



ELISA

Enabling **Linux** in
Safety Applications

ELISA Workshop London, UK

June 9-11, 2026
Co-hosted with Canonical





ELISA

Enabling **Linux** in
Safety Applications

Improving kernel test coverage with stress-ng

Colin Ian King

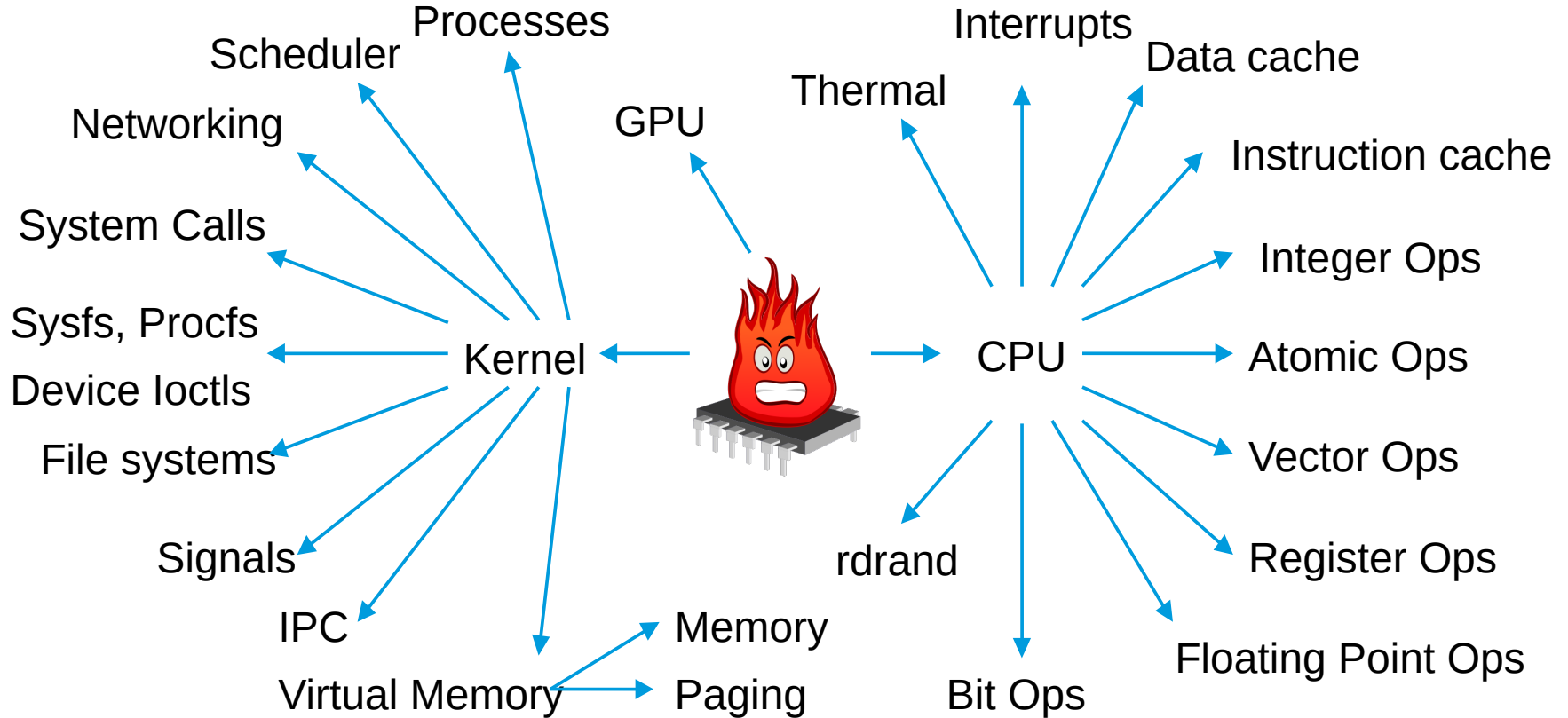
ELISA Workshop: London, UK

June 9-11, 2026

Co-hosted with Canonical



Stress-ng – stressors types

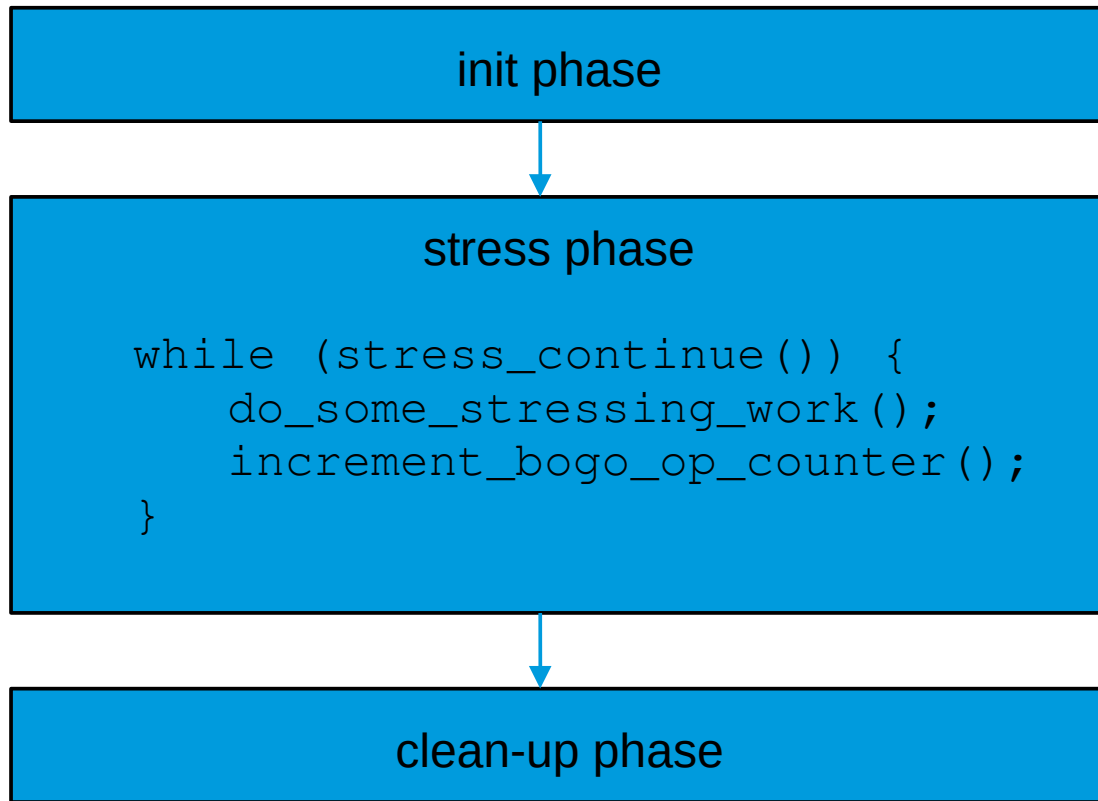


Stress-ng – what is a stressor?

Normally a single process forked from stress-ng

Stressor may be one or more child process or one or more pthreads in more complex stress cases.

Stressor terminates on SIGALRM or reached maximum bogo-op count



Examples of using stress-ng

- `stress-ng --mmap 8 --timeout 5m`
 - Run 8 mmap stressors for 5 minutes
- `stress-ng --mmap -1 --pageswap 1 -t 1h`
 - Run mmap stressors on all online CPUs and 1 instance of swapping pages for 1 hour
- `stress-ng --class scheduler --seq 4 -t 30s`
 - Run sequentially through all scheduler stressors, 4 instances of each stressor, 30 seconds per stressor
- `stress-ng --seq -1 -t 1m --progress --metrics --tz --c-states`
 - Run sequentially through all stressors on all online CPUs, showing progress, performance metrics, thermal zone temperatures and CPU C-states

The on-going stress-ng work

- Add stress tests for new system calls
- Add stress tests for new kernel/user API interfaces such as new ioctls() and system call options/flags
- Add stress tests for new /proc and /sys files
- Clean up stress-ng option handling and core helper code (now completed)
- Improve net core coverage
- Kernel gcov analysis, fill in untested core kernel code paths
- Merge contributor code and fix bugs
- Make regular releases (4-6 week release cadence)
- **Big thank you** for Linux Foundation ELISA project for sponsoring stress-ng work!

