Documentation for /proc/sys/kernel/

This file contains documentation for the sysctl files in `/proc/sys/kernel/`.

The files in this directory can be used to tune and monitor miscellaneous and general things in the operation of the Linux kernel. Since some of the files *can* be used to screw up your system, it is advisable to read both documentation and source before actually making adjustments.

Currently, these files might (depending on your configuration) show up in `/proc/sys/kernel`:

- perf_event_paranoid
- perf_event_max_stack
- perf_event_mlock_kb
- perf_event_max_contexts_per_stack
- perf_user_access (arm64 only)
- pid_max
- ns_last_pid
- powersave-nap (PPC only)
- printk
- printk_delay
- printk_ratelimit
- printk_ratelimit_burst
- printk_devkmsg
- pty
- random
- randomize_va_space
- real-root-dev
- reboot-cmd (SPARC only)
- sched_energy_aware
- task_delayacct
- sched_schedstats
- sched_util_clamp_min
- sched_util_clamp_max
- sched_util_clamp_min_rt_default
- seccomp
- sg-big-buff
- shmall
- shmmax
- shmmni
- shm_rmid_forced
- sysctl_writes_strict
- softlockup_all_cpu_backtrace
- softlockup_panic
- soft_watchdog
- stack_erasing
- stop-a (SPARC only)
- sysrq
- tainted
- threads-max
- traceoff_on_warning
- tracepoint_printk
- unaligned-dump-stack (ia64)
- unaligned-trap
- unknown_nmi_panic
- unprivileged_bpf_disabled
- watchdog
- watchdog_cpumask
- watchdog_thresh

acct

```
highwater lowwater frequency
```

If BSD-style process accounting is enabled these values control its behaviour. If free space on filesystem where the log lives goes below `lowwater``% accounting suspends. If free space gets above ``highwater``% accounting resumes. ``frequency` determines how often do we check the amount of free space (value is in seconds). Default:

```
4 2 30
```

That is, suspend accounting if free space drops below 2%; resume it if it increases to at least 4%; consider information about amount of free space valid for 30 seconds.

acpi_video_flags

See Documentation/power/video.rst. This allows the video resume mode to be set, in a similar fashion to the `acpi_sleep` kernel parameter, by combining the following values:

1	s3_bios
2	s3_mode
4	s3_beep

auto_msgmni

This variable has no effect and may be removed in future kernel releases. Reading it always returns 0. Up to Linux 3.17, it enabled/disabled automatic recomputing of msgmni upon memory add/remove or upon IPC namespace creation/removal. Echoing "1" into this file enabled msgmni automatic recomputing. Echoing "0" turned it off. The default value was 1.

bootloader_type (x86 only)

This gives the bootloader type number as indicated by the bootloader, shifted left by 4, and OR'd with the low four bits of the bootloader version. The reason for this encoding is that this used to match the `type_of_loader` field in the kernel header; the encoding is kept for backwards compatibility. That is, if the full bootloader type number is 0x15 and the full version number is 0x234, this file will contain the value 340 = 0x154.

See the `type_of_loader` and `ext_loader_type` fields in Documentation/x86/boot.rst for additional information.

bootloader_version (x86 only)

The complete bootloader version number. In the example above, this file will contain the value 564 = 0x234.

See the `type_of_loader` and `ext_loader_ver` fields in Documentation/x86/boot.rst for additional information.

bpf_stats_enabled

Controls whether the kernel should collect statistics on BPF programs (total time spent running, number of times run...). Enabling statistics causes a slight reduction in performance on each program run. The statistics can be seen using `bpftool`.

0	Don't collect statistics (default).
1	Collect statistics.

cad_pid

This is the pid which will be signalled on reboot (notably, by Ctrl-Alt-Delete). Writing a value to this file which doesn't correspond to a running process will result in `-ESRCH`.

See also ctrl-alt-del.

cap_last_cap

Highest valid capability of the running kernel. Exports `CAP_LAST_CAP` from the kernel.

core_pattern

`core_pattern` is used to specify a core dumpfile pattern name.

- max length 127 characters; default value is "core"
- `core_pattern` is used as a pattern template for the output filename; certain string patterns (beginning with '%') are substituted with their actual values.
- backward compatibility with `core_uses_pid`:

  If `core_pattern` does not include "%p" (default does not) and `core_uses_pid` is set, then .PID will be appended to the filename.

