Stress NG

Christian Külker

2024-03-16

Contents

1 Introduction

stress-ng is a tool designed for imposing a high load on various computer subsystems. This tool is capable of loading and stressing multiple aspects of a computer, including the CPU, cache, disk, memory, socket, and pipe I/O, as well as scheduling. Developed as an enhanced version of Amos Waterland's original stress tool, stress-ng introduces numerous additional features. These include the ability to define the number of bogo operations, gather execution metrics, verify stress on memory and compute operations, and implement a wider range of stress mechanisms.

The primary function of stress-ng is to conduct stress tests on a computer system through different methods. It targets both the physical subsystems of a computer and the interfaces of the operating system kernel. Furthermore, stress-ng contains an array of CPU-focused stress tests, which target various operations including floating-point calculations, integer processing, bit manipulation, and control flow.

Originally, the purpose of stress-ng was to heavily load a system to identify hardware issues like thermal overruns and operating system bugs that emerge under extreme conditions. Caution is advised when using stress-ng, as some tests may cause systems with poor design to overheat or experience significant system overload, potentially leading to difficult-to-stop conditions.

In addition to its primary stress-testing functions, stress-ng can be used to measure test throughput rates. This feature is helpful for observing performance variances across different hardware or operating system versions. However, stress-ng is not designed as a benchmarking tool and should **not** be used for precise benchmark testing.

Executing stress-ng with root privileges on Linux systems allows for the adjustment of out-of-memory settings, rendering stressors unkillable in low-memory scenarios. This feature should be used carefully. With the necessary privileges, stress-ng also permits the modification of 'ionice' classes and levels, which again requires cautious application.

Users can specify the number of processes for each stress test type. Setting this number to zero defaults to the number of processors available as defined by _sysconf(SC_NPROCESSORS_CONF). If this cannot be determined, the number of online CPUs is used. A negative value defaults to the number of online CPUs.

stress-ng includes over 220 different stress tests (stressors), encompassing a broad spectrum of system components:

- CPU-related tests (cache: icache, dcache; compute operations like: integer, float, string, searching)
- Process management (fork, vfork, clone, kill, pthread)
- Device interaction (block and /dev)

- File system and I/O operations (file handling, attributes, directories, links, renaming)
- Interrupt handling (IRQs and soft interrupts)
- Memory-related tests (throughput, virtual memory, RAM tests, paging, stack, brk, mmap)
- Networking aspects (tcp, udp, sctp, dccp, netlink, sockfd)
- Kernel (system calls, /sys, /proc interactions)
- Security (AppArmor, seccomp)
- Inter-process communication (pipes, shared memory, semaphores, mutexes)

The tool offers a wide range of stress mechanisms, each with a detailed description available in the manual page. This document serves as a quick-start reference guide, outlining common use cases for stress-ng.

> Exercise caution when using the stress-ng tool. Certain tests have the potential to affect thermal zone trip points in systems with suboptimal hardware design. This can lead to degraded system performance and excessive system thrashing, which may present challenges in halting. Additionally, some fault stressors may render the operating system inoperable, necessitating manual intervention. This is an important consideration for users who do not have direct physical access to the system.

2 Debian

#	Debian	Version
13		0.17.05
12	Bookworm	0.15.06-2
11	Bullseye	0.12.06
10	Buster	0.09.50

3 Dependencies

- libapparmor1 (>= 2.10)
- libbsd0 (>= 0.6.0)
- libc6 (>= 2.36)
- libcrypt1 (>= 1:4.1.0)
- libegl1
- libgbm1 (>= 8.1~0)

- libgles2
- libipsec-mb1 (>= 1.3)
- libjpeg62-turbo (>= 1.3.1)
- libjudydebian1
- libkmod2 (>= 5~)
- libsctp1 (>= 1.0.10+dfsg)
- libxxhash0 (>= 0.6.5)
- zlib1g (>= 1:1.1.4)

4 Installation

Installation of the distribution package is straight forward.

```
aptitude install stress-ng
```

5 Basic Usage

Amazingly stress-ng has approximately over 926 command line options[1]. Consequently, this guide will not attempt to cover all of them. The manual page for stress-ng includes 18 usage examples. The usage examples provided here are configured for brief durations, allowing for quick testing. However, for substantive stress testing, longer run times are typically recommended.

