NAME

stress−ng − a tool to load and stress a computer system

SYNOPSIS

stress−ng [*OPTION* [*ARG*]] ...

DESCRIPTION

stress−ng will stress test a computer system in various selectable ways. It was designed to exercise various physical subsystems of a computer as well as the various operating system kernel interfaces. stress−ng also has a wide range of CPU specific stress tests that exercise floating point, integer, bit manipulation and control flow.

stress−ng was originally intended to make a machine work hard and trip hardware issues such as thermal overruns as well as operating system bugs that only occur when a system is being thrashed hard. Use stress−ng with caution as some of the tests can make a system run hot on poorly designed hardware and also can cause excessive system thrashing which may be difficult to stop.

stress−ng can also measure test throughput rates; this can be useful to observe performance changes across different operating system releases or types of hardware. However, it has never been intended to be used as a precise benchmark test suite, so do NOT use it in this manner.

Running stress−ng with root privileges will adjust out of memory settings on Linux systems to make the stressors unkillable in low memory situations, so use this judiciously. With the appropriate privilege, stress−ng can allow the ionice class and ionice levels to be adjusted, again, this should be used with care.

One can specify the number of processes to invoke per type of stress test; specifying a negative or zero value will select the number of processors available as defined by sysconf(_SC_NPROCESSORS_CONF).

OPTIONS

General stress−ng control options:

−−abort

this option will force all running stressors to abort (terminate) if any other stressor terminates prematurely because of a failure.

−−aggressive

enables more file, cache and memory aggressive options. This may slow tests down, increase latencies and reduce the number of bogo ops as well as changing the balance of user time vs system time used depending on the type of stressor being used.

−a N, −−all N, −−parallel N

start N instances of all stressors in parallel. If N is less than zero, then the number of CPUs online is used for the number of instances. If N is zero, then the number of CPUs in the system is used.

−b N, −−backoff N

wait N microseconds between the start of each stress worker process. This allows one to ramp up the stress tests over time.

−−class name

specify the class of stressors to run. Stressors are classified into one or more of the following classes: cpu, cpu-cache, device, io, interrupt, filesystem, memory, network, os, pipe, scheduler and vm. Some stressors fall into just one class. For example the 'get' stressor is just in the 'os' class. Other stressors fall into more than one class, for example, the 'lsearch' stressor falls into the 'cpu', 'cpu-cache' and 'memory' classes as it exercises all these three. Selecting a specific class will run all the stressors that fall into that class only when run with the −−sequential option.

Specifying a name followed by a question mark (for example −−class vm?) will print out all the stressors in that specific class.

−n, −−dry−run
parse options, but do not run stress tests. A no-op.

−−ftrace
enable kernel function call tracing (Linux only). This will use the kernel debugfs ftrace mechanism to record all the kernel functions used on the system while stress-ng is running. This is only as accurate as the kernel ftrace output, so there may be some variability on the data reported.

−h, −−help
show help.

−−ignite−cpu
alter kernel controls to try and maximize the CPU. This requires root privilege to alter various /sys interface controls. Currently this only works for Intel P-State enabled x86 systems on Linux.

−−ionice−class class
specify ionice class (only on Linux). Can be idle (default), besteffort, be, realtime, rt.

−−ionice−level level
specify ionice level (only on Linux). For idle, 0 is the only possible option. For besteffort or realtime values 0 (highest priority) to 7 (lowest priority). See ionice(1) for more details.

−−job jobfile
run stressors using a jobfile. The jobfile is essentially a file containing stress-ng options (without the leading −−) with one option per line. Lines may have comments with comment text proceeded by the # character. A simple example is as follows:

```
run sequential   # run stressors sequentially
verbose          # verbose output
metrics-brief    # show metrics at end of run
timeout 60s      # stop each stressor after 60 seconds
#
# vm stressor options:
#
vm 2             # 2 vm stressors
vm-bytes 128M    # 128MB available memory
vm-keep          # keep vm mapping
vm-populate      # populate memory
#
# memcpy stressor options:
#
memcpy 5         # 5 memcpy stressors
```

The job file introduces the run command that specifies how to run the stressors:

run sequential − run stressors sequentially
run parallel − run stressors together in parallel

Note that 'run parallel' is the default.

−k, −−keep−name
by default, stress−ng will attempt to change the name of the stress processes according to their functionality; this option disables this and keeps the process names to be the name of the parent process, that is, stress−ng.

−−log−brief
by default stress−ng will report the name of the program, the message type and the process id as a prefix to all output. The −−log−brief option will output messages without these fields to produce a less verbose output.

−−log−file filename
write messages to the specified log file.

−−maximize
overrides the default stressor settings and instead sets these to the maximum settings allowed. These defaults can always be overridden by the per stressor settings options if required.

−−max−fd N
set the maximum limit on file descriptors (value or a % of system allowed maximum). By default, stress-ng can use all the available file descriptors; this option sets the limit in the range from 10 up to the maximum limit of RLIMIT_NOFILE. One can use a % setting too, e.g. 50% is half the maximum allowed file descriptors. Note that stress-ng will use about 5 of the available file descriptors so take this into consideration when using this setting.

−−metrics
output number of bogo operations in total performed by the stress processes. Note that these are not a reliable metric of performance or throughput and have not been designed to be used for benchmarking whatsoever. The metrics are just a useful way to observe how a system behaves when under various kinds of load.

The following columns of information are output:

Column Heading	**Explanation**
bogo ops	number of iterations of the stressor during the run. This is metric of how much overall "work" has been achieved in bogo operations.
real time (secs)	average wall clock duration (in seconds) of the stressor. This is the total wall clock time of all the instances of that particular stressor divided by the number of these stressors being run.
usr time (secs)	total user time (in seconds) consumed running all the instances of the stressor.
sys time (secs)	total system time (in seconds) consumed running all the instances of the stressor.
bogo ops/s (real time)	total bogo operations per second based on wall clock run time. The wall clock time reflects the apparent run time. The more processors one has on a system the more the work load can be distributed onto these and hence the wall clock time will reduce and the bogo ops rate will increase. This is essentially the "apparent" bogo ops rate of the system.
bogo ops/s (usr+sys time)	total bogo operations per second based on cumulative user and system time. This is the real bogo ops rate of the system taking into consideration the actual time execution time of the stressor across all the processors. Generally this will decrease as one adds more concurrent stressors due to contention on cache, memory, execution units, buses and I/O devices.

−−metrics−brief
enable metrics and only output metrics that are non-zero.

−−minimize
overrides the default stressor settings and instead sets these to the minimum settings allowed. These defaults can always be overridden by the per stressor settings options if required.

−−no−madvise
from version 0.02.26 stress−ng automatically calls madvise(2) with random advise options before each mmap and munmap to stress the vm subsystem a little harder. The −−no−advise option turns this default off.

−−no−rand−seed
Do not seed the stress-ng pseudo-random number generator with a quasi random start seed, but instead seed it with constant values. This forces tests to run each time using the same start conditions which can be useful when one requires reproducible stress tests.

−−oomable
Do not respawn a stressor if it gets killed by the Out-of-Memory (OOM) killer. The default behaviour is to restart a new instance of a stressor if the kernel OOM killer terminates the process. This option disables this default behaviour.

−−page−in
touch allocated pages that are not in core, forcing them to be paged back in. This is a useful option to force all the allocated pages to be paged in when using the bigheap, mmap and vm stressors. It will severely degrade performance when the memory in the system is less than the allocated buffer sizes. This uses mincore(2) to determine the pages that are not in core and hence need touching to page them back in.

−−pathological
enable stressors that are known to hang systems. Some stressors can quickly consume resources in such a way that they can rapidly hang a system before the kernel can OOM kill them. These stressors are not enabled by default, this option enables them, but you probably don't want to do this. You have been warned.

−−perf measure processor and system activity using perf events. Linux only and caveat emptor, according to perf_event_open(2): "Always double-check your results! Various generalized events have had wrong values.". Note that with Linux 4.7 one needs to have CAP_SYS_ADMIN capabilities for this option to work, or adjust /proc/sys/kernel/perf_event_paranoid to below 2 to use this without CAP_SYS_ADMIN.

−q, −−quiet
do not show any output.

−r N, −−random N
start N random stress workers. If N is 0, then the number of configured processors is used for N.

−−sched scheduler
select the named scheduler (only on Linux). To see the list of available schedulers use: stress−ng −−sched which

−−sched−prio prio
select the scheduler priority level (only on Linux). If the scheduler does not support this then the default priority level of 0 is chosen.

−−sched−period period
select the period parameter for deadline scheduler (only on Linux). Default value is 0 (in nanoseconds).

−−sched−runtime runtime
select the runtime parameter for deadline scheduler (only on Linux). Default value is 99999 (in nanoseconds).

−−sched−deadline deadline
select the deadline parameter for deadline scheduler (only on Linux). Default value is 100000 (in nanoseconds).

−−sched−reclaim
use cpu bandwidth reclaim feature for deadline scheduler (only on Linux).

−−sequential N
sequentially run all the stressors one by one for a default of 60 seconds. The number of instances of each of the individual stressors to be started is N. If N is less than zero, then the number of CPUs online is used for the number of instances. If N is zero, then the number of CPUs in the

system is used. Use the −−timeout option to specify the duration to run each stressor.

−−stressors
output the names of the available stressors.

−−syslog
log output (except for verbose −v messages) to the syslog.

−−taskset list
set CPU affinity based on the list of CPUs provided; stress-ng is bound to just use these CPUs (Linux only). The CPUs to be used are specified by a comma separated list of CPU (0 to N-1). One can specify a range of CPUs using '-', for example: −−taskset 0,2-3,6,7-11

−−temp−path path
specify a path for stress−ng temporary directories and temporary files; the default path is the current working directory. This path must have read and write access for the stress-ng stress processes.

−−thrash
This can only be used when running on Linux and with root privilege. This option starts a background thrasher process that works through all the processes on a system and tries to page as many pages in the processes as possible. This will cause considerable amount of thrashing of swap on an over-committed system.

−t N, −−timeout T
stop stress test after T seconds. One can also specify the units of time in seconds, minutes, hours, days or years with the suffix s, m, h, d or y. Note: A timeout of 0 will run stress-ng without any timeouts (run forever).

−-timestamp
add a timestamp in hours, minutes, seconds and hundredths of a second to the log output.

−−timer−slack N
adjust the per process timer slack to N nanoseconds (Linux only). Increasing the timer slack allows the kernel to coalesce timer events by adding some fuzziness to timer expiration times and hence reduce wakeups. Conversely, decreasing the timer slack will increase wakeups. A value of 0 for the timer-slack will set the system default of 50,000 nanoseconds.

−−times
show the cumulative user and system times of all the child processes at the end of the stress run. The percentage of utilisation of available CPU time is also calculated from the number of on-line CPUs in the system.

−−tz collect temperatures from the available thermal zones on the machine (Linux only). Some devices may have one or more thermal zones, where as others may have none.

−v, −−verbose
show all debug, warnings and normal information output.

−−verify
verify results when a test is run. This is not available on all tests. This will sanity check the computations or memory contents from a test run and report to stderr any unexpected failures.

−V, −−version
show version of stress-ng, version of toolchain used to build stress-ng and system information.

−x, −−exclude list
specify a list of one or more stressors to exclude (that is, do not run them). This is useful to exclude specific stressors when one selects many stressors to run using the −−class option, −−sequential, −−all and −−random options. Example, run the cpu class stressors concurrently and exclude the numa and search stressors:

stress−ng −−class cpu −−all 1 −x numa,bsearch,hsearch,lsearch

−Y, −−yaml filename
output gathered statistics to a YAML formatted file named 'filename'.

Stressor specific options:

−−access N
start N workers that work through various settings of file mode bits (read, write, execute) for the file owner and checks if the user permissions of the file using access(2) and faccessat(2) are sane.

−−access−ops N
stop access workers after N bogo access sanity checks.

−−affinity N
start N workers that run 16 processes that rapidly change CPU affinity (only on Linux). Rapidly switching CPU affinity can contribute to poor cache behaviour and high context switch rate.

−−affinity−ops N
stop affinity workers after N bogo affinity operations (only on Linux). Note that the counters across the 16 processes are not locked to improve affinity test rates so the final number of bogo-ops will be equal or more than the specified ops stop threshold because of racy unlocked bogo-op counting.

−−affinity−rand
switch CPU affinity randomly rather than the default of sequentially.

−−af−alg N
start N workers that exercise the AF_ALG socket domain by hashing and encrypting various sized random messages. This exercises the available hashes, ciphers, rng and aead crypto engines in the Linux kernel.

−−af−alg−ops N
stop af−alg workers after N AF_ALG messages are hashed.

−−af−alg−dump
dump the internal list representing cryptographic algorithms parsed from the /proc/crypto file to standard output (stdout).

−−aio N
start N workers that issue multiple small asynchronous I/O writes and reads on a relatively small temporary file using the POSIX aio interface. This will just hit the file system cache and soak up a lot of user and kernel time in issuing and handling I/O requests. By default, each worker process will handle 16 concurrent I/O requests.

−−aio−ops N
stop POSIX asynchronous I/O workers after N bogo asynchronous I/O requests.

−−aio−requests N
specify the number of POSIX asynchronous I/O requests each worker should issue, the default is 16; 1 to 4096 are allowed.

−−aiol N
start N workers that issue multiple 4K random asynchronous I/O writes using the Linux aio system calls io_setup(2), io_submit(2), io_getevents(2) and io_destroy(2). By default, each worker process will handle 16 concurrent I/O requests.

−−aiol−ops N
stop Linux asynchronous I/O workers after N bogo asynchronous I/O requests.

−−aiol−requests N
specify the number of Linux asynchronous I/O requests each worker should issue, the default is 16; 1 to 4096 are allowed.

−−apparmor N
start N workers that exercise various parts of the AppArmor interface. Currently one needs root permission to run this particular test. Only available on Linux systems with AppArmor support and requires the CAP_MAC_ADMIN capability.

−−apparmor-ops
stop the AppArmor workers after N bogo operations.

−−atomic N
start N workers that exercise various GCC __atomic_*() built in operations on 8, 16, 32 and 64 bit integers that are shared among the N workers. This stressor is only available for builds using GCC 4.7.4 or higher. The stressor forces many front end cache stalls and cache references.

−−atomic−ops N
stop the atomic workers after N bogo atomic operations.

−−bad−altstack N
start N workers that create broken alternative signal stacks for SIGSEGV handling that in turn create secondary SIGSEGVs. A variety of randonly selected methods are used to create the stacks:

a) Unmapping the alternative signal stack, before triggering the signal handling.

b) Changing the alternative signal stack to just being read only, write only, execute only.

c) Using a NULL alternative signal stack.

d) Using the signal handler object as the alternative signal stack.

e) Unmapping the alternative signal stack during execution of the signal handler.

−−bad−altstack−ops N
stop the bad alternative stack stressors after N SIGSEGV bogo operations.

−−bad−ioctl N
start N workers that perform a range of illegal bad read ioctls (using _IOR) across the device drivers. This exercises page size, 64 bit, 32 bit, 16 bit and 8 bit reads as well as NULL addresses, non-readable pages and PROT_NONE mapped pages. Currently only for Linux and requires the --pathological option.

−−bad−ioctl−ops N
stop the bad ioctl stressors after N bogo ioctl operations.

−B N, −−bigheap N
start N workers that grow their heaps by reallocating memory. If the out of memory killer (OOM) on Linux kills the worker or the allocation fails then the allocating process starts all over again. Note that the OOM adjustment for the worker is set so that the OOM killer will treat these workers as the first candidate processes to kill.

−−bigheap−ops N
stop the big heap workers after N bogo allocation operations are completed.

−−bigheap−growth N
specify amount of memory to grow heap by per iteration. Size can be from 4K to 64MB. Default is 64K.

−−binderfs N
start N workers that mount, exercise and unmount binderfs. The binder control device is exercised with 256 sequential BINDER_CTL_ADD ioctl calls per loop.

−−binderfs−ops N
stop after N binderfs cycles.

−−bind−mount N
start N workers that repeatedly bind mount / to / inside a user namespace. This can consume resources rapidly, forcing out of memory situations. Do not use this stressor unless you want to risk hanging your machine.

−−bind−mount−ops N
stop after N bind mount bogo operations.

−−branch N
start N workers that randomly jump to 256 randomly selected locations and hence exercise the CPU branch prediction logic.

−−branch−ops N
stop the branch stressors after N jumps

−−brk N
start N workers that grow the data segment by one page at a time using multiple brk(2) calls. Each successfully allocated new page is touched to ensure it is resident in memory. If an out of memory condition occurs then the test will reset the data segment to the point before it started and repeat the data segment resizing over again. The process adjusts the out of memory setting so that it may be killed by the out of memory (OOM) killer before other processes. If it is killed by the OOM killer then it will be automatically re-started by a monitoring parent process.

−−brk−ops N
stop the brk workers after N bogo brk operations.

−−brk−mlock
attempt to mlock future brk pages into memory causing more memory pressure. If mlock(MCL_FUTURE) is implemented then this will stop new brk pages from being swapped out.

−−brk−notouch
do not touch each newly allocated data segment page. This disables the default of touching each newly allocated page and hence avoids the kernel from necessarily backing the page with real physical memory.

−−bsearch N
start N workers that binary search a sorted array of 32 bit integers using bsearch(3). By default, there are 65536 elements in the array. This is a useful method to exercise random access of memory and processor cache.

−−bsearch−ops N
stop the bsearch worker after N bogo bsearch operations are completed.

−−bsearch−size N
specify the size (number of 32 bit integers) in the array to bsearch. Size can be from 1K to 4M.

−C N, −−cache N
start N workers that perform random wide spread memory read and writes to thrash the CPU cache. The code does not intelligently determine the CPU cache configuration and so it may be sub-optimal in producing hit-miss read/write activity for some processors.