- corename format specifiers

%<NUL>	'%' is dropped
%%	output one '%'
%p	pid
%P	global pid (init PID namespace)
%i	tid
%I	global tid (init PID namespace)
%u	uid (in initial user namespace)
%g	gid (in initial user namespace)

%d	dump mode, matches `PR_SET_DUMPABLE` and `/proc/sys/fs/suid_dumpable`
%s	signal number
%t	UNIX time of dump
%h	hostname
%e	executable filename (may be shortened, could be changed by prctl etc)
%f	executable filename
%E	executable path
%c	maximum size of core file by resource limit RLIMIT_CORE
% <OTHER>	both are dropped

- If the first character of the pattern is a '|', the kernel will treat the rest of the pattern as a command to run. The core dump will be written to the standard input of that program instead of to a file.

core_pipe_limit

This sysctl is only applicable when core_pattern is configured to pipe core files to a user space helper (when the first character of `core_pattern` is a '|', see above). When collecting cores via a pipe to an application, it is occasionally useful for the collecting application to gather data about the crashing process from its `/proc/pid` directory. In order to do this safely, the kernel must wait for the collecting process to exit, so as not to remove the crashing processes proc files prematurely. This in turn creates the possibility that a misbehaving userspace collecting process can block the reaping of a crashed process simply by never exiting. This sysctl defends against that. It defines how many concurrent crashing processes may be piped to user space applications in parallel. If this value is exceeded, then those crashing processes above that value are noted via the kernel log and their cores are skipped. 0 is a special value, indicating that unlimited processes may be captured in parallel, but that no waiting will take place (i.e. the collecting process is not guaranteed access to `/proc/<crashing pid>/`). This value defaults to 0.

core_uses_pid

The default coredump filename is "core". By setting `core_uses_pid` to 1, the coredump filename becomes core.PID. If core_pattern does not include "%p" (default does not) and `core_uses_pid` is set, then .PID will be appended to the filename.

ctrl-alt-del

When the value in this file is 0, ctrl-alt-del is trapped and sent to the `init(1)` program to handle a graceful restart. When, however, the value is > 0, Linux's reaction to a Vulcan Nerve Pinch (tm) will be an immediate reboot, without even syncing its dirty buffers.

Note:

when a program (like dosemu) has the keyboard in 'raw' mode, the ctrl-alt-del is intercepted by the program before it ever reaches the kernel tty layer, and it's up to the program to decide what to do with it.

dmesg_restrict

This toggle indicates whether unprivileged users are prevented from using `dmesg(8)` to view messages from the kernel's log buffer. When `dmesg_restrict` is set to 0 there are no restrictions. When `dmesg_restrict` is set to 1, users must have `CAP_SYSLOG` to use `dmesg(8)`.

The kernel config option `CONFIG_SECURITY_DMESG_RESTRICT` sets the default value of `dmesg_restrict`.

domainname & hostname

These files can be used to set the NIS/YP domainname and the hostname of your box in exactly the same way as the commands domainname and hostname, i.e.:

```
# echo "darkstar" > /proc/sys/kernel/hostname
# echo "mydomain" > /proc/sys/kernel/domainname
```

has the same effect as:

```
# hostname "darkstar"
# domainname "mydomain"
```

Note, however, that the classic darkstar.frop.org has the hostname "darkstar" and DNS (Internet Domain Name Server) domainname "frop.org", not to be confused with the NIS (Network Information Service) or YP (Yellow Pages) domainname. These two domain names are in general different. For a detailed discussion see the `hostname(1)` man page.

firmware_config

See Documentation/driver-api/firmware/fallback-mechanisms.rst.

The entries in this directory allow the firmware loader helper fallback to be controlled:

- `force_sysfs_fallback`, when set to 1, forces the use of the fallback;
- `ignore_sysfs_fallback`, when set to 1, ignores any fallback.

ftrace_dump_on_oops

Determines whether `ftrace_dump()` should be called on an oops (or kernel panic). This will output the contents of the ftrace buffers to the console. This is very useful for capturing traces that lead to crashes and outputting them to a serial console.

0	Disabled (default).
1	Dump buffers of all CPUs.
2	Dump the buffer of the CPU that triggered the oops.

ftrace_enabled, stack_tracer_enabled

See Documentation/trace/ftrace.rst.

hardlockup_all_cpu_backtrace

This value controls the hard lockup detector behavior when a hard lockup condition is detected as to whether or not to gather further debug information. If enabled, arch-specific all-CPU stack dumping will be initiated.

0	Do nothing. This is the default behavior.
1	On detection capture more debug information.

hardlockup_panic

This parameter can be used to control whether the kernel panics when a hard lockup is detected.

0	Don't panic on hard lockup.
1	Panic on hard lockup.

See Documentation/admin-guide/lockup-watchdogs.rst for more information. This can also be set using the nmi_watchdog kernel parameter.

hotplug

Path for the hotplug policy agent. Default value is `CONFIG_UEVENT_HELPER_PATH`, which in turn defaults to the empty string.

This file only exists when `CONFIG_UEVENT_HELPER` is enabled. Most modern systems rely exclusively on the netlink-based uevent source and don't need this.

hung_task_all_cpu_backtrace

If this option is set, the kernel will send an NMI to all CPUs to dump their backtraces when a hung task is detected. This file shows up if CONFIG_DETECT_HUNG_TASK and CONFIG_SMP are enabled.

0: Won't show all CPUs backtraces when a hung task is detected. This is the default behavior.

1: Will non-maskably interrupt all CPUs and dump their backtraces when a hung task is detected.

hung_task_panic

Controls the kernel's behavior when a hung task is detected. This file shows up if `CONFIG_DETECT_HUNG_TASK` is enabled.

