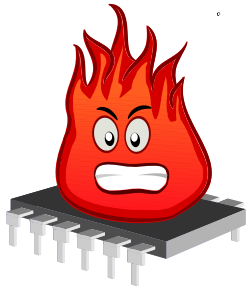
Introduction: Who am I?

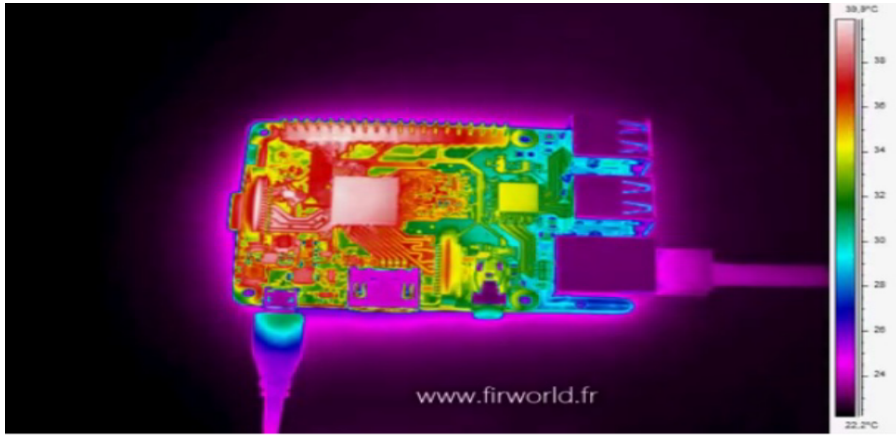
Colin Ian King

- Debian Maintainer (20 packages)
- Principle Engineer @ Intel
- Former Senior Kernel engineer @ Canonical
- Kernel bug fixing + Janitorial work (~3800+ commits)
- Firmware Test Suite (original developer)
- Stress-ng
- I like finding and fixing bugs (breaking/testing H/W and S/W)
- I like free H/W to test on too :-)

Stress-ng: the early days

- Stress “next generation”
- Clean room re-working of stress tool by Amos Waterland
- Originally intended to cause thermal overruns on laptops to sanity check Intel thermal-daemon
- Make CPUs and memory run hot





Stress-ng: Now

- Attempt to break/crash kernels (Linux, BSD, etc)
- Exercise memory, CPU, cache, devices, file systems
- Use (and abuse) kernel interfaces
- Put pressure on memory, scheduler, I/O
- Concurrency – locking and race conditions
- Portability – POSIX and compiler friendly
- Sledge Hammer + Swiss Army Knife


Kernel

Breaking Kernels (DragonFlyBSD)

```
spin_lock: vm_page_spin_lock(0xffffff8043262930), indefinite wait (51 secs)!
spin_lock: vm_page_spin_lock(0xffffff8043262930), indefinite wait (52 secs)!
spin_lock: vm_page_spin_lock(0xffffff8043262930), indefinite wait (53 secs)!
spin_lock: vm_page_spin_lock(0xffffff8043262930), indefinite wait (54 secs)!
spin_lock: vm_page_spin_lock(0xffffff8043262930), indefinite wait (55 secs)!
spin_lock: vm_page_spin_lock(0xffffff8043262930), indefinite wait (56 secs)!
spin_lock: vm_page_spin_lock(0xffffff8043262930), indefinite wait (57 secs)!
spin_lock: vm_page_spin_lock(0xffffff8043262930), indefinite wait (58 secs)!
spin_lock: vm_page_spin_lock(0xffffff8043262930), indefinite wait (59 secs)!
panic with 1 spinlocks held
panic: spin_lock: vm_page_spin_lock(0xffffff8043262930), indefinite wait!
cpuid = 0
Trace beginning at frame 0xffffff80f10f3568
panic() at panic+0x236 0xffffffff805e5d76
panic() at panic+0x236 0xffffffff805e5d76
spin_indefinite_check() at spin_indefinite_check+0xab 0xffffffff8060206b
_spin_lock_contested() at _spin_lock_contested+0xb3 0xffffffff806021b3
vm_page_alloc() at vm_page_alloc+0x2ef 0xffffffff808b5bdf
vm_fault_object() at vm_fault_object+0x804 0xffffffff8089fe24
Debugger("panic")

CPU0 stopping CPUs: 0x00000002
 stopped
Stopped at      Debugger+0x7c:  movb    $0,0xd9cb79(%rip)
db>
```

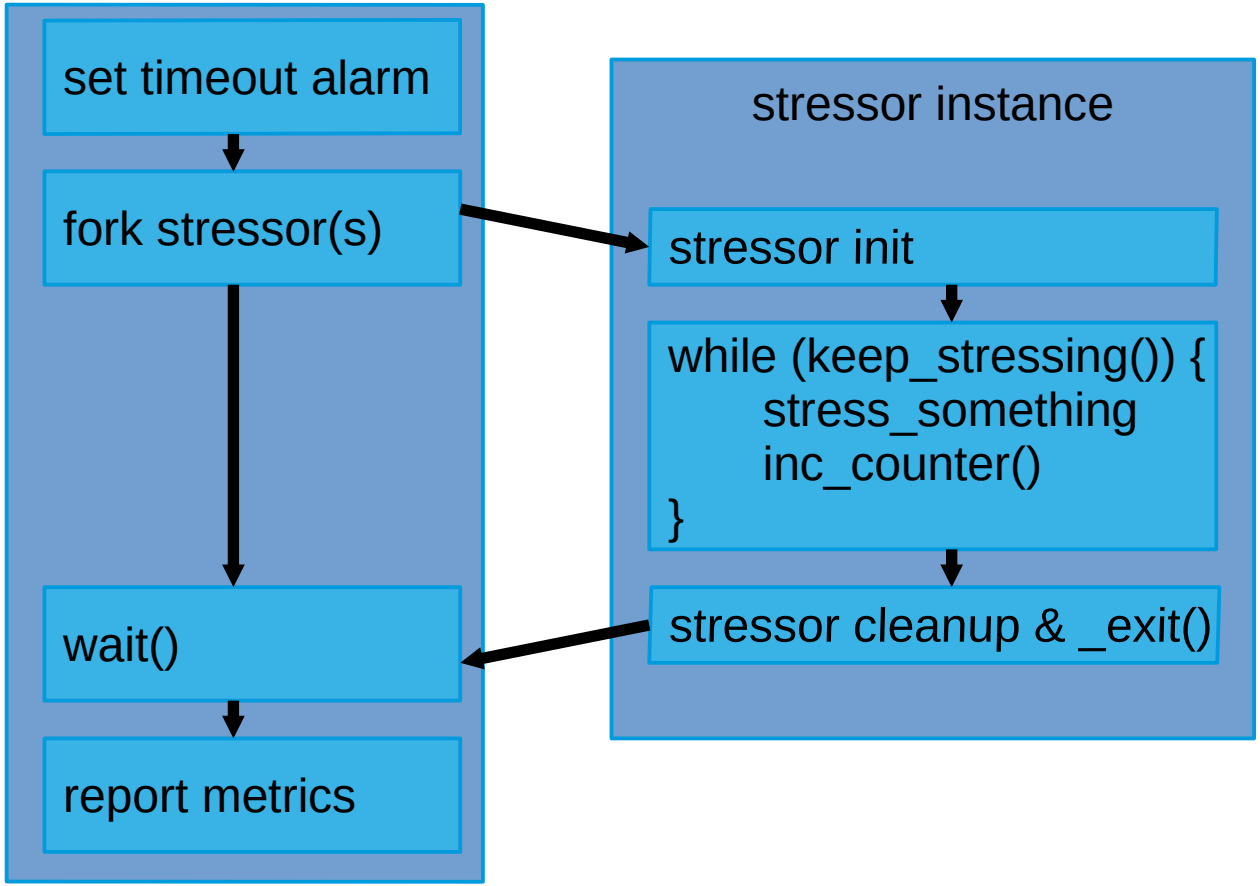
Breaking Kernels (OpenIndiana / Solaris)

```
stress-ng: debug: [19984] stress-ng-fault: page faults: minor: 0, major: 0
stress-ng: debug: [19984] stress-ng-fault: exited [19984] (instance 2)
stress-ng: debug: [19982] stress-ng-fault: exited [19982] (instance 0)
stress-ng: debug: [19981] process [19982] terminated
stress-ng: debug: [19981] process [19983] terminated
stress-ng: debug: [19981] process [19984] terminated
stress-ng: debug: [19981] process [19985] terminated
stress-ng: info:  [19981] successful run completed in 1.12s
fault PASSED
fcntl at May 14, 2018 at 01:11:29 PM UTC
stress-ng: debug: [19988] 4 processors online, 4 processors configured
stress-ng: info:  [19988] dispatching hogs: 4 fcntl
stress-ng: debug: [19988] cache allocate: default cache size: 2048K
stress-ng: debug: [19988] starting stressors
stress-ng: debug: [19989] stress-ng-fcntl: started [19989] (instance 0)
panic[cpu1]/thread=ffffff0264fdeba0: assertion failed: lckdat->l_start == 0, fil
e: ../../common/os/flock.c, line: 312

ffffff0009159ac0 fffffffffba75b18 ()
ffffff0009159c50 genunix:ofdlock+370 ()
ffffff0009159ec0 genunix:fcntl+c13 ()
ffffff0009159f10 unix:brand_sys_sysenter+1c9 ()

dumping to /dev/zvol/dsk/rpool/dump, offset 65536, content: kernel
dumping:  0:01  23% done
```