−−cache−fence
force write serialization on each store operation (x86 only). This is a no-op for non-x86 architectures.

−−cache−flush
force flush cache on each store operation (x86 only). This is a no-op for non-x86 architectures.

−−cache−level N
specify level of cache to exercise (1=L1 cache, 2=L2 cache, 3=L3/LLC cache (the default)). If the cache hierarchy cannot be determined, built-in defaults will apply.

−−cache−no−affinity
do not change processor affinity when **−−cache** is in effect.

−−cache−sfence
force write serialization on each store operation using the sfence instruction (x86 only). This is a no-op for non-x86 architectures.

−−cache−ops N
stop cache thrash workers after N bogo cache thrash operations.

−−cache−prefetch
force read prefetch on next read address on architectures that support prefetching.

−−cache−ways N
specify the number of cache ways to exercise. This allows a subset of the overall cache size to be exercised.

−−cap N
start N workers that read per process capabilities via calls to capget(2) (Linux only).

−−cap−ops N
stop after N cap bogo operations.

−−chattr N
start N workers that attempt to exercise file attributes via the EXT2_IOC_SETFLAGS ioctl. This is intended to be intentionally racy and exercise a range of chattr attributes by enabling and disabling them on a file shared amongst the N chattr stressor processes. (Linux only).

−−chattr−ops N
stop after N chattr bogo operations.

−−chdir N
start N workers that change directory between directories using chdir(2).

−−chdir−ops N
stop after N chdir bogo operations.

−−chdir−dirs N
exercise chdir on N directories. The default is 8192 directories, this allows 64 to 65536 directories to be used instead.

−−chmod N
start N workers that change the file mode bits via chmod(2) and fchmod(2) on the same file. The greater the value for N then the more contention on the single file. The stressor will work through all the combination of mode bits.

−−chmod−ops N
stop after N chmod bogo operations.

−−chown N
start N workers that exercise chown(2) on the same file. The greater the value for N then the more contention on the single file.

−−chown−ops N
stop the chown workers after N bogo chown(2) operations.

−−chroot N
start N workers that exercise chroot(2) on various valid and invalid chroot paths. Only available on Linux systems and requires the CAP_SYS_ADMIN capability.

−−chroot−ops N
stop the chroot workers after N bogo chroot(2) operations.

−−clock N
start N workers exercising clocks and POSIX timers. For all known clock types this will exercise clock_getres(2), clock_gettime(2) and clock_nanosleep(2). For all known timers it will create a 50000ns timer and busy poll this until it expires. This stressor will cause frequent context switching.

−−clock−ops N
stop clock stress workers after N bogo operations.

−−clone N
start N workers that create clones (via the clone(2) and clone3() system calls). This will rapidly try to create a default of 8192 clones that immediately die and wait in a zombie state until they are reaped. Once the maximum number of clones is reached (or clone fails because one has reached the maximum allowed) the oldest clone thread is reaped and a new clone is then created in a first-in first-out manner, and then repeated. A random clone flag is selected for each clone to try to exercise different clone operations. The clone stressor is a Linux only option.

−−clone−ops N
stop clone stress workers after N bogo clone operations.

−−clone−max N
try to create as many as N clone threads. This may not be reached if the system limit is less than N.

−−close N
start N workers that try to force race conditions on closing opened file descriptors. These file descriptors have been opened in various ways to try and exercise different kernel close handlers.

−−close−ops N
stop close workers after N bogo close operations.

−−context N
start N workers that run three threads that use swapcontext(3) to implement the thread-to-thread context switching. This exercises rapid process context saving and restoring and is bandwidth limited by register and memory save and restore rates.

−−context−ops N
stop context workers after N bogo context switches. In this stressor, 1 bogo op is equivalent to 1000 swapcontext calls.

−−copy−file N
start N stressors that copy a file using the Linux copy_file_range(2) system call. 2MB chunks of data are copied from random locations from one file to random locations to a destination file. By default, the files are 256 MB in size. Data is sync'd to the filesystem after each copy_file_range(2) call.

−−copy−file−ops N
stop after N copy_file_range() calls.

−−copy−file−bytes N
copy file size, the default is 256 MB. One can specify the size as % of free space on the file system or in units of Bytes, KBytes, MBytes and GBytes using the suffix b, k, m or g.

−c N, −−cpu N
start N workers exercising the CPU by sequentially working through all the different CPU stress methods. Instead of exercising all the CPU stress methods, one can specify a specific CPU stress method with the −−cpu−method option.

−−cpu−ops N
stop cpu stress workers after N bogo operations.

−l P, −−cpu−load P
load CPU with P percent loading for the CPU stress workers. 0 is effectively a sleep (no load) and 100 is full loading. The loading loop is broken into compute time (load%) and sleep time (100% - load%). Accuracy depends on the overall load of the processor and the responsiveness of the scheduler, so the actual load may be different from the desired load. Note that the number of bogo CPU operations may not be linearly scaled with the load as some systems employ CPU frequency scaling and so heavier loads produce an increased CPU frequency and greater CPU bogo operations.

Note: This option only applies to the −−cpu stressor option and not to all of the cpu class of stressors.

−−cpu−load−slice S
note − this option is only useful when −−cpu−load is less than 100%. The CPU load is broken into multiple busy and idle cycles. Use this option to specify the duration of a busy time slice. A negative value for S specifies the number of iterations to run before idling the CPU (e.g. -30 invokes 30 iterations of a CPU stress loop). A zero value selects a random busy time between 0 and 0.5 seconds. A positive value for S specifies the number of milliseconds to run before idling the CPU (e.g. 100 keeps the CPU busy for 0.1 seconds). Specifying small values for S lends to small time slices and smoother scheduling. Setting −−cpu−load as a relatively low value and −−cpu−load−slice to be large will cycle the CPU between long idle and busy cycles and exercise different CPU frequencies. The thermal range of the CPU is also cycled, so this is a good mechanism to exercise the scheduler, frequency scaling and passive/active thermal cooling mechanisms.

Note: This option only applies to the −−cpu stressor option and not to all of the cpu class of stressors.

−−cpu−method method
specify a cpu stress method. By default, all the stress methods are exercised sequentially, however one can specify just one method to be used if required. Available cpu stress methods are described as follows:

Method	**Description**
all	iterate over all the below cpu stress methods

ackermann	Ackermann function: compute A(3, 9), where:
	A(m, n) = n + 1 if m = 0;
	A(m - 1, 1) if m > 0 and n = 0;
	A(m - 1, A(m, n - 1)) if m > 0 and n > 0

apery	calculate Apery's constant $\zeta(3)$; the sum of 1/(n ↑ 3) for to a precision of 1.0x10↑14
bitops	various bit operations from bithack, namely: reverse bits, parity check, bit count, round to nearest power of 2
callfunc	recursively call 8 argument C function to a depth of 1024 calls and unwind
cfloat	1000 iterations of a mix of floating point complex operations
cdouble	1000 iterations of a mix of double floating point complex operations
clongdouble	1000 iterations of a mix of long double floating point complex operations
collatz	compute the 1348 steps in the collatz sequence from starting number 989345275647. Where f(n) = n / 2 (for even n) and f(n) = 3n + 1 (for odd n).
correlate	perform a 8192 × 512 correlation of random doubles
cpuid	fetch cpu specific information using the cpuid instruction (x86 only)
crc16	compute 1024 rounds of CCITT CRC16 on random data
decimal32	1000 iterations of a mix of 32 bit decimal floating point operations (GCC only)

decimal64	1000 iterations of a mix of 64 bit decimal floating point operations (GCC only)
decimal128	1000 iterations of a mix of 128 bit decimal floating point operations (GCC only)
dither	Floyd–Steinberg dithering of a 1024 × 768 random image from 8 bits down to 1 bit of depth
div64	50,000 64 bit unsigned integer divisions
djb2a	128 rounds of hash DJB2a (Dan Bernstein hash using the xor variant) on 128 to 1 bytes of random strings
double	1000 iterations of a mix of double precision floating point operations
euler	compute e using n = (1 + (1 ÷ n)) ↑ n
explog	iterate on n = exp(log(n) ÷ 1.00002)
factorial	find factorials from 1..150 using Stirling's and Ramanujan's approximations
fibonacci	compute Fibonacci sequence of 0, 1, 1, 2, 5, 8...
fft	4096 sample Fast Fourier Transform
float	1000 iterations of a mix of floating point operations
float16	1000 iterations of a mix of 16 bit floating point operations
float32	1000 iterations of a mix of 32 bit floating point operations
float80	1000 iterations of a mix of 80 bit floating point operations
float128	1000 iterations of a mix of 128 bit floating point operations
floatconversion	perform 65536 iterations of floating point conversions between float, double and long double floating point variables.
fnv1a	128 rounds of hash FNV-1a (Fowler–Noll–Vo hash using the xor then multiply variant) on 128 to 1 bytes of random strings
gamma	calculate the Euler–Mascheroni constant γ using the limiting difference between the harmonic series (1 + 1/2 + 1/3 + 1/4 + 1/5 ... + 1/n) and the natural logarithm ln(n), for n = 80000.
gcd	compute GCD of integers
gray	calculate binary to gray code and gray code back to binary for integers from 0 to 65535

hamming	compute Hamming H(8,4) codes on 262144 lots of 4 bit data. This turns 4 bit data into 8 bit Hamming code containing 4 parity bits. For data bits d1..d4, parity bits are computed as:
	p1 = d2 + d3 + d4
	p2 = d1 + d3 + d4
	p3 = d1 + d2 + d4
	p4 = d1 + d2 + d3

hanoi	solve a 21 disc Towers of Hanoi stack using the recursive solution
hyperbolic	compute $\sinh(\theta) \times \cosh(\theta) + \sinh(2\theta) + \cosh(3\theta)$ for float, double and long double hyperbolic sine and cosine functions where $\theta = 0$ to 2π in 1500 steps
idct	8 × 8 IDCT (Inverse Discrete Cosine Transform)
int8	1000 iterations of a mix of 8 bit integer operations
int16	1000 iterations of a mix of 16 bit integer operations
int32	1000 iterations of a mix of 32 bit integer operations
int64	1000 iterations of a mix of 64 bit integer operations
int128	1000 iterations of a mix of 128 bit integer operations (GCC only)
int32float	1000 iterations of a mix of 32 bit integer and floating point operations
int32double	1000 iterations of a mix of 32 bit integer and double precision floating point operations
int32longdouble	1000 iterations of a mix of 32 bit integer and long double precision floating point operations
int64float	1000 iterations of a mix of 64 bit integer and floating point operations

int64double	1000 iterations of a mix of 64 bit integer and double precision floating point operations
int64longdouble	1000 iterations of a mix of 64 bit integer and long double precision floating point operations
int128float	1000 iterations of a mix of 128 bit integer and floating point operations (GCC only)
int128double	1000 iterations of a mix of 128 bit integer and double precision floating point operations (GCC only)
int128longdouble	1000 iterations of a mix of 128 bit integer and long double precision floating point operations (GCC only)
int128decimal32	1000 iterations of a mix of 128 bit integer and 32 bit decimal floating point operations (GCC only)
int128decimal64	1000 iterations of a mix of 128 bit integer and 64 bit decimal floating point operations (GCC only)
int128decimal128	1000 iterations of a mix of 128 bit integer and 128 bit decimal floating point operations (GCC only)
intconversion	perform 65536 iterations of integer conversions between int16, int32 and int64 variables.
jenkin	Jenkin's integer hash on 128 rounds of 128..1 bytes of random data
jmp	Simple unoptimised compare >, <, == and jmp branching
ln2	compute ln(2) based on series: 1 - 1/2 + 1/3 - 1/4 + 1/5 - 1/6 ...
longdouble	1000 iterations of a mix of long double precision floating point operations
loop	simple empty loop
matrixprod	matrix product of two 128 × 128 matrices of double floats. Testing on 64 bit x86 hardware shows that this is provides a good mix of memory, cache and floating point operations and is probably the best CPU method to use to make a CPU run hot.
nsqrt	compute sqrt() of long doubles using Newton-Raphson
omega	compute the omega constant defined by $\Omega e \uparrow \Omega = 1$ using efficient iteration of $\Omega n+1 = (1 + \Omega n) / (1 + e \uparrow \Omega n)$
parity	compute parity using various methods from the Standford Bit Twiddling Hacks. Methods employed are: the naïve way, the naïve way with the Brian Kernigan bit counting optimisation, the multiply way, the parallel way, and the lookup table ways (2 variations).
phi	compute the Golden Ratio ϕ using series
pi	compute π using the Srinivasa Ramanujan fast convergence algorithm
pjw	128 rounds of hash pjw function on 128 to 1 bytes of random strings
prime	find the first 10000 prime numbers using a slightly optimised brute force naïve trial division search
psi	compute ψ (the reciprocal Fibonacci constant) using the sum of the reciprocals of the Fibonacci numbers
queens	compute all the solutions of the classic 8 queens problem for board sizes 1..11
rand	16384 iterations of rand(), where rand is the MWC pseudo random number generator. The MWC random function concatenates two 16 bit multiply−with−carry generators: $x(n) = 36969 \times x(n - 1) + carry,$ $y(n) = 18000 \times y(n - 1) + carry \bmod 2 \uparrow 16$ and has period of around $2 \uparrow 60$
rand48	16384 iterations of drand48(3) and lrand48(3)
rgb	convert RGB to YUV and back to RGB (CCIR 601)

sdbm	128 rounds of hash sdbm (as used in the SDBM database and GNU awk) on 128 to 1 bytes of random strings
sieve	find the first 10000 prime numbers using the sieve of Eratosthenes
stats	calculate minimum, maximum, arithmetic mean, geometric mean, harmoninc mean and standard deviation on 250 randomly generated positive double precision value.
sqrt	compute sqrt(rand()), where rand is the MWC pseudo random number generator
trig	compute $\sin(\theta) \times \cos(\theta) + \sin(2\theta) + \cos(3\theta)$ for float, double and long double sine and cosine functions where $\theta = 0$ to 2π in 1500 steps
union	perform integer arithmetic on a mix of bit fields in a C union. This exercises how well the compiler and CPU can perform integer bit field loads and stores.
zeta	compute the Riemann Zeta function $\zeta(s)$ for s = 2.0..10.0

Note that some of these methods try to exercise the CPU with computations found in some real world use cases. However, the code has not been optimised on a per-architecture basis, so may be a sub-optimal compared to hand-optimised code used in some applications. They do try to represent the typical instruction mixes found in these use cases.

−−cpu−online N
start N workers that put randomly selected CPUs offline and online. This Linux only stressor requires root privilege to perform this action. By default the first CPU (CPU 0) is never offlined as this has been found to be problematic on some systems and can result in a shutdown.

−−cpu−online−all
The default is to never offline the first CPU. This option will offline and online all the CPUs include CPU 0. This may cause some systems to shutdown.

−−cpu−online−ops N
stop after offline/online operations.

−−crypt N
start N workers that encrypt a 16 character random password using crypt(3). The password is encrypted using MD5, SHA-256 and SHA-512 encryption methods.

−−crypt−ops N
stop after N bogo encryption operations.

−−cyclic N
start N workers that exercise the real time FIFO or Round Robin schedulers with cyclic nanosecond sleeps. Normally one would just use 1 worker instance with this stressor to get reliable statistics. This stressor measures the first 10 thousand latencies and calculates the mean, mode, minimum, maximum latencies along with various latency percentiles for the just the first cyclic stressor instance. One has to run this stressor with CAP_SYS_NICE capability to enable the real time scheduling policies. The FIFO scheduling policy is the default.

−−cyclic−ops N
stop after N sleeps.

−−cyclic−dist N
calculate and print a latency distribution with the interval of N nanoseconds. This is helpful to see where the latencies are clustering.

−−cyclic−method [clock_ns | itimer | poll | posix_ns | pselect | usleep]
specify the cyclic method to be used, the default is clock_ns. The available cyclic methods are as follows:

Method	Description

clock_ns	sleep for the specified time using the clock_nanosleep(2) high resolution nanosleep and the CLOCK_REALTIME real time clock.
itimer	wakeup a paused process with a CLOCK_REALTIME itimer signal.
poll	delay for the specified time using a poll delay loop that checks for time changes using clock_gettime(2) on the CLOCK_REALTIME clock.
posix_ns	sleep for the specified time using the POSIX nanosleep(2) high resolution nanosleep.
pselect	sleep for the specified time using pselect(2) with null file descriptors.
usleep	sleep to the nearest microsecond using usleep(2).

−−cyclic−policy [fifo | rr]
specify the desired real time scheduling policy, ff (first-in, first-out) or rr (round robin).

−−cyclic−prio P
specify the scheduling priority P. Range from 1 (lowest) to 100 (highest).

−−cyclic−sleep N
sleep for N nanoseconds per test cycle using clock_nanosleep(2) with the CLOCK_REALTIME timer. Range from 1 to 1000000000 nanoseconds.

−−daemon N
start N workers that each create a daemon that dies immediately after creating another daemon and so on. This effectively works through the process table with short lived processes that do not have a parent and are waited for by init. This puts pressure on init to do rapid child reaping. The daemon processes perform the usual mix of calls to turn into typical UNIX daemons, so this artificially mimics very heavy daemon system stress.

−−daemon−ops N
stop daemon workers after N daemons have been created.

−−dccp N
start N workers that send and receive data using the Datagram Congestion Control Protocol (DCCP) (RFC4340). This involves a pair of client/server processes performing rapid connect, send and receives and disconnects on the local host.

−−dccp−domain D
specify the domain to use, the default is ipv4. Currently ipv4 and ipv6 are supported.

−−dccp−port P
start DCCP at port P. For N dccp worker processes, ports P to P - 1 are used.