0	Continue operation. This is the default behavior.
1	Panic immediately.

hung_task_check_count

The upper bound on the number of tasks that are checked. This file shows up if `CONFIG_DETECT_HUNG_TASK` is enabled.

hung_task_timeout_secs

When a task in D state did not get scheduled for more than this value report a warning. This file shows up if `CONFIG_DETECT_HUNG_TASK` is enabled.

0 means infinite timeout, no checking is done.

Possible values to set are in range {0:LONG_MAX/HZ}.

hung_task_check_interval_secs

Hung task check interval. If hung task checking is enabled (see hung_task_timeout_secs), the check is done every hung_task_check_interval_secs seconds. This file shows up if CONFIG_DETECT_HUNG_TASK is enabled.

0 (default) means use hung_task_timeout_secs as checking interval.

Possible values to set are in range {0:LONG_MAX/HZ}.

hung_task_warnings

The maximum number of warnings to report. During a check interval if a hung task is detected, this value is decreased by 1. When this value reaches 0, no more warnings will be reported. This file shows up if CONFIG_DETECT_HUNG_TASK is enabled.

-1: report an infinite number of warnings.

hyperv_record_panic_msg

Controls whether the panic kmsg data should be reported to Hyper-V.

0	Do not report panic kmsg data.
1	Report the panic kmsg data. This is the default behavior.

ignore-unaligned-usertrap

On architectures where unaligned accesses cause traps, and where this feature is supported (CONFIG_SYSCTL_ARCH_UNALIGN_NO_WARN; currently, arc and ia64), controls whether all unaligned traps are logged.

0	Log all unaligned accesses.
1	Only warn the first time a process traps. This is the default setting.

See also unaligned-trap and unaligned-dump-stack. On ia64, this allows system administrators to override the IA64_THREAD_UAC_NOPRINT prctl and avoid logs being flooded.

kexec_load_disabled

A toggle indicating if the kexec_load syscall has been disabled. This value defaults to 0 (false: kexec_load enabled), but can be set to 1 (true: kexec_load disabled). Once true, kexec can no longer be used, and the toggle cannot be set back to false. This allows a kexec image to be loaded before disabling the syscall, allowing a system to set up (and later use) an image without it being altered. Generally used together with the modules_disabled sysctl.

kptr_restrict

This toggle indicates whether restrictions are placed on exposing kernel addresses via /proc and other interfaces.

When kptr_restrict is set to 0 (the default) the address is hashed before printing. (This is the equivalent to %p.)

When kptr_restrict is set to 1, kernel pointers printed using the %pK format specifier will be replaced with 0s unless the user has CAP_SYSLOG and effective user and group ids are equal to the real ids. This is because %pK checks are done at read() time rather than open() time, so if permissions are elevated between the open() and the read() (e.g via a setuid binary) then %pK will not leak kernel pointers to unprivileged users. Note, this is a temporary solution only. The correct long-term solution is to do the permission checks at open() time. Consider removing world read permissions from files that use %pK, and using dmesg_restrict to protect against uses of %pK in dmesg(8) if leaking kernel pointer values to unprivileged users is a concern.

When kptr_restrict is set to 2, kernel pointers printed using %pK will be replaced with 0s regardless of privileges.

modprobe

The full path to the usermode helper for autoloading kernel modules, by default CONFIG_MODPROBE_PATH, which in turn defaults to "/sbin/modprobe". This binary is executed when the kernel requests a module. For example, if userspace passes an unknown filesystem type to mount(), then the kernel will automatically request the corresponding filesystem module by executing this usermode helper. This usermode helper should insert the needed module into the kernel.

This sysctl only affects module autoloading. It has no effect on the ability to explicitly insert modules.

This sysctl can be used to debug module loading requests:

```
echo '#! /bin/sh' > /tmp/modprobe
```

```
echo 'echo "$@" >> /tmp/modprobe.log' >> /tmp/modprobe
echo 'exec /sbin/modprobe "$@"' >> /tmp/modprobe
chmod a+x /tmp/modprobe
echo /tmp/modprobe > /proc/sys/kernel/modprobe
```

Alternatively, if this sysctl is set to the empty string, then module autoloading is completely disabled. The kernel will not try to execute a usermode helper at all, nor will it call the kernel_module_request LSM hook.

If CONFIG_STATIC_USERMODEHELPER=y is set in the kernel configuration, then the configured static usermode helper overrides this sysctl, except that the empty string is still accepted to completely disable module autoloading as described above.

modules_disabled

A toggle value indicating if modules are allowed to be loaded in an otherwise modular kernel. This toggle defaults to off (0), but can be set true (1). Once true, modules can be neither loaded nor unloaded, and the toggle cannot be set back to false. Generally used with the kexec_load_disabled toggle.

msgmax, msgmnb, and msgmni

`msgmax` is the maximum size of an IPC message, in bytes. 8192 by default (`MSGMAX`).

`msgmnb` is the maximum size of an IPC queue, in bytes. 16384 by default (`MSGMNB`).

`msgmni` is the maximum number of IPC queues. 32000 by default (`MSGMNI`).

msg_next_id, sem_next_id, and shm_next_id (System V IPC)

These three toggles allows to specify desired id for next allocated IPC object: message, semaphore or shared memory respectively.