Plugging untested holes using gcov (part 1)

LCOV - code coverage report

Current view: top level - ipc		Coverage	Total	Hit
Test: kernel.info	Lines:	75.7 %	4151	3143
Test Date: 2026-06-02 09:40:30	Functions:	69.2 %	325	225

File	Line Coverage ↕			Function Coverage ↕		
	Rate	Total	Hit	Rate	Total	Hit
compat.c	<div></div>	0.0 %	32		0.0 %	4
ipc_sysctl.c	<div></div>	84.1 %	107	90	81.8 %	11
mq_sysctl.c	<div></div>	90.6 %	53	48	100.0 %	6
mqueue.c	<div></div>	83.1 %	816	678	64.4 %	73
msg.c	<div></div>	72.6 %	634	460	60.0 %	50
msgutil.c	<div></div>	89.7 %	87	78	83.3 %	6
namespace.c	<div></div>	67.7 %	127	86	90.9 %	11
sem.c	<div></div>	77.3 %	1064	823	69.4 %	62
shm.c	<div></div>	69.5 %	862	599	68.7 %	67
syscall.c	<div></div>	0.0 %	43		0.0 %	3
util.c	<div></div>	86.3 %	292	252	90.3 %	31
util.h	<div></div>	85.3 %	34	29	100.0 %	1

Plugging untested holes using gcov (part 2)

ipc/shm.c, shmctl_down():

```
1016      1141355 :      switch (cmd) {
1017      1141355 :      case IPC_RMID:
1018      1141355 :          ipc_lock_object(&shp->shm_perm);
1019      :          /* do_shm_rmid unlocks the ipc object and rcu */
1020      1141355 :          do_shm_rmid(ns, ipcp);
1021      1141355 :          goto out_up;
1022      0 :      case IPC_SET:
1023      0 :          ipc_lock_object(&shp->shm_perm);
1024      0 :          err = ipc_update_perm(&shmid64->shm_perm, ipcp);
1025      0 :          if (err)
1026      0 :              goto out_unlock0;
1027      0 :          shp->shm_ctim = ktime_get_real_seconds();
1028      0 :          break;
1029      0 :      default:
1030      0 :          err = -EINVAL;
1031      0 :          goto out_unlock1;
1032      :      }
```

Need to add extra code to stress-shm.c stressor to exercise the red code paths

Plugging untested holes using gcov (part 3)

ipc/shm.c, do_shmat()

```
1620      1493734 :      err = -ENOMEM;
1621      1493734 :      sfd = kzalloc_obj(*sfd);
1622      1490630 :      if (!sfd) {
1623          0 :          fput(base);
1624          0 :          goto out_nattch;
1625          :      }
```

Force out of memory conditions:

- Set appropriate fault injection settings in `/sys/kernel/debug/fail*`
- `sudo stress-ng --shm 1 --make-it-fail`

see <https://docs.kernel.org/fault-injection/fault-injection.html>

New stressors since June 2024 (kernel scheduling)

- `cpu-sched` exercise CPU affinity and scheduler policies
- `min-nanosleep` nanosleep sleeps, attempts smallest feasible sleeps
- `timermix` mix of high frequency timer and itimer signals
- `varyload` scheduler variable load generator
- `sighup` `SIGHUP` (hang-up signal)
- `sigurg` socket `SIGURG` (urgent signal)
- `sigill` `SIGILL` (illegal instruction signal)
- `sigvtalrm` `SIGVTALRM` (itimer virtual alarm signal)

New stressors since June 2024 (filesystem)

- file-ioctl file ioctl() operations
- filehole randomly create and zero fill holes in files
- filerace randomize racy operations on files
- lockmix mixed file locking
- pseek randomized pread(), pwrite(), lseek() file operations
- rofs exercise read-only file systems
- umask exercise all umask() flag combinations

New stressors since June 2024 (miscellaneous)