5.1 CPU

```
# Run 1 CPU threads (stressor) for 1 minute
stress-ng --cpu 1 --timeout 1m
stress-ng: info: [106383] setting to a 60 second run per stressor
stress-ng: info: [106383] dispatching hogs: 1 cpu
stress-ng: info: [106383] successful run completed in 60.00s (1 min, 0.00
↪  secs)
```

In case you would like to run a special CPU test, you can query which tests are available.

```
stress-ng --cpu-method which
cpu-method must be one of: all ackermann apery bitops callfunc cdouble \
```

[1] Not counting the short options, just the long options parsed with:

```
man stress-ng|grep '       \--'|sort|uniq|wc -l
```

```
cfloat clongdouble collatz correlate crc16 decimal32 decimal64 \
decimal128 dither div8 div16 div32 div64 div128 double euler explog \
factorial fibonacci fft fletcher16 float float32 float64 float80 \
float128 floatconversion gamma gcd gray hamming hanoi hyperbolic idct \
int128 int64 int32 int16 int8 int128float int128double int128longdouble \
int128decimal32 int128decimal64 int128decimal128 int64float int64double \
int64longdouble int32float int32double int32longdouble intconversion \
ipv4checksum jmp lfs r32 ln2 logmap longdouble loop matrixprod nsqrt \
omega parity phi pi prime psi queens rand rand48 rgb sieve stats sqrt \
trig union zeta
```

5.2 Memory

Execute two virtual memory stressors, adjusting the number to align with the core or thread count of your system. These stressors should collectively utilize 80% of the available memory for a duration of one minute. Accordingly, each stressor will consume approximately 40% of the total available memory.

```
stress-ng --vm 2 --vm-bytes 80% -t 1m
stress-ng: info: [119819] setting to a 60 second run per stressor
stress-ng: info: [119819] dispatching hogs: 2 vm
stress-ng: info: [119819] successful run completed in 60.12s (1 min, 0.12
↪  secs)
```

5.3 IO

Execute two instances of mixed I/O stressors, allocating a total of 10% of the available file system space for a duration of 1 minute. Each stressor is configured to utilize 5% of the available file system space.

```
stress-ng --iomix 2 --iomix-bytes 10% -t 1m
stress-ng: info: [106940] setting to a 60 second run per stressor
stress-ng: info: [106940] dispatching hogs: 2 iomix
stress-ng: info: [106940] successful run completed in 60.01s (1 min, 0.01
↪  secs)
```

5.4 Devices

Measure real-time scheduling latencies induced by the HDD stressor using a high-resolution nanosecond clock. This measurement involves assessing latencies during sleep intervals of 10,000 nanoseconds. After a one-minute stress period, the tool displays

