trusty (5) proc.5.gz
Provided by: manpages_3.54-1ubuntu1_all

NAME

proc - process information pseudo-filesystem

DESCRIPTION

The proc filesystem is a pseudo-filesystem which provides an interface to kernel data structures. It is commonly mounted at /proc. Most of it is read-only, but some files allow kernel variables to be changed.

The following list describes many of the files and directories under the /proc hierarchy.

/proc/[pid]
There is a numerical subdirectory for each running process; the subdirectory is named by the process ID. Each such subdirectory contains the following pseudo-files and directories.

/proc/[pid]/auxv (since 2.6.0-test7)
This contains the contents of the ELF interpreter information passed to the process at exec time. The format is one unsigned long ID plus one unsigned long value for each entry. The last entry contains two zeros.

/proc/[pid]/cgroup (since Linux 2.6.24)
This file describes control groups to which the process/task belongs. For each cgroup hierarchy there is one entry containing colon-separated fields of the form:

```
5:cpuacct,cpu,cpuset:/daemons
```

The colon-separated fields are, from left to right:

1. hierarchy ID number
2. set of subsystems bound to the hierarchy
3. control group in the hierarchy to which the process belongs

This file is present only if the **CONFIG_CGROUPS** kernel configuration option is enabled.

/proc/[pid]/cmdline
This holds the complete command line for the process, unless the process is a zombie. In the latter case, there is nothing in this file: that is, a read on this file will return 0 characters. The command-line arguments appear in this file as a set of strings separated by null bytes ('\0'), with a further null byte after the last string.

/proc/[pid]/coredump_filter (since kernel 2.6.23)
See **core**(5).

/proc/[pid]/cpuset (since kernel 2.6.12)
See **cpuset**(7).

/proc/[pid]/cwd
This is a symbolic link to the current working directory of the process. To find out the current working directory of process 20, for instance, you can do this:

```
$ cd /proc/20/cwd; /bin/pwd
```

Note that the pwd command is often a shell built-in, and might

not work properly. In **bash**(1), you may use pwd -P.

In a multithreaded process, the contents of this symbolic link are not available if the main thread has already terminated (typically by calling **pthread_exit**(3)).

/proc/[pid]/environ
This file contains the environment for the process. The entries are separated by null bytes ('\0'), and there may be a null byte at the end. Thus, to print out the environment of process 1, you would do:

```
$ strings /proc/1/environ
```

/proc/[pid]/exe
Under Linux 2.2 and later, this file is a symbolic link containing the actual pathname of the executed command. This symbolic link can be dereferenced normally; attempting to open it will open the executable. You can even type /proc/[pid]/exe to run another copy of the same executable as is being run by process [pid]. In a multithreaded process, the contents of this symbolic link are not available if the main thread has already terminated (typically by calling **pthread_exit**(3)).

Under Linux 2.0 and earlier /proc/[pid]/exe is a pointer to the binary which was executed, and appears as a symbolic link. A **readlink**(2) call on this file under Linux 2.0 returns a string in the format:

```
[device]:inode
```

For example, [0301]:1502 would be inode 1502 on device major 03 (IDE, MFM, etc. drives) minor 01 (first partition on the first drive).

find(1) with the -inum option can be used to locate the file.

/proc/[pid]/fd/
This is a subdirectory containing one entry for each file which the process has open, named by its file descriptor, and which is a symbolic link to the actual file. Thus, 0 is standard input, 1 standard output, 2 standard error, etc.

For file descriptors for pipes and sockets, the entries will be symbolic links whose content is the file type with the inode. A **readlink**(2) call on this file returns a string in the format:

```
type:[inode]
```

For example, socket:[2248868] will be a socket and its inode is 2248868. For sockets, that inode can be used to find more information in one of the files under /proc/net/.

For file descriptors that have no corresponding inode (e.g., file descriptors produced by **epoll_create**(2), **eventfd**(2), **inotify_init**(2), **signalfd**(2), and **timerfd**(2)), the entry will be a symbolic link with contents of the form

```
anon_inode:<file-type>
```

In some cases, the file-type is surrounded by square brackets.

For example, an epoll file descriptor will have a symbolic link whose content is the string anon_inode:[eventpoll].

In a multithreaded process, the contents of this directory are not available if the main thread has already terminated (typically by calling **pthread_exit**(3)).