−−dccp−ops N
stop dccp stress workers after N bogo operations.

−−dccp−opts [send | sendmsg | sendmmsg]
by default, messages are sent using send(2). This option allows one to specify the sending method using send(2), sendmsg(2) or sendmmsg(2). Note that sendmmsg is only available for Linux systems that support this system call.

−D N, −−dentry N
start N workers that create and remove directory entries. This should create file system meta data activity. The directory entry names are suffixed by a gray-code encoded number to try to mix up the hashing of the namespace.

−−dentry−ops N
stop denty thrash workers after N bogo dentry operations.

−−dentry−order [forward | reverse | stride | random]
specify unlink order of dentries, can be one of forward, reverse, stride or random. By default, dentries are unlinked in random order. The forward order will unlink them from first to last, reverse order will unlink them from last to first, stride order will unlink them by stepping around order in a quasi-random pattern and random order will randomly select one of forward, reverse or stride

orders.

−−dentries N
create N dentries per dentry thrashing loop, default is 2048.

−−dev N
start N workers that exercise the /dev devices. Each worker runs 5 concurrent threads that perform open(2), fstat(2), lseek(2), poll(2), fcntl(2), mmap(2), munmap(2), fsync(2) and close(2) on each device. Note that watchdog devices are not exercised.

−−dev−ops N
stop dev workers after N bogo device exercising operations.

−−dev−file filename
specify the device file to exercise, for example, /dev/null. By default the stressor will work through all the device files it can fine, however, this option allows a single device file to be exercised.

−−dev−shm N
start N workers that fallocate large files in /dev/shm and then mmap these into memory and touch all the pages. This exercises pages being moved to/from the buffer cache. Linux only.

−−dev−shm−ops N
stop after N bogo allocation and mmap /dev/shm operations.

−−dir N
start N workers that create and remove directories using mkdir and rmdir.

−−dir−ops N
stop directory thrash workers after N bogo directory operations.

−−dir−dirs N
exercise dir on N directories. The default is 8192 directories, this allows 64 to 65536 directories to be used instead.

−−dirdeep N
start N workers that create a depth-first tree of directories to a maximum depth as limited by PATH_MAX or ENAMETOOLONG (which ever occurs first). By default, each level of the tree contains one directory, but this can be increased to a maximum of 10 sub-trees using the −−dirdeep−dir option. To stress inode creation, a symlink and a hardlink to a file at the root of the tree is created in each level.

−−dirdeep−ops N
stop directory depth workers after N bogo directory operations.

−−dirdeep−dirs N
create N directories at each tree level. The default is just 1 but can be increased to a maximum of 10 per level.

−−dirdeep−inodes N
consume up to N inodes per dirdeep stressor while creating directories and links. The value N can be the number of inodes or a percentage of the total available free inodes on the filesystem being used.

−−dnotify N
start N workers performing file system activities such as making/deleting files/directories, renaming files, etc. to stress exercise the various dnotify events (Linux only).

−−dnotify−ops N
stop inotify stress workers after N dnotify bogo operations.

−−dup N
start N workers that perform dup(2) and then close(2) operations on /dev/zero. The maximum opens at one time is system defined, so the test will run up to this maximum, or 65536 open file descriptors, which ever comes first.

−−dup−ops N
stop the dup stress workers after N bogo open operations.

−−dynlib N
start N workers that dynamically load and unload various shared libraries. This exercises memory mapping and dynamic code loading and symbol lookups. See dlopen(3) for more details of this mechanism.

−−dynlib−ops N
stop workers after N bogo load/unload cycles.

−−efivar N
start N works that exercise the Linux /sys/firmware/efi/vars interface by reading the EFI variables. This is a Linux only stress test for platforms that support the EFI vars interface and requires the CAP_SYS_ADMIN capability.

−−efivar-ops N
stop the efivar stressors after N EFI variable read operations.

−−enosys N
start N workers that exercise non-functional system call numbers. This calls a wide range of system call numbers to see if it can break a system where these are not wired up correctly. It also keeps track of system calls that exist (ones that don't return ENOSYS) so that it can focus on purely finding and exercising non-functional system calls. This stressor exercises system calls from 0 . .__NR_syscalls + 1024, random system calls within constrained in the ranges of 0 to 2^8, 2^16, 2^24, 2^32, 2^40, 2^48, 2^56 and 2^64 bits, high system call numbers and various other bit patterns to try to get wide good coverage. To keep the environment clean, each system call being tested runs in a child process with reduced capabilities.

−−enosys−ops N
stop after N bogo enosys system call attempts

−−env N
start N workers that creates numerous large environment variables to try to trigger out of memory conditions using setenv(3). If ENOMEM occurs then the environment is emptied and another memory filling retry occurs. The process is restarted if it is killed by the Out Of Memory (OOM) killer.

−−env−ops N
stop after N bogo setenv/unsetenv attempts.

−−epoll N
start N workers that perform various related socket stress activity using epoll_wait(2) to monitor and handle new connections. This involves client/server processes performing rapid connect, send/receives and disconnects on the local host. Using epoll allows a large number of connections to be efficiently handled, however, this can lead to the connection table filling up and blocking further socket connections, hence impacting on the epoll bogo op stats. For ipv4 and ipv6 domains, multiple servers are spawned on multiple ports. The epoll stressor is for Linux only.

−−epoll−domain D
specify the domain to use, the default is unix (aka local). Currently ipv4, ipv6 and unix are supported.

−−epoll−port P
start at socket port P. For N epoll worker processes, ports P to (P * 4) - 1 are used for ipv4, ipv6 domains and ports P to P - 1 are used for the unix domain.

−−epoll−ops N
stop epoll workers after N bogo operations.

−−eventfd N
start N parent and child worker processes that read and write 8 byte event messages between them via the eventfd mechanism (Linux only).

−−eventfd−ops N
stop eventfd workers after N bogo operations.

−−eventfd−nonblock N
enable EFD_NONBLOCK to allow non-blocking on the event file descriptor. This will cause reads and writes to return with EAGAIN rather the blocking and hence causing a high rate of polling I/O.

−−exec N
start N workers continually forking children that exec stress-ng and then exit almost immediately. If a system has pthread support then 1 in 4 of the exec's will be from inside a pthread to exercise exec'ing from inside a pthread context.

−−exec−ops N
stop exec stress workers after N bogo operations.

−−exec−max P
create P child processes that exec stress-ng and then wait for them to exit per iteration. The default is just 1; higher values will create many temporary zombie processes that are waiting to be reaped. One can potentially fill up the process table using high values for −−exec−max and −−exec.

−F N, −−fallocate N
start N workers continually fallocating (preallocating file space) and ftruncating (file truncating) temporary files. If the file is larger than the free space, fallocate will produce an ENOSPC error which is ignored by this stressor.

−−fallocate−bytes N
allocated file size, the default is 1 GB. One can specify the size as % of free space on the file system or in units of Bytes, KBytes, MBytes and GBytes using the suffix b, k, m or g.

−−fallocate−ops N
stop fallocate stress workers after N bogo fallocate operations.

−−fanotify N
start N workers performing file system activities such as creating, opening, writing, reading and unlinking files to exercise the fanotify event monitoring interface (Linux only). Each stressor runs a child process to generate file events and a parent process to read file events using fanotify. Has to be run with CAP_SYS_ADMIN capability.

−−fanotify-ops N
stop fanotify stress workers after N bogo fanotify events.

−−fault N
start N workers that generates minor and major page faults.

−−fault−ops N
stop the page fault workers after N bogo page fault operations.

−−fcntl N
start N workers that perform fcntl(2) calls with various commands. The exercised commands (if available) are: F_DUPFD, F_DUPFD_CLOEXEC, F_GETFD, F_SETFD, F_GETFL, F_SETFL, F_GETOWN, F_SETOWN, F_GETOWN_EX, F_SETOWN_EX, F_GETSIG, F_SETSIG, F_GETLK, F_SETLK, F_SETLKW, F_OFD_GETLK, F_OFD_SETLK and F_OFD_SETLKW.

−−fcntl−ops N
stop the fcntl workers after N bogo fcntl operations.

−−fiemap N
start N workers that each create a file with many randomly changing extents and has 4 child processes per worker that gather the extent information using the FS_IOC_FIEMAP ioctl(2).

−−fiemap−ops N
stop after N fiemap bogo operations.

−−fiemap−bytes N
specify the size of the fiemap'd file in bytes. One can specify the size as % of free space on the file system or in units of Bytes, KBytes, MBytes and GBytes using the suffix b, k, m or g. Larger files will contain more extents, causing more stress when gathering extent information.

−−fifo N
start N workers that exercise a named pipe by transmitting 64 bit integers.

−−fifo-ops N
stop fifo workers after N bogo pipe write operations.

−−fifo-readers N
for each worker, create N fifo reader workers that read the named pipe using simple blocking reads.

−−file−ioctl N
start N workers that exercise various file specific ioctl(2) calls. This will attempt to use the FIONBIO, FIOQSIZE, FIGETBSZ, FIOCLEX, FIONCLEX, FIONBIO, FIOASYNC, FIOQSIZE, FIFREEZE, FITHAW, FICLONE, FICLONERANGE, FIONREAD, FIONWRITE and FS_IOC_RESVSP ioctls if these are defined.

−−file−ioctl−ops N
stop file−ioctl workers after N file ioctl bogo operations.

−−filename N
start N workers that exercise file creation using various length filenames containing a range of allowed filename characters. This will try to see if it can exceed the file system allowed filename length was well as test various filename lengths between 1 and the maximum allowed by the file system.

−−filename-ops N
stop filename workers after N bogo filename tests.

−−filename-opts opt
use characters in the filename based on option 'opt'. Valid options are:

Option	**Description**
probe	default option, probe the file system for valid allowed characters in a file name and use these
posix	use characters as specified by The Open Group Base Specifications Issue 7, POSIX.1-2008, 3.278 Portable Filename Character Set
ext	use characters allowed by the ext2, ext3, ext4 file systems, namely any 8 bit character apart from NUL and /

−−flock N
start N workers locking on a single file.

−−flock−ops N
stop flock stress workers after N bogo flock operations.

−f N, −−fork N
start N workers continually forking children that immediately exit.

−−fork−ops N
stop fork stress workers after N bogo operations.

−−fork−max P
create P child processes and then wait for them to exit per iteration. The default is just 1; higher values will create many temporary zombie processes that are waiting to be reaped. One can potentially fill up the process table using high values for −−fork−max and −−fork.

−−fp−error N
start N workers that generate floating point exceptions. Computations are performed to force and check for the FE_DIVBYZERO, FE_INEXACT, FE_INVALID, FE_OVERFLOW and FE_UNDERFLOW exceptions. EDOM and ERANGE errors are also checked.

−−fp−error−ops N
stop after N bogo floating point exceptions.

−−fstat N
start N workers fstat'ing files in a directory (default is /dev).

−−fstat−ops N
stop fstat stress workers after N bogo fstat operations.

−−fstat−dir directory
specify the directory to fstat to override the default of /dev. All the files in the directory will be fstat'd repeatedly.

−−full N
start N workers that exercise /dev/full. This attempts to write to the device (which should always get error ENOSPC), to read from the device (which should always return a buffer of zeros) and to seek randomly on the device (which should always succeed). (Linux only).

−−full−ops N
stop the stress full workers after N bogo I/O operations.

−−funccall N
start N workers that call functions of 1 through to 9 arguments. By default functions with uint64_t arguments are called, however, this can be changed using the −−funccall−method option.

−−funccall−ops N
stop the funccall workers after N bogo function call operations. Each bogo operation is 1000 calls of functions of 1 through to 9 arguments of the chosen argument type.

−−funccall−method method
specify the method of funccall argument type to be used. The default is uint64_t but can be one of uint8 uint16 uint32 uint64 uint128 float double longdouble float80 float128 decimal32 decimal64 and decimal128. Note that some of these types are only available with specific architectures and compiler versions.

−−funcret N
start N workers that pass and return by value various small to large data types.

−−funcret−ops N
stop the funcret workers after N bogo function call operations.

−−funcret−method method
specify the method of funcret argument type to be used. The default is uint64_t but can be one of uint8 uint16 uint32 uint64 uint128 float double longdouble float80 float128 decimal32 decimal64 decimal128 uint8x32 uint8x128 uint64x128.

−−futex N
start N workers that rapidly exercise the futex system call. Each worker has two processes, a futex waiter and a futex waker. The waiter waits with a very small timeout to stress the timeout and rapid polled futex waiting. This is a Linux specific stress option.

−−futex−ops N
stop futex workers after N bogo successful futex wait operations.

−−get N
start N workers that call system calls that fetch data from the kernel, currently these are: getpid, getppid, getcwd, getgid, getegid, getuid, getgroups, getpgrp, getpgid, getpriority, getresgid, getresuid, getrlimit, prlimit, getrusage, getsid, gettid, getcpu, gettimeofday, uname, adjtimex, sysfs. Some of these system calls are OS specific.

−−get−ops N
stop get workers after N bogo get operations.

−−getdent N
start N workers that recursively read directories /proc, /dev/, /tmp, /sys and /run using getdents and getdents64 (Linux only).

−−getdent−ops N
stop getdent workers after N bogo getdent bogo operations.

−−getrandom N
start N workers that get 8192 random bytes from the /dev/urandom pool using the getrandom(2) system call (Linux) or getentropy(2) (OpenBSD).

−−getrandom−ops N
stop getrandom workers after N bogo get operations.

−−handle N
start N workers that exercise the name_to_handle_at(2) and open_by_handle_at(2) system calls. (Linux only).

−−handle−ops N
stop after N handle bogo operations.

−d N, −−hdd N
start N workers continually writing, reading and removing temporary files. The default mode is to stress test sequential writes and reads. With the −−aggressive option enabled without any −−hdd−opts options the hdd stressor will work through all the −−hdd−opt options one by one to cover a range of I/O options.

−−hdd−bytes N
write N bytes for each hdd process, the default is 1 GB. One can specify the size as % of free space on the file system or in units of Bytes, KBytes, MBytes and GBytes using the suffix b, k, m or g.

−−hdd−opts list
specify various stress test options as a comma separated list. Options are as follows:

Option	**Description**
direct	try to minimize cache effects of the I/O. File I/O writes are performed directly from user space buffers and synchronous transfer is also attempted. To guarantee synchronous I/O, also use the sync option.
dsync	ensure output has been transferred to underlying hardware and file metadata has been updated (using the O_DSYNC open flag). This is equivalent to each write(2) being followed by a call to fdatasync(2). See also the fdatasync option.
fadv−dontneed	advise kernel to expect the data will not be accessed in the near future.
fadv−noreuse	advise kernel to expect the data to be accessed only once.
fadv−normal	advise kernel there are no explicit access pattern for the data. This is the default advice assumption.
fadv−rnd	advise kernel to expect random access patterns for the data.
fadv−seq	advise kernel to expect sequential access patterns for the data.

fadv−willneed	advise kernel to expect the data to be accessed in the near future.
fsync	flush all modified in-core data after each write to the output device using an explicit fsync(2) call.
fdatasync	similar to fsync, but do not flush the modified metadata unless metadata is required for later data reads to be handled correctly. This uses an explicit fdatasync(2) call.
iovec	use readv/writev multiple buffer I/Os rather than read/write. Instead of 1 read/write operation, the buffer is broken into an iovec of 16 buffers.
noatime	do not update the file last access timestamp, this can reduce metadata writes.
sync	ensure output has been transferred to underlying hardware (using the O_SYNC open flag). This is equivalent to a each write(2) being followed by a call to fsync(2). See also the fsync option.
rd−rnd	read data randomly. By default, written data is not read back, however, this option will force it to be read back randomly.
rd−seq	read data sequentially. By default, written data is not read back, however, this option will force it to be read back sequentially.
syncfs	write all buffered modifications of file metadata and data on the filesystem that contains the hdd worker files.
utimes	force update of file timestamp which may increase metadata writes.
wr−rnd	write data randomly. The wr−seq option cannot be used at the same time.
wr−seq	write data sequentially. This is the default if no write modes are specified.

Note that some of these options are mutually exclusive, for example, there can be only one method of writing or reading. Also, fadvise flags may be mutually exclusive, for example fadv-willneed cannot be used with fadv-dontneed.

−−hdd−ops N
stop hdd stress workers after N bogo operations.

−−hdd−write−size N
specify size of each write in bytes. Size can be from 1 byte to 4MB.

−−heapsort N
start N workers that sort 32 bit integers using the BSD heapsort.

−−heapsort−ops N
stop heapsort stress workers after N bogo heapsorts.

−−heapsort−size N
specify number of 32 bit integers to sort, default is 262144 (256 × 1024).

−−hrtimers N
start N workers that exercise high resolution times at a high frequency. Each stressor starts 32 processes that run with random timer intervals of 0..499999 nanoseconds. Running this stressor with appropriate privilege will run these with the SCHED_RR policy.

−−hrtimers−ops N
stop hrtimers stressors after N timer event bogo operations

−−hsearch N
start N workers that search a 80% full hash table using hsearch(3). By default, there are 8192 elements inserted into the hash table. This is a useful method to exercise access of memory and processor cache.

−−hsearch−ops N
stop the hsearch workers after N bogo hsearch operations are completed.

−−hsearch−size N
specify the number of hash entries to be inserted into the hash table. Size can be from 1K to 4M.

−−icache N
start N workers that stress the instruction cache by forcing instruction cache reloads. This is achieved by modifying an instruction cache line, causing the processor to reload it when we call a function in inside it. Currently only verified and enabled for Intel x86 CPUs.

−−icache−ops N
stop the icache workers after N bogo icache operations are completed.

−−icmp−flood N
start N workers that flood localhost with randonly sized ICMP ping packets. This stressor requires the CAP_NET_RAW capbility.