- fd-abuse perform valid and invalid operations on file descriptors (files, sockets, etc)
- fd-race pthread racy shared file descriptor operations over a socket
- eth-sniff simple and fast ethernet packet sniffer
- easy-opcode simple op-codes to overdrive instruction fetch and decoder pipeline

New stressors since June 2024 (compute)

- bitops bit-wise operations (and,or,exor,not..)
- chyperbolic complex hyperbolic trigonometric functions (libm)
- crc cyclic redundancy check
- ctrig complex trigonometric functions (libm)
- dfp decimal floating point math (libm)
- intmath integer math operations (various widths)
- fp-misc floating point misc macros/functions (libm)
- hyperbolic hyperbolic trigonometric functions (libm)
- strnum string to/from number conversions (libc) (can overheat x86 CPUs!)
- veccmp vector comparisons
- regex regular expressions

New stressors since June 2024 (cache/memory)

- bubblesort classic bubblesort
- cachehammer random cache operations (flush, prefetch, invalidate, read/write, etc)
- fibsearch search using Fibonacci series for search steps
- flipflop pthreads contending memory on cmpxchg operation
- tlb-numa NUMA Translation Lookaside Buffer (TLB) shutdowns
- mapcow memory mapping Copy-on-Write page operations
- mmaprandom randomised mix of memory mapping operations
- mmaptorture randomised file base memory mapping torture operations
- numacopy NUMA page copying operations
- physmmap /dev/mem physical page mapping
- ptr-chase cache/memory page-size pointer chasing
- spinmem synchronisation using memory based spin loop (cache/snoop stress)