Programs that will take a filename as a command-line argument, but will not take input from standard input if no argument is supplied, or that write to a file named as a command-line argument, but will not send their output to standard output if no argument is supplied, can nevertheless be made to use standard input or standard out using /proc/[pid]/fd. For example, assuming that -i is the flag designating an input file and -o is the flag designating an output file:

```
$ foobar -i /proc/self/fd/0 -o /proc/self/fd/1 ...
```

and you have a working filter.

/proc/self/fd/N is approximately the same as /dev/fd/N in some UNIX and UNIX-like systems. Most Linux MAKEDEV scripts symbolically link /dev/fd to /proc/self/fd, in fact.

Most systems provide symbolic links /dev/stdin, /dev/stdout, and /dev/stderr, which respectively link to the files 0, 1, and 2 in /proc/self/fd. Thus the example command above could be written as:

```
$ foobar -i /dev/stdin -o /dev/stdout ...
```

/proc/[pid]/fdinfo/ (since kernel 2.6.22)
This is a subdirectory containing one entry for each file which the process has open, named by its file descriptor. The contents of each file can be read to obtain information about the corresponding file descriptor, for example:

```
$ cat /proc/12015/fdinfo/4
pos:    1000
flags:  01002002
```

The pos field is a decimal number showing the current file offset. The flags field is an octal number that displays the file access mode and file status flags (see **open**(2)).

The files in this directory are readable only by the owner of the process.

/proc/[pid]/io (since kernel 2.6.20)
This file contains I/O statistics for the process, for example:

```
# cat /proc/3828/io
rchar: 323934931
wchar: 323929600
syscr: 632687
syscw: 632675
read_bytes: 0
write_bytes: 323932160
cancelled_write_bytes: 0
```

The fields are as follows:

rchar: characters read
The number of bytes which this task has caused to be read from storage. This is simply the sum of bytes which this process passed to **read**(2) and similar system calls. It includes things such as terminal I/O and is unaffected by whether or not actual physical disk I/O was required (the read might have been satisfied from pagecache).

wchar: characters written
The number of bytes which this task has caused, or shall cause to be written to disk. Similar caveats apply here as with rchar.

syscr: read syscalls
Attempt to count the number of read I/O operations—that is, system calls such as **read**(2) and **pread**(2).

syscw: write syscalls
Attempt to count the number of write I/O operations—that is, system calls such as **write**(2) and **pwrite**(2).

read_bytes: bytes read
Attempt to count the number of bytes which this process really did cause to be fetched from the storage layer. This is accurate for block-backed filesystems.

write_bytes: bytes written
Attempt to count the number of bytes which this process caused to be sent to the storage layer.

cancelled_write_bytes:
The big inaccuracy here is truncate. If a process writes 1MB to a file and then deletes the file, it will in fact perform no writeout. But it will have been accounted as having caused 1MB of write. In other words: this field represents the number of bytes which this process caused to not happen, by truncating pagecache. A task can cause "negative" I/O too. If this task truncates some dirty

pagecache, some I/O which another task has been accounted for (in its write_bytes) will not be happening.

Note: In the current implementation, things are a bit racy on 32-bit systems: if process A reads process B's /proc/[pid]/io while process B is updating one of these 64-bit counters, process A could see an intermediate result.

/proc/[pid]/limits (since kernel 2.6.24)
This file displays the soft limit, hard limit, and units of measurement for each of the process's resource limits (see **getrlimit**(2)). Up to and including Linux 2.6.35, this file is protected to allow reading only by the real UID of the process. Since Linux 2.6.36, this file is readable by all users on the system.

/proc/[pid]/map_files/ (since kernel 3.3)
This subdirectory contains entries corresponding to memory-mapped files (see **mmap**(2)). Entries are named by memory region start and end address pair (expressed as hexadecimal numbers), and are symbolic links to the mapped files themselves. Here is an example, with the output wrapped and reformatted to fit on an 80-column display:

```
$ ls -l /proc/self/map_files/
lr--------. 1 root root 64 Apr 16 21:31
            3252e00000-3252e20000 -> /usr/lib64/ld-2.15.so
...
```

Although these entries are present for memory regions that were mapped with the **MAP_FILE** flag, the way anonymous shared memory (regions created with the **MAP_ANON | MAP_SHARED** flags) is implemented in Linux means that such regions also appear on this directory. Here is an example where the target file is the deleted /dev/zero one:

```
lrw-------. 1 root root 64 Apr 16 21:33
            7fc075d2f000-7fc075e6f000 -> /dev/zero (deleted)
```

This directory appears only if the **CONFIG_CHECKPOINT_RESTORE** kernel configuration option is enabled.