−−icmp−flood−ops N
stop icmp flood workers after N ICMP ping packets have been sent.

−−idle−scan N
start N workers that scan the idle page bitmap across a range of physical pages. This sets and checks for idle pages via the idle page tracking interface /sys/kernel/mm/page_idle/bitmap. This is for Linux only.

−−idle−scan−ops N
stop after N bogo page scan operations. Currently one bogo page scan operation is equivalent to setting and checking 64 physical pages.

−−idle−page N
start N workers that walks through every page exercising the Linux /sys/kernel/mm/page_idle/bitmap interface. Requires CAP_SYS_RESOURCE capability.

−−idle−page−ops N
stop after N bogo idle page operations.

−−inode-flags N
start N workers that exercise inode flags using the FS_IOC_GETFLAGS and FS_IOC_SETFLAGS ioctl(2). This attempts to apply all the available inode flags onto a directory and file even if the underlying file system may not support these flags (errors are just ignored). Each worker runs 4 threads that exercise the flags on the same directory and file to try to force races. This is a Linux only stressor, see ioctl_iflags(2) for more details.

−−inode-flags-ops N
stop the inode-flags workers after N ioctl flag setting attempts.

−−inotify N
start N workers performing file system activities such as making/deleting files/directories, moving files, etc. to stress exercise the various inotify events (Linux only).

−−inotify−ops N
stop inotify stress workers after N inotify bogo operations.

−i N, −−io N
start N workers continuously calling sync(2) to commit buffer cache to disk. This can be used in conjunction with the −−hdd options.

−−io−ops N
stop io stress workers after N bogo operations.

−−iomix N
start N workers that perform a mix of sequential, random and memory mapped read/write operations as well as forced sync'ing and (if run as root) cache dropping. Multiple child processes are spawned to all share a single file and perform different I/O operations on the same file.

−−iomix−bytes N
write N bytes for each iomix worker process, the default is 1 GB. One can specify the size as % of free space on the file system or in units of Bytes, KBytes, MBytes and GBytes using the suffix b,

k, m or g.

−−iomix−ops N
stop iomix stress workers after N bogo iomix I/O operations.

−−ioport N
start N workers than perform bursts of 16 reads and 16 writes of ioport 0x80 (x86 Linux systems only). I/O performed on x86 platforms on port 0x80 will cause delays on the CPU performing the I/O.

−−ioport−ops N
stop the ioport stressors after N bogo I/O operations

−−ioport−opts [in | out | inout]
specify if port reads in, port read writes out or reads and writes are to be performed. The default is both in and out.

−−ioprio N
start N workers that exercise the ioprio_get(2) and ioprio_set(2) system calls (Linux only).

−−ioprio−ops N
stop after N io priority bogo operations.

−−io−uring N
start N workers that perform iovec write and read I/O operations using the Linux io-uring interface. On each bogo-loop 1024 × 512 byte writes and 1024 × reads are performed on a temporary file.

−−io−uring−ops
stop after N rounds of write and reads.

−−ipsec−mb N
start N workers that perform cryptographic processing using the highly optimized Intel Multi-Buffer Crypto for IPsec library. Depending on the features available, SSE, AVX, AVX and AVX512 CPU features will be used on data encrypted by SHA, DES, CMAC, CTR, HMAC MD5, HMAC SHA1 and HMAC SHA512 cryptographic routines. This is only available for x86-64 modern Intel CPUs.

−−ipsec−mb−ops N
stop after N rounds of processing of data using the cryptographic routines.

−−ipsec−mb−feature [sse | avx | avx2 | avx512]
Just use the specified processor CPU feature. By default, all the available features for the CPU are exercised.

−−itimer N
start N workers that exercise the system interval timers. This sets up an ITIMER_PROF itimer that generates a SIGPROF signal. The default frequency for the itimer is 1 MHz, however, the Linux kernel will set this to be no more that the jiffy setting, hence high frequency SIGPROF signals are not normally possible. A busy loop spins on getitimer(2) calls to consume CPU and hence decrement the itimer based on amount of time spent in CPU and system time.

−−itimer−ops N
stop itimer stress workers after N bogo itimer SIGPROF signals.

−−itimer−freq F
run itimer at F Hz; range from 1 to 1000000 Hz. Normally the highest frequency is limited by the number of jiffy ticks per second, so running above 1000 Hz is difficult to attain in practice.

−−itimer−rand
select an interval timer frequency based around the interval timer frequency +/- 12.5% random jitter. This tries to force more variability in the timer interval to make the scheduling less predictable.

−−judy N
start N workers that insert, search and delete 32 bit integers in a Judy array using a predictable yet sparse array index. By default, there are 131072 integers used in the Judy array. This is a useful method to exercise random access of memory and processor cache.

−−judy−ops N
stop the judy workers after N bogo judy operations are completed.

−−judy−size N
specify the size (number of 32 bit integers) in the Judy array to exercise. Size can be from 1K to 4M 32 bit integers.

−−kcmp N
start N workers that use kcmp(2) to compare parent and child processes to determine if they share kernel resources. Supported only for Linux and requires CAP_SYS_PTRACE capability.

−−kcmp−ops N
stop kcmp workers after N bogo kcmp operations.

−−key N
start N workers that create and manipulate keys using add_key(2) and ketctl(2). As many keys are created as the per user limit allows and then the following keyctl commands are exercised on each key: KEYCTL_SET_TIMEOUT, KEYCTL_DESCRIBE, KEYCTL_UPDATE, KEYCTL_READ, KEYCTL_CLEAR and KEYCTL_INVALIDATE.

−−key−ops N
stop key workers after N bogo key operations.

−−kill N
start N workers sending SIGUSR1 kill signals to a SIG_IGN signal handler. Most of the process time will end up in kernel space.

−−kill−ops N
stop kill workers after N bogo kill operations.

−−klog N
start N workers exercising the kernel syslog(2) system call. This will attempt to read the kernel log with various sized read buffers. Linux only.

−−klog−ops N
stop klog workers after N syslog operations.

−−lease N
start N workers locking, unlocking and breaking leases via the fcntl(2) F_SETLEASE operation. The parent processes continually lock and unlock a lease on a file while a user selectable number of child processes open the file with a non-blocking open to generate SIGIO lease breaking notifications to the parent. This stressor is only available if F_SETLEASE, F_WRLCK and F_UNLCK support is provided by fcntl(2).

−−lease−ops N
stop lease workers after N bogo operations.

−−lease−breakers N
start N lease breaker child processes per lease worker. Normally one child is plenty to force many SIGIO lease breaking notification signals to the parent, however, this option allows one to specify more child processes if required.

−−link N
start N workers creating and removing hardlinks.

−−link−ops N
stop link stress workers after N bogo operations.

--lockbus N
start N workers that rapidly lock and increment 64 bytes of randomly chosen memory from a 16MB mmap'd region (Intel x86 and ARM CPUs only). This will cause cacheline misses and stalling of CPUs.

--lockbus-ops N
stop lockbus workers after N bogo operations.

--locka N
start N workers that randomly lock and unlock regions of a file using the POSIX advisory locking mechanism (see fcntl(2), F_SETLK, F_GETLK). Each worker creates a 1024 KB file and attempts to hold a maximum of 1024 concurrent locks with a child process that also tries to hold 1024 concurrent locks. Old locks are unlocked in a first-in, first-out basis.

--locka-ops N
stop locka workers after N bogo locka operations.

--lockf N
start N workers that randomly lock and unlock regions of a file using the POSIX lockf(3) locking mechanism. Each worker creates a 64 KB file and attempts to hold a maximum of 1024 concurrent locks with a child process that also tries to hold 1024 concurrent locks. Old locks are unlocked in a first-in, first-out basis.

--lockf-ops N
stop lockf workers after N bogo lockf operations.

--lockf-nonblock
instead of using blocking F_LOCK lockf(3) commands, use non-blocking F_TLOCK commands and re-try if the lock failed. This creates extra system call overhead and CPU utilisation as the number of lockf workers increases and should increase locking contention.

--lockofd N
start N workers that randomly lock and unlock regions of a file using the Linux open file description locks (see fcntl(2), F_OFD_SETLK, F_OFD_GETLK). Each worker creates a 1024 KB file and attempts to hold a maximum of 1024 concurrent locks with a child process that also tries to hold 1024 concurrent locks. Old locks are unlocked in a first-in, first-out basis.

--lockofd-ops N
stop lockofd workers after N bogo lockofd operations.

--longjmp N
start N workers that exercise setjmp(3)/longjmp(3) by rapid looping on longjmp calls.

--longjmp-ops N
stop longjmp stress workers after N bogo longjmp operations (1 bogo op is 1000 longjmp calls).

--loop N
start N workers that exercise the loopback control device. This creates 2MB loopback devices, expands them to 4MB, performs some loopback status information get and set operations and then destoys them. Linux only and requires CAP_SYS_ADMIN capability.

--loop-ops N
stop after N bogo loopback creation/deletion operations.

--lsearch N
start N workers that linear search a unsorted array of 32 bit integers using lsearch(3). By default, there are 8192 elements in the array. This is a useful method to exercise sequential access of memory and processor cache.

--lsearch-ops N
stop the lsearch workers after N bogo lsearch operations are completed.

−−lsearch−size N
specify the size (number of 32 bit integers) in the array to lsearch. Size can be from 1K to 4M.

−−madvise N
start N workers that apply random madvise(2) advise settings on pages of a 4MB file backed shared memory mapping.

−−madvise−ops N
stop madvise stressors after N bogo madvise operations.

−−malloc N
start N workers continuously calling malloc(3), calloc(3), realloc(3) and free(3). By default, up to 65536 allocations can be active at any point, but this can be altered with the −−malloc−max option. Allocation, reallocation and freeing are chosen at random; 50% of the time memory is allocation (via malloc, calloc or realloc) and 50% of the time allocations are free'd. Allocation sizes are also random, with the maximum allocation size controlled by the −−malloc−bytes option, the default size being 64K. The worker is re-started if it is killed by the out of memory (OOM) killer.

−−malloc−bytes N
maximum per allocation/reallocation size. Allocations are randomly selected from 1 to N bytes. One can specify the size as % of total available memory or in units of Bytes, KBytes, MBytes and GBytes using the suffix b, k, m or g. Large allocation sizes cause the memory allocator to use mmap(2) rather than expanding the heap using brk(2).

−−malloc−max N
maximum number of active allocations allowed. Allocations are chosen at random and placed in an allocation slot. Because about 50%/50% split between allocation and freeing, typically half of the allocation slots are in use at any one time.

−−malloc−ops N
stop after N malloc bogo operations. One bogo operations relates to a successful malloc(3), calloc(3) or realloc(3).

−−malloc−thresh N
specify the threshold where malloc uses mmap(2) instead of sbrk(2) to allocate more memory. This is only available on systems that provide the GNU C mallopt(3) tuning function.

−−matrix N
start N workers that perform various matrix operations on floating point values. Testing on 64 bit x86 hardware shows that this provides a good mix of memory, cache and floating point operations and is an excellent way to make a CPU run hot.

By default, this will exercise all the matrix stress methods one by one. One can specify a specific matrix stress method with the −−matrix−method option.

−−matrix−ops N
stop matrix stress workers after N bogo operations.

−−matrix−method method
specify a matrix stress method. Available matrix stress methods are described as follows:

Method	**Description**
all	iterate over all the below matrix stress methods
add	add two N × N matrices
copy	copy one N × N matrix to another
div	divide an N × N matrix by a scalar
frobenius	Frobenius product of two N × N matrices
hadamard	Hadamard product of two N × N matrices
identity	create an N × N identity matrix
mean	arithmetic mean of two N × N matrices

mult	multiply an N × N matrix by a scalar
negate	negate an N × N matrix
prod	product of two N × N matrices
sub	subtract one N × N matrix from another N × N matrix
square	multiply an N × N matrix by itself
trans	transpose an N × N matrix
zero	zero an N × N matrix

−−matrix−size N
specify the N × N size of the matrices. Smaller values result in a floating point compute throughput bound stressor, where as large values result in a cache and/or memory bandwidth bound stressor.

−−matrix−yx
perform matrix operations in order y by x rather than the default x by y. This is suboptimal ordering compared to the default and will perform more data cache stalls.

−−matrix-3d N
start N workers that perform various 3D matrix operations on floating point values. Testing on 64 bit x86 hardware shows that this provides a good mix of memory, cache and floating point operations and is an excellent way to make a CPU run hot.

By default, this will exercise all the 3D matrix stress methods one by one. One can specify a specific 3D matrix stress method with the −−matrix−3d−method option.

−−matrix−3d−ops N
stop the 3D matrix stress workers after N bogo operations.

−−matrix−3d−method method
specify a 3D matrix stress method. Available 3D matrix stress methods are described as follows:

Method	**Description**
all	iterate over all the below matrix stress methods
add	add two N × N × N matrices
copy	copy one N × N × N matrix to another
div	divide an N × N × N matrix by a scalar
frobenius	Frobenius product of two N × N × N matrices
hadamard	Hadamard product of two N × N × N matrices
identity	create an N × N × N identity matrix
mean	arithmetic mean of two N × N × N matrices
mult	multiply an N × N × N matrix by a scalar
negate	negate an N × N × N matrix
sub	subtract one N × N × N matrix from another N × N × N matrix
trans	transpose an N × N × N matrix
zero	zero an N × N × N matrix

−−matrix−3d−size N
specify the N × N × N size of the matrices. Smaller values result in a floating point compute throughput bound stressor, where as large values result in a cache and/or memory bandwidth bound stressor.

−−matrix−3d−zyx
perform matrix operations in order z by y by x rather than the default x by y by z. This is suboptimal ordering compared to the default and will perform more data cache stalls.

−−mcontend N
start N workers that produce memory contention read/write patterns. Each stressor runs with 5 threads that read and write to two different mappings of the same underlying physical page. Various caching operations are also exercised to cause sub-optimal memory access patterns. The threads also randomly change CPU affinity to exercise CPU and memory migration stress.

−−mcontend−ops N
stop mcontend stressors after N bogo read/write operations.

−−membarrier N
start N workers that exercise the membarrier system call (Linux only).

−−membarrier−ops N
stop membarrier stress workers after N bogo membarrier operations.

−−memcpy N
start N workers that copy 2MB of data from a shared region to a buffer using memcpy(3) and then move the data in the buffer with memmove(3) with 3 different alignments. This will exercise processor cache and system memory.

−−memcpy−ops N
stop memcpy stress workers after N bogo memcpy operations.

−−memcpy−method [all | libc | builtin | naive]
specify a memcpy copying method. Available memcpy methods are described as follows:

Method	**Description**
all	use libc, builtin and naive methods
libc	use libc memcpy and memmove functions, this is the default
builtin	use the compiler built in optimized memcpy and memmove functions
naive	use unoptimized naive byte by byte copying and memory moving

−−memfd N
start N workers that create allocations of 1024 pages using memfd_create(2) and ftruncate(2) for allocation and mmap(2) to map the allocation into the process address space. (Linux only).

−−memfd−bytes N
allocate N bytes per memfd stress worker, the default is 256MB. One can specify the size in as % of total available memory or in units of Bytes, KBytes, MBytes and GBytes using the suffix b, k, m or g.

−−memfd−fds N
create N memfd file descriptors, the default is 256. One can select 8 to 4096 memfd file descriptions with this option.

−−memfd−ops N
stop after N memfd-create(2) bogo operations.

−−memhotplug N
start N workers that offline and online memory hotplug regions. Linux only and requires CAP_SYS_ADMIN capabilities.

−−memhotplug−ops N
stop memhotplug stressors after N memory offline and online bogo operations.

−−memrate N
start N workers that exercise a buffer with 64, 32, 16 and 8 bit reads and writes. This memory stressor allows one to also specify the maximum read and write rates. The stressors will run at maximum speed if no read or write rates are specified.

−−memrate−ops N
stop after N bogo memrate operations.

−−memrate−bytes N
specify the size of the memory buffer being exercised. The default size is 256MB. One can specify the size in units of Bytes, KBytes, MBytes and GBytes using the suffix b, k, m or g.

−−memrate−rd−mbs N
specify the maximum allowed read rate in MB/sec. The actual read rate is dependent on scheduling jitter and memory accesses from other running processes.

−−memrate−wr−mbs N
specify the maximum allowed read rate in MB/sec. The actual write rate is dependent on scheduling jitter and memory accesses from other running processes.

−−memthrash N
start N workers that thrash and exercise a 16MB buffer in various ways to try and trip thermal overrun. Each stressor will start 1 or more threads. The number of threads is chosen so that there will be at least 1 thread per CPU. Note that the optimal choice for N is a value that divides into the number of CPUs.

−−memthrash-ops N
stop after N memthrash bogo operations.

−−memthrash−method method
specify a memthrash stress method. Available memthrash stress methods are described as follows:

Method	**Description**
all	iterate over all the below memthrash methods
chunk1	memset 1 byte chunks of random data into random locations
chunk8	memset 8 byte chunks of random data into random locations
chunk64	memset 64 byte chunks of random data into random locations
chunk256	memset 256 byte chunks of random data into random locations
chunkpage	memset page size chunks of random data into random locations
flip	flip (invert) all bits in random locations
flush	flush cache line in random locations
lock	lock randomly choosing locations (Intel x86 and ARM CPUs only)
matrix	treat memory as a 2×2 matrix and swap random elements
memmove	copy all the data in buffer to the next memory location
memset	memset the memory with random data
mfence	stores with write serialization
prefetch	prefetch data at random memory locations
random	randomly run any of the memthrash methods except for ’random’ and ’all’
spinread	spin loop read the same random location 2^19 times
spinwrite	spin loop write the same random location 2^19 times
swap	step through memory swapping bytes in steps of 65 and 129 byte strides

--mergesort N
start N workers that sort 32 bit integers using the BSD mergesort.