the latency distribution in 2500 nanosecond intervals. It is important to note that this test requires the `CAP_SYS_NICE` capability for enabling real-time scheduling, which is essential for obtaining accurate measurements.

```
stress-ng --cyclic 1 --cyclic-dist 2500 --cyclic-method clock_ns \
--cyclic-prio 100 --cyclic-sleep 10000 --hdd 0 -t 1m
stress-ng: info: [107070] setting to a 60 second run per stressor
stress-ng: info: [107070] dispatching hogs: 1 cyclic, 4 hdd
stress-ng: info: [107071] cyclic: sched SCHED_DEADLINE: 10000 ns delay, \
10000 samples
stress-ng: info: [107071] cyclic:   mean: 4312.11 ns, mode: 3641 ns
stress-ng: info: [107071] cyclic:   min: 3564 ns, max: 24051 ns, std.dev. \
992.30
stress-ng: info: [107071] cyclic: latency percentiles:
stress-ng: info: [107071] cyclic:   25.00%:       3668 ns
stress-ng: info: [107071] cyclic:   50.00%:       3993 ns
stress-ng: info: [107071] cyclic:   75.00%:       4667 ns
stress-ng: info: [107071] cyclic:   90.00%:       5289 ns
stress-ng: info: [107071] cyclic:   95.40%:       6014 ns
stress-ng: info: [107071] cyclic:   99.00%:       8153 ns
stress-ng: info: [107071] cyclic:   99.50%:       9663 ns
stress-ng: info: [107071] cyclic:   99.90%:      12810 ns
stress-ng: info: [107071] cyclic:   99.99%:      24051 ns
stress-ng: info: [107071] cyclic: latency distribution (2500 ns
 ↪  intervals):
stress-ng: info: [107071] cyclic: (for the first 10 buckets of 10)
stress-ng: info: [107071] cyclic: latency (ns)  frequency
stress-ng: info: [107071] cyclic:            0          0
stress-ng: info: [107071] cyclic:         2500       8476
stress-ng: info: [107071] cyclic:         5000       1389
stress-ng: info: [107071] cyclic:         7500         92
stress-ng: info: [107071] cyclic:        10000         31
stress-ng: info: [107071] cyclic:        12500          8
stress-ng: info: [107071] cyclic:        15000          2
stress-ng: info: [107071] cyclic:        17500          1
stress-ng: info: [107071] cyclic:        20000          0
stress-ng: info: [107071] cyclic:        22500          1
stress-ng: info: [107071] cyclic: Note: --cyclic-samples needed to be \
4374911 to capture all the data for this run
stress-ng: info: [107070] successful run completed in 60.75s (1 min, 0.75
 ↪  secs)
```

5.5 Interrupts

Executing timers at a high frequency can result in a substantial increase in interrupt load. Utilizing the `--timer` stressor with a suitably chosen timer frequency can induce a high number of interrupts per second. It is important to note that executing this operation requires root permissions.

```
stress-ng --timer 32 --timer-freq 1000000 -t 1m
stress-ng: info: [108527] setting to a 60 second run per stressor
stress-ng: info: [108527] dispatching hogs: 32 timer
stress-ng: info: [108527] successful run completed in 60.69s (1 min, 0.69
↪  secs)
```

In this example, stress-ng tests 32 instances at 1MHz.

5.6 Combining

The combination of tasks (stressors) is easily possible by adding more command line options. Example CPU + IO + Memory.

```
stress-ng --cpu 4 --io 2 --vm 1 --vm-bytes 1G --timeout 60s
stress-ng: info: [108632] setting to a 60 second run per stressor
stress-ng: info: [108632] dispatching hogs: 4 cpu, 2 io, 1 vm
stress-ng: info: [108637] io: this is a legacy I/O sync stressor, \
consider using iomix instead
stress-ng: info: [108632] successful run completed in 60.06s (1 min, 0.06
↪  secs)

# Replacing io with iomix:
stress-ng --cpu 4 --iomix 2 --vm 1 --vm-bytes 1G --timeout 60s
stress-ng: info: [108673] setting to a 60 second run per stressor
stress-ng: info: [108673] dispatching hogs: 4 cpu, 2 iomix, 1 vm
stress-ng: info: [108673] successful run completed in 60.67s (1 min, 0.67
↪  secs)
```

5.7 Faults

For system benchmarking or evaluating systems produced by an Original Equipment Manufacturer (OEM), injecting faults can be a useful approach to observe system reactions.

Page Faults:

stress-ng enables the testing and analysis of page fault rates by generating major page faults in pages that are not currently loaded in memory. In newer kernel versions, the

