:~$ echo 'while [ $x -gt 0 ]; do x=$(( x-1 ));done'>>count1.sh
ian@attic4:~$ echo 'echo "$2" $(date)'>>count1.sh
ian@attic4:~$ cat count1.sh
x="$1"
echo "$2" $(date)
while [ $x -gt 0 ]; do x=$(( x-1 ));done
echo "$2" $(date)
```

If you run this on your own system, you might see output similar to Listing 5. Depending on the speed of your system, you may have to increase the count value to even see a difference in the times. This script uses lots of CPU, as we'll see in a moment. If your default shell is not Bash, and if the script does not work for you, then use the second form of calling shown below. If you are not using your own workstation, make sure that it is okay to use lots of CPU before you run the script.

Listing 5. Running count1.sh

```
ian@attic4:~$ sh count1.sh 10000 A
A Wed Jan 20 08:34:16 EST 2010
A Wed Jan 20 08:34:16 EST 2010
ian@attic4:~$ bash count1.sh 99000 A
A Wed Jan 20 08:34:20 EST 2010
A Wed Jan 20 08:34:22 EST 2010
```

So far, so good. Now let's create a command list to run the script in background and launch the `top` command to see how much CPU the script is using. (See the previous article "Learn Linux 101: The Linux command line" for a refresher on command lists.) The command list is shown in Listing 6 and the output from `top` in Listing 7.

Listing 6. Running count1.sh and top

```
ian@attic4:~$ (sh count1.sh 5000000 A&);top
```

Listing 7. Using lots of CPU

```
top - 15:41:15 up 1 day, 17:59,  6 users,  load average: 0.20, 0.06, 0.02
Tasks: 169 total,   2 running, 167 sleeping,   0 stopped,   0 zombie
Cpu(s): 52.1%us,  0.7%sy,  0.0%ni, 47.3%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   4057976k total,  1393772k used,  2664204k free,   235596k buffers
Swap: 10241428k total,        0k used, 10241428k free,   662592k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
26756 ian       20   0  4004  588  496 R  100  0.0   0:03.53 sh
11220 ian       20   0  555m 101m  27m S    5  2.6  57:58.07 firefox
26757 ian       20   0 19132 1364  980 R    0  0.0   0:00.03 top
    1 root      20   0 19584 1888 1196 S    0  0.0   0:00.89 init
    2 root      15  -5     0    0    0 S    0  0.0   0:00.01 kthreadd
```

Not bad. We are using 100% of one of the CPUs on this system with just a simple script. If you want to stress multiple CPUs, you can add an extra invocation of `count1.sh` to the command list. If

we had a long running job such as this, we might find that it interfered with our ability (or the ability of other users) to do other work on our system.

Using nice to set priorities

Now that we can keep a CPU busy for a while, we'll see how to set a priority for a process. To summarize what we've learned so far:

- Linux and UNIX® systems use a priority system with 40 priorities, ranging from -20 (highest priority) to 19 (lowest priority.
- Processes started by regular users usually have priority 0.
- The `ps` command can display the priority (nice, or NI, level, for example) using the `-l` option.
- The `nice` command displays our default priority.

The `nice` command can also be used to start a process with a different priority. You use the `-n` or (`--adjustment`) option with a positive value to increase the priority value and a negative value to decrease it. Remember that processes with the lowest priority value run at highest scheduling priority, so think of increasing the priority value as being *nice* to other processes. Note that you usually need to be the superuser (root) to specify negative priority adjustments. In other words, regular users can usually only make their processes nicer.

To demonstrate the use of `nice` to set priorities, let's start two copies of the `count1.sh` script in different subshells at the same time, but give one the maximum niceness of 19. After a second we'll use `ps -l` to display the process status, including niceness. Finally, we'll add an arbitrary 30-second sleep to ensure the command sequence finishes after the two subshells do. That way, we won't get a new prompt while we're still waiting for output. The result is shown in Listing 8.

Listing 8. Using nice to set priorities for a pair of processes

```
ian@attic4:~$ (sh count1.sh 2000000 A&);(nice -n 19 sh count1.sh 2000000 B&);\
> sleep 1;ps -l;sleep 10
A Thu Jan 21 14:38:39 EST 2010
B Thu Jan 21 14:38:39 EST 2010
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 R  1000   946     1 99  80   0 -  1001 -      pts/3    00:00:01 sh
0 R  1000   948     1 99  99  19 -  1001 -      pts/3    00:00:01 sh
0 R  1000   952 32408  0  80   0 -  1684 -      pts/3    00:00:00 ps
0 S  1000 32408 32407  0  80   0 -  5368 wait   pts/3    00:00:02 bash
A Thu Jan 21 14:38:45 EST 2010
B Thu Jan 21 14:38:45 EST 2010
```

Are you surprised that the two jobs finished at the same time? What happened to our priority setting? Remember that the script occupied **one** of our CPUs. This particular system runs on an AMD Athlon™ 7750 dual-core processor, which is very lightly loaded, so each core ran one process, and there wasn't any need to prioritize them.

So let's try starting four processes at four different niceness levels (0, 6, 12, and18) and see what happens. We'll increase the busy count parameter for each so they run a little longer. Before you look at Listing 9, think about what you might expect, given what you've already seen.

Listing 9. Using nice to set priorities for four of processes