By default they are equal to -1, which means generic allocation logic. Possible values to set are in range {0:`INT_MAX`}.

Notes:

1. kernel doesn't guarantee, that new object will have desired id. So, it's up to userspace, how to handle an object with "wrong" id.
2. Toggle with non-default value will be set back to -1 by kernel after successful IPC object allocation. If an IPC object allocation syscall fails, it is undefined if the value remains unmodified or is reset to -1.

ngroups_max

Maximum number of supplementary groups, _i.e._ the maximum size which `setgroups` will accept. Exports `NGROUPS_MAX` from the kernel.

nmi_watchdog

This parameter can be used to control the NMI watchdog (i.e. the hard lockup detector) on x86 systems.

0	Disable the hard lockup detector.
1	Enable the hard lockup detector.

The hard lockup detector monitors each CPU for its ability to respond to timer interrupts. The mechanism utilizes CPU performance counter registers that are programmed to generate Non-Maskable Interrupts (NMIs) periodically while a CPU is busy. Hence, the alternative name 'NMI watchdog'.

The NMI watchdog is disabled by default if the kernel is running as a guest in a KVM virtual machine. This default can be overridden by adding:

```
nmi_watchdog=1
```

to the guest kernel command line (see Documentation/admin-guide/kernel-parameters.rst).

numa_balancing

Enables/disables and configures automatic page fault based NUMA memory balancing. Memory is moved automatically to nodes that access it often. The value to set can be the result of ORing the following:

0	NUMA_BALANCING_DISABLED
1	NUMA_BALANCING_NORMAL
2	NUMA_BALANCING_MEMORY_TIERING

Or NUMA_BALANCING_NORMAL to optimize page placement among different NUMA nodes to reduce remote accessing. On NUMA machines, there is a performance penalty if remote memory is accessed by a CPU. When this feature is enabled the kernel samples what task thread is accessing memory by periodically unmapping pages and later trapping a page fault. At the time of the

page fault, it is determined if the data being accessed should be migrated to a local memory node.

The unmapping of pages and trapping faults incur additional overhead that ideally is offset by improved memory locality but there is no universal guarantee. If the target workload is already bound to NUMA nodes then this feature should be disabled.

Or NUMA_BALANCING_MEMORY_TIERING to optimize page placement among different types of memory (represented as different NUMA nodes) to place the hot pages in the fast memory. This is implemented based on unmapping and page fault too.

oops_all_cpu_backtrace

If this option is set, the kernel will send an NMI to all CPUs to dump their backtraces when an oops event occurs. It should be used as a last resort in case a panic cannot be triggered (to protect VMs running, for example) or kdump can't be collected. This file shows up if CONFIG_SMP is enabled.

0: Won't show all CPUs backtraces when an oops is detected. This is the default behavior.

1: Will non-maskably interrupt all CPUs and dump their backtraces when an oops event is detected.

osrelease, ostype & version

```
# cat osrelease
2.1.88
# cat ostype
Linux
# cat version
#5 Wed Feb 25 21:49:24 MET 1998
```

The files `osrelease` and `ostype` should be clear enough. `version` needs a little more clarification however. The '#5' means that this is the fifth kernel built from this source base and the date behind it indicates the time the kernel was built. The only way to tune these values is to rebuild the kernel :-)

overflowgid & overflowuid

if your architecture did not always support 32-bit UIDs (i.e. arm, i386, m68k, sh, and sparc32), a fixed UID and GID will be returned to applications that use the old 16-bit UID/GID system calls, if the actual UID or GID would exceed 65535.

These sysctls allow you to change the value of the fixed UID and GID. The default is 65534.

panic

The value in this file determines the behaviour of the kernel on a panic:

- if zero, the kernel will loop forever;
- if negative, the kernel will reboot immediately;
- if positive, the kernel will reboot after the corresponding number of seconds.

When you use the software watchdog, the recommended setting is 60.

panic_on_io_nmi

Controls the kernel's behavior when a CPU receives an NMI caused by an IO error.

0	Try to continue operation (default).
1	Panic immediately. The IO error triggered an NMI. This indicates a serious system condition which could result in IO data corruption. Rather than continuing, panicking might be a better choice. Some servers issue this sort of NMI when the dump button is pushed, and you can use this option to take a crash dump.

panic_on_oops

Controls the kernel's behaviour when an oops or BUG is encountered.

0	Try to continue operation.
1	Panic immediately. If the *panic* sysctl is also non-zero then the machine will be rebooted.

panic_on_stackoverflow

Controls the kernel's behavior when detecting the overflows of kernel, IRQ and exception stacks except a user stack. This file shows up if `CONFIG_DEBUG_STACKOVERFLOW` is enabled.

0	Try to continue operation.
1	Panic immediately.

panic_on_unrecovered_nmi

The default Linux behaviour on an NMI of either memory or unknown is to continue operation. For many environments such as scientific computing it is preferable that the box is taken out and the error dealt with than an uncorrected parity/ECC error get propagated.