Stress-ng Design



keep_stressing():

- check for alarm
- check bogo-op counter

inc_counter()

- increment bogo-op counter every loop

counter is in shared memory

Stress-ng Overview

- Stress tool with over 280 stress tests (“stressors”)
- 1 or more instances of each stressor can be run in parallel
- Many different stressors can be run in parallel
- 1 to 4096 stressor instances allowed
- Each stressor is a specific set of related stress cases
- Can mix and match stressors
- Options to report throughput stats, load, perf monitors, system call usage, memory usage, etc.

Stress-ng: Some examples

1 instance of CPU stressor, 5 minutes run time:

stress-ng --cpu 1 -t 5m

8 instances of cache stressor & 4 instances of memory contention stressor for 1 hour:

stress-ng --cache 8 --mcontend 4 -t 1h

0 instances (number of CPUs) of vm stressor with verification, use 95% of available memory, verify tests and run for 1 day:

stress-ng --vm 0 --vm-bytes 95% --verify -t 1d

Stress-ng Size and Time Specifiers

stress-ng --vm 1 --vm-bytes 20G

stress-ng --hdd 1 --hdd-bytes 64M

Sizes can be K (kilobytes), M, G, T, default is bytes

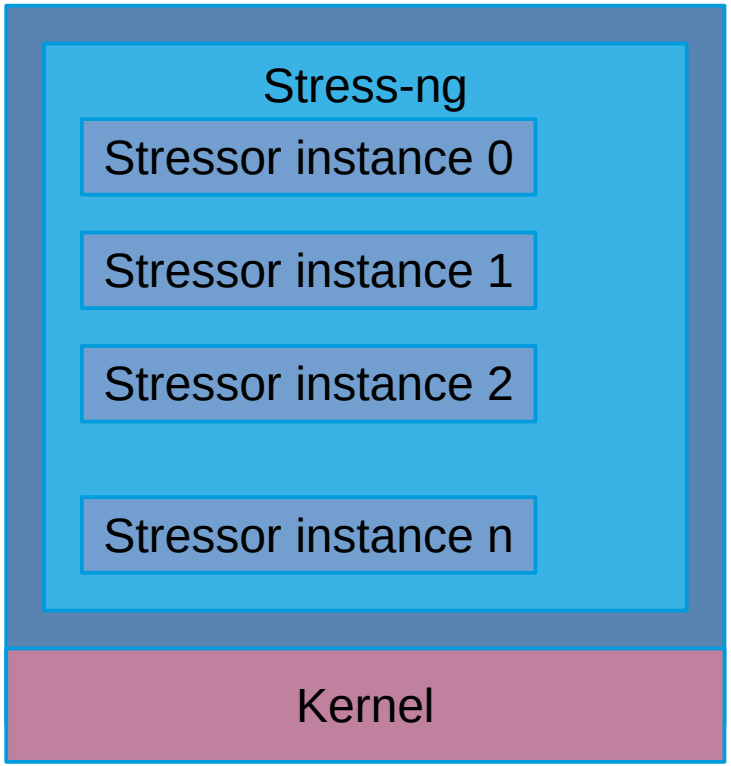
stress-ng --vm 1 --vm-bytes 90%

Sizes can be % of a total resource

stress-ng --cpu 1 -t 1h

Time in seconds or m, h, d, y, default is seconds

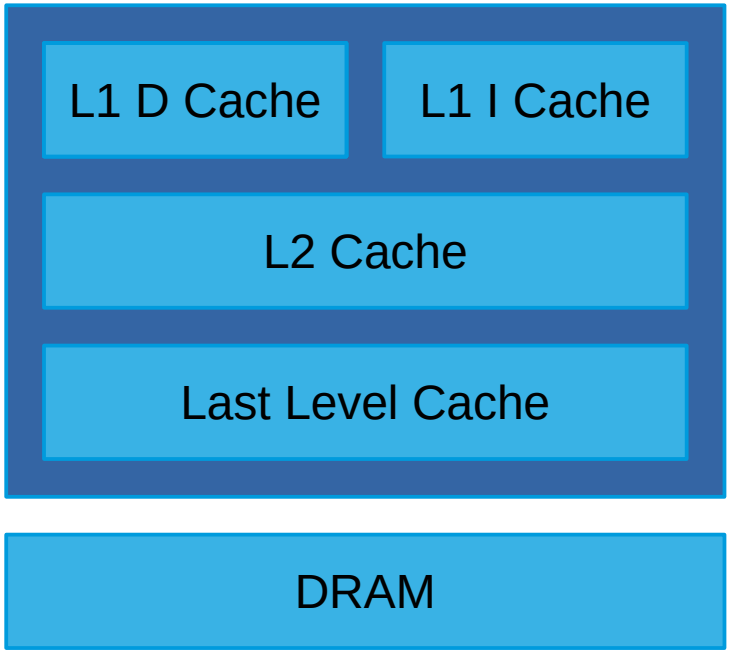
Stressing Using Concurrency



Multiple instances of stressors:

- Each stressor is a child process
- Thousands of instances
- Re-spawned if killed by kernel OOM killer (can be disabled, --oomable)
- Stressors may have their own children (e.g. client/server network tests)
- Stressors may have **many** pthreads
- Try to trigger soft/hard locks and race conditions in kernel and libraries.