−−mergesort−ops N
stop mergesort stress workers after N bogo mergesorts.

−−mergesort−size N
specify number of 32 bit integers to sort, default is 262144 (256×1024).

−−mincore N
start N workers that walk through all of memory 1 page at a time checking if the page mapped and also is resident in memory using mincore(2). It also maps and unmaps a page to check if the page is mapped or not using mincore(2).

−−mincore−ops N
stop after N mincore bogo operations. One mincore bogo op is equivalent to a 300 mincore(2) calls. **−−mincore−random** instead of walking through pages sequentially, select pages at random. The chosen address is iterated over by shifting it right one place and checked by mincore until the address is less or equal to the page size.

−−mknod N
start N workers that create and remove fifos, empty files and named sockets using mknod and unlink.

−−mknod−ops N
stop directory thrash workers after N bogo mknod operations.

−−mlock N
start N workers that lock and unlock memory mapped pages using mlock(2), munlock(2), mlockall(2) and munlockall(2). This is achieved by the mapping of three contiguous pages and then locking the second page, hence ensuring non-contiguous pages are locked . This is then repeated until the maximum allowed mlocks or a maximum of 262144 mappings are made. Next, all future mappings are mlocked and the worker attempts to map 262144 pages, then all pages are munlocked and the pages are unmapped.

−−mlock−ops N
stop after N mlock bogo operations.

−−mlockmany N
start N workers that fork off up to 1024 child processes in total; each child will attempt to anonymously mmap and mlock the maximum allowed mlockable memory size. The stress test attempts to avoid swapping by tracking low memory and swap allocations (but some swapping may occur). Once either the maximum number of child process is reached or all mlockable in-core memory is locked then child processes are killed and the stress test is repeated.

−−mlockmany−ops N
stop after N mlockmany (mmap and mlock) operations.

−−mmap N
start N workers continuously calling mmap(2)/munmap(2). The initial mapping is a large chunk (size specified by −−mmap−bytes) followed by pseudo-random 4K unmappings, then pseudo-random 4K mappings, and then linear 4K unmappings. Note that this can cause systems to trip the kernel OOM killer on Linux systems if not enough physical memory and swap is not available. The MAP_POPULATE option is used to populate pages into memory on systems that support this. By default, anonymous mappings are used, however, the −−mmap−file and −−mmap−async options allow one to perform file based mappings if desired.

−−mmap−ops N
stop mmap stress workers after N bogo operations.

−−mmap−async
enable file based memory mapping and use asynchronous msync'ing on each page, see −−mmap−file.

−−mmap−bytes N
allocate N bytes per mmap stress worker, the default is 256MB. One can specify the size as % of total available memory or in units of Bytes, KBytes, MBytes and GBytes using the suffix b, k, m or g.

−−mmap−file
enable file based memory mapping and by default use synchronous msync'ing on each page.

−−mmap−mmap2
use mmap2 for 4K page aligned offsets if mmap2 is available, otherwise fall back to mmap.

−−mmap−mprotect
change protection settings on each page of memory. Each time a page or a group of pages are mapped or remapped then this option will make the pages read-only, write-only, exec-only, and read-write.

−−mmap−odirect
enable file based memory mapping and use O_DIRECT direct I/O.

−−mmap−osync
enable file based memory mapping and used O_SYNC synchronous I/O integrity completion.

−−mmapaddr N
start N workers that memory map pages at a random memory location that is not already mapped. On 64 bit machines the random address is randomly chosen 32 bit or 64 bit address. If the mapping works a second page is memory mapped from the first mapped address. The stressor exercises mmap/munmap, mincore and segfault handling.

−−mmapaddr−ops N
stop after N random address mmap bogo operations.

−−mmapfork N
start N workers that each fork off 32 child processes, each of which tries to allocate some of the free memory left in the system (and trying to avoid any swapping). The child processes then hint that the allocation will be needed with madvise(2) and then memset it to zero and hint that it is no longer needed with madvise before exiting. This produces significant amounts of VM activity, a lot of cache misses and with minimal swapping.

−−mmapfork−ops N
stop after N mmapfork bogo operations.

−−mmapfixed N
start N workers that perform fixed address allocations from the top virtual address down to 128K. The allocated sizes are from 1 page to 8 pages and various random mmap flags are used MAP_SHARED/MAP_PRIVATE, MAP_LOCKED, MAP_NORESERVE, MAP_POPULATE. If successfully map'd then the allocation is remap'd to an address that is several pages higher in memory. Mappings and remappings are madvised with random madvise options to further exercise the mappings.

−−mmapfixed−ops N
stop after N mmapfixed memory mapping bogo operations.

−−mmapmany N
start N workers that attempt to create the maximum allowed per-process memory mappings. This is achieved by mapping 3 contiguous pages and then unmapping the middle page hence splitting the mapping into two. This is then repeated until the maximum allowed mappings or a maximum of 262144 mappings are made.

−−mmapmany−ops N
stop after N mmapmany bogo operations

−−mq N
start N sender and receiver processes that continually send and receive messages using POSIX message queues. (Linux only).

−−mq−ops N
stop after N bogo POSIX message send operations completed.

−−mq−size N
specify size of POSIX message queue. The default size is 10 messages and most Linux systems this is the maximum allowed size for normal users. If the given size is greater than the allowed message queue size then a warning is issued and the maximum allowed size is used instead.

−−mremap N
start N workers continuously calling mmap(2), mremap(2) and munmap(2). The initial anonymous mapping is a large chunk (size specified by −−mremap−bytes) and then iteratively halved in size by remapping all the way down to a page size and then back up to the original size. This worker is only available for Linux.

−−mremap−ops N
stop mremap stress workers after N bogo operations.

−−mremap−bytes N
initially allocate N bytes per remap stress worker, the default is 256MB. One can specify the size in units of Bytes, KBytes, MBytes and GBytes using the suffix b, k, m or g.

−−mremap−mlock
attempt to mlock remapped pages into memory prohibiting them from being paged out. This is a no-op if mlock(2) is not available.

−−msg N
start N sender and receiver processes that continually send and receive messages using System V message IPC.

−−msg−ops N
stop after N bogo message send operations completed.

−−msg−types N
select the quality of message types (mtype) to use. By default, msgsnd sends messages with a mtype of 1, this option allows one to send messages types in the range 1..N to exercise the message queue receive ordering. This will also impact throughput performance.

−−msync N
start N stressors that msync data from a file backed memory mapping from memory back to the file and msync modified data from the file back to the mapped memory. This exercises the msync(2) MS_SYNC and MS_INVALIDATE sync operations.

−−msync−ops N
stop after N msync bogo operations completed.

−−msync−bytes N
allocate N bytes for the memory mapped file, the default is 256MB. One can specify the size as % of total available memory or in units of Bytes, KBytes, MBytes and GBytes using the suffix b, k, m or g.

−−nanosleep N
start N workers that each run 256 pthreads that call nanosleep with random delays from 1 to 2^18 nanoseconds. This should exercise the high resolution timers and scheduler.

−−nanosleep−ops N
stop the nanosleep stressor after N bobo nanosleep operations.

−−netdev N
start N workers that exercise various netdevice ioctl commands across all the available network devices. The ioctls exercised by this stressor are as follows: SIOCGIFCONF, SIOCGIFINDEX, SIOCGIFNAME, SIOCGIFFLAGS, SIOCGIFADDR, SIOCGIFNETMASK, SIOCGIFMETRIC, SIOCGIFMTU, SIOCGIFHWADDR, SIOCGIFMAP and SIOCGIFTXQLEN. See netdevice(7) for more details of these ioctl commands.

−−netdev−ops N
stop after N netdev bogo operations completed.

−−netlink−proc N
start N workers that spawn child processes and monitor fork/exec/exit process events via the proc netlink connector. Each event received is counted as a bogo op. This stressor can only be run on Linux and requires CAP_NET_ADMIN capability.

−−netlink−proc−ops N
stop the proc netlink connector stressors after N bogo ops.

−−netlink−task N
start N workers that collect task statistics via the netlink taskstats interface. This stressor can only be run on Linux and requires CAP_NET_ADMIN capability.

−−netlink−task−ops N
stop the taskstats netlink connector stressors after N bogo ops.

−−nice N
start N cpu consuming workers that exercise the available nice levels. Each iteration forks off a child process that runs through the all the nice levels running a busy loop for 0.1 seconds per level and then exits.

−−nice−ops N
stop after N nice bogo nice loops

−−nop N
start N workers that consume cpu cycles issuing no-op instructions. This stressor is available if the assembler supports the "nop" instruction.

−−nop−ops N
stop nop workers after N no-op bogo operations. Each bogo-operation is equivalent to 256 loops of 256 no-op instructions.

−−null N
start N workers writing to /dev/null.

−−null−ops N
stop null stress workers after N /dev/null bogo write operations.

−−numa N
start N workers that migrate stressors and a 4MB memory mapped buffer around all the available NUMA nodes. This uses migrate_pages(2) to move the stressors and mbind(2) and move_pages(2) to move the pages of the mapped buffer. After each move, the buffer is written to force activity over the bus which results cache misses. This test will only run on hardware with NUMA enabled and more than 1 NUMA node.

−−numa−ops N
stop NUMA stress workers after N bogo NUMA operations.

−−oom−pipe N
start N workers that create as many pipes as allowed and exercise expanding and shrinking the pipes from the largest pipe size down to a page size. Data is written into the pipes and read out again to fill the pipe buffers. With the −−aggressive mode enabled the data is not read out when the pipes are shrunk, causing the kernel to OOM processes aggressively. Running many instances of this stressor will force kernel to OOM processes due to the many large pipe buffer allocations.

−−oom−pipe−ops N
stop after N bogo pipe expand/shrink operations.

−−opcode N
start N workers that fork off children that execute randomly generated executable code. This will generate issues such as illegal instructions, bus errors, segmentation faults, traps, floating point errors that are handled gracefully by the stressor.

−−opcode−ops N
stop after N attempts to execute illegal code.

−−opcode−method [inc | mixed | random | text]
select the opcode generation method. By default, random bytes are used to generate the executable code. This option allows one to select one of the three methods:

Method	**Description**
inc	use incrementing 32 bit opcode patterns from 0x00000000 to 0xfffffff inclusive.

mixed	use a mix of incrementing 32 bit opcode patterns and random 32 bit opcode patterns that are also inverted, encoded with gray encoding and bit reversed.
random	generate opcodes using random bytes from a mwc random generator.
text	copies random chunks of code from the stress-ng text segment and randomly flips single bits in a random choice of 1/8th of the code.

−o N, −−open N
start N workers that perform open(2) and then close(2) operations on /dev/zero. The maximum opens at one time is system defined, so the test will run up to this maximum, or 65536 open file descriptors, which ever comes first.

−−open−ops N
stop the open stress workers after N bogo open operations.

−−open−fd
run a child process that scans /proc/$PID/fd and attempts to open the files that the stressor has opened. This exercises racing open/close operations on the proc interface.

−−personality N
start N workers that attempt to set personality and get all the available personality types (process execution domain types) via the personality(2) system call. (Linux only).

−−personality−ops N
stop personality stress workers after N bogo personality operations.

−−physpage N
start N workers that use /proc/self/pagemap and /proc/kpagecount to determine the physical page and page count of a virtual mapped page and a page that is shared among all the stressors. Linux only and requires the CAP_SYS_ADMIN capabilities.

−−physpage−ops N
stop physpage stress workers after N bogo physical address lookups.

−−pidfd N
start N workers that exercise signal sending via the pidfd_send_signal system call. This stressor creates child processes and checks if they exist and can be stopped, restarted and killed using the pidfd_send_signal system call.

−−pidfd−ops N
stop pidfd stress workers after N child processes have been created, tested and killed with pidfd_send_signal.

−p N, −−pipe N
start N workers that perform large pipe writes and reads to exercise pipe I/O. This exercises memory write and reads as well as context switching. Each worker has two processes, a reader and a writer.

−−pipe−ops N
stop pipe stress workers after N bogo pipe write operations.

−−pipe−data−size N
specifies the size in bytes of each write to the pipe (range from 4 bytes to 4096 bytes). Setting a small data size will cause more writes to be buffered in the pipe, hence reducing the context switch rate between the pipe writer and pipe reader processes. Default size is the page size.

−−pipe−size N
specifies the size of the pipe in bytes (for systems that support the F_SETPIPE_SZ fcntl() command). Setting a small pipe size will cause the pipe to fill and block more frequently, hence increasing the context switch rate between the pipe writer and the pipe reader processes. Default size is 512 bytes.

−−pipeherd N
start N workers that pass a 64 bit token counter to/from 100 child processes over a shared pipe. This forces a high context switch rate and can trigger a "thundering herd" of wakeups on processes that are blocked on pipe waits.

−−pipeherd−ops N
stop pipe stress workers after N bogo pipe write operations.

−−pipeherd−yield
force a scheduling yield after each write, this increases the context switch rate.

−−pkey N
start N workers that change memory protection using a protection key (pkey) and the pkey_mprotect call (Linux only). This will try to allocate a pkey and use this for the page protection, however, if this fails then the special pkey -1 will be used (and the kernel will use the normal mprotect mechanism instead). Various page protection mixes of read/write/exec/none will be cycled through on randomly chosen pre-allocated pages.

−−pkey−ops N
stop after N pkey_mprotect page protection cycles.

−P N, −−poll N
start N workers that perform zero timeout polling via the poll(2), select(2) and sleep(3) calls. This wastes system and user time doing nothing.

−−poll−ops N
stop poll stress workers after N bogo poll operations.

−−prctl N
start N workers that exercise the majority of the prctl(2) system call options. Each batch of prctl calls is performed inside a new child process to ensure the limit of prctl is contained inside a new process every time. Some prctl options are architecture specific, however, this stressor will exercise these even if they are not implemented.

−−prctl−ops N
stop prctl workers after N batches of prctl calls

−−procfs N
start N workers that read files from /proc and recursively read files from /proc/self (Linux only).

−−procfs−ops N
stop procfs reading after N bogo read operations. Note, since the number of entries may vary between kernels, this bogo ops metric is probably very misleading.

−−pthread N
start N workers that iteratively creates and terminates multiple pthreads (the default is 1024 pthreads per worker). In each iteration, each newly created pthread waits until the worker has created all the pthreads and then they all terminate together.

−−pthread−ops N
stop pthread workers after N bogo pthread create operations.

−−pthread−max N
create N pthreads per worker. If the product of the number of pthreads by the number of workers is greater than the soft limit of allowed pthreads then the maximum is re-adjusted down to the maximum allowed.

−−ptrace N
start N workers that fork and trace system calls of a child process using ptrace(2).

−−ptrace−ops N
stop ptracer workers after N bogo system calls are traced.

−−pty N
start N workers that repeatedly attempt to open pseudoterminals and perform various pty ioctls upon the ptys before closing them.

−−pty−ops N
stop pty workers after N pty bogo operations.

−−pty−max N
try to open a maximum of N pseudoterminals, the default is 65536. The allowed range of this setting is 8..65536.

−Q, −−qsort N
start N workers that sort 32 bit integers using qsort.

−−qsort−ops N
stop qsort stress workers after N bogo qsorts.

−−qsort−size N
specify number of 32 bit integers to sort, default is 262144 (256 × 1024).

−−quota N
start N workers that exercise the Q_GETQUOTA, Q_GETFMT, Q_GETINFO, Q_GETSTATS and Q_SYNC quotactl(2) commands on all the available mounted block based file systems. Requires CAP_SYS_ADMIN capability to run.

−−quota−ops N
stop quota stress workers after N bogo quotactl operations.

−−radixsort N
start N workers that sort random 8 byte strings using radixsort.

−−radixsort−ops N
stop radixsort stress workers after N bogo radixsorts.

−−radixsort−size N
specify number of strings to sort, default is 262144 (256 × 1024).

−−ramfs N
start N workers mounting a memory based file system using ramfs and tmpfs (Linux only). This alternates between mounting and umounting a ramfs or tmpfs file system using the traditional mount(2) and umount(2) system call as well as the newer Linux 5.2 fsopen(2), fsmount(2), fsconfig(2) and move_mount(2) system calls if they are available. The default ram file system size is 2MB.

−−ramfs−ops N
stop after N ramfs mount operations.

−−ramfs−size N
set the ramfs size (must be multiples of the page size).

−−rawdev N
start N workers that read the underlying raw drive device using direct IO reads. The device (with minor number 0) that stores the current working directory is the raw device to be read by the stressor. The read size is exactly the size of the underlying device block size. By default, this stressor will exercise all the of the rawdev methods (see the −−rawdev−method option). This is a Linux only stressor and requires root privilege to be able to read the raw device.

−−rawdev−ops N
stop the rawdev stress workers after N raw device read bogo operations.

−−rawdev−method M
Available rawdev stress methods are described as follows:

Method	**Description**

all	iterate over all the rawdev stress methods as listed below:
sweep	repeatedly read across the raw device from the 0th block to the end block in steps of the number of blocks on the device / 128 and back to the start again.
wiggle	repeatedly read across the raw device in 128 evenly steps with each step reading 1024 blocks backwards from each step.
ends	repeatedly read the first and last 128 start and end blocks of the raw device alternating from start of the device to the end of the device.
random	repeatedly read 256 random blocks
burst	repeatedly read 256 sequential blocks starting from a random block on the raw device.

−−rawsock N
start N workers that send and receive packet data using raw sockets on the localhost. Requires CAP_NET_RAW to run.

−−rawsock-ops N
stop rawsock workers after N packets are received.

−−rawpkt N
start N workers that sends and receives ethernet packets using raw packets on the localhost via the loopback device. Requires CAP_NET_RAW to run.

−−rawpkt−ops N
stop rawpkt workers after N packets from the sender process are received.