userfaultfd mechanism alerts fault-handling threads about page faults within the virtual memory layout of a process. Executing these operations requires root privileges on the system.

```
# For older kernel (output was run on newer kernel, stress-ng + PID
# removed)
# Linux 6.1.0-18-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.76-1 \
# (2024-02-01) x86_64 GNU/Linux
# 1. Simple Run without performance reporting (we can not see that it do
#    not fit well)
stress-ng --fault 0 -t 1m
setting to a 60 second run per stressor
dispatching hogs: 4 fault
successful run completed in 60.00s (1 min, 0.00 secs)
```

```
# 2. Run with performance reporting (we can see that it do not fit well: a
# lot of 0.000 /sec reporting
stress-ng --fault 0 --perf -t 1m
setting to a 60 second run per stressor
dispatching hogs: 4 fault
fault:
544,855,786,816 CPU Cycles                    8.794 B/sec
371,921,049,104 Instructions                  6.003 B/sec (0.683 instr.
↪  per cycle)
 70,548,210,352 Branch Instructions           1.139 B/sec
  3,463,128,704 Branch Misses                55.894 M/sec ( 4.909%)
 21,793,985,384 Bus Cycles                    0.352 B/sec
479,480,334,820 Total Cycles                  7.739 B/sec
 14,189,251,132 Cache References              0.229 B/sec
    878,261,112 Cache Misses                 14.175 M/sec ( 6.190%)
108,783,796,528 Cache L1D Read                1.756 B/sec
  4,686,096,088 Cache L1D Read Miss          75.633 M/sec ( 4.308%)
 70,957,241,756 Cache L1D Write               1.145 B/sec
  4,238,030,032 Cache L1I Read Miss          68.401 M/sec
  1,720,397,160 Cache LL Read                27.767 M/sec
    151,743,796 Cache LL Read Miss            2.449 M/sec ( 8.820%)
    415,481,916 Cache LL Write                6.706 M/sec
     30,201,260 Cache LL Write Miss           0.487 M/sec ( 7.269%)
108,737,787,928 Cache DTLB Read               1.755 B/sec
     59,822,784 Cache DTLB Read Miss          0.966 M/sec ( 0.055%)
 70,922,004,348 Cache DTLB Write              1.145 B/sec
     17,035,460 Cache DTLB Write Miss         0.275 M/sec ( 0.024%)
 70,553,791,700 Cache BPU Read                1.139 B/sec
```

```
  3,457,202,928 Cache BPU Read Miss           55.799 M/sec ( 4.900%)
    150,661,108 Cache NODE Read                2.432 M/sec
              0 Cache NODE Read Miss           0.000 /sec  ( 0.000%)
     30,356,440 Cache NODE Write               0.490 M/sec
              0 Cache NODE Write Miss          0.000 /sec  ( 0.000%)
219,716,211,724 CPU Clock                      3.546 B/sec
219,750,070,380 Task Clock                     3.547 B/sec
      2,590,976 Page Faults Total             41.818 K/sec
      1,554,604 Page Faults Minor             25.091 K/sec
      1,036,372 Page Faults Major             16.727 K/sec
         48,180 Context Switches             777.619 /sec
         48,052 Cgroup Switches              775.553 /sec
            284 CPU Migrations                 4.584 /sec
              0 Alignment Faults               0.000 /sec
              0 Emulation Faults               0.000 /sec
      2,590,972 Page Faults User              41.818 K/sec
              4 Page Faults Kernel             0.065 /sec
      6,737,088 System Call Enter              0.109 M/sec
      6,737,084 System Call Exit               0.109 M/sec
      2,082,852 Kmalloc                       33.617 K/sec
      2,776,412 Kfree                         44.811 K/sec
     17,315,696 Kmem Cache Alloc               0.279 M/sec
     18,768,992 Kmem Cache Free                0.303 M/sec
      1,640,580 MM Page Alloc                 26.479 K/sec
      1,568,584 MM Page Free                  25.317 K/sec
      7,254,672 MMAP lock start                0.117 M/sec
      7,254,672 MMAP lock release              0.117 M/sec
        670,928 RCU Utilization               10.829 K/sec
              0 RCU Stall Warning              0.000 /sec
            912 Sched Migrate Task            14.720 /sec