A small number of systems do generate NMIs for bizarre random reasons such as power management so the default is off. That sysctl works like the existing panic controls already in that directory.

panic_on_warn

Calls panic() in the WARN() path when set to 1. This is useful to avoid a kernel rebuild when attempting to kdump at the location of a WARN().

0	Only WARN(), default behaviour.
1	Call panic() after printing out WARN() location.

panic_print

Bitmask for printing system info when panic happens. User can chose combination of the following bits:

bit 0	print all tasks info
bit 1	print system memory info
bit 2	print timer info
bit 3	print locks info if `CONFIG_LOCKDEP` is on
bit 4	print ftrace buffer
bit 5	print all printk messages in buffer
bit 6	print all CPUs backtrace (if available in the arch)

So for example to print tasks and memory info on panic, user can:

```
echo 3 > /proc/sys/kernel/panic_print
```

panic_on_rcu_stall

When set to 1, calls panic() after RCU stall detection messages. This is useful to define the root cause of RCU stalls using a vmcore.

0	Do not panic() when RCU stall takes place, default behavior.
1	panic() after printing RCU stall messages.

perf_cpu_time_max_percent

Hints to the kernel how much CPU time it should be allowed to use to handle perf sampling events. If the perf subsystem is informed that its samples are exceeding this limit, it will drop its sampling frequency to attempt to reduce its CPU usage.

Some perf sampling happens in NMIs. If these samples unexpectedly take too long to execute, the NMIs can become stacked up next to each other so much that nothing else is allowed to execute.

0	Disable the mechanism. Do not monitor or correct perf's sampling rate no matter how CPU time it takes.
1-100	Attempt to throttle perf's sample rate to this percentage of CPU. Note: the kernel calculates an "expected" length of each sample event. 100 here means 100% of that expected length. Even if this is set to 100, you may still see sample throttling if this length is exceeded. Set to 0 if you truly do not care how much CPU is consumed.

perf_event_paranoid

Controls use of the performance events system by unprivileged users (without CAP_PERFMON). The default value is 2.

For backward compatibility reasons access to system performance monitoring and observability remains open for CAP_SYS_ADMIN privileged processes but CAP_SYS_ADMIN usage for secure system performance monitoring and observability operations is discouraged with respect to CAP_PERFMON use cases.

-1	Allow use of (almost) all events by all users. Ignore mlock limit after perf_event_mlock_kb without `CAP_IPC_LOCK`.
>=0	Disallow ftrace function tracepoint by users without `CAP_PERFMON`. Disallow raw tracepoint access by users without `CAP_PERFMON`.
>=1	Disallow CPU event access by users without `CAP_PERFMON`.
>=2	Disallow kernel profiling by users without `CAP_PERFMON`.

perf_event_max_stack

Controls maximum number of stack frames to copy for (`attr.sample_type & PERF_SAMPLE_CALLCHAIN`) configured events, for instance, when using '`perf record -g`' or '`perf trace --call-graph fp`'.

This can only be done when no events are in use that have callchains enabled, otherwise writing to this file will return `-EBUSY`.

The default value is 127.

perf_event_mlock_kb

Control size of per-cpu ring buffer not counted against mlock limit.

The default value is 512 + 1 page

perf_event_max_contexts_per_stack

Controls maximum number of stack frame context entries for (`attr.sample_type & PERF_SAMPLE_CALLCHAIN`) configured events, for instance, when using '`perf record -g`' or '`perf trace --call-graph fp`'.

This can only be done when no events are in use that have callchains enabled, otherwise writing to this file will return `-EBUSY`.

The default value is 8.

perf_user_access (arm64 only)

Controls user space access for reading perf event counters. When set to 1, user space can read performance monitor counter registers directly.

The default value is 0 (access disabled).

See Documentation/arm64/perf.rst for more information.

pid_max

PID allocation wrap value. When the kernel's next PID value reaches this value, it wraps back to a minimum PID value. PIDs of value `pid_max` or larger are not allocated.

ns_last_pid

The last pid allocated in the current (the one task using this sysctl lives in) pid namespace. When selecting a pid for a next task on fork kernel tries to allocate a number starting from this one.

powersave-nap (PPC only)

If set, Linux-PPC will use the 'nap' mode of powersaving, otherwise the 'doze' mode will be used.

---

printk

The four values in printk denote: `console_loglevel`, `default_message_loglevel`, `minimum_console_loglevel` and `default_console_loglevel` respectively.

These values influence printk() behavior when printing or logging error messages. See '`man 2 syslog`' for more info on the different loglevels.

console_loglevel	messages with a higher priority than this will be printed to the console
default_message_loglevel	messages without an explicit priority will be printed with this priority
minimum_console_loglevel	minimum (highest) value to which console_loglevel can be set
default_console_loglevel	default value for console_loglevel

printk_delay

Delay each printk message in `printk_delay` milliseconds

Value from 0 - 10000 is allowed.

printk_ratelimit

Some warning messages are rate limited. `printk_ratelimit` specifies the minimum length of time between these messages (in seconds). The default value is 5 seconds.