−−rawpkt−port N
start at port P. For N rawpkt worker processes, ports P to (P * 4) - 1 are used. The default starting port is port 14000.

−−rawudp N
start N workers that send and receive UDP packets using raw sockets on the localhost. Requires CAP_NET_RAW to run.

−−rawudp−ops N
stop rawudp workers after N packets are received.

−−rawudp−port N
start at port P. For N rawudp worker processes, ports P to (P * 4) - 1 are used. The default starting port is port 13000.

−−rdrand N
start N workers that read a random number from an on-chip random number generator This uses the rdrand instruction on Intel processors or the darn instruction on Power9 processors.

−−rdrand−ops N
stop rdrand stress workers after N bogo rdrand operations (1 bogo op = 2048 random bits successfully read).

−−readahead N
start N workers that randomly seek and perform 4096 byte read/write I/O operations on a file with readahead. The default file size is 64 MB. Readaheads and reads are batched into 16 readaheads and then 16 reads.

−−readahead−bytes N
set the size of readahead file, the default is 1 GB. One can specify the size as % of free space on the file system or in units of Bytes, KBytes, MBytes and GBytes using the suffix b, k, m or g.

−−readahead−ops N
stop readahead stress workers after N bogo read operations.

−−reboot N
start N workers that exercise the reboot(2) system call. When possible, it will create a process in a PID namespace and perform a reboot power off command that should shutdown the process.

Also, the stressor exercises invalid reboot magic values and invalid reboots when there are insufficient privileges that will not actually reboot the system.

−−reboot−ops N
stop the reboot stress workers after N bogo reboot cycles.

−−remap N
start N workers that map 512 pages and re-order these pages using the deprecated system call remap_file_pages(2). Several page re-orderings are exercised: forward, reverse, random and many pages to 1 page.

−−remap−ops N
stop after N remapping bogo operations.

−R N, −−rename N
start N workers that each create a file and then repeatedly rename it.

−−rename−ops N
stop rename stress workers after N bogo rename operations.

−−resources N
start N workers that consume various system resources. Each worker will spawn 1024 child processes that iterate 1024 times consuming shared memory, heap, stack, temporary files and various file descriptors (eventfds, memoryfds, userfaultfds, pipes and sockets).

−−resources−ops N
stop after N resource child forks.

−−revio N
start N workers continually writing in reverse position order to temporary files. The default mode is to stress test reverse position ordered writes with randomly sized sparse holes between each write. With the −−aggressive option enabled without any −−revio−opts options the revio stressor will work through all the −−revio−opt options one by one to cover a range of I/O options.

−−revio−bytes N
write N bytes for each revio process, the default is 1 GB. One can specify the size as % of free space on the file system or in units of Bytes, KBytes, MBytes and GBytes using the suffix b, k, m or g.

−−revio−opts list
specify various stress test options as a comma separated list. Options are the same as −−hdd−opts but without the iovec option.

−−revio−ops N
stop revio stress workers after N bogo operations.

−−revio−write−size N
specify size of each write in bytes. Size can be from 1 byte to 4MB.

−−rlimit N
start N workers that exceed CPU and file size resource imits, generating SIGXCPU and SIGXFSZ signals.

−−rlimit−ops N
stop after N bogo resource limited SIGXCPU and SIGXFSZ signals have been caught.

−−rmap N
start N workers that exercise the VM reverse-mapping. This creates 16 processes per worker that write/read multiple file-backed memory mappings. There are 64 lots of 4 page mappings made onto the file, with each mapping overlapping the previous by 3 pages and at least 1 page of non-mapped memory between each of the mappings. Data is synchronously msync'd to the file 1 in every 256 iterations in a random manner.

−−rmap−ops N
stop after N bogo rmap memory writes/reads.

−−rseq N
start N workers that exercise restartable sequences via the rseq(2) system call. This loops over a long duration critical section that is likely to be interrupted. A rseq abort handler keeps count of the number of interruptions and a SIGSEV handler also tracks any failed rseq aborts that can occur if there is a mistmatch in a rseq check signature. Linux only.

−−rseq−ops N
stop after N bogo rseq operations. Each bogo rseq operation is equivalent to 10000 iterations over a long duration rseq handled criticial section.

−−rtc N
start N workers that exercise the real time clock (RTC) interfaces via /dev/rtc and /sys/class/rtc/rtc0. No destructive writes (modifications) are performed on the RTC. This is a Linux only stressor.

−−rtc−ops N
stop after N bogo RTC interface accesses.

−−schedpolicy N
start N workers that work set the worker to various available scheduling policies out of SCHED_OTHER, SCHED_BATCH, SCHED_IDLE, SCHED_FIFO, SCHED_RR and SCHED_DEADLINE. For the real time scheduling policies a random sched priority is selected between the minimum and maximum scheduling priority settings.

−−schedpolicy−ops N
stop after N bogo scheduling policy changes.

−−sctp N
start N workers that perform network sctp stress activity using the Stream Control Transmission Protocol (SCTP). This involves client/server processes performing rapid connect, send/receives and disconnects on the local host.

−−sctp−domain D
specify the domain to use, the default is ipv4. Currently ipv4 and ipv6 are supported.

−−sctp−ops N
stop sctp workers after N bogo operations.

−−sctp−port P
start at sctp port P. For N sctp worker processes, ports P to (P * 4) - 1 are used for ipv4, ipv6 domains and ports P to P - 1 are used for the unix domain.

−−seal N
start N workers that exercise the fcntl(2) SEAL commands on a small anonymous file created using memfd_create(2). After each SEAL command is issued the stressor also sanity checks if the seal operation has sealed the file correctly. (Linux only).

−−seal−ops N
stop after N bogo seal operations.

−−seccomp N
start N workers that exercise Secure Computing system call filtering. Each worker creates child processes that write a short message to /dev/null and then exits. 2% of the child processes have a seccomp filter that disallows the write system call and hence it is killed by seccomp with a SIGSYS. Note that this stressor can generate many audit log messages each time the child is killed. Requires CAP_SYS_ADMIN to run.

−−seccomp-ops N
stop seccomp stress workers after N seccomp filter tests.

−−seek N
start N workers that randomly seeks and performs 512 byte read/write I/O operations on a file. The default file size is 16 GB.

−−seek−ops N
stop seek stress workers after N bogo seek operations.

−−seek−punch
punch randomly located 8K holes into the file to cause more extents to force a more demanding seek stressor, (Linux only).

−−seek−size N
specify the size of the file in bytes. Small file sizes allow the I/O to occur in the cache, causing greater CPU load. Large file sizes force more I/O operations to drive causing more wait time and more I/O on the drive. One can specify the size in units of Bytes, KBytes, MBytes and GBytes using the suffix b, k, m or g.

−−sem N
start N workers that perform POSIX semaphore wait and post operations. By default, a parent and 4 children are started per worker to provide some contention on the semaphore. This stresses fast semaphore operations and produces rapid context switching.

−−sem−ops N
stop semaphore stress workers after N bogo semaphore operations.

−−sem−procs N
start N child workers per worker to provide contention on the semaphore, the default is 4 and a maximum of 64 are allowed.

−−sem−sysv N
start N workers that perform System V semaphore wait and post operations. By default, a parent and 4 children are started per worker to provide some contention on the semaphore. This stresses fast semaphore operations and produces rapid context switching.

−−sem−sysv−ops N
stop semaphore stress workers after N bogo System V semaphore operations.

−−sem−sysv−procs N
start N child processes per worker to provide contention on the System V semaphore, the default is 4 and a maximum of 64 are allowed.

−−sendfile N
start N workers that send an empty file to /dev/null. This operation spends nearly all the time in the kernel. The default sendfile size is 4MB. The sendfile options are for Linux only.

−−sendfile−ops N
stop sendfile workers after N sendfile bogo operations.

−−sendfile−size S
specify the size to be copied with each sendfile call. The default size is 4MB. One can specify the size in units of Bytes, KBytes, MBytes and GBytes using the suffix b, k, m or g.

−−session N
start N workers that create child and grandchild processes that set and get their session ids. 25% of the grandchild processes are not waited for by the child to create orphaned sessions that need to be reaped by init.

−−session−ops N
stop session workers after N child processes are spawned and reaped.

−−set N
start N workers that call system calls that try to set data in the kernel, currently these are: setgid, sethostname, setpgid, setpgrp, setuid, setgroups, setreuid, setregid, setresuid, setresgid and

setrlimit. Some of these system calls are OS specific.

−−set−ops N
stop set workers after N bogo set operations.

−−shellsort N
start N workers that sort 32 bit integers using shellsort.

−−shellsort−ops N
stop shellsort stress workers after N bogo shellsorts.

−−shellsort−size N
specify number of 32 bit integers to sort, default is 262144 (256 × 1024).

−−shm N
start N workers that open and allocate shared memory objects using the POSIX shared memory interfaces. By default, the test will repeatedly create and destroy 32 shared memory objects, each of which is 8MB in size.

−−shm−ops N
stop after N POSIX shared memory create and destroy bogo operations are complete.

−−shm−bytes N
specify the size of the POSIX shared memory objects to be created. One can specify the size as % of total available memory or in units of Bytes, KBytes, MBytes and GBytes using the suffix b, k, m or g.

−−shm−objs N
specify the number of shared memory objects to be created.

−−shm−sysv N
start N workers that allocate shared memory using the System V shared memory interface. By default, the test will repeatedly create and destroy 8 shared memory segments, each of which is 8MB in size.

−−shm−sysv−ops N
stop after N shared memory create and destroy bogo operations are complete.

−−shm−sysv−bytes N
specify the size of the shared memory segment to be created. One can specify the size as % of total available memory or in units of Bytes, KBytes, MBytes and GBytes using the suffix b, k, m or g.

−−shm−sysv−segs N
specify the number of shared memory segments to be created. The default is 8 segments.

−−sigabrt N
start N workers that create children that are killed by SIGABRT signals or by calling abort(3).

−−sigabrt−ops N
stop the sigabrt workers after N SIGABRT signals are successfully handled.

−−sigchld N
start N workers that create children to generate SIGCHLD signals. This exercises children that exit (CLD_EXITED), get killed (CLD_KILLED), get stopped (CLD_STOPPED) or continued (CLD_CONTINUED).

−−sigchld−ops N
stop the sigchld workers after N SIGCHLD signals are successfully handled.

−−sigfd N
start N workers that generate SIGRT signals and are handled by reads by a child process using a file descriptor set up using signalfd(2). (Linux only). This will generate a heavy context switch load when all CPUs are fully loaded.

−−sigfd−ops
stop sigfd workers after N bogo SIGUSR1 signals are sent.

−−sigfpe N
start N workers that rapidly cause division by zero SIGFPE faults.

−−sigfpe−ops N
stop sigfpe stress workers after N bogo SIGFPE faults.

−−sigio N
start N workers that read data from a child process via a pipe and generate SIGIO signals. This exercises asynchronous I/O via SIGIO.

−−sigio−ops N
stop sigio stress workers after handling N SIGIO signals.

−−signal N
start N workers that exercise the signal system call three different signal handlers, SIG_IGN (ignore), a SIGCHLD handler and SIG_DFL (default action). For the SIGCHLD handler, the stressor sends itself a SIGCHLD signal and checks if it has been handled. For other handlers, the stressor checks that the SIGCHLD handler has not been called. This stress test calls the signal system call directly when possible and will try to avoid the C library attempt to replace signal with the more modern sigaction system call.

−−signal−ops N
stop signal stress workers after N rounds of signal handler setting.

−−sigpending N
start N workers that check if SIGUSR1 signals are pending. This stressor masks SIGUSR1, generates a SIGUSR1 signal and uses sigpending(2) to see if the signal is pending. Then it unmasks the signal and checks if the signal is no longer pending.

−−sigpending-ops N
stop sigpending stress workers after N bogo sigpending pending/unpending checks.

−−sigpipe N
start N workers that repeatedly spawn off child process that exits before a parent can complete a pipe write, causing a SIGPIPE signal. The child process is either spawned using clone(2) if it is available or use the slower fork(2) instead.

−−sigpipe−ops N
stop N workers after N SIGPIPE signals have been caught and handled.

−−sigq N
start N workers that rapidly send SIGUSR1 signals using sigqueue(3) to child processes that wait for the signal via sigwaitinfo(2).

−−sigq−ops N
stop sigq stress workers after N bogo signal send operations.

−−sigrt N
start N workers that each create child processes to handle SIGRTMIN to SIGRMAX real time signals. The parent sends each child process a RT signal via siqueue(2) and the child process waits for this via sigwaitinfo(2). When the child receives the signal it then sends a RT signal to one of the other child processes also via sigqueue(2).

−−sigrt−ops N
stop sigrt stress workers after N bogo sigqueue signal send operations.

−−sigsegv N
start N workers that rapidly create and catch segmentation faults.

−−sigsegv−ops N
stop sigsegv stress workers after N bogo segmentation faults.

−−sigsuspend N
start N workers that each spawn off 4 child processes that wait for a SIGUSR1 signal from the parent using sigsuspend(2). The parent sends SIGUSR1 signals to each child in rapid succession. Each sigsuspend wakeup is counted as one bogo operation.

−−sigsuspend-ops N
stop sigsuspend stress workers after N bogo sigsuspend wakeups.

−−sigtrap N
start N workers that exercise the SIGTRAP signal. For systems that support SIGTRAP, the signal is generated using raise(SIGTRAP). Only x86 Linux systems the SIGTRAP is also generated by an int 3 instruction.

−−sigtrap-ops N
stop sigtrap stress workers after N SIGTRAPs have been handled.

−−skiplist N
start N workers that store and then search for integers using a skiplist. By default, 65536 integers are added and searched. This is a useful method to exercise random access of memory and processor cache.

−−skiplist−ops N
stop the skiplist worker after N skiplist store and search cycles are completed.

−−skiplist−size N
specify the size (number of integers) to store and search in the skiplist. Size can be from 1K to 4M.

−−sleep N
start N workers that spawn off multiple threads that each perform multiple sleeps of ranges 1us to 0.1s. This creates multiple context switches and timer interrupts.

−−sleep−ops N
stop after N sleep bogo operations.

−−sleep−max P
start P threads per worker. The default is 1024, the maximum allowed is 30000.

−S N, −−sock N
start N workers that perform various socket stress activity. This involves a pair of client/server processes performing rapid connect, send and receives and disconnects on the local host.

−−sock−domain D
specify the domain to use, the default is ipv4. Currently ipv4, ipv6 and unix are supported.

−−sock−nodelay
This disables the TCP Nagle algorithm, so data segments are always sent as soon as possible. This stops data from being buffered before being transmitted, hence resulting in poorer network utilisation and more context switches between the sender and receiver.

−−sock−port P
start at socket port P. For N socket worker processes, ports P to P - 1 are used.

−−sock−ops N
stop socket stress workers after N bogo operations.

−−sock−opts [random | send | sendmsg | sendmmsg]
by default, messages are sent using send(2). This option allows one to specify the sending method using send(2), sendmsg(2), sendmmsg(2) or a random selection of one of thse 3 on each iteration. Note that sendmmsg is only available for Linux systems that support this system call.

−−sock−type [stream | seqpacket]
specify the socket type to use. The default type is stream. seqpacket currently only works for the unix socket domain.

−−sockabuse N
start N workers that abuse a socket file descriptor with various file based system that don't normally act on sockets. The kernel should handle these illegal and unexpected calls gracefully.

−−sockabuse−ops N
stop after N iterations of the socket abusing stressor loop.

−−sockdiag N
start N workers that exercise the Linux sock_diag netlink socket diagnostics (Linux only). This currently requests diagnostics using UDIAG_SHOW_NAME, UDIAG_SHOW_VFS, UDIAG_SHOW_PEER, UDIAG_SHOW_ICONS, UDIAG_SHOW_RQLEN and UDIAG_SHOW_MEMINFO for the AF_UNIX family of socket connections.

−−sockdiag−ops N
stop after receiving N sock_diag diagnostic messages.

−−sockfd N
start N workers that pass file descriptors over a UNIX domain socket using the CMSG(3) ancillary data mechanism. For each worker, pair of client/server processes are created, the server opens as many file descriptors on /dev/null as possible and passing these over the socket to a client that reads these from the CMSG data and immediately closes the files.

−−sockfd−ops N
stop sockfd stress workers after N bogo operations.

−−sockfd−port P
start at socket port P. For N socket worker processes, ports P to P - 1 are used.

−−sockmany N
start N workers that use a client process to attempt to open as many as 100000 TCP/IP socket connections to a server on port 10000.

−−sockmany−ops N
stop after N connections.

−−sockpair N
start N workers that perform socket pair I/O read/writes. This involves a pair of client/server processes performing randomly sized socket I/O operations.

−−sockpair−ops N
stop socket pair stress workers after N bogo operations.

−−softlockup N
start N workers that flip between with the "real-time" SCHED_FIO and SCHED_RR scheduling policies at the highest priority to force softlockups. This can only be run with CAP_SYS_NICE capability and for best results the number of stressors should be at least the number of online CPUs. Once running, this is practically impossible to stop and it will force softlockup issues and may trigger watchdog timeout reboots.

−−softlockup−ops N
stop softlockup stress workers after N bogo scheduler policy changes.

−−spawn N
start N workers continually spawn children using posix_spawn(3) that exec stress-ng and then exit almost immediately. Currently Linux only.

−−spawn−ops N
stop spawn stress workers after N bogo spawns.

−−splice N
move data from /dev/zero to /dev/null through a pipe without any copying between kernel address space and user address space using splice(2). This is only available for Linux.

−−splice-ops N
stop after N bogo splice operations.

−−splice-bytes N
transfer N bytes per splice call, the default is 64K. One can specify the size as % of total available memory or in units of Bytes, KBytes, MBytes and GBytes using the suffix b, k, m or g.