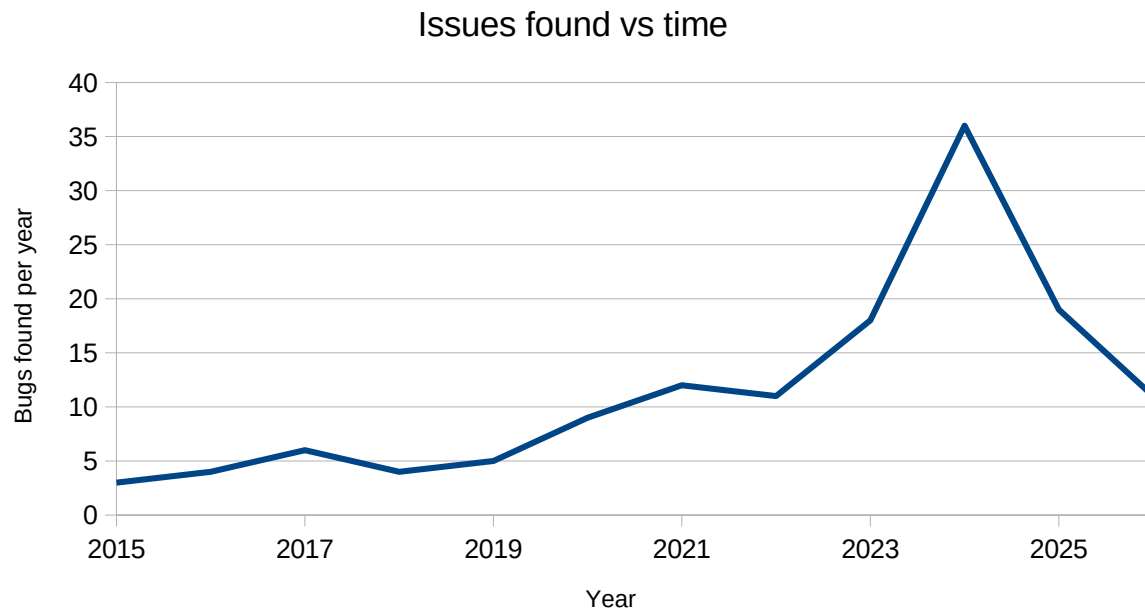
New --class options since June 2024

Example: `stress-ng --class hot --seq 0 -t 60 --progress -tz`

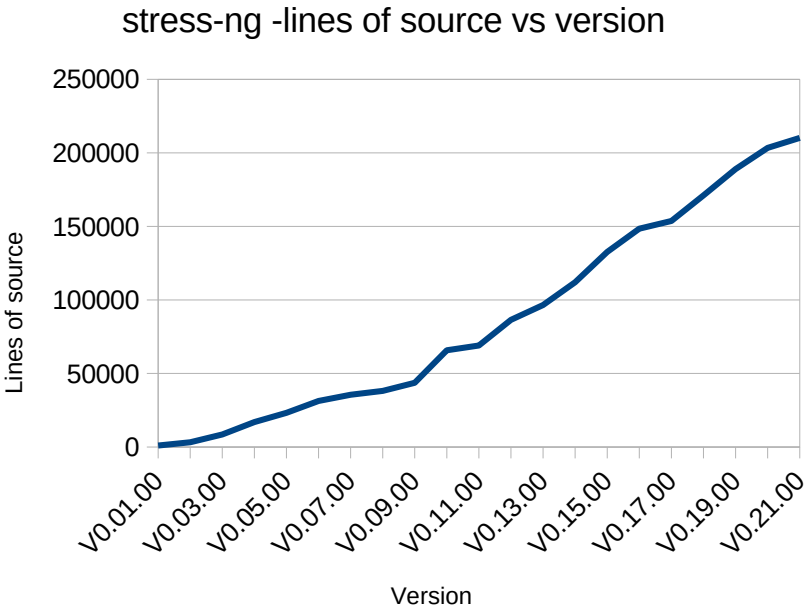
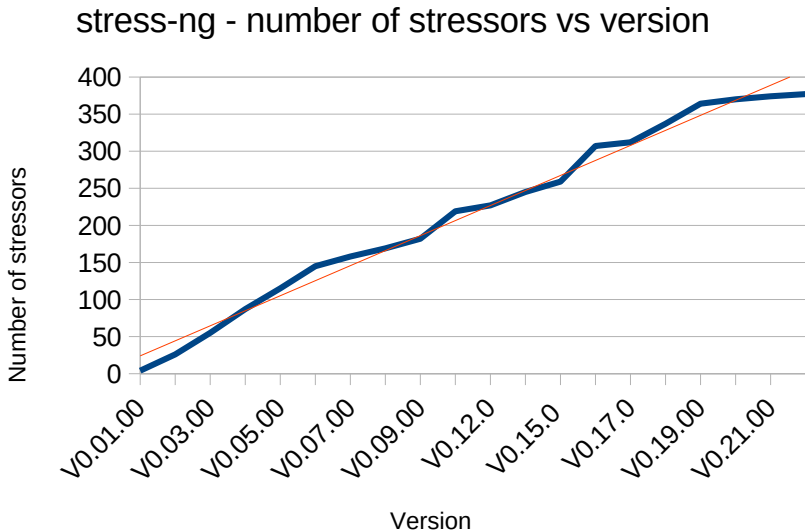
- `fp` - floating point operations (libm and cpu compute)
- `hot` - make system run hot (cache, cpu, memory, vector)
- `integer` - integer operations (cpu compute)
- `ipc` - kernel inter-process communication system calls (mutex, semaphores, pipes)
- `sort` - sorting and searching (memory, cache)
- `tlb` - translation lookaside buffer shootdowns (cache, TLB)
- `vector` - integer and floating point vector operations

Available classes: `compute` `cpu-cache` `cpu` `device` `integer` `filesystem` `fp` `gpu` `hot` `interrupt` `io` `ipc` `memory` `network` `os` `pipe` `scheduler` `search` `security` `signal` `sort` `tlb` `vector` `vm`