A value of 0 will disable rate limiting.

printk_ratelimit_burst

While long term we enforce one message per printk_ratelimit seconds, we do allow a burst of messages to pass through. `printk_ratelimit_burst` specifies the number of messages we can send before ratelimiting kicks in.

The default value is 10 messages.

printk_devkmsg

Control the logging to `/dev/kmsg` from userspace:

ratelimit	default, ratelimited
on	unlimited logging to /dev/kmsg from userspace
off	logging to /dev/kmsg disabled

The kernel command line parameter `printk.devkmsg=` overrides this and is a one-time setting until next reboot: once set, it cannot be changed by this sysctl interface anymore.

---

pty

See Documentation/filesystems/devpts.rst.

random

This is a directory, with the following entries:

- `boot_id`: a UUID generated the first time this is retrieved, and unvarying after that;
- `entropy_avail`: the pool's entropy count, in bits;
- `poolsize`: the entropy pool size, in bits;
- `urandom_min_reseed_secs`: obsolete (used to determine the minimum number of seconds between urandom pool reseeding). This file is writable for compatibility purposes, but writing to it has no effect on any RNG behavior.
- `uuid`: a UUID generated every time this is retrieved (this can thus be used to generate UUIDs at will);
- `write_wakeup_threshold`: when the entropy count drops below this (as a number of bits), processes waiting to write to `/dev/random` are woken up. This file is writable for compatibility purposes, but writing to it has no effect on any RNG behavior.

randomize_va_space

This option can be used to select the type of process address space randomization that is used in the system, for architectures that support this feature.

0	Turn the process address space randomization off. This is the default for architectures that do not support this feature anyways, and kernels that are booted with the "norandmaps" parameter.
1	Make the addresses of mmap base, stack and VDSO page randomized. This, among other things, implies that shared libraries will be loaded to random addresses. Also for PIE-linked binaries, the location of code start is randomized. This is the default if the `CONFIG_COMPAT_BRK` option is enabled.
2	Additionally enable heap randomization. This is the default if `CONFIG_COMPAT_BRK` is disabled. There are a few legacy applications out there (such as some ancient versions of libc.so.5 from 1996) that assume that brk area starts just after the end of the code+bss. These applications break when start of the brk area is randomized. There are however no known non-legacy applications that would be broken this way, so for most systems it is safe to choose full randomization. Systems with ancient and/or broken binaries should be configured with `CONFIG_COMPAT_BRK` enabled, which excludes the heap from process address space randomization.

real-root-dev

See Documentation/admin-guide/initrd.rst.

reboot-cmd (SPARC only)

??? This seems to be a way to give an argument to the Sparc ROM/Flash boot loader. Maybe to tell it what to do after rebooting.

???

sched_energy_aware

Enables/disables Energy Aware Scheduling (EAS). EAS starts automatically on platforms where it can run (that is, platforms with asymmetric CPU topologies and having an Energy Model available). If your platform happens to meet the requirements for EAS but you do not want to use it, change this value to 0.

task_delayacct

Enables/disables task delay accounting (see Documentation/accounting/delay-accounting.rst. Enabling this feature incurs a small amount of overhead in the scheduler but is useful for debugging and performance tuning. It is required by some tools such as iotop.

sched_schedstats

Enables/disables scheduler statistics. Enabling this feature incurs a small amount of overhead in the scheduler but is useful for debugging and performance tuning.

sched_util_clamp_min

Max allowed *minimum* utilization.

Default value is 1024, which is the maximum possible value.

It means that any requested uclamp.min value cannot be greater than sched_util_clamp_min, i.e., it is restricted to the range [0:sched_util_clamp_min].

sched_util_clamp_max

Max allowed *maximum* utilization.

Default value is 1024, which is the maximum possible value.

It means that any requested uclamp.max value cannot be greater than sched_util_clamp_max, i.e., it is restricted to the range [0:sched_util_clamp_max].

sched_util_clamp_min_rt_default

By default Linux is tuned for performance. Which means that RT tasks always run at the highest frequency and most capable (highest capacity) CPU (in heterogeneous systems).

Uclamp achieves this by setting the requested uclamp.min of all RT tasks to 1024 by default, which effectively boosts the tasks to run at the highest frequency and biases them to run on the biggest CPU.

This knob allows admins to change the default behavior when uclamp is being used. In battery powered devices particularly, running at the maximum capacity and frequency will increase energy consumption and shorten the battery life.

This knob is only effective for RT tasks which the user hasn't modified their requested uclamp.min value via sched_setattr() syscall.

This knob will not escape the range constraint imposed by sched_util_clamp_min defined above.

For example if

    sched_util_clamp_min_rt_default = 800 sched_util_clamp_min = 600

Then the boost will be clamped to 600 because 800 is outside of the permissible range of [0:600]. This could happen for instance if a powersave mode will restrict all boosts temporarily by modifying sched_util_clamp_min. As soon as this restriction is lifted, the requested sched_util_clamp_min_rt_default will take effect.

seccomp

See Documentation/userspace-api/seccomp_filter.rst.

sg-big-buff

This file shows the size of the generic SCSI (sg) buffer. You can't tune it just yet, but you could change it on compile time by editing `include/scsi/sg.h` and changing the value of `SG_BIG_BUFF`.

