UNIX AS A SECOND LANGUAGE
By Sandra Henry-Stocker, Unix Dweeb, Network World
JUN 25, 2020 8:59 AM PDT

How to stress test your Linux system

Stressing your Linux servers can be a good idea if you'd like to see how well they function when they're loaded down. In this post, we'll look at some tools that can help you add stress and gauge the results.

Why would you ever want to stress your Linux system? Because sometimes you might want to know how a system will behave when it's under a lot of pressure due to a large number of running processes, heavy network traffic, excessive memory use and so on. This kind of testing can help to ensure that a system is ready to "go public".

If you need to predict how long applications might take to respond and what, if any, processes might fail or run slowly under a heavy load, doing the stress testing up front can be a very good idea.

Fortunately for those who need to be able to predict how a Linux system will react under stress, there are some helpful techniques you can employ and tools that you can use to make the process easier. In this post, we examine a few options.

Do it yourself loops

This first technique involves running some loops on the command line and watching how they affect the system. This technique burdens the CPUs by greatly increasing the load. The results can easily be seen using the **uptime** or similar commands.

In the command below, we kick off four endless loops. You can increase the number of loops by adding digits or using a **bash** expression like **{1..6}** in place of "1 2 3 4".

```
for i in 1 2 3 4; do while : ; do : ; done & done
```

Typed on the command line, this command will start four endless loops in the background.

```
$ for i in 1 2 3 4; do while : ; do : ; done & done
[1] 205012
[2] 205013
[3] 205014
[4] 205015
```

In this case, jobs 1-4 were kicked off. Both the job numbers and process IDs are displayed.

To observe the effect on load averages, use a command like the one shown below. In this case, the **uptime** command is run every 30 seconds:


```
$ while true; do uptime; sleep 30; done
```

If you intend to run tests like this periodically, you can put the loop command into a script:

```
#!/bin/bash

while true
do
  uptime
  sleep 30
done
```

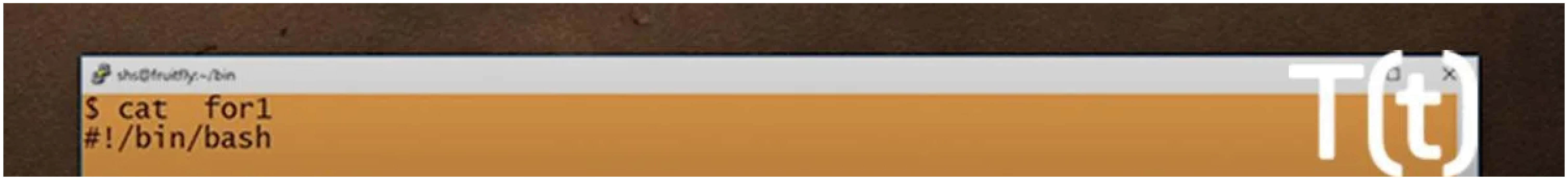
In the output, you can see how the load averages increase and then start going down again once the loops have been ended.

```
 11:25:34 up 5 days, 17:27,  2 users,  load average: 0.15, 0.14, 0.08
 11:26:04 up 5 days, 17:27,  2 users,  load average: 0.09, 0.12, 0.08
 11:26:34 up 5 days, 17:28,  2 users,  load average: 1.42, 0.43, 0.18
 11:27:04 up 5 days, 17:28,  2 users,  load average: 2.50, 0.79, 0.31
 11:27:34 up 5 days, 17:29,  2 users,  load average: 3.09, 1.10, 0.43
 11:28:04 up 5 days, 17:29,  2 users,  load average: 3.45, 1.38, 0.54
 11:28:34 up 5 days, 17:30,  2 users,  load average: 3.67, 1.63, 0.66
 11:29:04 up 5 days, 17:30,  2 users,  load average: 3.80, 1.86, 0.76
 11:29:34 up 5 days, 17:31,  2 users,  load average: 3.88, 2.06, 0.87
 11:30:04 up 5 days, 17:31,  2 users,  load average: 3.93, 2.25, 0.97
 11:30:34 up 5 days, 17:32,  2 users,  load average: 3.64, 2.35, 1.04 <== loops
 11:31:04 up 5 days, 17:32,  2 users,  load average: 2.20, 2.13, 1.01     stopped
 11:31:34 up 5 days, 17:33,  2 users,  load average: 1.40, 1.94, 0.98
```

Because the loads shown represent averages over 1, 5 and 15 minutes, the values will take a while to go back to what is likely normal for the system.

To stop the loops, issue a **kill** command like this one below – assuming the job numbers are 1-4 as was shown earlier in this post. If you're unsure, use the jobs command to verify the job IDs.

```
$ kill %1 %2 %3 %4
```



Specialized tools for adding stress

Another way to create system stress involves using a tool that was specifically built to stress the system for you. One of these is called “stress” and can stress the system in a number of ways. The **stress** tool is a workload generator that provides CPU, memory and disk I/O stress tests.

