CANONICAL

stress-ng

A stress-testing Swiss army knife

Presentation by
Colin Ian King
colin.king@canonical.com
www.canonical.com
October 2019

Designed to stress a computer system:

- Originally designed to trip hardware issues (make test systems hot!)
- Exercises a wide range of system calls, /dev, /sys, /proc interfaces
- Micro-benchmarking (bogo-ops throughput metrics)
- Real Time / Low-latency cyclic measurements
- System burn-in tests
- Kernel regression testing (22 bugs found so far)
- Kernel coverage testing
- Verify option for deeper system failure checking
- Used by researchers for stress testing

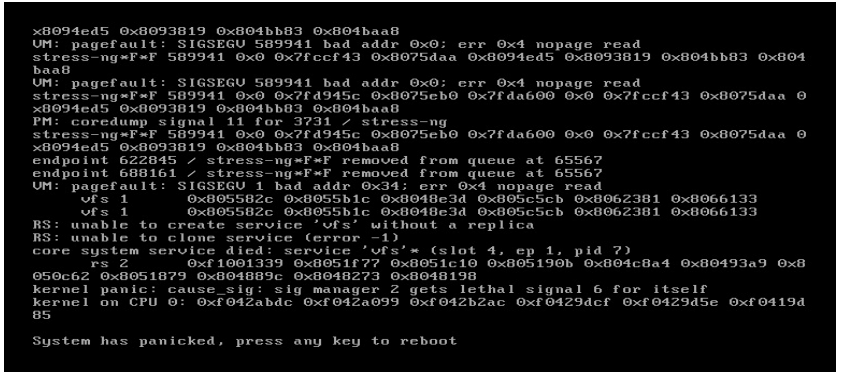
Over 220 stress tests (aka stressors)

- CPU cache (icache, dcache), CPU compute (integer, float, string, searching..)
- Process management (fork, vfork, clone, kill, pthread)
- Device (block and /dev)
- File system and I/O (files, attributes, directories, links, renaming, etc)
- Interrupts (IRQs and soft interrupts)
- Memory (throughput, VM, RAM tests, paging, stack, brk, mmap)
- Networking (tcp, udp, sctp, dccp, netlink, sockfd)
- Kernel (system calls and /sys, /proc interfaces)
- Security (AppArmor, seccomp)
- IPC (pipes, shared memory, sempahores, mutexes)

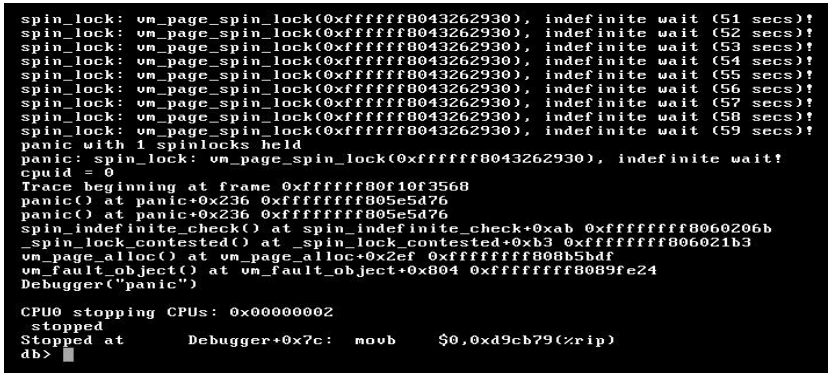
Designed to be portable:

- Linux (multiple arches, multiple distros)
- FreeBSD, OpenBSD, NetBSD, DragonFlyBSD
- Solaris (OpenIndiana)
- Minix
- Android (static image)
- Mac OS X
- Haiku
- GNU/HURD
- Compiles with GCC, Clang and tcc