Stressing CPU Caches



Cache stressing:

- Data cache: prefetch, fence, sfence, invalidate, flush, etc..
- Exercise cache hit/misses
- Instruction cache: modifying code, cache flushing, cache misses, randomized branching.
- Streaming memory read/writes
- Randomized memory read/writes
- SMP + shared memory exercising

Stressing CPU Caches (examples)

Exercise data cache flushing:

stress-ng --cache 0 --cache-flush

Exercise instruction cache reloads (e.g. x86 self modifying code):

stress-ng --icache 0

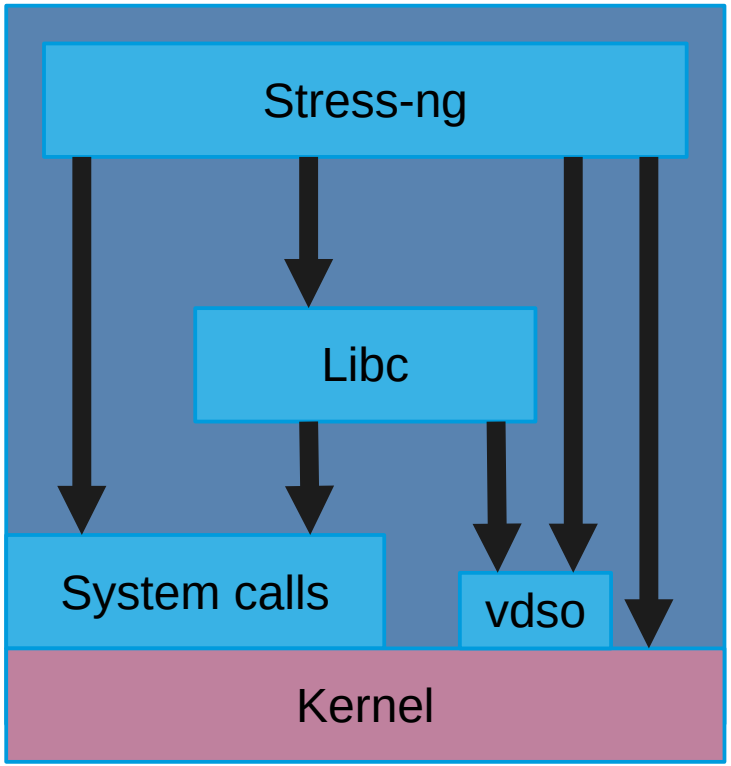
Exercise and verify Level 1 cache on 1 CPU for 1 minute:

stress-ng --l1cache 1 --verify -t 1m

Exercise and benchmark cache prefetching (L3 cache size buffer):

stress-ng --prefetch 1 --metrics -t 1m

Stressing Kernel System Calls



Exercising kernel entry/exit:

- Via libc call interfaces, e.g. select()
- Via libc syscall()
- Via vdso, e.g. gettimeofday()
- Via syscall x86 instruction
- Via x86 int trap
- ~98% of all system calls covered
- Valid and invalid arguments used
- **Not as powerful as syzkaller**

Stressing System Calls (examples)

Exercise system calls with permutations of random invalid arguments:

stress-ng --sysinval 1

Exercise invalid system call numbers:

stress-ng --enosys 1

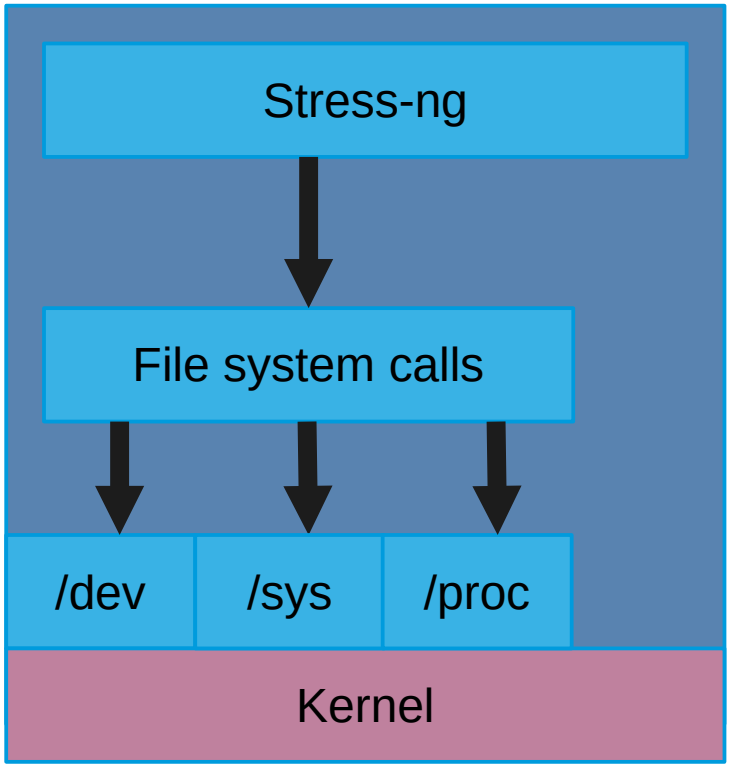
Exercise system calls via the vdso (and benchmark them too)

stress-ng --vdso 1

Exercise x86 system call instruction

stress-ng --x86syscall 1

Stressing Kernel Interfaces



Kernel interfaces, exercise with open(),close(),ioctl(),fcntl(), mmap(),read(), write(), etc..

- Device nodes, /dev
- Sysfs, /sys
- Procfs, /proc
- Multiple processes to force race conditions
- Time-out handling in case interfaces get blocked

Stressing Kernel Interfaces (examples)

Exercising dev interfaces, check for any kernel error messages:

sudo stress-ng --dev 0 --klog-check -t 5m

Exercise /sysfs

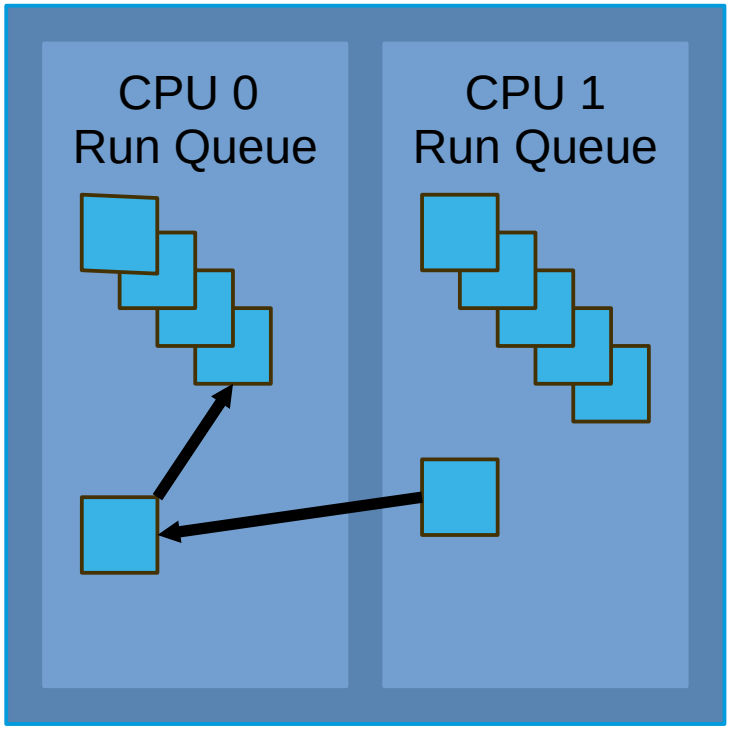
sudo stress-ng --sysfs 0 --klog-check -t 5m

Exercise /procfs

sudo stress-ng --procfs 0 --klog-check -t 5m

Many entries in /dev, /sysfs, /procfs so run stressors for several minutes

Stressing CPU scheduler



- Force heavy context switching
- Create excessive loads and large run queues
- Exercise fork/clone/kill/exit/wait*
- Scheduling / priority / niceness
- NUMA migrations
- Affinity (pin processes to CPUs)
- Locking + priority inversion

Stressing CPU Scheduler (examples)

Process creation using fork, vfork, clone and daemons:

stress-ng --fork 0 --vfork 0 --clone 0 --daemon 0

Create Zombie processes (excessive wait reaping duration):

stress-ng --zombies 0

Excessive context switching:

stress-ng --switch 0

High resolution timer wakeup scheduling, adjust to max freq:

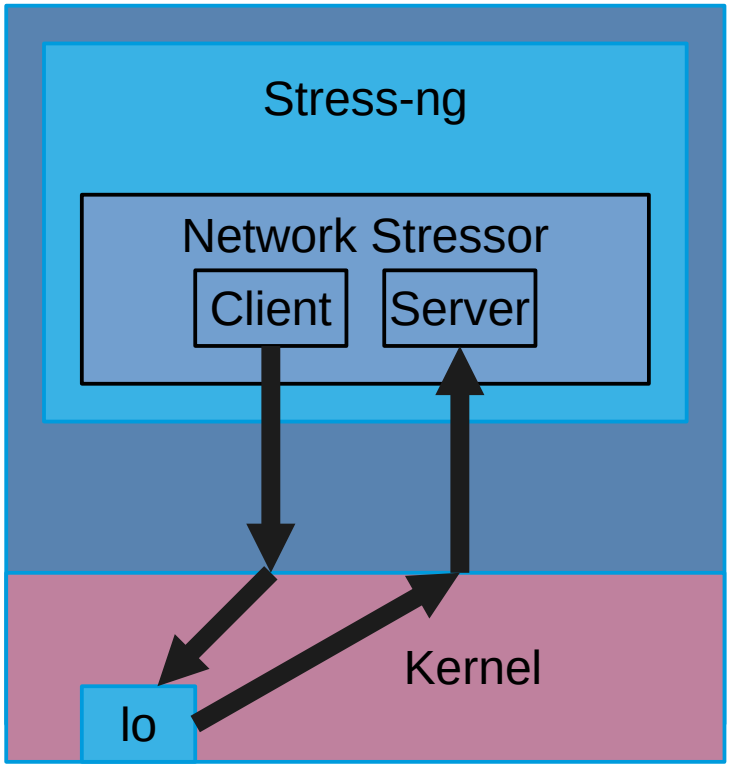
stress-ng --hrtimer 0 --hrtimer-adjust

Stressing CPU Scheduler (cont..)

CPU scheduler stressors:

affinity, clock, clone, cyclic, daemon, exec, exit-group, fifo, fork, futex, hrtimers, itimer, loadavg, mq, msg, mutex, nanosleep, nice, numa, pidfd, pipe, pipeherd, poll, pthread, resched, schedpolicy, sem, sem-sysv, sigabrt, sigchld, sigfd, sigpipe, sigrt, sleep, softlockup, spawn, switch, syncload, timer, timerfd, vfork, vforkmany, wait, yield

Stressing Networking



Partial coverage of networking stack

- UDP, UDP-lite, TCP/IP, MPTCP, DCCP, SCTP, TUN
- ICMP & PING flooding (local host!)
- Raw socket/udp (client/server)
- ioctls() and setsockopt()
- send(), sendmsg(), sendmmsg(), sendto(), write(), etc..
- IPv4, IPv6,UNIX socket

Stress Networking (examples)

Exercise UDP-lite over ipv6 on loopback:

stress-ng --udp 0 --udp-lite --udp-domain ipv6

Exercise TCP/IP socket using sendmsg + zerocopy:

stress-ng --sock 0 --sock-zerocopy --sock-opts sendmsg

Exercise as many TCP/IP socket connections as possible:

stress-ng --sockmany 200 --sockmany-if enp0s31f6

Exercise DCCP and SCTP protocols:

stress-ng --dccp 8 --sctp 4

Stress Networking (examples)

Network stressors:

dccp, epoll, icmp-flood, ping-sock, rawsock, rawpkt, rawudp, sctp, sock, sockabuse, sockdiag, sockfd, sockmany, sockpair, tun, udp, udp-flood

Netlink stressors:

netlink-proc, netlink-task

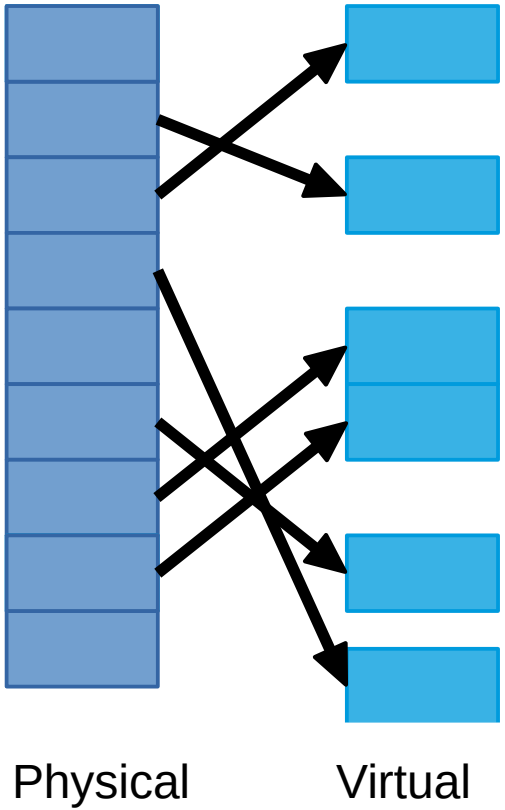
Network device stressor:

netdev

AF_ALG socket crypto:

af-alg

Stressing Virtual Memory



Virtual Memory:

- Pages mapped into memory, mmap()
- Exercise mapping / unmapping / re-mapping
- Change map protection bits
- Exercise page sharing
- Map many pages, force swapping, eviction, etc
- Exercise madvise()
- Exercise page traversal (page table entries etc)
- Exercise Translation Lookaside Buffer

Stressing Virtual Memory (examples)

Exercise 99% of all VM on 8 CPUs, pre-fault in all pages:

stress-ng --vm 8 --vm-bytes 99% --vm-populate --verify

Exercise 32GB of memory using single bit flips and verify:

stress-ng --vm 1 --vm-bytes 32G --vm-method flip --verify

Eat up heap and stack by expanding brk point and stack:

stress-ng --brk 0 --stack 0

Create page faults (minor + major):

stress-ng --fault 0

Stressing Virtual Memory (examples)

Apply random page madvise hints:

stress-ng --madvise 0

Exercise mlock'ing on contiguous pages shared by 1024 children:

stress-ng --mlockmany 1

Exercise random memory mapping on a 4GB file:

stress-ng --mmap 0 --mmap-file --mmap-bytes 4G --verify

Exercise page remapping on 8GB of memory, force mlock'ing:

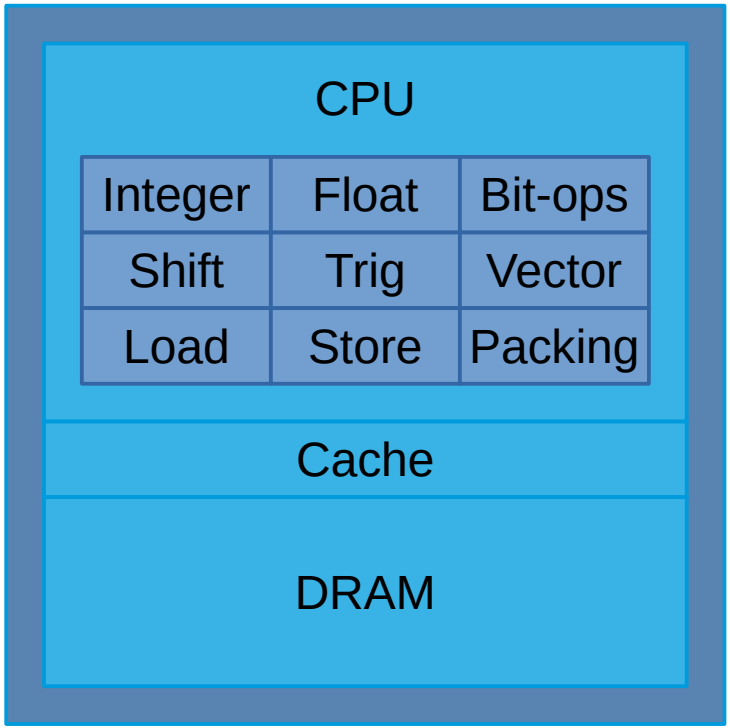
stress-ng --mremap 0 --mremap-bytes 8G --mremap-mlock