/proc/[pid]/maps
A file containing the currently mapped memory regions and their access permissions. See **mmap**(2) for some further information about memory mappings.

The format of the file is:

```
address           perms offset  dev   inode       pathname
00400000-00452000 r-xp 00000000 08:02 173521      /usr/bin/dbus-daemon
00651000-00652000 r--p 00051000 08:02 173521      /usr/bin/dbus-daemon
00652000-00655000 rw-p 00052000 08:02 173521      /usr/bin/dbus-daemon
00e03000-00e24000 rw-p 00000000 00:00 0           [heap]
00e24000-011f7000 rw-p 00000000 00:00 0           [heap]
...
35b1800000-35b1820000 r-xp 00000000 08:02 135522  /usr/lib64/ld-2.15.so
35b1a1f000-35b1a20000 r--p 0001f000 08:02 135522  /usr/lib64/ld-2.15.so
35b1a20000-35b1a21000 rw-p 00020000 08:02 135522  /usr/lib64/ld-2.15.so
35b1a21000-35b1a22000 rw-p 00000000 00:00 0
35b1c00000-35b1dac000 r-xp 00000000 08:02 135870  /usr/lib64/libc-2.15.so
35b1dac000-35b1fac000 ---p 001ac000 08:02 135870  /usr/lib64/libc-2.15.so
35b1fac000-35b1fb0000 r--p 001ac000 08:02 135870  /usr/lib64/libc-2.15.so
35b1fb0000-35b1fb2000 rw-p 001b0000 08:02 135870  /usr/lib64/libc-2.15.so
...
f2c6ff8c000-7f2c7078c000 rw-p 00000000 00:00 0    [stack:986]
...
7fffb2c0d000-7fffb2c2e000 rw-p 00000000 00:00 0   [stack]
7fffb2d48000-7fffb2d49000 r-xp 00000000 00:00 0   [vdso]
```

The address field is the address space in the process that the mapping occupies. The perms field is a set of permissions:

```
r = read
w = write
x = execute
s = shared
p = private (copy on write)
```

The offset field is the offset into the file/whatever; dev is the device (major:minor); inode is the inode on that device. 0

indicates that no inode is associated with the memory region, as would be the case with BSS (uninitialized data).

The pathname field will usually be the file that is backing the mapping. For ELF files, you can easily coordinate with the offset field by looking at the Offset field in the ELF program headers (readelf -l).

There are additional helpful pseudo-paths:

[stack]
The initial process's (also known as the main thread's) stack.

[stack:<tid>] (since Linux 3.4)
A thread's stack (where the <tid> is a thread ID). It corresponds to the /proc/[pid]/task/[tid]/ path.

[vdso] The virtual dynamically linked shared object.

[heap] The process's heap.

If the pathname field is blank, this is an anonymous mapping as obtained via the **mmap**(2) function. There is no easy way to coordinate this back to a process's source, short of running it through **gdb**(1), **strace**(1), or similar.

Under Linux 2.0 there is no field giving pathname.

/proc/[pid]/mem
This file can be used to access the pages of a process's memory through **open**(2), **read**(2), and **lseek**(2).

/proc/[pid]/mountinfo (since Linux 2.6.26)
This file contains information about mount points. It contains lines of the form:

```
36 35 98:0 /mnt1 /mnt2 rw,noatime master:1 - ext3 /dev/root rw,errors=continue
(1)(2)(3)   (4)   (5)      (6)      (7)   (8) (9)   (10)         (11)
```

The numbers in parentheses are labels for the descriptions below:

(1) mount ID: unique identifier of the mount (may be reused after **umount**(2)).

(2) parent ID: ID of parent mount (or of self for the top of the mount tree).

(3) major:minor: value of st_dev for files on filesystem (see **stat**(2)).

(4) root: root of the mount within the filesystem.

(5) mount point: mount point relative to the process's root.

(6) mount options: per-mount options.

(7) optional fields: zero or more fields of the form "tag[:value]".

(8) separator: marks the end of the optional fields.

(9) filesystem type: name of filesystem in the form "type[.subtype]".

(10) mount source: filesystem-specific information or "none".

(11) super options: per-super block options.

Parsers should ignore all unrecognized optional fields. Currently the possible optional fields are:

shared:X	mount is shared in peer group X
master:X	mount is slave to peer group X
propagate_from:X	mount is slave and receives propagation from peer group X (*)
unbindable	mount is unbindable

(*) X is the closest dominant peer group under the process's root. If X is the immediate master of the mount, or if there is no dominant peer group under the same root, then only the "master:X" field is present and not the "propagate_from:X" field.