With the **--cpu** option, the **stress** command uses a square-root function to force the CPUs to work hard. The higher the number of CPUs specified, the faster the loads will ramp up.

A second **watch-it** script (**watch-it-2**) can be used to gauge the effect on system memory usage. Note that it uses the **free** command to see the effect of the stressing.

```
$ cat watch-it-2
#!/bin/bash

while true
do
  free
  sleep 30
done
```

UNITED STATES

Kicking off and observing the stress:

```
$ stress --cpu 2
```

```
$ ./watch-it
 13:09:14 up 5 days, 19:10,  2 users,  load average: 0.00, 0.00, 0.00
 13:09:44 up 5 days, 19:11,  2 users,  load average: 0.68, 0.16, 0.05
 13:10:14 up 5 days, 19:11,  2 users,  load average: 1.20, 0.34, 0.12
 13:10:44 up 5 days, 19:12,  2 users,  load average: 1.52, 0.50, 0.18
 13:11:14 up 5 days, 19:12,  2 users,  load average: 1.71, 0.64, 0.24
 13:11:44 up 5 days, 19:13,  2 users,  load average: 1.83, 0.77, 0.30
```

The more CPUs specified on the command line, the faster the load will ramp up.

```
$ stress --cpu 4
$ ./watch-it
 13:47:49 up 5 days, 19:49,  2 users,  load average: 0.00, 0.00, 0.00
 13:48:19 up 5 days, 19:49,  2 users,  load average: 1.58, 0.38, 0.13
 13:48:49 up 5 days, 19:50,  2 users,  load average: 2.61, 0.75, 0.26
 13:49:19 up 5 days, 19:50,  2 users,  load average: 3.16, 1.06, 0.38
 13:49:49 up 5 days, 19:51,  2 users,  load average: 3.49, 1.34, 0.50
 13:50:19 up 5 days, 19:51,  2 users,  load average: 3.69, 1.60, 0.61
```

The **stress** command can also stress the system by adding I/O and memory load with its **--io** (input/output) and **--vm** (memory) options.

In this next example, this command for adding memory stress is run, and then the **watch-it-2** script is started:

```
$ stress --vm 2
```

```
$ watch-it-2                  UNITED STATES ▾
              total        used        free      shared  buff/cache   available
Mem:        6087064      662160     2519164        8868     2905740     5117548
Swap:       2097148           0     2097148
              total        used        free      shared  buff/cache   available
Mem:        6087064      803464     2377832        8864     2905768     4976248
Swap:       2097148           0     2097148
              total        used        free      shared  buff/cache   available
Mem:        6087064      968512     2212772        8864     2905780     4811200
Swap:       2097148           0     2097148
```

Another option for **stress** is to use the **--io** option to add input/output activity to the system. In this case, you would use a command like this:

```
$ stress --io 4
```

You could then observe the stressed IO using **iotop**. Note that **iotop** requires root privilege.

before

```
$ sudo iotop -o
Total DISK READ:         0.00 B/s | Total DISK WRITE:        19.36 K/s
Current DISK READ:       0.00 B/s | Current DISK WRITE:      27.10 K/s
    TID  PRIO  USER     DISK READ  DISK WRITE  SWAPIN     IO>    COMMAND
 269308 be/4 root        0.00 B/s    0.00 B/s  0.00 %  1.24 % [kworker~fficient]
    283 be/3 root        0.00 B/s   19.36 K/s  0.00 %  0.26 % [jbd2/sda1-8]
```

after

```
Total DISK READ:         0.00 B/s | Total DISK WRITE:         0.00 B/s
Current DISK READ:       0.00 B/s | Current DISK WRITE:       0.00 B/s
    TID  PRIO  USER     DISK READ  DISK WRITE  SWAPIN     IO>    COMMAND
 270983 be/4 shs         0.00 B/s    0.00 B/s  0.00 % 51.45 % stress --io 4
 270984 be/4 shs         0.00 B/s    0.00 B/s  0.00 % 51.36 % stress --io 4
 270985 be/4 shs         0.00 B/s    0.00 B/s  0.00 % 50.95 % stress --io 4
 270982 be/4 shs         0.00 B/s    0.00 B/s  0.00 % 50.80 % stress --io 4
 269308 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.09 % [kworker~fficient]
```

Stress is just one of a number of tools for adding stress to a system. Another and newer tool, **stress-ng**, will be covered in a future post.

Wrap-Up

Various tools for stress-testing a system will help you anticipate how systems will respond in real world situations in which they are subjected to increased traffic and computing demands.

While what we've shown in the post are ways to create and measure various types of stress, the ultimate benefit is how the stress helps in determining how well your system or application responds to it.

Join the Network World communities on Facebook and LinkedIn to comment on topics that are top of mind.

Sandra Henry-Stocker has been administering Unix systems for more than 30 years. She describes herself as "USL" (Unix as a second language) but remembers enough English to write books and buy groceries. She lives in the mountains in Virginia where, when not working with or writing about Unix, she's chasing the bears away from her bird feeders.

Follow