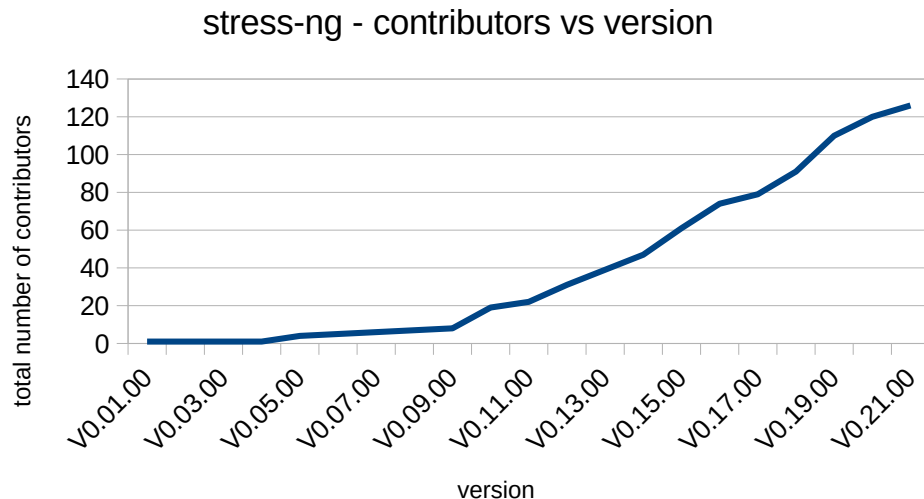
Stress-ng – issues found so far..



Stress-ng – growth in stressors; 40+ new stressors since June 2024



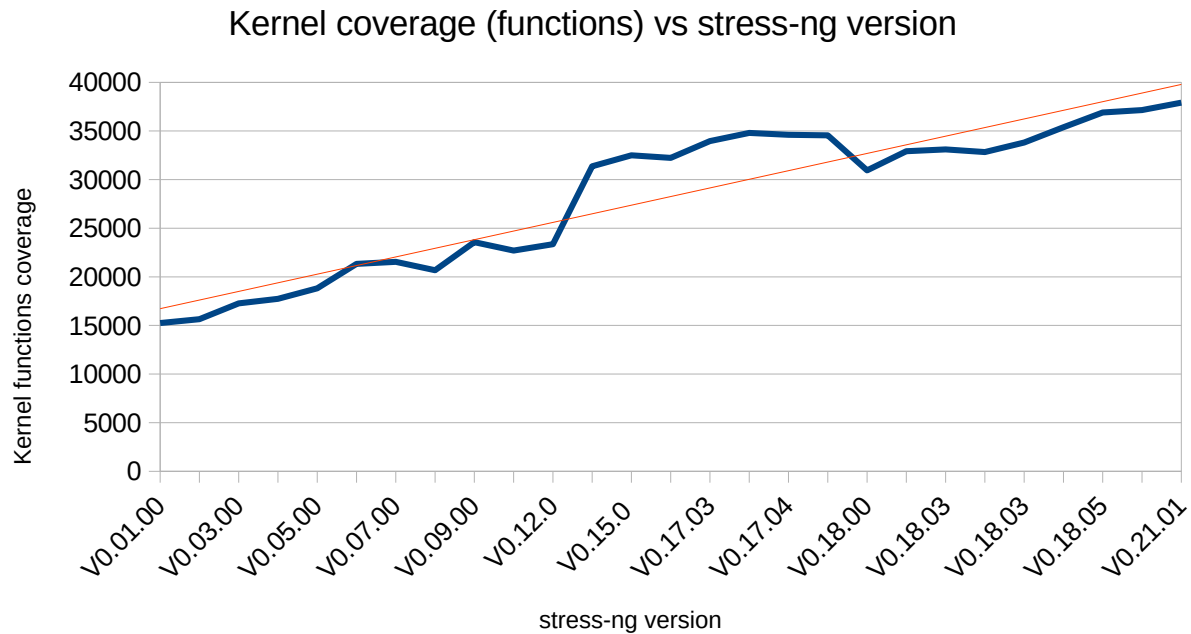
Stress-ng – growth in project contributors (expanding community)