              0 Sched Move NUMA                0.000 /sec
         47,784 Sched Wakeup                 771.228 /sec
              0 Sched Proc Exec                0.000 /sec
              0 Sched Proc Exit                0.000 /sec
              0 Sched Proc Fork                0.000 /sec
              8 Sched Proc Free                0.129 /sec
              0 Sched Proc Hang                0.000 /sec
              0 Sched Proc Wait                0.000 /sec
         48,184 Sched Switch                 777.684 /sec
              0 New Task                       0.000 /sec
              0 Context User Exit              0.000 /sec
              8 Signal Generate                0.129 /sec
```

```
               4 Signal Deliver                  0.065 /sec
             736 IRQ Entry                      11.879 /sec
             736 IRQ Exit                       11.879 /sec
         288,772 Soft IRQ Entry                  4.661 K/sec
         288,772 Soft IRQ Exit                   4.661 K/sec
              48 NMI handler                     0.775 /sec
           1,424 Block BIO Complete             22.983 /sec
               0 IO uring submit                0.000 /sec
               0 IO uring complete              0.000 /sec
               0 Writeback Dirty Inode          0.000 /sec
               0 Migrate MM Pages               0.000 /sec
              36 SKB Consume                    0.581 /sec
               0 SKB Kfree                      0.000 /sec
          26,652 Lock Contention Begin        430.160 /sec
          26,652 Lock Contention End          430.160 /sec
               0 IOMMU IO Page Fault            0.000 /sec
             132 IOMMU Map                      2.130 /sec
             532 IOMMU Unmap                    8.586 /sec
         518,188 Filemap page-cache add         8.363 K/sec
       1,036,376 Filemap page-cache del        16.727 K/sec
               0 OOM Compact Retry              0.000 /sec
               0 OOM Wake Reaper                0.000 /sec
               0 OOM Score Adjust Update        0.000 /sec
               0 Thermal Zone Trip              0.000 /sec
successful run completed in 61.96s (1 min, 1.96 secs)
```

```
# For newer kernel
# (output was run on newer kernel, stress-ng + PID removed)
# Linux 6.1.0-18-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.76-1 \
# (2024-02-01) x86_64 GNU/Linux
# 1. Simple run
stress-ng --userfaultfd 0  -t 1m
setting to a 60 second run per stressor
dispatching hogs: 4 userfaultfd
successful run completed in 60.00s (1 min, 0.00 secs)
```

```
# 2. Run with performance report
stress-ng --userfaultfd 0 --perf -t 1m
setting to a 60 second run per stressor
dispatching hogs: 4 userfaultfd
userfaultfd:
573,784,529,308 CPU Cycles                      9.563 B/sec
```

```
367,401,565,752 Instructions                    6.123 B/sec (0.640 instr.
↪  per cycle)
 75,247,240,040 Branch Instructions             1.254 B/sec
    597,506,500 Branch Misses                   9.958 M/sec ( 0.794%)
 22,975,717,548 Bus Cycles                      0.383 B/sec
505,493,763,492 Total Cycles                    8.425 B/sec
 16,891,417,900 Cache References                0.282 B/sec
    778,519,852 Cache Misses                   12.975 M/sec ( 4.609%)
100,779,216,856 Cache L1D Read                  1.680 B/sec
 12,832,061,780 Cache L1D Read Miss             0.214 B/sec (12.733%)
 76,371,736,040 Cache L1D Write                 1.273 B/sec
  5,657,176,416 Cache L1I Read Miss            94.282 M/sec
  3,251,015,820 Cache LL Read                  54.181 M/sec
     46,779,708 Cache LL Read Miss              0.780 M/sec ( 1.439%)
  1,282,146,056 Cache LL Write                 21.368 M/sec
    504,428,128 Cache LL Write Miss             8.407 M/sec (39.342%)
101,289,571,044 Cache DTLB Read                 1.688 B/sec
     57,563,032 Cache DTLB Read Miss            0.959 M/sec ( 0.057%)
 76,447,085,328 Cache DTLB Write                1.274 B/sec
     85,701,532 Cache DTLB Write Miss           1.428 M/sec ( 0.112%)