−−stack N
start N workers that rapidly cause and catch stack overflows by use of large recursive stack allocations. Much like the brk stressor, this can eat up pages rapidly and may trigger the kernel OOM killer on the process, however, the killed stressor is respawned again by a monitoring parent process.

−−stack−fill
the default action is to touch the lowest page on each stack allocation. This option touches all the pages by filling the new stack allocation with zeros which forces physical pages to be allocated and hence is more aggressive.

−−stack−mlock
attempt to mlock stack pages into memory prohibiting them from being paged out. This is a no-op if mlock(2) is not available.

−−stack−ops N
stop stack stress workers after N bogo stack overflows.

−−stackmmap N
start N workers that use a 2MB stack that is memory mapped onto a temporary file. A recursive function works down the stack and flushes dirty stack pages back to the memory mapped file using msync(2) until the end of the stack is reached (stack overflow). This exercises dirty page and stack exception handling.

−−stackmmap−ops N
stop workers after N stack overflows have occurred.

−−str N
start N workers that exercise various libc string functions on random strings.

−−str-method strfunc
select a specific libc string function to stress. Available string functions to stress are: all, index, rindex, strcasecmp, strcat, strchr, strcoll, strcmp, strcpy, strlen, strncasecmp, strncat, strncmp, strrchr and strxfrm. See string(3) for more information on these string functions. The 'all' method is the default and will exercise all the string methods.

−−str-ops N
stop after N bogo string operations.

−−stream N
start N workers exercising a memory bandwidth stressor loosely based on the STREAM "Sustainable Memory Bandwidth in High Performance Computers" benchmarking tool by John D. McCalpin, Ph.D. This stressor allocates buffers that are at least 4 times the size of the CPU L2 cache and continually performs rounds of following computations on large arrays of double precision floating point numbers:

Operation	**Description**
copy	c[i] = a[i]
scale	b[i] = scalar * c[i]
add	c[i] = a[i] + b[i]

triad	a[i] = b[i] + (c[i] * scalar)

Since this is loosely based on a variant of the STREAM benchmark code, DO NOT submit results based on this as it is intended to in stress-ng just to stress memory and compute and NOT intended for STREAM accurate tuned or non-tuned benchmarking whatsoever. Use the official STREAM benchmarking tool if you desire accurate and standardised STREAM benchmarks.

−−stream−ops N
stop after N stream bogo operations, where a bogo operation is one round of copy, scale, add and triad operations.

−−stream−index N
specify number of stream indices used to index into the data arrays a, b and c. This adds indirection into the data lookup by using randomly shuffled indexing into the three data arrays. Level 0 (no indexing) is the default, and 3 is where all 3 arrays are indexed via 3 different randomly shuffled indexes. The higher the index setting the more impact this has on L1, L2 and L3 caching and hence forces higher memory read/write latencies.

−−stream−l3−size N
Specify the CPU Level 3 cache size in bytes. One can specify the size in units of Bytes, KBytes, MBytes and GBytes using the suffix b, k, m or g. If the L3 cache size is not provided, then stress-ng will attempt to determine the cache size, and failing this, will default the size to 4MB.

−−stream−madvise [hugepage | nohugepage | normal]
Specify the madvise options used on the memory mapped buffer used in the stream stressor. Non-linux systems will only have the ’normal’ madvise advice. The default is ’normal’.

−−swap N
start N workers that add and remove small randomly sizes swap partitions (Linux only). Note that if too many swap partitions are added then the stressors may exit with exit code 3 (not enough resources). Requires CAP_SYS_ADMIN to run.

−−swap−ops N
stop the swap workers after N swapon/swapoff iterations.

−s N, −−switch N
start N workers that send messages via pipe to a child to force context switching.

−−switch−ops N
stop context switching workers after N bogo operations.

−−switch−rate R
run the context switching at the rate of R context switches per second. Note that the specified switch rate may not be achieved because of CPU speed and memory bandwidth limitations.

−−symlink N
start N workers creating and removing symbolic links.

−−symlink−ops N
stop symlink stress workers after N bogo operations.

−−sync−file N
start N workers that perform a range of data syncs across a file using sync_file_range(2). Three mixes of syncs are performed, from start to the end of the file, from end of the file to the start, and a random mix. A random selection of valid sync types are used, covering the SYNC_FILE_RANGE_WAIT_BEFORE, SYNC_FILE_RANGE_WRITE and SYNC_FILE_RANGE_WAIT_AFTER flag bits.

−−sync−file−ops N
stop sync−file workers after N bogo sync operations.

−−sync−file−bytes N
specify the size of the file to be sync'd. One can specify the size as % of free space on the file system in units of Bytes, KBytes, MBytes and GBytes using the suffix b, k, m or g.

−−sysbadaddr N
start N workers that pass bad addresses to system calls to exercise bad address and fault handling. The addresses used are null pointers, read only pages, write only pages, unmapped addresses, text only pages, unaligned addresses and top of memory addresses.

−−sysbadaddr−ops N
stop the sysbadaddr stressors after N bogo system calls.

−−sysinfo N
start N workers that continually read system and process specific information. This reads the process user and system times using the times(2) system call. For Linux systems, it also reads overall system statistics using the sysinfo(2) system call and also the file system statistics for all mounted file systems using statfs(2).

−−sysinfo−ops N
stop the sysinfo workers after N bogo operations.

−−sysinval N
start N workers that exercise system calls in random order with permutations of invalid arguments to force kernel error handling checks. The stress test autodetects system calls that cause processes to crash or exit prematurely and will blocklist these after several repeated breakages. System call arguments that cause system calls to work successfully are also detected an blocklisted too. Linux only.

−−sysinval-ops N
stop systinval workers after N system call attempts.

−−sysfs N
start N workers that recursively read files from /sys (Linux only). This may cause specific kernel drivers to emit messages into the kernel log.

−−sys−ops N
stop sysfs reading after N bogo read operations. Note, since the number of entries may vary between kernels, this bogo ops metric is probably very misleading.

−−tee N
move data from a writer process to a reader process through pipes and to /dev/null without any copying between kernel address space and user address space using tee(2). This is only available for Linux.

−−tee-ops N
stop after N bogo tee operations.

−T N, −−timer N
start N workers creating timer events at a default rate of 1 MHz (Linux only); this can create a many thousands of timer clock interrupts. Each timer event is caught by a signal handler and counted as a bogo timer op.

−−timer−ops N
stop timer stress workers after N bogo timer events (Linux only).

−−timer−freq F
run timers at F Hz; range from 1 to 1000000000 Hz (Linux only). By selecting an appropriate frequency stress−ng can generate hundreds of thousands of interrupts per second. Note: it is also worth using −−timer−slack 0 for high frequencies to stop the kernel from coalescing timer events.

−−timer−rand
select a timer frequency based around the timer frequency +/- 12.5% random jitter. This tries to force more variability in the timer interval to make the scheduling less predictable.

−−timerfd N
start N workers creating timerfd events at a default rate of 1 MHz (Linux only); this can create a many thousands of timer clock events. Timer events are waited for on the timer file descriptor using select(2) and then read and counted as a bogo timerfd op.

−−timerfd−ops N
stop timerfd stress workers after N bogo timerfd events (Linux only).

−−timerfd−freq F
run timers at F Hz; range from 1 to 1000000000 Hz (Linux only). By selecting an appropriate frequency stress−ng can generate hundreds of thousands of interrupts per second.

−−timerfd−rand
select a timerfd frequency based around the timer frequency +/- 12.5% random jitter. This tries to force more variability in the timer interval to make the scheduling less predictable.

−−tlb−shootdown N
start N workers that force Translation Lookaside Buffer (TLB) shootdowns. This is achieved by creating up to 16 child processes that all share a region of memory and these processes are shared amongst the available CPUs. The processes adjust the page mapping settings causing TLBs to be force flushed on the other processors, causing the TLB shootdowns.

−−tlb−shootdown−ops N
stop after N bogo TLB shootdown operations are completed.

−−tmpfs N
start N workers that create a temporary file on an available tmpfs file system and perform various file based mmap operations upon it.

−−tmpfs−ops N
stop tmpfs stressors after N bogo mmap operations.

−−tmpfs−mmap−async
enable file based memory mapping and use asynchronous msync'ing on each page, see −−tmpfs−mmap−file.

−−tmpfs−mmap−file
enable tmpfs file based memory mapping and by default use synchronous msync'ing on each page.

−−tree N
start N workers that exercise tree data structures. The default is to add, find and remove 250,000 64 bit integers into AVL (avl), Red-Black (rb), Splay (splay) and binary trees. The intention of this stressor is to exercise memory and cache with the various tree operations.

−−tree−ops N
stop tree stressors after N bogo ops. A bogo op covers the addition, finding and removing all the items into the tree(s).

−−tree−size N
specify the size of the tree, where N is the number of 64 bit integers to be added into the tree.

−−tree−method [all | avl | binary | rb | splay]
specify the tree to be used. By default, both the rb ad splay trees are used (the 'all' option).

−−tsc N
start N workers that read the Time Stamp Counter (TSC) 256 times per loop iteration (bogo operation). This exercises the tsc instruction for x86, the mftb instruction for ppc64 and the rdcycle instruction for RISC-V.

−−tsc−ops N
stop the tsc workers after N bogo operations are completed.

−−tsearch N
start N workers that insert, search and delete 32 bit integers on a binary tree using tsearch(3), tfind(3) and tdelete(3). By default, there are 65536 randomized integers used in the tree. This is a useful method to exercise random access of memory and processor cache.

−−tsearch−ops N
stop the tsearch workers after N bogo tree operations are completed.

−−tsearch−size N
specify the size (number of 32 bit integers) in the array to tsearch. Size can be from 1K to 4M.

−−tun N
start N workers that create a network tunnel device and sends and receives packets over the tunnel using UDP and then destroys it. A new random 192.168.*.* IPv4 address is used each time a tunnel is created.

−−tun−ops N
stop after N iterations of creating/sending/receiving/destroying a tunnel.

−−tun−tap
use network tap device using level 2 frames (bridging) rather than a tun device for level 3 raw packets (tunnelling).

−−udp N
start N workers that transmit data using UDP. This involves a pair of client/server processes performing rapid connect, send and receives and disconnects on the local host.

−−udp−domain D
specify the domain to use, the default is ipv4. Currently ipv4, ipv6 and unix are supported.

−−udp−lite
use the UDP-Lite (RFC 3828) protocol (only for ipv4 and ipv6 domains).

−−udp−ops N
stop udp stress workers after N bogo operations.

−−udp−port P
start at port P. For N udp worker processes, ports P to P - 1 are used. By default, ports 7000 upwards are used.

−−udp−flood N
start N workers that attempt to flood the host with UDP packets to random ports. The IP address of the packets are currently not spoofed. This is only available on systems that support AF_PACKET.

−−udp−flood−domain D
specify the domain to use, the default is ipv4. Currently ipv4 and ipv6 are supported.

−−udp−flood−ops N
stop udp-flood stress workers after N bogo operations.

−−unshare N
start N workers that each fork off 32 child processes, each of which exercises the unshare(2) system call by disassociating parts of the process execution context. (Linux only).

−−unshare−ops N
stop after N bogo unshare operations.

−−uprobe N
start N workers that trace the entry to libc function getpid() using the Linux uprobe kernel tracing mechanism. This requires CAP_SYS_ADMIN capabilities and a modern Linux uprobe capable kernel.

−−uprobe−ops N
stop uprobe tracing after N trace events of the function that is being traced.

−u N, −−urandom N
start N workers reading /dev/urandom (Linux only). This will load the kernel random number source.

−−urandom−ops N
stop urandom stress workers after N urandom bogo read operations (Linux only).

−−userfaultfd N
start N workers that generate write page faults on a small anonymously mapped memory region and handle these faults using the user space fault handling via the userfaultfd mechanism. This will generate a large quantity of major page faults and also context switches during the handling of the page faults. (Linux only).

−−userfaultfd-ops N
stop userfaultfd stress workers after N page faults.

−−userfaultfd-bytes N
mmap N bytes per userfaultfd worker to page fault on, the default is 16MB. One can specify the size as % of total available memory or in units of Bytes, KBytes, MBytes and GBytes using the suffix b, k, m or g.

−−utime N
start N workers updating file timestamps. This is mainly CPU bound when the default is used as the system flushes metadata changes only periodically.

−−utime−ops N
stop utime stress workers after N utime bogo operations.

−−utime−fsync
force metadata changes on each file timestamp update to be flushed to disk. This forces the test to become I/O bound and will result in many dirty metadata writes.

−−vdso N
start N workers that repeatedly call each of the system call functions in the vDSO (virtual dynamic shared object). The vDSO is a shared library that the kernel maps into the address space of all user-space applications to allow fast access to kernel data to some system calls without the need of performing an expensive system call.

−−vdso−ops N
stop after N vDSO functions calls.

−−vdso−func F
Instead of calling all the vDSO functions, just call the vDSO function F. The functions depend on the kernel being used, but are typically clock_gettime, getcpu, gettimeofday and time.

−−vecmath N
start N workers that perform various unsigned integer math operations on various 128 bit vectors. A mix of vector math operations are performed on the following vectors: 16×8 bits, 8×16 bits, 4×32 bits, 2×64 bits. The metrics produced by this mix depend on the processor architecture and the vector math optimisations produced by the compiler.

−−vecmath−ops N
stop after N bogo vector integer math operations.

−−verity N
start N workers that exercise read-only file based authenticy protection using the verity ioctls FS_IOC_ENABLE_VERITY and FS_IOC_MEASURE_VERITY. This requires file systems with verity support (currently ext4 and f2fs on Linux) with the verity feature enabled. The test attempts to creates a small file with multiple small extents and enables verity on the file and verifies it. It also checks to see if the file has verity enabled with the FS_VERITY_FL bit set on the file flags.

−−verity−ops N
stop the verity workers after N file create, enable verity, check verity and unlink cycles.

−−vfork N
start N workers continually vforking children that immediately exit.

−−vfork−ops N
stop vfork stress workers after N bogo operations.

−−vfork−max P
create P processes and then wait for them to exit per iteration. The default is just 1; higher values will create many temporary zombie processes that are waiting to be reaped. One can potentially fill up the process table using high values for −−vfork−max and −−vfork.

−−vforkmany N
start N workers that spawn off a chain of vfork children until the process table fills up and/or vfork fails. vfork can rapidly create child processes and the parent process has to wait until the child dies, so this stressor rapidly fills up the process table.

−−vforkmany−ops N
stop vforkmany stressors after N vforks have been made.

−m N, −−vm N
start N workers continuously calling mmap(2)/munmap(2) and writing to the allocated memory. Note that this can cause systems to trip the kernel OOM killer on Linux systems if not enough physical memory and swap is not available.

−−vm−bytes N
mmap N bytes per vm worker, the default is 256MB. One can specify the size as % of total available memory or in units of Bytes, KBytes, MBytes and GBytes using the suffix b, k, m or g.

−−vm−ops N
stop vm workers after N bogo operations.

−−vm−hang N
sleep N seconds before unmapping memory, the default is zero seconds. Specifying 0 will do an infinite wait.

−−vm−keep
do not continually unmap and map memory, just keep on re-writing to it.

−−vm−locked
Lock the pages of the mapped region into memory using mmap MAP_LOCKED (since Linux 2.5.37). This is similar to locking memory as described in mlock(2).

−−vm−madvise advice
Specify the madvise ’advice’ option used on the memory mapped regions used in the vm stressor. Non-linux systems will only have the ’normal’ madvise advice, linux systems support ’dontneed’, ’hugepage’, ’mergeable’ , ’nohugepage’, ’normal’, ’random’, ’sequential’, ’unmergeable’ and ’willneed’ advice. If this option is not used then the default is to pick random madvise advice for each mmap call. See madvise(2) for more details.

−−vm−method m
specify a vm stress method. By default, all the stress methods are exercised sequentially, however one can specify just one method to be used if required. Each of the vm workers have 3 phases:

1. Initialised. The anonymously memory mapped region is set to a known pattern.

2. Exercised. Memory is modified in a known predictable way. Some vm workers alter memory sequentially, some use small or large strides to step along memory.

3. Checked. The modified memory is checked to see if it matches the expected result.

The vm methods containing ’prime’ in their name have a stride of the largest prime less than 2ˆ64, allowing to them to thoroughly step through memory and touch all locations just once while also doing without touching memory cells next to each other. This strategy exercises the cache and page non-locality.

Since the memory being exercised is virtually mapped then there is no guarantee of touching page addresses in any particular physical order. These workers should not be used to test that all the system's memory is working correctly either, use tools such as memtest86 instead.

The vm stress methods are intended to exercise memory in ways to possibly find memory issues and to try to force thermal errors.