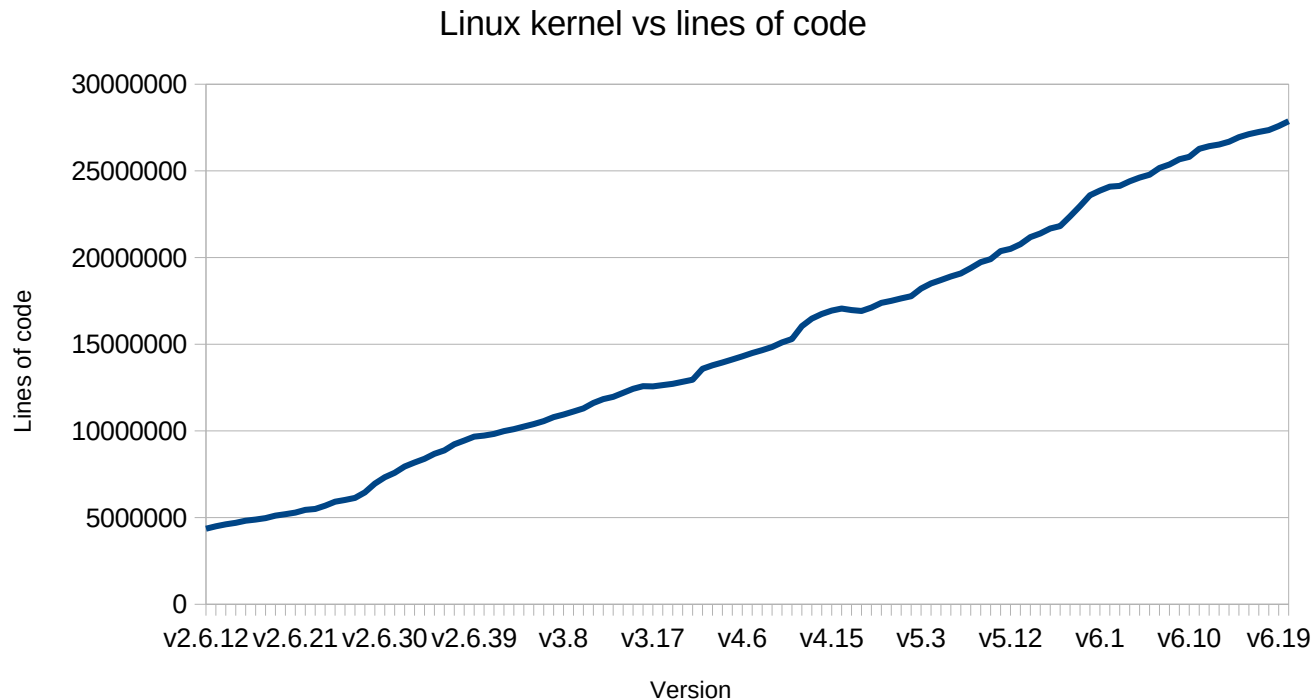
New features and fixes are always welcome!

- Merge requests at <https://github.com/ColinIanKing/stress-ng>
- Patches sent to colin.i.king@gmail.com