Stressing More Memory

Way too many stressors to provide worked examples:

bigheap, brk, bsearch, fault, heapsort, hsearch, idle-page, judy, list, lockbus, lsearch, madvise, malloc, mcontend, membarrier, memcpy, memfd, memhotplug, memrate, memthrash, mergesort, mincore, misaligned, mlock, mlockmany, mmap, mmapaddr, mmapfork, mmapfixed, mmaphuge, mmapmany, mprotect, mremap, msync, msyncmany, munmap, numa, oom-pipe, pageswap, physpage, pkey, prefetch, qsort, radixsort, ramfs, randlist, remap, rmap, secretmem, shellsort, shm, shm-sysv, skiplist, sparsematrix, stack, stackmmap, stream, swap, tlb-shootdown, tmpfs, tree, tsearch, userfaultfd, vm, vm-addr, vm-rw, vm-segv, vm-splice

Stressing Thermal Overrun



- CPU: float, double, decimal, complex, integer, bit-ops, vector ops, shift, add, sub, multiply, divide. Use gcc/clang built-in intrinsics where possible.
- Use gcc target clones for CPU specific optimizations.
- Algorithms: e.g. FFT, hamming code, CRC16, pi, prime, parity, trig functions, ipv4checksum, compression, pack/unpack, etc.
- Cache + DRAM: loads/stores.

Stressing Thermal Overrun (examples)

Exercise mix of CPU stress tests for 5 min, report thermal zone info

stress-ng --cpu 0 -t 5m --tz

Exercise compute + load/stores (mostly cache)

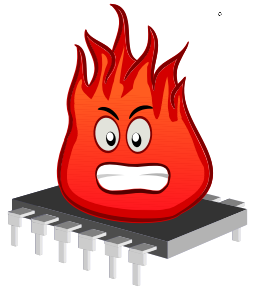
stress-ng --matrix 0 -t 5m --tz

Exercise mixed 128 bit integer and 32 bit decimal math operations:

stress-ng --cpu 0 --cpu-method int128decimal32

Exercise Fast Fourier Transform CPU method

stress-ng --cpu 0 --cpu-method fft



Thermal Zone Information

```
demo$ stress-ng --matrix 0 -t 5m --tz
stress-ng: info:  [350105] setting to a 300 second (5 mins, 0.00 secs) run per stressor
stress-ng: info:  [350105] dispatching hogs: 8 matrix
stress-ng: info:  [350105] matrix:
stress-ng: info:  [350105]                    B0D4   90.05 C (363.20 K)
stress-ng: info:  [350105]      INT3400_Thermal   55.02 C (328.17 K)
stress-ng: info:  [350105]                    SEN1   52.37 C (325.52 K)
stress-ng: info:  [350105]                  acpitz   61.77 C (334.92 K)
stress-ng: info:  [350105]              iwlwifi_1   56.22 C (329.37 K)
stress-ng: info:  [350105]           pch_skylake   58.37 C (331.52 K)
stress-ng: info:  [350105]          x86_pkg_temp   60.87 C (334.02 K)
stress-ng: info:  [350105] successful run completed in 300.62s (5 mins, 0.62 secs)
```

Stressing Compute (examples)

Jpeg compression (using libjpeg)

stress-ng --jpeg 0 --metrics

Zlib compression:

stress-ng --zlib 0 --metrics

Vector instructions (sse, avx, etc) gcc “target clones” used

stress-ng --vecmath 1 --vecwide 1 --metrics

Hashing functions (integer, shift, xor, multiply, add, mem+cache):

stress-ng --hash 0 -v --metrics

Stressing Compute (examples)

Stressing wide generic mix of CPU compute instructions:

stress-ng --cpu 0

Stressing with a specific CPU method (sin/cos/tan trig functions)

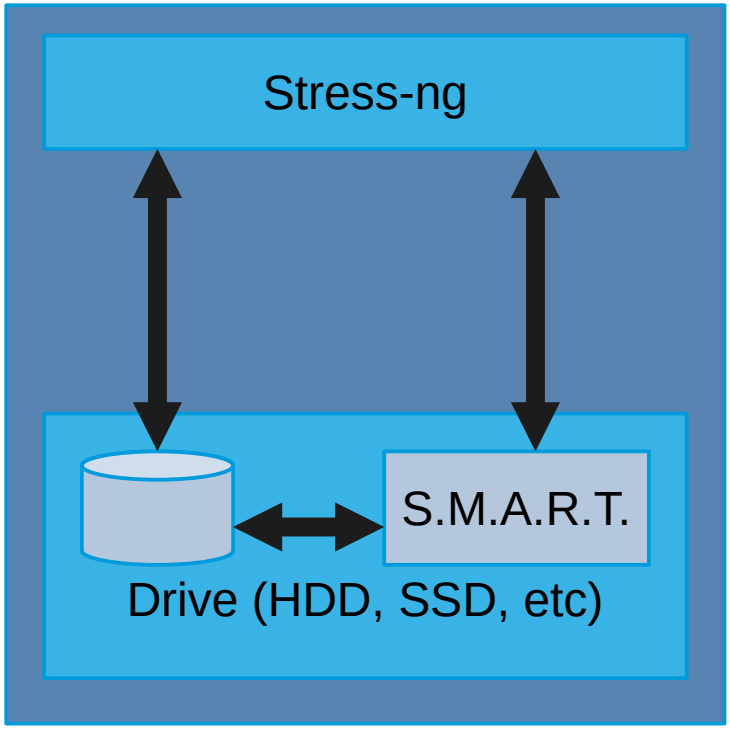
stress-ng --cpu 0 --cpu-method trig

Over 80 CPU methods, show them using:

stress-ng --cpu-method which

ackermann apery bitops callfunc cdouble cfloat clongdouble collatz correlate cpuid crc16 decimal32 decimal64 decimal128 dither div8 div16 div32 div64 div128 double euler explog factorial fibonacci fft fletcher16 float float32 float64 float80 float128 … zeta

Stressing Storage



Exercise:

- raw device (read) & file systems (r/w)
- directories (dentries)
- read/write, sequential/random I/O
- Direct, sync and async I/O
- Syncing, caching, allocation, holes
- File ioctls, fcntl()
- Locking
- **Not a replacement for fio**

Stressing Storage (examples)

Mix of I/O types, check drive S.M.A.R.T., use 10% of free space:

stress-ng --iomix 4 --smart --iomix-bytes 10%

I/O on 1GB file, Random writes, Sequential Reads, I/O stats:

stress-ng --hdd 0 --hdd-bytes 1G --hdd-opts wr-rnd,rd-seq --iostat 1

POSIX and Linux Asynchronous I/O on file system on /mnt:

stress-ng --aio 1 --aiol 1 --temp-path /mnt

Exercise file allocation, 256MB file for 2 hours:

stress-ng --fallocate 0 --fallocate-bytes 256M -t 2h

Stressing Storage (examples)

Misc. stressors:

chattr, chdir, chmod, chown, copy-file, dentry, dir, dirdeep, dirmany, dnotify, fanotify, fcntl, fiemap, file-ioctl, filename, fpunch, getdent, inode-flags, inotify, io, io-uring, lease, link, locka, lockf, lockofd, loop, madvise, quota, ramfs, rawdev, rename, revio, seal, seek, sendfile, sync-file, tmpfs, utime