There shouldn't be any reason to change this value. If you can come up with one, you probably know what you are doing anyway :)

shmall

shmall

This parameter sets the total amount of shared memory pages that can be used system wide. Hence, `shmall` should always be at least `ceil(shmmax/PAGE_SIZE)`.

If you are not sure what the default `PAGE_SIZE` is on your Linux system, you can run the following command:

```
# getconf PAGE_SIZE
```

shmmax

This value can be used to query and set the run time limit on the maximum shared memory segment size that can be created. Shared memory segments up to 1Gb are now supported in the kernel. This value defaults to `SHMMAX`.

shmmni

This value determines the maximum number of shared memory segments. 4096 by default (`SHMMNI`).

shm_rmid_forced

Linux lets you set resource limits, including how much memory one process can consume, via `setrlimit(2)`. Unfortunately, shared memory segments are allowed to exist without association with any process, and thus might not be counted against any resource limits. If enabled, shared memory segments are automatically destroyed when their attach count becomes zero after a detach or a process termination. It will also destroy segments that were created, but never attached to, on exit from the process. The only use left for `IPC_RMID` is to immediately destroy an unattached segment. Of course, this breaks the way things are defined, so some applications might stop working. Note that this feature will do you no good unless you also configure your resource limits (in particular, `RLIMIT_AS` and `RLIMIT_NPROC`). Most systems don't need this.

Note that if you change this from 0 to 1, already created segments without users and with a dead originative process will be destroyed.

sysctl_writes_strict

Control how file position affects the behavior of updating sysctl values via the `/proc/sys` interface:

-1	Legacy per-write sysctl value handling, with no printk warnings. Each write syscall must fully contain the sysctl value to be written, and multiple writes on the same sysctl file descriptor will rewrite the sysctl value, regardless of file position.
0	Same behavior as above, but warn about processes that perform writes to a sysctl file descriptor when the file position is not 0.
1	(default) Respect file position when writing sysctl strings. Multiple writes will append to the sysctl value buffer. Anything past the max length of the sysctl value buffer will be ignored. Writes to numeric sysctl entries must always be at file position 0 and the value must be fully contained in the buffer sent in the write syscall.

softlockup_all_cpu_backtrace

This value controls the soft lockup detector thread's behavior when a soft lockup condition is detected as to whether or not to gather further debug information. If enabled, each cpu will be issued an NMI and instructed to capture stack trace.

This feature is only applicable for architectures which support NMI.

0	Do nothing. This is the default behavior.
1	On detection capture more debug information.

softlockup_panic

This parameter can be used to control whether the kernel panics when a soft lockup is detected.

0	Don't panic on soft lockup.
1	Panic on soft lockup.

This can also be set using the softlockup_panic kernel parameter.

soft_watchdog

This parameter can be used to control the soft lockup detector.

0	Disable the soft lockup detector.
1	Enable the soft lockup detector.

The soft lockup detector monitors CPUs for threads that are hogging the CPUs without rescheduling voluntarily, and thus prevent the 'migration/N' threads from running, causing the watchdog work fail to execute. The mechanism depends on the CPUs ability to respond to timer interrupts which are needed for the watchdog work to be queued by the watchdog timer function, otherwise the NMI watchdog â€" if enabled â€" can detect a hard lockup condition.

stack_erasing

This parameter can be used to control kernel stack erasing at the end of syscalls for kernels built with `CONFIG_GCC_PLUGIN_STACKLEAK`.

That erasing reduces the information which kernel stack leak bugs can reveal and blocks some uninitialized stack variable attacks. The tradeoff is the performance impact: on a single CPU system kernel compilation sees a 1% slowdown, other systems and workloads may vary.

0	Kernel stack erasing is disabled, STACKLEAK_METRICS are not updated.
1	Kernel stack erasing is enabled (default), it is performed before returning to the userspace at the end of syscalls.

stop-a (SPARC only)

Controls Stop-A:

0	Stop-A has no effect.
1	Stop-A breaks to the PROM (default).

Stop-A is always enabled on a panic, so that the user can return to the boot PROM.

sysrq

See Documentation/admin-guide/sysrq.rst.

tainted

Non-zero if the kernel has been tainted. Numeric values, which can be ORed together. The letters are seen in "Tainted" line of Oops reports.

1	*(P)*	proprietary module was loaded
2	*(F)*	module was force loaded
4	*(S)*	kernel running on an out of specification system
8	*(R)*	module was force unloaded
16	*(M)*	processor reported a Machine Check Exception (MCE)
32	*(B)*	bad page referenced or some unexpected page flags
64	*(U)*	taint requested by userspace application
128	*(D)*	kernel died recently, i.e. there was an OOPS or BUG
256	*(A)*	an ACPI table was overridden by user
512	*(W)*	kernel issued warning
1024	*(C)*	staging driver was loaded
2048	*(I)*	workaround for bug in platform firmware applied
4096	*(O)*	externally-built ("out-of-tree") module was loaded
8192	*(E)*	unsigned module was loaded
16384	*(L)*	soft lockup occurred
32768	*(K)*	kernel has been live patched
65536	*(X)*	Auxiliary taint, defined and used by for distros
131072	*(T)*	The kernel was built with the struct randomization plugin

See Documentation/admin-guide/tainted-kernels.rst for more information.