For more information on mount propagation see: Documentation/filesystems/sharedsubtree.txt in the Linux kernel source tree.

/proc/[pid]/mounts (since Linux 2.4.19)
This is a list of all the filesystems currently mounted in the process's mount namespace. The format of this file is documented in **fstab**(5). Since kernel version 2.6.15, this file is pollable: after opening the file for reading, a change in this file (i.e., a filesystem mount or unmount) causes **select**(2) to mark the file descriptor as readable, and **poll**(2) and **epoll_wait**(2) mark the file as having an error condition.

/proc/[pid]/mountstats (since Linux 2.6.17)
This file exports information (statistics, configuration information) about the mount points in the process's name space. Lines in this file have the form:

```
device /dev/sda7 mounted on /home with fstype ext3 [statistics]
(       1      )            ( 2 )             (3 ) (4)
```

The fields in each line are:

(1) The name of the mounted device (or "nodevice" if there is no corresponding device).

(2) The mount point within the filesystem tree.

(3) The filesystem type.

(4) Optional statistics and configuration information. Currently (as at Linux 2.6.26), only NFS filesystems export information via this field.

This file is readable only by the owner of the process.

/proc/[pid]/ns/ (since Linux 3.0)
This is a subdirectory containing one entry for each namespace that supports being manipulated by **setns**(2). For information about namespaces, see **clone**(2).

/proc/[pid]/ns/ipc (since Linux 3.0)
Bind mounting this file (see **mount**(2)) to somewhere else in the filesystem keeps the IPC namespace of the process specified by pid alive even if all processes currently in the namespace terminate.

Opening this file returns a file handle for the IPC namespace of the process specified by pid. As long as this file descriptor remains open, the IPC namespace will remain alive, even if all processes in the namespace terminate. The file descriptor can be passed to **setns**(2).

/proc/[pid]/ns/net (since Linux 3.0)
Bind mounting this file (see **mount**(2)) to somewhere else in the filesystem keeps the network namespace of the process specified by pid alive even if all processes in the namespace terminate.

Opening this file returns a file handle for the network namespace of the process specified by pid. As long as this file descriptor remains open, the network namespace will remain alive, even if all processes in the namespace terminate. The file descriptor can be passed to **setns**(2).

/proc/[pid]/ns/uts (since Linux 3.0)
Bind mounting this file (see **mount**(2)) to somewhere else in the filesystem keeps the UTS namespace of the process specified by pid alive even if all processes currently in the namespace terminate.

Opening this file returns a file handle for the UTS namespace of the process specified by pid. As long as this file descriptor remains open, the UTS namespace will remain alive, even if all processes in the namespace terminate. The file descriptor can

be passed to **setns**(2).

/proc/[pid]/numa_maps (since Linux 2.6.14)
See **numa**(7).

/proc/[pid]/oom_adj (since Linux 2.6.11)
This file can be used to adjust the score used to select which process should be killed in an out-of-memory (OOM) situation. The kernel uses this value for a bit-shift operation of the process's oom_score value: valid values are in the range -16 to +15, plus the special value -17, which disables OOM-killing altogether for this process. A positive score increases the likelihood of this process being killed by the OOM-killer; a negative score decreases the likelihood.

The default value for this file is 0; a new process inherits its parent's oom_adj setting. A process must be privileged (**CAP_SYS_RESOURCE**) to update this file.

Since Linux 2.6.36, use of this file is deprecated in favor of /proc/[pid]/oom_score_adj.

/proc/[pid]/oom_score (since Linux 2.6.11)
This file displays the current score that the kernel gives to this process for the purpose of selecting a process for the OOM-killer. A higher score means that the process is more likely to be selected by the OOM-killer. The basis for this score is the amount of memory used by the process, with increases (+) or decreases (-) for factors including:

* whether the process creates a lot of children using **fork**(2) (+);

* whether the process has been running a long time, or has used a lot of CPU time (-);

* whether the process has a low nice value (i.e., > 0) (+);

* whether the process is privileged (-); and

* whether the process is making direct hardware access (-).

The oom_score also reflects the adjustment specified by the oom_score_adj or oom_adj setting for the process.

/proc/[pid]/oom_score_adj (since Linux 2.6.36)
This file can be used to adjust the badness heuristic used to select which process gets killed in out-of-memory conditions.