 75,143,922,784 Cache BPU Read                  1.252 B/sec
    598,705,764 Cache BPU Read Miss             9.978 M/sec ( 0.797%)
     49,948,888 Cache NODE Read                 0.832 M/sec
              0 Cache NODE Read Miss            0.000 /sec  ( 0.000%)
    493,044,204 Cache NODE Write                8.217 M/sec
              0 Cache NODE Write Miss           0.000 /sec  ( 0.000%)
235,288,674,364 CPU Clock                       3.921 B/sec
235,309,819,424 Task Clock                      3.922 B/sec
      8,465,072 Page Faults Total               0.141 M/sec
            224 Page Faults Minor               3.733 /sec
      8,464,844 Page Faults Major               0.141 M/sec
     16,939,336 Context Switches                0.282 M/sec
         15,748 Cgroup Switches               262.455 /sec
          1,204 CPU Migrations                 20.066 /sec
              0 Alignment Faults                0.000 /sec
              0 Emulation Faults                0.000 /sec
      8,465,052 Page Faults User                0.141 M/sec
             20 Page Faults Kernel              0.333 /sec
     33,959,376 System Call Enter               0.566 M/sec
     33,959,376 System Call Exit                0.566 M/sec
         66,364 Kmalloc                         1.106 K/sec
         99,728 Kfree                           1.662 K/sec
```

```
       134,364 Kmem Cache Alloc              2.239 K/sec
       138,432 Kmem Cache Free               2.307 K/sec
    16,940,184 MM Page Alloc                 0.282 M/sec
    16,929,232 MM Page Free                  0.282 M/sec
    25,395,468 MMAP lock start               0.423 M/sec
    25,395,468 MMAP lock release             0.423 M/sec
    34,085,752 RCU Utilization               0.568 M/sec
             0 RCU Stall Warning             0.000 /sec
         3,312 Sched Migrate Task           55.197 /sec
             0 Sched Move NUMA               0.000 /sec
    13,425,252 Sched Wakeup                  0.224 M/sec
             0 Sched Proc Exec               0.000 /sec
             4 Sched Proc Exit               0.067 /sec
             8 Sched Proc Fork               0.133 /sec
             4 Sched Proc Free               0.067 /sec
             0 Sched Proc Hang               0.000 /sec
             8 Sched Proc Wait               0.133 /sec
    16,939,336 Sched Switch                  0.282 M/sec
             8 New Task                      0.133 /sec
             0 Context User Exit             0.000 /sec
            12 Signal Generate               0.200 /sec
             8 Signal Deliver                0.133 /sec
           404 IRQ Entry                     6.733 /sec
           404 IRQ Exit                      6.733 /sec
        61,384 Soft IRQ Entry                1.023 K/sec
        61,384 Soft IRQ Exit                 1.023 K/sec
            40 NMI handler                   0.667 /sec
            20 Block BIO Complete            0.333 /sec
             0 IO uring submit               0.000 /sec
             0 IO uring complete             0.000 /sec
             4 Writeback Dirty Inode         0.067 /sec
             0 Migrate MM Pages              0.000 /sec
           316 SKB Consume                   5.266 /sec
             4 SKB Kfree                     0.067 /sec
       783,916 Lock Contention Begin        13.065 K/sec
       783,908 Lock Contention End          13.065 K/sec
             0 IOMMU IO Page Fault           0.000 /sec
             4 IOMMU Map                     0.067 /sec
            68 IOMMU Unmap                   1.133 /sec
             0 Filemap page-cache add        0.000 /sec
             0 Filemap page-cache del        0.000 /sec
             0 OOM Compact Retry             0.000 /sec
```

```
              0 OOM Wake Reaper                0.000 /sec
              4 OOM Score Adjust Update        0.067 /sec
              0 Thermal Zone Trip              0.000 /sec
successful run completed in 60.00s (1 min, 0.00 secs)
```