Example:

stress-ng --revio 1 --io-uring 1 --seek 1 --punch 1 --locka 1

stress-ng --tmpfs 1 --tmpfs-mmap-file --ramfs 1 --ramfs-size 256M

Breaking Filesystem (Minix)

```
x8094ed5 0x8093819 0x804bb83 0x804baa8
VM: pagefault: SIGSEGV 589941 bad addr 0x0; err 0x4 nopage read
stress-ng*F*F 589941 0x0 0x7fccf43 0x8075daa 0x8094ed5 0x8093819 0x804bb83 0x804
baa8
VM: pagefault: SIGSEGV 589941 bad addr 0x0; err 0x4 nopage read
stress-ng*F*F 589941 0x0 0x7fd945c 0x8075eb0 0x7fda600 0x0 0x7fccf43 0x8075daa 0
x8094ed5 0x8093819 0x804bb83 0x804baa8
PM: coredump signal 11 for 3731 / stress-ng
stress-ng*F*F 589941 0x0 0x7fd945c 0x8075eb0 0x7fda600 0x0 0x7fccf43 0x8075daa 0
x8094ed5 0x8093819 0x804bb83 0x804baa8
endpoint 622845 / stress-ng*F*F removed from queue at 65567
endpoint 688161 / stress-ng*F*F removed from queue at 65567
VM: pagefault: SIGSEGV 1 bad addr 0x34; err 0x4 nopage read
     vfs 1      0x805582c 0x8055b1c 0x8048e3d 0x805c5cb 0x8062381 0x8066133
     vfs 1      0x805582c 0x8055b1c 0x8048e3d 0x805c5cb 0x8062381 0x8066133
RS: unable to create service 'vfs' without a replica
RS: unable to clone service (error -1)
core system service died: service 'vfs'* (slot 4, ep 1, pid 7)
      rs 2      0xf1001339 0x8051f77 0x8051c10 0x805190b 0x804c8a4 0x80493a9 0x8
050c62 0x8051879 0x804889c 0x8048273 0x8048198
kernel panic: cause_sig: sig manager 2 gets lethal signal 6 for itself
kernel on CPU 0: 0xf042abdc 0xf042a099 0xf042b2ac 0xf0429dcf 0xf0429d5e 0xf0419d
85

System has panicked, press any key to reboot
```

Stressing Branch Prediction

```
for (;;) {
label0x000:
    seed = (a * seed + c);
    idx = (seed >> 22);
    goto *labels[idx];
…
label0x3fff:
    seed = ….
}
```

- 1024 branch labels
- Random(ish) branching, hard to predict
- Typically ~95% prediction miss rate

Stressing Branch Prediction (examples)

1024 randomized branches, high miss rate shown by perf:

sudo stress-ng --branch 1 --perf

```
demo$ sudo stress-ng --branch 0 --perf -t 60
stress-ng: info:  [351719] setting to a 60 second run per stressor
stress-ng: info:  [351719] dispatching hogs: 8 branch
stress-ng: info:  [351719] successful run completed in 60.50s (1 min, 0.50 secs)
stress-ng: info:  [351719] branch:
stress-ng: info:  [351719]          1,279,251,726,248 CPU Cycles                    21.14 B/sec
stress-ng: info:  [351719]            587,945,201,128 Instructions                   9.72 B/sec (0.460 instr. per cycle)
stress-ng: info:  [351719]             18,574,985,072 Branch Instructions            0.31 B/sec
stress-ng: info:  [351719]             17,394,553,176 Branch Misses                  0.29 B/sec (93.65%)
stress-ng: info:  [351719]             11,393,965,768 Bus Cycles                     0.19 B/sec
stress-ng: info:  [351719]            900,167,024,416 Total Cycles                  14.88 B/sec
```

1024 forward / reverse predictable goto branches, too many for the branch prediction logic to keep a history:

stress-ng --goto 1

Stressing Locking Primitives

Pthread Mutex:

stress-ng --mutex 0

Fast mutex system call (futex):

stress-ng --futex 0

User space shared memory locking mechanisms:

stress-ng --peterson 4 --dekker 4

Atomic operations (e.g. x86 lock'd instructions)

stress-ng --atomic 0

Stress Memory Bandwidth

Memory write/read streaming + some compute (copy, scale *, add +, load/add/scale/store):

stress-ng --stream 1 -t 60

```
demo$ stress-ng --stream 1 -t 1m
stress-ng: info:  [345637] setting to a 60 second run per stressor
stress-ng: info:  [345637] dispatching hogs: 1 stream
stress-ng: info:  [345638] stress-ng-stream: stressor loosely based on a variant of the STREAM benchmark code
stress-ng: info:  [345638] stress-ng-stream: do NOT submit any of these results to the STREAM benchmark results
stress-ng: info:  [345638] stress-ng-stream: Using CPU cache size of 6144K
stress-ng: info:  [345638] stress-ng-stream: memory rate: 18202.39 MB/sec, 7280.95 Mflop/sec (instance 0)
stress-ng: info:  [345637] successful run completed in 60.02s (1 min, 0.02 secs)
```

Stressing memory writes/reads

stress-ng --memrate 1 -t 60

```
demo$ stress-ng --memrate 1 -t 60
stress-ng: info:  [344991] setting to a 60 second run per stressor
stress-ng: info:  [344991] dispatching hogs: 1 memrate
stress-ng: info:  [344992] stress-ng-memrate: write128nt:     20998.77 MB/sec
stress-ng: info:  [344992] stress-ng-memrate:  write64nt:     25009.98 MB/sec
stress-ng: info:  [344992] stress-ng-memrate:  write32nt:     13130.62 MB/sec
stress-ng: info:  [344992] stress-ng-memrate:  write1024:     13139.70 MB/sec
stress-ng: info:  [344992] stress-ng-memrate:   write512:     12988.38 MB/sec
stress-ng: info:  [344992] stress-ng-memrate:   write256:     13049.26 MB/sec
stress-ng: info:  [344992] stress-ng-memrate:   write128:     13259.89 MB/sec
stress-ng: info:  [344992] stress-ng-memrate:    write64:     13023.69 MB/sec
stress-ng: info:  [344992] stress-ng-memrate:    write32:     10400.67 MB/sec
stress-ng: info:  [344992] stress-ng-memrate:    write16:      5646.29 MB/sec
stress-ng: info:  [344992] stress-ng-memrate:     write8:      3061.19 MB/sec
stress-ng: info:  [344992] stress-ng-memrate:   read64pf:     19744.93 MB/sec
stress-ng: info:  [344992] stress-ng-memrate:   read1024:     15680.30 MB/sec
stress-ng: info:  [344992] stress-ng-memrate:    read512:     13700.34 MB/sec
stress-ng: info:  [344992] stress-ng-memrate:    read256:     14762.04 MB/sec
stress-ng: info:  [344992] stress-ng-memrate:    read128:     13650.74 MB/sec
stress-ng: info:  [344992] stress-ng-memrate:     read64:     11547.13 MB/sec
stress-ng: info:  [344992] stress-ng-memrate:     read32:      8249.80 MB/sec
stress-ng: info:  [344992] stress-ng-memrate:     read16:      7632.16 MB/sec
stress-ng: info:  [344992] stress-ng-memrate:      read8:      4956.71 MB/sec
stress-ng: info:  [344991] successful run completed in 60.03s (1 min, 0.03 secs)
```

write*nt:

- non-temporal writes

read*pf:

- prefetched reads

Stressing Inter-Process Communication

POSIX & SYS-V APIs supported, 1 server + 1 client per stressor

Semaphores:

```
stress-ng --sem 1          # POSIX
stress-ng --sem-sysv 1     # SYS-V
```