stress-ng can break kernels

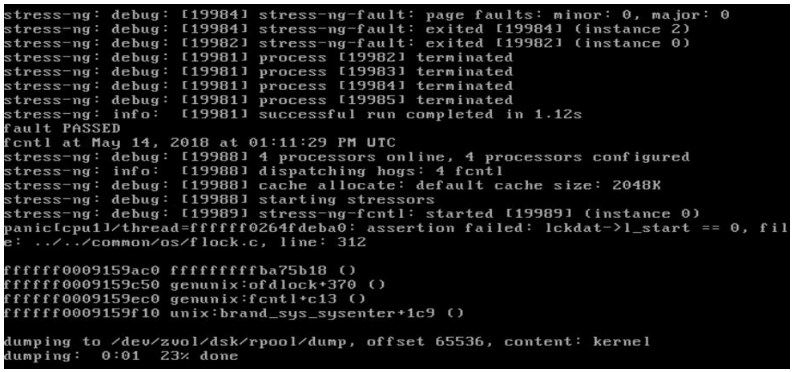


Minix 3.3 file system crash



DragonFlyBSD

Debian kFreeBSD



OpenIndiana (Solaris)

CANONICAL

stress-ng examples (1/4)

Run 1 iomix stressor (mix of I/O operations) for 20 seconds with verbose output:

```
stress-ng --iomix 1 -t 20 -v
```

Run 2 cpu stressors and 4 virtual memory stressors for 5 minutes:

```
stress-ng --cpu 2 --vm 4 -t 5m
```

Special mode with zero stressors will run a stressor on each of the currently on-line CPUs (no need to specify number of CPUs),

e.g. on a 8 thread machine, run 8 shared memory stressors:

```
stress-ng --shm 0
```

stress-ng examples (2/4)

Run all the stressors one by one on all CPUs; each stressor will run for 30 seconds and measure thermal zone temperatures:

```
sudo stress-ng --seq 0 -t 30 --tz -v
```

Generate major page faults and see the page fault rate using perf stats:

```
stress-ng --fault 0 --perf -t 1m
```

```
stress-ng --userfaultfd 0 --perf -t 1m
```

Generate large interrupt load with 32 timer stressors:

```
stress-ng --timer 32 --timer-freq 1000000
```

stress-ng examples (3/4)

Memory pressure and swapping:

```
stress-ng --brk 0 --stack 0 --bigheap 0
```

Stressor size options:

```
stress-ng --vm 1 --vm-bytes 2G
stress-ng --vm 1 --vm-bytes 50%
stress-ng --hdd 1 --hdd-bytes 10%
stress-ng --malloc 1 --malloc-bytes 120%
stress-ng --shm –-shm-bytes 256M
```

Can use percentage (%), or specific sizes in bytes, kilobytes (K), megabytes (M) or gigabytes (G)

stress-ng examples (4/4)

Highly configurable stressors:

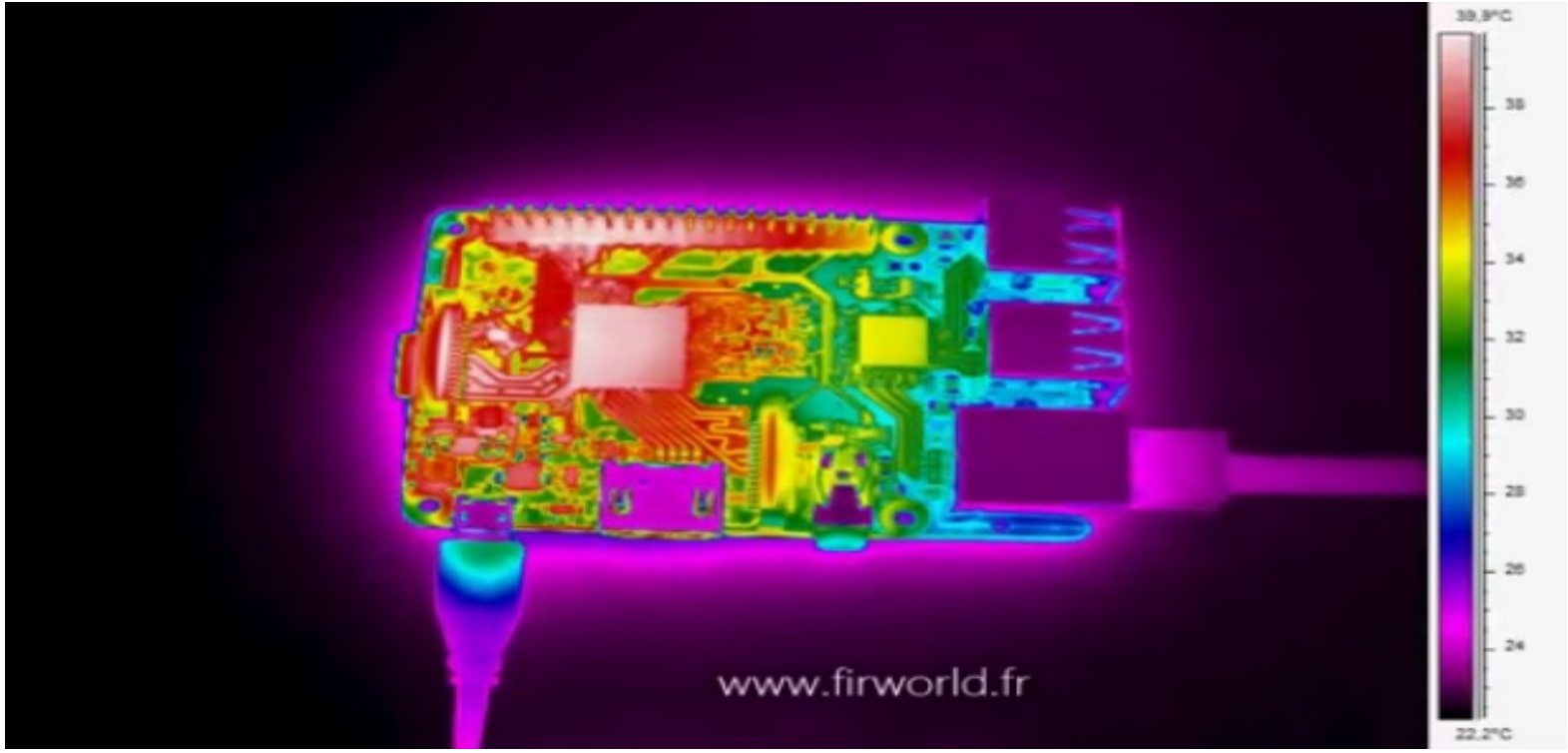
```
stress-ng --vm 1 --vm-locked --vm-populate --vm-madvise nohugepage --vm-method gray
--vm-bytes 128M --verify --metrics-brief --vm-ops 1000000
```

- Attempt to lock pages into memory using MAP_LOCKED
- Populate page tables for the memory mappings and don't use hugepages
- Fill memory with ascending gray codes and verify these are set correctly
- Repeat for 1,000,000 bogo ops

The vm stressor will cycle through memory mapping, filling and checking and unmapping the mapped region. If the stressor is OOM'd by the kernel then stress-ng will re-spawn the test.

Stress-ng thermal testing

https://www.youtube.com/watch?v=V4idnxE5AbE



stress-ng thermal zones

How Hot? Thermal zone information using the --tz option:

```
stress-ng --matrix 0 --tz -t 60 --log-brief
dispatching hogs: 4 matrix
successful run completed in 60.00s (1 min, 0.00 secs)
matrix:
        x86_pkg_temp   89.00 C (362.15 K)
              acpitz   88.50 C (361.65 K)

stress-ng --cpu 0 --tz -t 60 --log-brief
dispatching hogs: 4 cpu
successful run completed in 60.05s (1 min, 0.05 secs)
cpu:
        x86_pkg_temp   87.25 C (360.40 K)
              acpitz   87.12 C (360.27 K)
```

stress-ng metrics (1/2)

- Stress-ng uses a concept of bogo-ops per second as a measure of throughput.
- One bogo-op is one loop iteration of a stressor action.
- Bogo-op rates vary from stressor to stressor.
- Bogo-op rates may vary between releases of stress-ng due to compiler optimizations or code changes.
- Bogo-op rates will vary between kernels.
- Used by the Ubuntu Kernel team for performance regression testing.

```
stress-ng --dup 1 -t 1m --metrics --log-brief
dispatching hogs: 1 dup
successful run completed in 60.00s (1 min, 0.00 secs)
stressor       bogo ops real time  usr time  sys time   bogo ops/s   bogo ops/s
                        (secs)    (secs)    (secs)   (real time) (usr+sys time)
dup             21520821     60.00     33.79     26.16    358681.41    358979.50
```

stress-ng metrics (2/2)