5.8 Reporting

In the output below `stress-ng: REALM: [PID]` was removed.

`--metrics:`

In comparison to `--metrics-brief` this adds CPU used per instance (%) and RSS Max.

```
stress-ng --cpu 1 --timeout 1m --metrics
setting to a 60 second run per stressor
dispatching hogs: 1 cpu
stressor bogo ops real time usr time sys time bogo ops/s     bogo ops/s \
CPU used per RSS Max
                    (secs)   (secs)   (secs) (real time) (usr+sys time) \
instance (%)    (KB)
cpu         74433     60.00    60.00     0.00    1240.49        1240.59 \
       99.99    5824
successful run completed in 60.00s (1 min, 0.00 secs)
```

`--metrics-brief:`

Compared to `--metrics` the output fits better a 80 char terminal and it leaves out CPU used per instance (%) and RSS Max.

```
stress-ng --cpu 1 --timeout 1m --metrics-brief
setting to a 60 second run per stressor
dispatching hogs: 1 cpu
stressor bogo ops real time usr time sys time  bogo ops/s     bogo ops/s
                    (secs)   (secs)   (secs)  (real time) (usr+sys time)
cpu         71860     60.00    60.00     0.00     1197.59        1197.68
successful run completed in 60.01s (1 min, 0.01 secs)
```

`--tz:` (temperature)

```
stress-ng --cpu 0 --cpu-method all --verify -t 1m --tz
stress-ng: info: [107199] setting to a 60 second run per stressor
stress-ng: info: [107199] dispatching hogs: 4 cpu
stress-ng: info: [107199] cpu:
stress-ng: info: [107199]    pch_wildcat_point   47.50 C (320.65 K)
```

```
stress-ng: info: [107199]         x86_pkg_temp   64.50 C (337.65 K)
stress-ng: info: [107199] successful run completed in 60.01s (1 min, 0.01
↪  secs)
```

`--perf:`

Measure processor and system activity using perf events, a functionality specific to Linux. However, exercise caution as per the guidance in `perf_event_open(2)`, which advises to "Always double-check your results! Various generalized events have had wrong values." It's important to note that as of Linux version 4.7, `CAP_SYS_ADMIN` capabilities are required for this option to function[2].

6 Benchmarking

`--verify:`

Most stressors have a verification mode `--verify` to sanity check test operations. This adds overhead to bogo-ops rate so don't use it for benchmarking.

Test memory with different test patterns for 1 hour:

`stress-ng --vm 1 --vm-bytes 2G --verify -v -t 1h`

1 hour CPU computation soak test:

`stress-ng --cpu 0 --verify -t 1h`

`--perf:`

See previous section **Reporting** and **Faults**.

7 Exit

When executing stress-ng, it is important to check the exit status. The tool documents eight exit conditions, which are detailed in the manual page. Refer to the bottom of the manual page for comprehensive information on these conditions.

- 0: Success.
- 1: Error; Incorrect user options or a fatal resource issue in the stress-ng stressor harness
- 2: Error; One or more stressors failed

[2] Alternatively, adjusting `/proc/sys/kernel/perf_event_paranoid` to a value below 2 allows for the use of this feature without `CAP_SYS_ADMIN` capabilities.

- 3: Error; One or more stressors failed to initialise because of lack of resources
- 4: Error; One or more stressors were not implemented on a specific architecture or operating system
- 5: Error; A stressor has been killed by an unexpected signal
- 6: Error; A stressor exited by exit(2) which was not expected and timing metrics could not be gathered
- 7: Error; The bogo ops metrics maybe untrustworthy

```
# With ERROR:
stress-ng --cpu 0 --timeout 60s.;echo "Exit code: $?"
Illegal time specifier .
Exit code: 1

# With SUCCESS:
stress-ng --cpu 0 --timeout 60s;echo "Exit code: $?"
stress-ng: info: [108944] setting to a 60 second run per stressor
stress-ng: info: [108944] dispatching hogs: 4 cpu
stress-ng: info: [108944] successful run completed in 60.01s (1 min, 0.01
↪  secs)
Exit code: 0
```

8 Links

- Homepage: https://github.com/ColinIanKing/stress-ng
- Source: https://github.com/ColinIanKing/stress-ng
- Raspberry PI thermal stress test: https://www.youtube.com/watch?v=V4idnxE5AbE
- Guide: https://wiki.ubuntu.com/Kernel/Reference/stress-ng

9 History

Version	Date	Notes
0.1.1	2024-03-16	Fix example descriptions, formatting
0.1.0	2024-03-16	Initial release (to GitHub)

10 Disclaimer of Warranty

UMENT OR THE PROGRAM “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE INFORMATION, DOCUMENTS AND PROGRAMS IS WITH YOU. SHOULD THE INFORMATION, DOCUMENTS OR PROGRAMS PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

11 Limitation of Liability

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE INFORMATION, DOCUMENTS OR PROGRAMS AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE INFORMATION, DOCUMENTS OR PROGRAMS (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE INFORMATION, DOCUMENTS OR PROGRAMS TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.