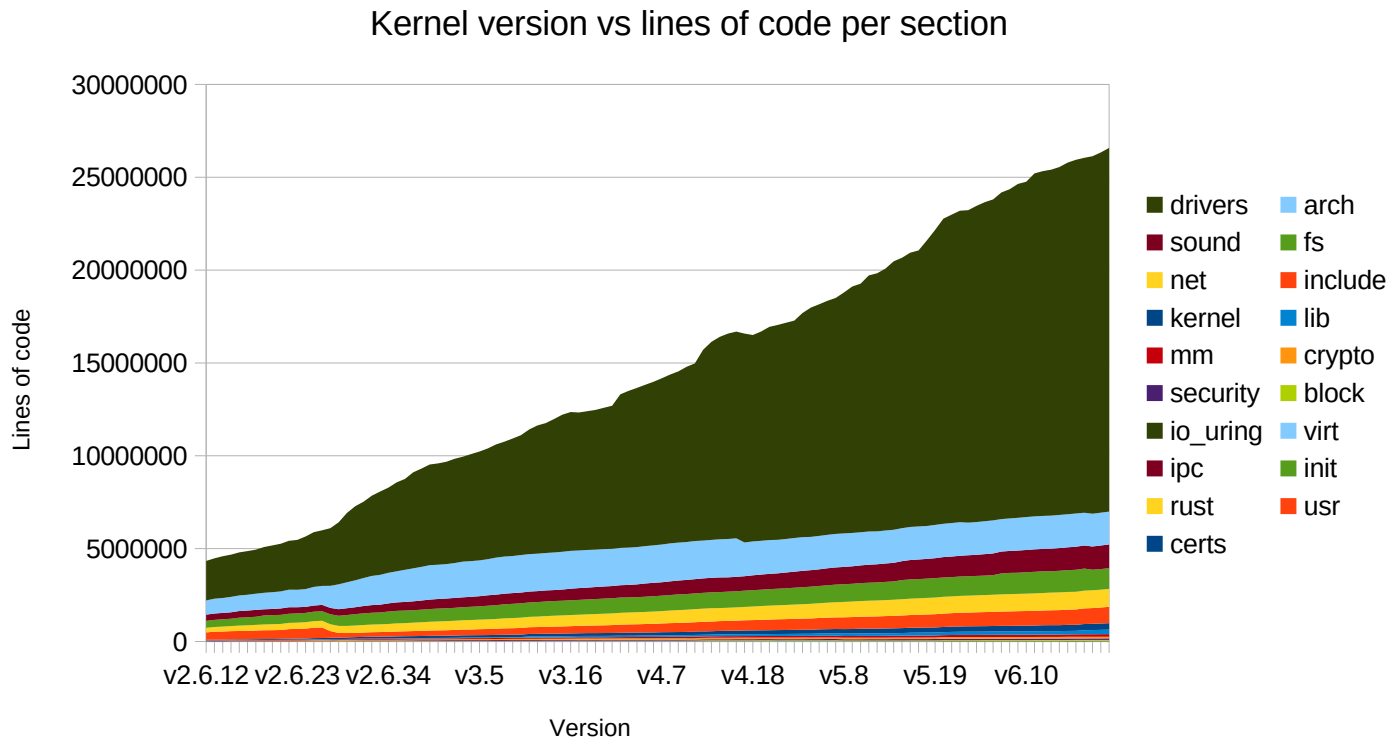
Linux kernel coverage with stress-ng (kernel functions)



Testing problem: Unstoppable kernel growth!



Where does all the kernel code go?



Stress-ng kernel coverage:

- Reasonably good coverage in kernel core directories:
 - ipc, virt, block, security, crypto, mm, kernel (2.55% of total kernel)
- Moderate coverage in kernel core directories:
 - lib, net, arch, io_uring (10.75% of total kernel)
- **Little to no coverage** for kernel drivers directories:
 - sound (4.59% of total kernel)
 - drivers (70.31% of total kernel)

Static analysis using CoverityScan shows that most bugs in the kernel occur in drivers, and these are the least tested code paths by stress-ng!

Kernel drivers are a problem!

- Probably the buggiest kernel code, and lowest code test coverage
- Hard to test drivers without hardware or suitable emulation
- Drivers normally require specialised stress-tests (/dev ioctl() interfaces, etc)
- Many drivers (8,773 driver modules in linux 7.0)
- Lots of functionality, not that much documentation, makes exhaustive testing hard

Kernel driver testing solutions – future work for 2026 onwards

- Identify common driver core code
- Identify drivers that are commonly used
- Identify well used and well documented core code interfaces to stress-test
- Work on low-hanging fruit – most impact with smallest amount of stress-test code to develop
- Try to be as generic as possible for classes of drivers
- Need input on what's a priority...
 - e.g i2c, network drivers, etc..?



Any questions?

Email: colin.i.king@gmail.com

Repo: <https://github.com/ColinIanKing/stress-ng>



Licensing of Workshop Results

All work created during the workshop is licensed under Creative Commons Attribution 4.0 International (CC-BY-4.0) [<https://creativecommons.org/licenses/by/4.0/>] by default, or under another suitable open-source license, e.g., GPL-2.0 for kernel code contributions.

You are free to:

- Share — copy and redistribute the material in any medium or format
- Adapt — remix, transform, and build upon the material for any purpose, even commercially.

The licensor cannot revoke these freedoms as long as you follow the license terms.

Under the following terms:

Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.

No additional restrictions — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.