Available vm stress methods are described as follows:

Method	**Description**
all	iterate over all the vm stress methods as listed below.
flip	sequentially work through memory 8 times, each time just one bit in memory flipped (inverted). This will effectively invert each byte in 8 passes.
galpat-0	galloping pattern zeros. This sets all bits to 0 and flips just 1 in 4096 bits to 1. It then checks to see if the 1s are pulled down to 0 by their neighbours or of the neighbours have been pulled up to 1.
galpat-1	galloping pattern ones. This sets all bits to 1 and flips just 1 in 4096 bits to 0. It then checks to see if the 0s are pulled up to 1 by their neighbours or of the neighbours have been pulled down to 0.
gray	fill the memory with sequential gray codes (these only change 1 bit at a time between adjacent bytes) and then check if they are set correctly.
incdec	work sequentially through memory twice, the first pass increments each byte by a specific value and the second pass decrements each byte back to the original start value. The increment/decrement value changes on each invocation of the stressor.
inc-nybble	initialise memory to a set value (that changes on each invocation of the stressor) and then sequentially work through each byte incrementing the bottom 4 bits by 1 and the top 4 bits by 15.
rand-set	sequentially work through memory in 64 bit chunks setting bytes in the chunk to the same 8 bit random value. The random value changes on each chunk. Check that the values have not changed.
rand-sum	sequentially set all memory to random values and then summate the number of bits that have changed from the original set values.
read64	sequentially read memory using 32 x 64 bit reads per bogo loop. Each loop equates to one bogo operation. This exercises raw memory reads.
ror	fill memory with a random pattern and then sequentially rotate 64 bits of memory right by one bit, then check the final load/rotate/stored values.
swap	fill memory in 64 byte chunks with random patterns. Then swap each 64 chunk with a randomly chosen chunk. Finally, reverse the swap to put the chunks back to their original place and check if the data is correct. This exercises adjacent and random memory load/stores.
move-inv	sequentially fill memory 64 bits of memory at a time with random values, and then check if the memory is set correctly. Next, sequentially invert each 64 bit pattern and again check if the memory is set as expected.

modulo-x	fill memory over 23 iterations. Each iteration starts one byte further along from the start of the memory and steps along in 23 byte strides. In each stride, the first byte is set to a random pattern and all other bytes are set to the inverse. Then it checks see if the first byte contains the expected random pattern. This exercises cache store/reads as well as seeing if neighbouring cells influence each other.
prime-0	iterate 8 times by stepping through memory in very large prime strides clearing just on bit at a time in every byte. Then check to see if all bits are set to zero.
prime-1	iterate 8 times by stepping through memory in very large prime strides setting just on bit at a time in every byte. Then check to see if all bits are set to one.
prime-gray-0	first step through memory in very large prime strides clearing just on bit (based on a gray code) in every byte. Next, repeat this but clear the other 7 bits. Then check to see if all bits are set to zero.
prime-gray-1	first step through memory in very large prime strides setting just on bit (based on a gray code) in every byte. Next, repeat this but set the other 7 bits. Then check to see if all bits are set to one.
rowhammer	try to force memory corruption using the rowhammer memory stressor. This fetches two 32 bit integers from memory and forces a cache flush on the two addresses multiple times. This has been known to force bit flipping on some hardware, especially with lower frequency memory refresh cycles.
walk-0d	for each byte in memory, walk through each data line setting them to low (and the others are set high) and check that the written value is as expected. This checks if any data lines are stuck.
walk-1d	for each byte in memory, walk through each data line setting them to high (and the others are set low) and check that the written value is as expected. This checks if any data lines are stuck.
walk-0a	in the given memory mapping, work through a range of specially chosen addresses working through address lines to see if any address lines are stuck low. This works best with physical memory addressing, however, exercising these virtual addresses has some value too.
walk-1a	in the given memory mapping, work through a range of specially chosen addresses working through address lines to see if any address lines are stuck high. This works best with physical memory addressing, however, exercising these virtual addresses has some value too.
write64	sequentially write memory using 32 x 64 bit writes per bogo loop. Each loop equates to one bogo operation. This exercises raw memory writes. Note that memory writes are not checked at the end of each test iteration.
zero-one	set all memory bits to zero and then check if any bits are not zero. Next, set all the memory bits to one and check if any bits are not one.

−−vm−populate

populate (prefault) page tables for the memory mappings; this can stress swapping. Only available on systems that support MAP_POPULATE (since Linux 2.5.46).

−−vm−addr N

start N workers that exercise virtual memory addressing using various methods to walk through a memory mapped address range. This will exercise mapped private addresses from 8MB to 64MB per worker and try to generate cache and TLB inefficient addressing patterns. Each method will set the memory to a random pattern in a write phase and then sanity check this in a read phase.

−−vm−addr−ops N
stop N workers after N bogo addressing passes.

−−vm−addr−method M
specify a vm address stress method. By default, all the stress methods are exercised sequentially, however one can specify just one method to be used if required.

Available vm address stress methods are described as follows:

Method	**Description**
all	iterate over all the vm stress methods as listed below.
pwr2	work through memory addresses in steps of powers of two.
pwr2inv	like pwr2, but with the all relevant address bits inverted.
gray	work through memory with gray coded addresses so that each change of address just changes 1 bit compared to the previous address.
grayinv	like gray, but with the all relevant address bits inverted, hence all bits change apart from 1 in the address range.
rev	work through the address range with the bits in the address range reversed.
revinv	like rev, but with all the relevant address bits inverted.
inc	work through the address range forwards sequentially, byte by byte.
incinv	like inc, but with all the relevant address bits inverted.
dec	work through the address range backwards sequentially, byte by byte.
decinv	like dec, but with all the relevant address bits inverted.

−−vm−rw N
start N workers that transfer memory to/from a parent/child using process_vm_writev(2) and process_vm_readv(2). This is feature is only supported on Linux. Memory transfers are only verified if the −−verify option is enabled.

−−vm−rw−ops N
stop vm−rw workers after N memory read/writes.

−−vm−rw−bytes N
mmap N bytes per vm−rw worker, the default is 16MB. One can specify the size as % of total available memory or in units of Bytes, KBytes, MBytes and GBytes using the suffix b, k, m or g.

−−vm−segv N
start N workers that create a child process that unmaps its address space causing a SIGSEGV on return from the unmap.

−−vm−segv−ops N
stop after N bogo vm−segv SIGSEGV faults.

−−vm−splice N
move data from memory to /dev/null through a pipe without any copying between kernel address space and user address space using vmsplice(2) and splice(2). This is only available for Linux.

−−vm−splice-ops N
stop after N bogo vm−splice operations.

−−vm−splice-bytes N
transfer N bytes per vmsplice call, the default is 64K. One can specify the size as % of total available memory or in units of Bytes, KBytes, MBytes and GBytes using the suffix b, k, m or g.

−−wait N
start N workers that spawn off two children; one spins in a pause(2) loop, the other continually stops and continues the first. The controlling process waits on the first child to be resumed by the

delivery of SIGCONT using waitpid(2) and waitid(2).

−−wait−ops N
stop after N bogo wait operations.

−−watchdog N
start N workers that exercising the /dev/watchdog watchdog interface by opening it, perform various watchdog specific ioctl(2) commands on the device and close it. Before closing the special watchdog magic close message is written to the device to try and force it to never trip a watchdog reboot after the stressor has been run. Note that this stressor needs to be run as root with the −−pathological option and is only available on Linux.

−−watchdog−ops N
stop after N bogo operations on the watchdog device.

−−wcs N
start N workers that exercise various libc wide character string functions on random strings.

−−wcs-method wcsfunc
select a specific libc wide character string function to stress. Available string functions to stress are: all, wcscasecmp, wcscat, wcschr, wcscoll, wcscmp, wcscpy, wcslen, wcsncasecmp, wcsncat, wcsncmp, wcsrchr and wcsxfrm. The 'all' method is the default and will exercise all the string methods.

−−wcs-ops N
stop after N bogo wide character string operations.

−−x86syscall N
start N workers that repeatedly exercise the x86-64 syscall instruction to call the getcpu(2), gettimeofday(2) and time(2) system using the Linux vsyscall handler. Only for Linux.

−−x86syscall−ops N
stop after N x86syscall system calls.

−−x86syscall−func F
Instead of exercising the 3 syscall system calls, just call the syscall function F. The function F must be one of getcpu, gettimeofday and time.

−−xattr N
start N workers that create, update and delete batches of extended attributes on a file.

−−xattr−ops N
stop after N bogo extended attribute operations.

−y N, −−yield N
start N workers that call sched_yield(2). This stressor ensures that at least 2 child processes per CPU exercise shield_yield(2) no matter how many workers are specified, thus always ensuring rapid context switching.

−−yield−ops N
stop yield stress workers after N sched_yield(2) bogo operations.

−−zero N
start N workers reading /dev/zero.

−−zero−ops N
stop zero stress workers after N /dev/zero bogo read operations.

−−zlib N
start N workers compressing and decompressing random data using zlib. Each worker has two processes, one that compresses random data and pipes it to another process that decompresses the data. This stressor exercises CPU, cache and memory.

−−zlib−ops N
stop after N bogo compression operations, each bogo compression operation is a compression of 64K of random data at the highest compression level.

−−zlib−level L
specify the compression level (0..9), where 0 = no compression, 1 = fastest compression and 9 = best compression.

−−zlib−method method
specify the type of random data to send to the zlib library. By default, the data stream is created from a random selection of the different data generation processes. However one can specify just one method to be used if required. Available zlib data generation methods are described as follows:

Method	**Description**
00ff	randomly distributed 0x00 and 0xFF values.
ascii01	randomly distributed ASCII 0 and 1 characters.
asciidigits	randomly distributed ASCII digits in the range of 0 and 9.
bcd	packed binary coded decimals, 0..99 packed into 2 4-bit nybbles.
binary	32 bit random numbers.
brown	8 bit brown noise (Brownian motion/Random Walk noise).
double	double precision floating point numbers from sin(θ).
fixed	data stream is repeated 0x04030201.
gray	16 bit gray codes generated from an incrementing counter.
latin	Random latin sentences from a sample of Lorem Ipsum text.
logmap	Values generated from a logistical map of the equation $Xn+1 = r \times Xn \times (1 - Xn)$ where $r > \approx 3.56994567$ to produce chaotic data. The values are scaled by a large arbitrary value and the lower 8 bits of this value are compressed.
lfsr32	Values generated from a 32 bit Galois linear feedback shift register using the polynomial x↑32 + x↑31 + x↑29 + x + 1. This generates a ring of 2↑32 - 1 unique values (all 32 bit values except for 0).
lrand48	Uniformly distributed pseudo-random 32 bit values generated from lrand48(3).
morse	Morse code generated from random latin sentences from a sample of Lorem Ipsum text.
nybble	randomly distributed bytes in the range of 0x00 to 0x0f.
objcode	object code selected from a random start point in the stress-ng text segment.
parity	7 bit binary data with 1 parity bit.
pink	pink noise in the range 0..255 generated using the Gardner method with the McCartney selection tree optimization. Pink noise is where the power spectral density is inversely proportional to the frequency of the signal and hence is slightly compressible.
random	segments of the data stream are created by randomly calling the different data generation methods.
rarely1	data that has a single 1 in every 32 bits, randomly located.
rarely0	data that has a single 0 in every 32 bits, randomly located.
text	random ASCII text.
utf8	random 8 bit data encoded to UTF-8.
zero	all zeros, compresses very easily.

−−zlib−window-bits W
specify the window bits used to specify the history buffer size. The value is specified as the base two logarithm of the buffer size (e.g. value 9 is 2^9 = 512 bytes). Default is 15.

Values:
-8-(-15): raw deflate format
8-15: zlib format

24-31: gzip format
40-47: inflate auto format detection using zlib deflate format

−−zlib−mem-level L specify the reserved compression state memory for zlib. Default is 8.

Values:
1 = minimum memory usage
9 = maximum memory usage

−−zlib−strategy S
specifies the strategy to use when deflating data. This is used to tune the compression algorithm. Default is 0.

Values:
0: used for normal data (Z_DEFAULT_STRATEGY)
1: for data generated by a filter or predictor (Z_FILTERED)
2: forces huffman encoding (Z_HUFFMAN_ONLY)
3: Limit match distances to one run-length-encoding (Z_RLE)
4: prevents dynamic huffman codes (Z_FIXED)

−−zlib−stream-bytes S
specify the amount of bytes to deflate until deflate should finish the block and return with Z_STREAM_END. One can specify the size in units of Bytes, KBytes, MBytes and GBytes using the suffix b, k, m or g. Default is 0 which creates and endless stream until stressor ends.

Values:
0: creates an endless deflate stream until stressor stops
n: creates an stream of n bytes over and over again. Each block will be closed with Z_STREAM_END.

−−zombie N
start N workers that create zombie processes. This will rapidly try to create a default of 8192 child processes that immediately die and wait in a zombie state until they are reaped. Once the maximum number of processes is reached (or fork fails because one has reached the maximum allowed number of children) the oldest child is reaped and a new process is then created in a first-in first-out manner, and then repeated.

−−zombie−ops N
stop zombie stress workers after N bogo zombie operations.

−−zombie−max N
try to create as many as N zombie processes. This may not be reached if the system limit is less than N.

EXAMPLES

stress−ng −−vm 8 −−vm−bytes 80% -t 1h

run 8 virtual memory stressors that combined use 80% of the available memory for 1 hour. Thus each stressor uses 10% of the available memory.

stress−ng −−cpu 4 −−io 2 −−vm 1 −−vm−bytes 1G −−timeout 60s

runs for 60 seconds with 4 cpu stressors, 2 io stressors and 1 vm stressor using 1GB of virtual memory.

stress−ng −−iomix 2 −−iomix−bytes 10% -t 10m

runs 2 instances of the mixed I/O stressors using a total of 10% of the available file system space for 10 minutes. Each stressor will use 5% of the available file system space.

stress−ng −−cyclic 1 −−cyclic−dist 2500 −−cyclic−method clock_ns −−cyclic−prio 100 −−cyclic−sleep 10000 −−hdd 0 -t 1m

measures real time scheduling latencies created by the hdd stressor. This uses the high resolution nanosecond clock to measure latencies during sleeps of 10,000 nanoseconds. At the end of 1 minute of stressing, the latency distribution with 2500 ns intervals will be displayed. NOTE: this must be run with the CAP_SYS_NICE capability to enable the real time scheduling to get accurate measurements.

stress−ng −−cpu 8 −−cpu−ops 800000

runs 8 cpu stressors and stops after 800000 bogo operations.

stress−ng −−sequential 2 −−timeout 2m −−metrics

run 2 simultaneous instances of all the stressors sequentially one by one, each for 2 minutes and summarise with performance metrics at the end.

stress−ng −−cpu 4 −−cpu-method fft −−cpu-ops 10000 −−metrics−brief

run 4 FFT cpu stressors, stop after 10000 bogo operations and produce a summary just for the FFT results.

stress−ng −−cpu 0 −−cpu-method all −t 1h

run cpu stressors on all online CPUs working through all the available CPU stressors for 1 hour.

stress−ng −−all 4 −−timeout 5m

run 4 instances of all the stressors for 5 minutes.

stress−ng −−random 64

run 64 stressors that are randomly chosen from all the available stressors.

stress−ng −−cpu 64 −−cpu−method all −−verify −t 10m −−metrics−brief

run 64 instances of all the different cpu stressors and verify that the computations are correct for 10 minutes with a bogo operations summary at the end.

stress−ng −−sequential 0 −t 10m

run all the stressors one by one for 10 minutes, with the number of instances of each stressor matching the number of online CPUs.

stress−ng −−sequential 8 −−class io −t 5m −−times

run all the stressors in the io class one by one for 5 minutes each, with 8 instances of each stressor running concurrently and show overall time utilisation statistics at the end of the run.

stress−ng −−all 0 −−maximize −−aggressive

run all the stressors (1 instance of each per CPU) simultaneously, maximize the settings (memory sizes, file allocations, etc.) and select the most demanding/aggressive options.

stress−ng −−random 32 −x numa,hdd,key

run 32 randomly selected stressors and exclude the numa, hdd and key stressors

stress−ng −−sequential 4 −−class vm −−exclude bigheap,brk,stack

run 4 instances of the VM stressors one after each other, excluding the bigheap, brk and stack stressors

stress−ng −−taskset 0,2-3 −−cpu 3

run 3 instances of the CPU stressor and pin them to CPUs 0, 2 and 3.

EXIT STATUS

Status	Description
0	Success.
1	Error; incorrect user options or a fatal resource issue in the stress-ng stressor harness (for example, out of memory).
2	One or more stressors failed.

3	One or more stressors failed to initialise because of lack of resources, for example ENOMEM (no memory), ENOSPC (no space on file system) or a missing or unimplemented system call.
4	One or more stressors were not implemented on a specific architecture or operating system.
5	A stressor has been killed by an unexpected signal.
6	A stressor exited by exit(2) which was not expected and timing metrics could not be gathered.
7	The bogo ops metrics maybe untrustworthy. This is most likely to occur when a stress test is terminated during the update of a bogo-ops counter such as when it has been OOM killed. A less likely reason is that the counter ready indicator has been corrupted.

BUGS

File bug reports at:
https://launchpad.net/ubuntu/+source/stress−ng/+filebug

SEE ALSO

cpuburn(1), **perf**(1), **stress**(1), **taskset**(1)

AUTHOR

stress−ng was written by Colin King <colin.king@canonical.com> and is a clean room re-implementation and extension of the original stress tool by Amos Waterland. Thanks also for contributions from Abdul Haleem, André Wild, Baruch Siach, Carlos Santos, Christian Ehrhardt, Chunyu Hu, David Turner, Dominik B Czarnota, Fabrice Fontaine, Helmut Grohne, James Hunt, Jim Rowan, Joseph DeVincentis, Khalid Elmously, Khem Raj, Luca Pizzamiglio, Luis Henriques, Manoj Iyer, Matthew Tippett, Mauricio Faria de Oliveira, Piyush Goyal, Ralf Ramsauer, Rob Colclaser, Thadeu Lima de Souza Cascardo, Thia Wyrod, Tim Gardner, Tim Orling, Tommi Rantala, Zhiyi Sun and others.

NOTES

Sending a SIGALRM, SIGINT or SIGHUP to stress-ng causes it to terminate all the stressor processes and ensures temporary files and shared memory segments are removed cleanly.

Sending a SIGUSR2 to stress-ng will dump out the current load average and memory statistics.

Note that the stress−ng cpu, io, vm and hdd tests are different implementations of the original stress tests and hence may produce different stress characteristics. stress−ng does not support any GPU stress tests.

The bogo operations metrics may change with each release because of bug fixes to the code, new features, compiler optimisations or changes in system call performance.

COPYRIGHT