The --yaml option specifies a YAML output file containing test metrics.

```
stress-ng --cpu 0 -t 1m --metrics --yaml cpu-stats.yaml
```

The yaml file contains:

- system information:
  - stress-ng version, date, hostname, kernel version,
  - architecture, memory, CPU info, etc.
- per-stress test metrics:
  - stressor name, bogo-ops rates, wall-clock time,
  - user-time and system-time

Useful for automated benchmarking.

stress-ng perf metrics (1/2)

Perf stats on CPU cycles, instruction rate, branching, cache activity, page faults, context switching, page activity, system calls, TLB flushes, scheduling stats, signals, IRQs, filemap cache, OOMs and thermal zone trips.

```
sudo stress-ng --perf --matrix 1 -t 60  --log-brief
          175,852,773,535 CPU Cycles                       2.87 B/sec
          396,687,869,300 Instructions                     6.48 B/sec (2.256 instr. per cycle)
           50,130,992,422 Branch Instructions              0.82 B/sec
              389,648,188 Branch Misses                    6.37 M/sec ( 0.78%)
           74,228,869,562 Stalled Cycles Frontend          1.21 B/sec
            5,859,477,614 Bus Cycles                      95.77 M/sec
          146,503,609,353 Total Cycles                     2.39 B/sec
              300,031,623 Cache References                 4.90 M/sec
                6,795,960 Cache Misses                     0.11 M/sec ( 2.27%)
```

stress-ng perf metrics (2/2)

```
95,514,831,044 Cache L1D Read                    1.56 B/sec
49,657,247,152 Cache L1D Read Miss               0.81 B/sec
 1,123,259,756 Cache L1D Write                  18.36 M/sec
   596,354,025 Cache L1D Write Miss              9.75 M/sec
 2,771,837,260 Cache L1D Prefetch Miss          45.31 M/sec
    18,003,604 Cache L1I Read Miss               0.29 M/sec
   277,631,907 Cache LL Read                     4.54 M/sec
    19,435,044 Cache LL Write                    0.32 M/sec
   335,291,945 Cache LL Prefetch                 5.48 M/sec
95,347,100,060 Cache DTLB Read                   1.56 B/sec
    98,849,843 Cache DTLB Read Miss              1.62 M/sec
 1,118,723,178 Cache DTLB Write                 18.29 M/sec
       400,530 Cache DTLB Write Miss             6.55 K/sec
```

stress-ng cyclic latency measurements (1/3)

Much like the Real Time cyclictest tool, but can use any mix of stressors.

Example, run 1 cyclic benchmark with the virtual memory stressor for 60 seconds:

```
stress-ng --cyclic 1 --cyclic-dist 250 --cyclic-method clock_ns \
  --cyclic-sleep 20000 --cyclic-policy rr --vm 4 -t 60  --log-brief
```

- distribution stats @ 250 ns intervals
- using CLOCK_NANOSECOND timer, sleep interval of 20000 ns
- round robin scheduler policy
- exercise virtual memory with 4 vm stressors

stress-ng cyclic latency measurements (2/3)

```
dispatching hogs: 1 cyclic, 4 vm
stress-ng-cyclic: sched SCHED_RR: 20000 ns delay, 10000 samples
stress-ng-cyclic:   mean: 4164.04 ns, mode: 3791 ns
stress-ng-cyclic:   min: 3547 ns, max: 58286 ns, std.dev. 1068.23
stress-ng-cyclic: latency percentiles:
stress-ng-cyclic:   25.00%:       3813 ns
stress-ng-cyclic:   50.00%:       3993 ns
stress-ng-cyclic:   75.00%:       4233 ns
stress-ng-cyclic:   90.00%:       4588 ns
stress-ng-cyclic:   95.40%:       5025 ns
stress-ng-cyclic:   99.00%:       7397 ns
stress-ng-cyclic:   99.50%:       9936 ns
stress-ng-cyclic:   99.90%:      14758 ns
stress-ng-cyclic:   99.99%:      46148 ns
```