Note:

> writes to this sysctl interface will fail with `EINVAL` if the kernel is booted with the command line option `panic_on_taint=<bitmask>,nousertaint` and any of the ORed together values being written to `tainted` match with the bitmask declared on panic_on_taint. See Documentation/admin-guide/kernel-parameters.rst for more details on that particular kernel command line option and its optional `nousertaint` switch.

threads-max

This value controls the maximum number of threads that can be created using `fork()`.

During initialization the kernel sets this value such that even if the maximum number of threads is created, the thread structures occupy only a part (1/8th) of the available RAM pages.

The minimum value that can be written to `threads-max` is 1.

The maximum value that can be written to `threads-max` is given by the constant `FUTEX_TID_MASK` (0x3fffffff).

If a value outside of this range is written to `threads-max` an `EINVAL` error occurs.

traceoff_on_warning

When set, disables tracing (see Documentation/trace/ftrace.rst) when a `WARN()` is hit.

tracepoint_printk

When tracepoints are sent to printk() (enabled by the `tp_printk` boot parameter), this entry provides runtime control:

```
echo 0 > /proc/sys/kernel/tracepoint_printk
```

will stop tracepoints from being sent to printk(), and:

```
echo 1 > /proc/sys/kernel/tracepoint_printk
```

will send them to printk() again.

This only works if the kernel was booted with `tp_printk` enabled.

See Documentation/admin-guide/kernel-parameters.rst and Documentation/trace/boottime-trace.rst.

unaligned-dump-stack (ia64)

When logging unaligned accesses, controls whether the stack is dumped.

0	Do not dump the stack. This is the default setting.
1	Dump the stack.

See also ignore-unaligned-usertrap.

unaligned-trap

On architectures where unaligned accesses cause traps, and where this feature is supported (`CONFIG_SYSCTL_ARCH_UNALIGN_ALLOW`; currently, `arc` and `parisc`), controls whether unaligned traps are caught and emulated (instead of failing).

0	Do not emulate unaligned accesses.
1	Emulate unaligned accesses. This is the default setting.

See also ignore-unaligned-usertrap.

unknown_nmi_panic

The value in this file affects behavior of handling NMI. When the value is non-zero, unknown NMI is trapped and then panic occurs. At that time, kernel debugging information is displayed on console.

NMI switch that most IA32 servers have fires unknown NMI up, for example. If a system hangs up, try pressing the NMI switch.

unprivileged_bpf_disabled

Writing 1 to this entry will disable unprivileged calls to `bpf()`; once disabled, calling `bpf()` without `CAP_SYS_ADMIN` or `CAP_BPF` will return `-EPERM`. Once set to 1, this can't be cleared from the running kernel anymore.

Writing 2 to this entry will also disable unprivileged calls to `bpf()`, however, an admin can still change this setting later on, if needed, by writing 0 or 1 to this entry.

If `BPF_UNPRIV_DEFAULT_OFF` is enabled in the kernel config, then this entry will default to 2 instead of 0.

0	Unprivileged calls to `bpf()` are enabled
1	Unprivileged calls to `bpf()` are disabled without recovery
2	Unprivileged calls to `bpf()` are disabled

watchdog

This parameter can be used to disable or enable the soft lockup detector *and* the NMI watchdog (i.e. the hard lockup detector) at the same time.

0	Disable both lockup detectors.
1	Enable both lockup detectors.

The soft lockup detector and the NMI watchdog can also be disabled or enabled individually, using the `soft_watchdog` and `nmi_watchdog` parameters. If the `watchdog` parameter is read, for example by executing:

```
cat /proc/sys/kernel/watchdog
```

the output of this command (0 or 1) shows the logical OR of `soft_watchdog` and `nmi_watchdog`.

watchdog_cpumask

This value can be used to control on which cpus the watchdog may run. The default cpumask is all possible cores, but if `NO_HZ_FULL` is enabled in the kernel config, and cores are specified with the `nohz_full=` boot argument, those cores are excluded by default. Offline cores can be included in this mask, and if the core is later brought online, the watchdog will be started based on the mask value.

Typically this value would only be touched in the `nohz_full` case to re-enable cores that by default were not running the watchdog, if a kernel lockup was suspected on those cores.

The argument value is the standard cpulist format for cpumasks, so for example to enable the watchdog on cores 0, 2, 3, and 4 you might say:

```
echo 0,2-4 > /proc/sys/kernel/watchdog_cpumask
```

watchdog_thresh

This value can be used to control the frequency of hrtimer and NMI events and the soft and hard lockup thresholds. The default threshold is 10 seconds.

The softlockup threshold is (`2 * watchdog_thresh`). Setting this tunable to zero will disable lockup detection altogether.