```
ian@attic4:~$ (sh count1.sh 5000000 A&);(nice -n 6 sh count1.sh 5000000 B&);\
> (nice -n 12 sh count1.sh 5000000 C&);(nice -n 18 sh count1.sh 5000000 D&);\
> sleep 1;ps -l;sleep 30
A Thu Jan 21 16:06:00 EST 2010
C Thu Jan 21 16:06:00 EST 2010
D Thu Jan 21 16:06:00 EST 2010
B Thu Jan 21 16:06:00 EST 2010
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 R  1000  1422     1 94  80   0 -  1001 -      pts/3    00:00:00 sh
0 R  1000  1424     1 42  86   6 -  1001 -      pts/3    00:00:00 sh
0 R  1000  1427     1 56  92  12 -  1001 -      pts/3    00:00:00 sh
0 R  1000  1431     1 14  98  18 -  1001 -      pts/3    00:00:00 sh
0 R  1000  1435 32408  0  80   0 -  1684 -      pts/3    00:00:00 ps
0 S  1000 32408 32407  0  80   0 -  5368 wait   pts/3    00:00:02 bash
A Thu Jan 21 16:06:14 EST 2010
B Thu Jan 21 16:06:17 EST 2010
C Thu Jan 21 16:06:26 EST 2010
D Thu Jan 21 16:06:30 EST 2010
```

With four different priorities, we see the effect of the different niceness values as each job finishes in priority order. Try experimenting with different nice values to demonstrate the different possibilities for yourself.

A final note on starting processes with `nice`; as with the `nohup` command, you cannot use a command list or a pipeline as the argument of `nice`.

Changing priorities

renice

If you happen to start a process and realize that it should run at a different priority, there is a way to change it after it has started, using the `renice` command. You specify an absolute priority (and not an adjustment) for the process or processes to be changed as shown in Listing 10.

Listing 10. Using renice to change priorities

```
ian@attic4:~$ sh count1.sh 10000000 A&
[1] 1537
ian@attic4:~$ A Thu Jan 21 16:17:16 EST 2010
sh count1.sh 1renice 1 1537;ps -l 1537
1537: old priority 0, new priority 1
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY        TIME CMD
0 R  1000  1537 32408 99  81   1 -  1001 -      pts/3      0:13 sh count1.sh 100
ian@attic4:~$ renice +3 1537;ps -l 1537
1537: old priority 1, new priority 3
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY        TIME CMD
0 R  1000  1537 32408 99  83   3 -  1001 -      pts/3      0:18 sh count1.sh 100
```

Remember that you have to be the superuser to give your processes higher scheduling priority and make them less nice.

You can find more information on `nice` and `renice` in the man pages.

Resources

Learn

- Use the developerWorks roadmap for LPIC-1 to find the developerWorks articles to help you study for LPIC-1 certification based on the April 2009 objectives.
- At the LPIC Program site, find detailed objectives, task lists, and sample questions for the three levels of the Linux Professional Institute's Linux system administration certification. In particular, see their April 2009 objectives for LPI exam 101 and LPI exam 102. Always refer to the LPIC Program site for the latest objectives.
- Develop and deploy your next app on the IBM Bluemix cloud platform.
- Review the entire LPI exam prep series on developerWorks to learn Linux fundamentals and prepare for system administrator certification based on earlier LPI exam objectives prior to April 2009.
- In "Basic tasks for new Linux developers" (developerWorks, March 2005), learn how to open a terminal window or shell prompt and much more.
- The Linux Documentation Project has a variety of useful documents, especially its HOWTOs.
- In the developerWorks Linux zone, find more resources for Linux developers, and scan our most popular articles and tutorials.
- See all Linux tutorials and Linux tips on developerWorks.
- Stay current with developerWorks technical events and Webcasts.
- Follow developerWorks on Twitter.

Get products and technologies

- With IBM trial software, available for download directly from developerWorks, build your next development project on Linux.

Discuss

- Participate in the discussion forum for this content.
- Get involved in the My developerWorks community. Connect with other developerWorks users while exploring the developer-driven blogs, forums, groups, and wikis.

About the author

Ian Shields

Ian Shields works on a multitude of Linux projects for the developerWorks Linux zone. He is a Senior Programmer at IBM at the Research Triangle Park, NC. He joined IBM in Canberra, Australia, as a Systems Engineer in 1973, and has since worked on communications systems and pervasive computing in Montreal, Canada, and RTP, NC. He has several patents and has published several papers. His undergraduate degree is in pure mathematics and philosophy from the Australian National University. He has an M.S. and Ph.D. in computer science from North Carolina State University. Learn more about Ian in Ian's profile on developerWorks Community.