The badness heuristic assigns a value to each candidate task ranging from 0 (never kill) to 1000 (always kill) to determine which process is targeted. The units are roughly a proportion along that range of allowed memory the process may allocate from, based on an estimation of its current memory and swap use. For example, if a task is using all allowed memory, its badness score will be 1000. If it is using half of its allowed memory, its score will be 500.

There is an additional factor included in the badness score: root processes are given 3% extra memory over other tasks.

The amount of "allowed" memory depends on the context in which the OOM-killer was called. If it is due to the memory assigned to the allocating task's cpuset being exhausted, the allowed memory represents the set of mems assigned to that cpuset (see **cpuset**(7)). If it is due to a mempolicy's node(s) being exhausted, the allowed memory represents the set of mempolicy nodes. If it is due to a memory limit (or swap limit) being reached, the allowed memory is that configured limit. Finally, if it is due to the entire system being out of memory, the allowed memory represents all allocatable resources.

The value of oom_score_adj is added to the badness score before it is used to determine which task to kill. Acceptable values range from -1000 (OOM_SCORE_ADJ_MIN) to +1000 (OOM_SCORE_ADJ_MAX). This allows user space to control the preference for OOM-killing, ranging from always preferring a certain task or completely disabling it from OOM-killing. The lowest possible value, -1000, is equivalent to disabling OOM-killing entirely for that task, since it will always report a badness score of 0.

Consequently, it is very simple for user space to define the amount of memory to consider for each task. Setting a oom_score_adj value of +500, for example, is roughly equivalent to allowing the remainder of tasks sharing the same system, cpuset, mempolicy, or memory controller resources to use at least 50% more memory. A value of -500, on the other hand, would be roughly equivalent to discounting 50% of the task's allowed memory from being considered as scoring against the task.

For backward compatibility with previous kernels, /proc/[pid]/oom_adj can still be used to tune the badness score. Its value is scaled linearly with oom_score_adj.

Writing to /proc/[pid]/oom_score_adj or /proc/[pid]/oom_adj will change the other with its scaled value.

/proc/[pid]/root

UNIX and Linux support the idea of a per-process root of the filesystem, set by the **chroot**(2) system call. This file is a symbolic link that points to the process's root directory, and behaves as exe, fd/*, etc. do.

In a multithreaded process, the contents of this symbolic link are not available if the main thread has already terminated (typically by calling **pthread_exit**(3)).

/proc/[pid]/smaps (since Linux 2.6.14)

This file shows memory consumption for each of the process's mappings. For each of mappings there is a series of lines such as the following:

```
08048000-080bc000 r-xp 00000000 03:02 13130      /bin/bash
Size:                464 kB
Rss:                 424 kB
Shared_Clean:        424 kB
Shared_Dirty:          0 kB
Private_Clean:         0 kB
Private_Dirty:         0 kB
```

The first of these lines shows the same information as is displayed for the mapping in /proc/[pid]/maps. The remaining lines show the size of the mapping, the amount of the mapping that is currently resident in RAM, the number of clean and dirty shared pages in the mapping, and the number of clean and dirty private pages in the mapping.

This file is present only if the **CONFIG_MMU** kernel configuration option is enabled.

/proc/[pid]/stat

Status information about the process. This is used by **ps**(1). It is defined in /usr/src/linux/fs/proc/array.c.

The fields, in order, with their proper **scanf**(3) format specifiers, are:

pid %d (1) The process ID.

comm %s (2) The filename of the executable, in parentheses. This is visible whether or not the executable is swapped out.

state %c (3) One character from the string "RSDZTW" where R is running, S is sleeping in an interruptible wait, D is waiting in uninterruptible disk sleep, Z is zombie, T is traced or stopped (on a signal), and W is paging.

ppid %d (4) The PID of the parent.

pgrp %d (5) The process group ID of the process.

session %d (6) The session ID of the process.

tty_nr %d (7) The controlling terminal of the process. (The minor device number is contained in the combination of bits 31 to 20 and 7 to 0; the major device number is in bits 15 to 8.)

tpgid %d (8) The ID of the foreground process group of the controlling terminal of the process.

flags %u (%lu before Linux 2.6.22)
(9) The kernel flags word of the process. For bit meanings, see the PF_* defines in the Linux kernel source file include/linux/sched.h. Details depend on the kernel version.

minflt %lu (10) The number of minor faults the process has made which have not required loading a memory page from disk.