stress-ng cyclic latency measurements (3/3)

```
stress-ng-cyclic: latency distribution (250 ns intervals):
stress-ng-cyclic: (for the first 234 buckets of 234)
stress-ng-cyclic: latency (ns)  frequency
stress-ng-cyclic:            0          0
stress-ng-cyclic:          250          0
...
stress-ng-cyclic:         3250          0
stress-ng-cyclic:         3500       1526
stress-ng-cyclic:         3750       3543
stress-ng-cyclic:         4000       2546
stress-ng-cyclic:         4250       1194
stress-ng-cyclic:         4500        450
stress-ng-cyclic:         4750        267
stress-ng-cyclic:         5000        124
```

stress-ng stressor methods (1/2)

Some stressors have many different methods to stress a system.

```
stress-ng --tree 1 --tree-method avl -t 15s --metrics --log-brief
dispatching hogs: 1 tree
successful run completed in 15.00s
stressor       bogo ops real time  usr time  sys time   bogo ops/s   bogo ops/s
                        (secs)    (secs)    (secs)   (real time) (usr+sys time)
tree                 94     15.00     14.96      0.01         6.27         6.28
```

One can see all the in-built methods using the 'which' option, e.g.

```
stress-ng --vm-method which
vm-method must be one of: all flip galpat-0 galpat-1 gray rowhammer incdec inc-nybble
rand-set rand-sum read64 ror swap move-inv modulo-x prime-0 prime-1 prime-gray-0
prime-gray-1 prime-incdec walk-0d walk-1d walk-0a walk-1a write64 zero-one
```

stress-ng stressor methods (2/2)

The cpu stressor has over **75** different methods, so plenty of different ways to exercise the CPU: float, integer, vector math, mixed math, etc. See the manual for more details.

```
stress-ng --cpu 4 --cpu-method fft -t 10 --metrics --log-brief
dispatching hogs: 4 cpu
successful run completed in 10.00s
stressor       bogo ops real time  usr time  sys time   bogo ops/s   bogo ops/s
                           (secs)    (secs)    (secs)   (real time) (usr+sys time)
cpu                23462     10.00     38.58      0.00      2346.02       608.14
```

Stress tests that have multiple methods will cycle through all the methods by default unless a specific stressor method is specified.

stress-ng verification mode

Most stressors have a verification mode to sanity check test operations.

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
```

stress-ng references

Quick start guide:

https://wiki.ubuntu.com/Kernel/Reference/stress-ng

Main project page:

https://kernel.ubuntu.com/~cking/stress-ng/

GitHub Repo:

https://github.com/ColinIanKing/stress-ng

Manual:

https://kernel.ubuntu.com/~cking/stress-ng/stress-ng.pdf

Kernel Coverage:

https://kernel.ubuntu.com/~cking/kernel-coverage/stress-ng

Patches are always appreciated!

Questions please
Thank you

Colin Ian King
colin.king@canonical.com
https://kernel.ubuntu.com/~cking/stress-ng/
www.canonical.com

CANONICAL

stress-ng bonus material: stressor classes

Each stressor is in one or more classes of stress test. All the stressors in a particular class can be run using the --class option.

Classes are: cpu-cache cpu device filesystem interrupt io memory network os pipe scheduler security vm

Example: run sequentially 2 instances of each CPU cache stressing test for 1 minute per stress test:

```
stress-ng --class cpu-cache --seq 2 -t 1m -v
```

Example: run in parallel all the virtual memory stressors, 1 instance of each stressor:

```
stress-ng --class vm --all 1 -t 1m -v
```

stress-ng bonus material: stressor jobs

One can script stress-ng stress tests using the –jobs or #!/usr/bin/stress-ng.

The stress-ng long options can be put into the script (without the long option dashes). One option per line, the interpreter is very simple.

```
#!/usr/bin/stress-ng
run parallel          # run jobs in parallel
brief                 # metrics
verbose               # verbose output
timeout 5m            # run 5 minutes
af-alg 2              # 2 instances
atomic 4              # 4 instances
bsearch 1             # 1 instance
```

See stress-ng source example-jobs for some job file examples.