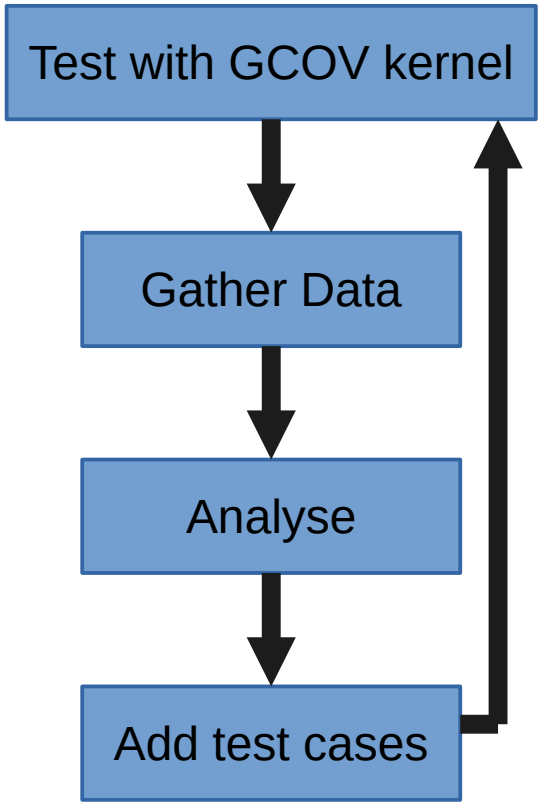
Message Queues:

```
stress-ng --mq 1           # POSIX
stress-ng --msg 1          # SYS-V
```

Pipes:

```
stress-ng  --pipe 1        # see also --pipeherd
```

Data driven improvements: GCOV



GCOV analysis:

- Enable GCOV in kernel
- Run a wide mix of stress cases (see kernel-coverage.sh)

  (Takes 12-15 hours to run in a VM)
- Generate HTML showing test coverage in kernel source using LCOV
- Examine areas where coverage is lacking
- Implement new stress test cases
- Rinse and repeat

LCOV Kernel code coverage

LCOV - code coverage report

			Hit	Total	Coverage
Current view:	top level - fs	**Lines:**	22510	37595	**59.9 %**
Test:	kernel.info	**Functions:**	1843	2925	**63.0 %**
Date:	2022-04-20 23:45:46				

Filename	Line Coverage		Functions	
aio.c	**62.7 %**	652 / 1040	**58.3 %**	49 / 84
anon_inodes.c	**56.8 %**	42 / 74	**70.0 %**	7 / 10
attr.c	**85.7 %**	120 / 140	**100.0 %**	7 / 7
bad_inode.c	**26.7 %**	16 / 60	**12.0 %**	3 / 25
binfmt_elf.c	**44.5 %**	403 / 906	**37.0 %**	10 / 27
binfmt_misc.c	**14.2 %**	57 / 401	**25.0 %**	5 / 20
binfmt_script.c	**85.5 %**	47 / 55	**50.0 %**	2 / 4
buffer.c	**77.8 %**	1220 / 1569	**86.7 %**	85 / 98
char_dev.c	**74.3 %**	208 / 280	**77.8 %**	21 / 27
coredump.c	**26.8 %**	140 / 523	**40.7 %**	11 / 27
d_path.c	**90.4 %**	169 / 187	**88.9 %**	16 / 18
dax.c	**1.4 %**	11 / 779	**4.7 %**	2 / 43
dcache.c	**77.0 %**	1102 / 1432	**79.4 %**	85 / 107
direct-io.c	**83.2 %**	436 / 524	**92.3 %**	24 / 26
drop_caches.c	**100.0 %**	35 / 35	**100.0 %**	2 / 2

LCOV coverage view of fs/fcntl.c

```
395     1660312 :             case F_GETOWN_EX:
396     1660312 :                     err = f_getown_ex(filp, arg);
397     1704642 :                     break;
398     2204914 :             case F_SETOWN_EX:
399     2204914 :                     err = f_setown_ex(filp, arg);
400     2233955 :                     break;
401           0 :             case F_GETOWNER_UIDS:
402           0 :                     err = f_getowner_uids(filp, arg);
403           0 :                     break;
404     3024513 :             case F_GETSIG:
405     3024513 :                     err = filp->f_owner.signum;
406     3024513 :                     break;
407             :             case F_SETSIG:
408             :                     /* arg == 0 restores default behaviour. */
409     3669520 :                     if (!valid_signal(arg)) {
410             :                             break;
411             :                     }
```

Micro-benchmarking

- Bogo-ops: **Bogus** measurement of **operations per second**
- Stressors iterate on a test pattern, 1 loop = 1 bogo op
- **May suffer from jitter**, may need to run stressors for tens of seconds or minutes depending on bogo-op rates
- Can change on newer releases (optimizations, bug fixes etc)
- **Treat with caution! Don’t compare metrics from different versions of stress-ng**
- --metrics option shows the bogo-ops rates

stress-ng --cpu 1 --cpu-ops 10000 --metrics --times

Micro-benchmarking bogo-ops metrics

stress-ng --matrix 1 -t 10 --metrics

```
demo$ stress-ng --matrix 1 -t 10 --metrics
stress-ng: info:  [213076] setting to a 10 second run per stressor
stress-ng: info:  [213076] dispatching hogs: 1 matrix
stress-ng: info:  [213076] stressor       bogo ops real time  usr time  sys time   bogo ops/s     bogo ops/s CPU used per
stress-ng: info:  [213076]                          (secs)    (secs)    (secs)   (real time) (usr+sys time) instance (%)
stress-ng: info:  [213076] matrix            33864     10.00      9.98      0.00      3386.41        3393.19        99.80
stress-ng: info:  [213076] successful run completed in 10.00s
```

stress-ng --matrix 8 -t 10 --metrics

```
demo$ stress-ng --matrix 8 -t 10 --metrics
stress-ng: info:  [213390] setting to a 10 second run per stressor
stress-ng: info:  [213390] dispatching hogs: 8 matrix
stress-ng: info:  [213390] stressor       bogo ops real time  usr time  sys time   bogo ops/s     bogo ops/s CPU used per
stress-ng: info:  [213390]                          (secs)    (secs)    (secs)   (real time) (usr+sys time) instance (%)
stress-ng: info:  [213390] matrix           150028     10.00     79.16      0.00     15002.80        1895.25        98.95
stress-ng: info:  [213390] successful run completed in 10.00s
```

Stress-ng + Perf

Get perf stats at the end of the run:

sudo stress-ng --stream 1 -t 15 --perf

```
stress-ng: info:  [346299]          36,404,208,191 CPU Cycles                       2.34 B/sec
stress-ng: info:  [346299]          36,855,579,442 Instructions                     2.37 B/sec (1.012 instr. per cycle)
stress-ng: info:  [346299]           4,940,759,436 Branch Instructions              0.32 B/sec
stress-ng: info:  [346299]               850,202 Branch Misses                     54.60 K/sec ( 0.02%)
stress-ng: info:  [346299]             355,877,210 Bus Cycles                      22.85 M/sec
stress-ng: info:  [346299]          28,110,403,675 Total Cycles                     1.81 B/sec
stress-ng: info:  [346299]           7,746,000,593 Cache References                 0.50 B/sec
stress-ng: info:  [346299]           4,589,566,752 Cache Misses                     0.29 B/sec (59.25%)
stress-ng: info:  [346299]           9,564,597,031 Cache L1D Read                   0.61 B/sec
stress-ng: info:  [346299]           3,716,027,832 Cache L1D Read Miss              0.24 B/sec
stress-ng: info:  [346299]           5,873,399,394 Cache L1D Write                  0.38 B/sec
stress-ng: info:  [346299]               9,487,835 Cache L1I Read Miss              0.61 M/sec
stress-ng: info:  [346299]           1,032,742,232 Cache LL Read                   66.32 M/sec
stress-ng: info:  [346299]             327,509,314 Cache LL Read Miss              21.03 M/sec
stress-ng: info:  [346299]           1,205,701,863 Cache LL Write                  77.42 M/sec
stress-ng: info:  [346299]             529,021,725 Cache LL Write Miss             33.97 M/sec
stress-ng: info:  [346299]           9,572,395,695 Cache DTLB Read                  0.61 B/sec
stress-ng: info:  [346299]              39,247,049 Cache DTLB Read Miss             2.52 M/sec
```