cminflt %lu (11) The number of minor faults that the process's waited-for children have made.

majflt %lu (12) The number of major faults the process has made which have required loading a memory page from disk.

cmajflt %lu (13) The number of major faults that the process's waited-for children have made.

utime %lu (14) Amount of time that this process has been scheduled in user mode, measured in clock ticks (divide by sysconf(_SC_CLK_TCK)). This includes guest time, guest_time (time spent running a virtual CPU, see below), so that applications that are not aware of the guest time field do not lose that time from their calculations.

stime %lu (15) Amount of time that this process has been scheduled in kernel mode, measured in clock ticks (divide by sysconf(_SC_CLK_TCK)).

cutime %ld (16) Amount of time that this process's waited-for children have been scheduled in user mode, measured in clock ticks (divide by sysconf(_SC_CLK_TCK)). (See also **times**(2).) This includes guest time, cguest_time (time spent running a virtual CPU, see below).

cstime %ld (17) Amount of time that this process's waited-for children have been scheduled in kernel mode, measured in clock ticks (divide by sysconf(_SC_CLK_TCK)).

priority %ld
(18) (Explanation for Linux 2.6) For processes running a real-time scheduling policy (policy below; see **sched_setscheduler**(2)), this is the negated scheduling priority, minus one; that is, a number in the range -2 to -100, corresponding to real-time priorities 1 to 99. For processes running under a non-real-time scheduling policy, this is the raw nice value (**setpriority**(2)) as represented in the kernel. The kernel stores nice values as numbers in the range 0 (high) to 39 (low), corresponding to the user-visible nice range of -20 to 19.

Before Linux 2.6, this was a scaled value based on the scheduler weighting given to this process.

nice %ld (19) The nice value (see **setpriority**(2)), a value in the range 19 (low priority) to -20 (high priority).

num_threads %ld
(20) Number of threads in this process (since Linux 2.6). Before kernel 2.6, this field was hard coded to 0 as a placeholder for an earlier removed field.

itrealvalue %ld
(21) The time in jiffies before the next **SIGALRM** is sent to the process due to an interval timer. Since kernel 2.6.17, this field is no longer maintained, and is hard coded as 0.

starttime %llu (was %lu before Linux 2.6)
(22) The time the process started after system boot. In kernels before Linux 2.6, this value was expressed in jiffies. Since Linux 2.6, the value is expressed in clock ticks (divide by

sysconf(_SC_CLK_TCK)).

vsize %lu (23) Virtual memory size in bytes.

rss %ld (24) Resident Set Size: number of pages the process has in real memory. This is just the pages which count toward text, data, or stack space. This does not include pages which have not been demand-loaded in, or which are swapped out.

rsslim %lu (25) Current soft limit in bytes on the rss of the process; see the description of **RLIMIT_RSS** in **getrlimit**(2).

startcode %lu
(26) The address above which program text can run.

endcode %lu (27) The address below which program text can run.

startstack %lu
(28) The address of the start (i.e., bottom) of the stack.

kstkesp %lu (29) The current value of ESP (stack pointer), as found in the kernel stack page for the process.

kstkeip %lu (30) The current EIP (instruction pointer).

signal %lu (31) The bitmap of pending signals, displayed as a decimal number. Obsolete, because it does not provide information on real-time signals; use /proc/[pid]/status instead.

blocked %lu (32) The bitmap of blocked signals, displayed as a decimal number. Obsolete, because it does not provide information on real-time signals; use /proc/[pid]/status instead.

sigignore %lu
(33) The bitmap of ignored signals, displayed as a decimal number. Obsolete, because it does not provide information on real-time signals; use /proc/[pid]/status instead.

sigcatch %lu
(34) The bitmap of caught signals, displayed as a decimal number. Obsolete, because it does not provide information on real-time signals; use /proc/[pid]/status instead.

wchan %lu (35) This is the "channel" in which the process is waiting. It is the address of a system call, and can be looked up in a namelist if you need a textual name. (If you have an up-to-date /etc/psdatabase, then try ps -l to see the WCHAN field in action.)

nswap %lu (36) Number of pages swapped (not maintained).

cnswap %lu (37) Cumulative nswap for child processes (not maintained).

exit_signal %d (since Linux 2.1.22)
(38) Signal to be sent to parent when we die.

processor %d (since Linux 2.2.8)
(39) CPU number last executed on.

rt_priority %u (since Linux 2.5.19; was %lu before Linux 2.6.22)
(40) Real-time scheduling priority, a number in the range 1 to 99 for processes scheduled under a real-time policy, or 0, for non-real-time processes (see **sched_setscheduler**(2)).

policy %u (since Linux 2.5.19; was %lu before Linux 2.6.22)
(41) Scheduling policy (see **sched_setscheduler**(2)). Decode using the SCHED_* constants in linux/sched.h.

delayacct_blkio_ticks %llu (since Linux 2.6.18)
(42) Aggregated block I/O delays, measured in clock ticks (centiseconds).

guest_time %lu (since Linux 2.6.24)
(43) Guest time of the process (time spent running a virtual CPU for a guest operating system), measured in clock ticks (divide by sysconf(_SC_CLK_TCK)).

cguest_time %ld (since Linux 2.6.24)
(44) Guest time of the process's children, measured in clock ticks (divide by sysconf(_SC_CLK_TCK)).

/proc/[pid]/statm
Provides information about memory usage, measured in pages. The columns are:

size	(1) total program size (same as VmSize in /proc/[pid]/status)
resident	(2) resident set size (same as VmRSS in /proc/[pid]/status)
share	(3) shared pages (i.e., backed by a file)
text	(4) text (code)
lib	(5) library (unused in Linux 2.6)
data	(6) data + stack
dt	(7) dirty pages (unused in Linux 2.6)

/proc/[pid]/status
Provides much of the information in /proc/[pid]/stat and /proc/[pid]/statm in a format that's easier for humans to parse. Here's an example:

```
$ cat /proc/$$/status
Name:   bash
State:  S (sleeping)
Tgid:   3515
Pid:    3515
PPid:   3452
TracerPid:      0
Uid:    1000    1000    1000    1000
Gid:    100     100     100     100
FDSize: 256
Groups: 16 33 100
VmPeak:     9136 kB
VmSize:     7896 kB
VmLck:         0 kB
VmHWM:      7572 kB
VmRSS:      6316 kB
VmData:     5224 kB
VmStk:        88 kB
VmExe:       572 kB
VmLib:      1708 kB
VmPTE:        20 kB
Threads:        1
SigQ:   0/3067
SigPnd: 0000000000000000
ShdPnd: 0000000000000000
SigBlk: 0000000000010000
SigIgn: 0000000000384004
SigCgt: 000000004b813efb
CapInh: 0000000000000000
CapPrm: 0000000000000000
CapEff: 0000000000000000
CapBnd: ffffffffffffffff
Cpus_allowed:   00000001
Cpus_allowed_list:      0
Mems_allowed:   1
Mems_allowed_list:      0
voluntary_ctxt_switches:        150
nonvoluntary_ctxt_switches:     545
```

The fields are as follows:

* Name: Command run by this process.

* State: Current state of the process. One of "R (running)", "S (sleeping)", "D (disk sleep)", "T (stopped)", "T (tracing stop)", "Z (zombie)", or "X (dead)".

* Tgid: Thread group ID (i.e., Process ID).

* Pid: Thread ID (see **gettid**(2)).

* PPid: PID of parent process.

* TracerPid: PID of process tracing this process (0 if not being traced).

* Uid, Gid: Real, effective, saved set, and filesystem UIDs (GIDs).

* FDSize: Number of file descriptor slots currently allocated.

* Groups: Supplementary group list.

* VmPeak: Peak virtual memory size.

* VmSize: Virtual memory size.

* VmLck: Locked memory size (see **mlock**(3)).

* VmHWM: Peak resident set size ("high water mark").

* VmRSS: Resident set size.

* VmData, VmStk, VmExe: Size of data, stack, and text segments.

* VmLib: Shared library code size.

* VmPTE: Page table entries size (since Linux 2.6.10).

* Threads: Number of threads in process containing this thread.

* SigQ: This field contains two slash-separated numbers that relate to queued signals for the real user ID of this process. The first of these is the number of currently queued signals for this real user ID, and the second is the resource limit on the number of queued signals for this process (see the description of **RLIMIT_SIGPENDING** in **getrlimit**(2)).

* SigPnd, ShdPnd: Number of signals pending for thread and for process as a whole (see **pthreads**(7) and **signal**(7)).

* SigBlk, SigIgn, SigCgt: Masks indicating signals being blocked, ignored, and caught (see **signal**(7)).

* CapInh, CapPrm, CapEff: Masks of capabilities enabled in inheritable, permitted, and effective sets (see **capabilities**(7)).