Stress-ng Run-Time Statistics

Show system utilization, e.g. every 2 seconds during run:

stress-ng --brk 1 --iomix 1 --vmstat 2

Show thermal measurements every second:

stress-ng --cpu 0 --thermalstat 1

Show I/O statistics every 5 seconds:

stress-ng --iomix 4 --iostat 5

```
demo$ stress-ng --iomix 4 --iostat 5
stress-ng: info:  [346534] defaulting to a 86400 second (1 day, 0.00 secs) run per stressor
stress-ng: info:  [346534] dispatching hogs: 4 iomix
stress-ng: info:  [346535] iostat: Inflght  Rd K/s   Wr K/s Dscd K/s     Rd/s     Wr/s   Dscd/s
stress-ng: info:  [346535] iostat       0       50    59425        0        3     7757        0
stress-ng: info:  [346535] iostat       0       68    93134        0        4     8090        0
stress-ng: info:  [346535] iostat       0      125    93085        0       11     8328        0
```

Stressor Classes

Stressors are classified into **one or more** classes:

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

E.g. async I/O stressor (aio) is in the I/O, interrupt and os classes

Run all filesystem stressors sequentially on 8 CPUs, each stressor will run for 1 minute before moving to next stressor:

stress-ng --class filesystem --seq 8 -t 1m

Stressing Real-Time and Low-Latency Kernels

Cyclic timer measurements measure latency jitter, e.g. for RT or LL kernel benchmarking

Run 1 instance of cyclic with some stressors

Specify latency distribution bucket size, the timer to use, scheduler policy and scheduling priority:

```
sudo stress-ng --cyclic 1 --cyclic-dist 10000 \
--cyclic-method clock_ns --cyclic-policy rr \
--cyclic-prio 90 --iomix 4 --mmap 8 -t 5m
```

Cyclic Stressor (Example)

```
stress-ng: info:  [349502] stress-ng-cyclic: sched SCHED_RR: 100000 ns delay, 10000 samples
stress-ng: info:  [349502] stress-ng-cyclic:   mean: 7787.84 ns, mode: 6182 ns
stress-ng: info:  [349502] stress-ng-cyclic:   min: 3777 ns, max: 464676 ns, std.dev. 13669.42
stress-ng: info:  [349502] stress-ng-cyclic: latency percentiles:
stress-ng: info:  [349502] stress-ng-cyclic:   25.00%:       5743 ns
stress-ng: info:  [349502] stress-ng-cyclic:   50.00%:       6383 ns
stress-ng: info:  [349502] stress-ng-cyclic:   75.00%:       7159 ns
stress-ng: info:  [349502] stress-ng-cyclic:   90.00%:       8495 ns
stress-ng: info:  [349502] stress-ng-cyclic:   95.40%:      10527 ns
stress-ng: info:  [349502] stress-ng-cyclic:   99.00%:      29767 ns
stress-ng: info:  [349502] stress-ng-cyclic:   99.50%:      84446 ns
stress-ng: info:  [349502] stress-ng-cyclic:   99.90%:     201913 ns
stress-ng: info:  [349502] stress-ng-cyclic:   99.99%:     464676 ns
stress-ng: info:  [349502] stress-ng-cyclic: latency distribution (10000 ns intervals):
stress-ng: info:  [349502] stress-ng-cyclic: (for the first 47 buckets of 47)
stress-ng: info:  [349502] stress-ng-cyclic: latency (ns)  frequency
stress-ng: info:  [349502] stress-ng-cyclic:            0       9446
stress-ng: info:  [349502] stress-ng-cyclic:        10000        398
stress-ng: info:  [349502] stress-ng-cyclic:        20000         57
stress-ng: info:  [349502] stress-ng-cyclic:        30000         25
stress-ng: info:  [349502] stress-ng-cyclic:        40000         10
stress-ng: info:  [349502] stress-ng-cyclic:        50000          3
stress-ng: info:  [349502] stress-ng-cyclic:        60000          5
stress-ng: info:  [349502] stress-ng-cyclic:        70000          4
stress-ng: info:  [349502] stress-ng-cyclic:        80000          5
stress-ng: info:  [349502] stress-ng-cyclic:        90000          0
```

Evil stressing

SMI (x86 only) – System Management Mode Interrupt (ring -2)

sudo stress-ng --smi --pathological

Out of Memory (OOM) and paging:

sudo stress-ng --brk 0 --oom-pipe 0 --stack 0 --page-in --thrash

Override x86 P-state controls (e.g. set by thermald)

sudo stress-ng --cpu 0 --ignite-cpu

Force bus locking memory accesses (bad on NUMA + SMP)

stress-ng --lockbus 0

Stress-ng is Portable

- Build-time auto-detection of system call APIs, structures, types, compiler features, assembler/instructions, optimizations..
- Compiles with: gcc, clang, tcc, pcc, icc
- Architectures: alpha, armhf, arm64, hppa, i386, m68k, mips32, mips64, ppc64el, RISC-V, s390x, sparc64, x86-64
- Operating systems: Linux, Solaris, *BSD, Minix, Android, MacOS X, GNU/Hurd, Haiku, POSIX systems.
- glibc or musl C

Building Stress-ng

Clone and build:

```
git clone https://github.com/ColinIanKing/stress-ng
cd stress-ng
make clean && make
make install  # optional
make pdf      # make PDF manual
```

Pull updates and force re-configuration and build:

```
git pull && git clean -f && make
make install  # optional
```

Stress-ng Structure

stress-ng.c	main stress framework
core*.c	core support code, helpers, abstraction shims
stress-*.c	stressors
test/*	build-time configuration tests
debian/*	Debian packaging
Makefile	GNU make file
Makefile.config	make file for build-time configuration checks

Stress-ng Pre-release checks

Quality checks:

- Static analysis using cppcheck, clang scan-build
- Build checks with gcc-8 to gcc-12, clang-9 to clang-13, tcc, pcc, icc
- PEDANTIC=1 UNEXPECTED=1 make
- Tested on Fedora, OpenSUSE, Slackware, Ubuntu, Debian, Solaris, NetBSD, OpenBSD, FreeBSD, DragonFlyBSD, Minix, OS X, Haiku, GNU/Hurd
- Tested on alpha, armhf, arm64, hppa, i386, m68k, mips32, mips64, ppc64el, risc-v, s390x, sparc64, x86-64
- Most tests using QEMU

Further Work

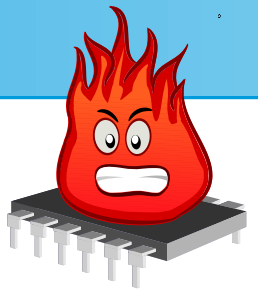
- Improve ioctl() coverage
- Add more network domain / protocols
- Add support for the family of new mount system calls
- Keep in sync with new system calls
- Improve non-x86 arch test coverage
- Improve driver test coverage – most of the kernel code and hence most of the kernel bugs

...contributions to stress-ng welcome!

Results

- 180+ stressors
- 45+ kernel bugs found (Linux, Minix, *BSD, etc)
- ~15 kernel improvements based on micro-benchmarking results
- 0-Day CI Linux Kernel Performance Testing
- Kernel regression testing (Ubuntu, etc)
- Used in research for cloud/kernel stress testing (50+ citations)
- Microcode regression testing (e.g. H/W clocks)

Project Information



- Project: https://github.com/ColinIanKing/stress-ng
- Bugs: https://github.com/ColinIanKing/stress-ng/issues
- Quickstart: https://wiki.ubuntu.com/Kernel/Reference/stress-ng
- 74 page manual (man page or PDF)
- Install:

  sudo apt install stress-ng

  yum install stress-ng

Stress-ng Community

Patches always welcome.
Fixes and bug reports are appreciated!