* CapBnd: Capability Bounding set (since kernel 2.6.26, see **capabilities**(7)).

* Cpus_allowed: Mask of CPUs on which this process may run (since Linux 2.6.24, see **cpuset**(7)).

* Cpus_allowed_list: Same as previous, but in "list format" (since Linux 2.6.26, see **cpuset**(7)).

* Mems_allowed: Mask of memory nodes allowed to this process (since Linux 2.6.24, see **cpuset**(7)).

* Mems_allowed_list: Same as previous, but in "list format" (since Linux 2.6.26, see **cpuset**(7)).

* voluntary_context_switches, nonvoluntary_context_switches: Number of voluntary and involuntary context switches (since Linux 2.6.23).

/proc/[pid]/task (since Linux 2.6.0-test6)
This is a directory that contains one subdirectory for each thread in the process. The name of each subdirectory is the numerical thread ID ([tid]) of the thread (see **gettid**(2)). Within each of these subdirectories, there is a set of files with the same names and contents as under the /proc/[pid] directories. For attributes that are shared by all threads, the contents for each of the files under the task/[tid] subdirectories will be the same as in the corresponding file in the parent /proc/[pid] directory (e.g., in a multithreaded process, all of the task/[tid]/cwd files will have the same value as the /proc/[pid]/cwd file in the parent directory, since all of the threads in a process share a working directory). For attributes that are distinct for each thread, the corresponding files under task/[tid] may have different values (e.g., various fields in each of the task/[tid]/status files may be different for each thread).

In a multithreaded process, the contents of the /proc/[pid]/task directory are not available if the main thread has already terminated (typically by calling **pthread_exit**(3)).

/proc/apm
Advanced power management version and battery information when **CONFIG_APM** is defined at kernel compilation time.

/proc/bus
Contains subdirectories for installed busses.

/proc/bus/pccard
Subdirectory for PCMCIA devices when **CONFIG_PCMCIA** is set at kernel compilation time.

/proc/bus/pccard/drivers

/proc/bus/pci
Contains various bus subdirectories and pseudo-files containing information about PCI busses, installed devices, and device drivers. Some of these files are not ASCII.

/proc/bus/pci/devices
Information about PCI devices. They may be accessed through **lspci**(8) and **setpci**(8).

/proc/cmdline
Arguments passed to the Linux kernel at boot time. Often done via a boot manager such as **lilo**(8) or **grub**(8).

/proc/config.gz (since Linux 2.6)
This file exposes the configuration options that were used to build the currently running kernel, in the same format as they would be shown in the .config file that resulted when configuring the kernel (using make xconfig, make config, or similar). The file contents are compressed; view or search them using **zcat**(1), **zgrep**(1), etc. As long as no changes have been made to the following file, the contents of /proc/config.gz are the same as those provided by :

```
cat /lib/modules/$(uname -r)/build/.config
```

/proc/config.gz is provided only if the kernel is configured with **CONFIG_IKCONFIG_PROC**.

/proc/cpuinfo
This is a collection of CPU and system architecture dependent items, for each supported architecture a different list. Two common entries are processor which gives CPU number and bogomips; a system constant that is calculated during kernel initialization. SMP machines have information for each CPU. The **lscpu**(1) command gathers its information from this file.

/proc/devices
Text listing of major numbers and device groups. This can be used by MAKEDEV scripts for consistency with the kernel.

/proc/diskstats (since Linux 2.5.69)
This file contains disk I/O statistics for each disk device. See the Linux kernel source file Documentation/iostats.txt for further information.

/proc/dma
This is a list of the registered ISA DMA (direct memory access) channels in use.

/proc/driver
Empty subdirectory.

/proc/execdomains
List of the execution domains (ABI personalities).

/proc/fb
Frame buffer information when **CONFIG_FB** is defined during kernel compilation.

/proc/filesystems
A text listing of the filesystems which are supported by the kernel, namely filesystems which were compiled into the kernel or whose kernel modules are currently loaded. (See also

filesystems(5).) If a filesystem is marked with "nodev", this means that it does not require a block device to be mounted (e.g., virtual filesystem, network filesystem).

Incidentally, this file may be used by **mount**(8) when no filesystem is specified and it didn't manage to determine the filesystem type. Then filesystems contained in this file are tried (excepted those that are marked with "nodev").

/proc/fs
Empty subdirectory.

/proc/ide
This directory exists on systems with the IDE bus. There are directories for each IDE channel and attached device. Files include:

cache	buffer size in KB
capacity	number of sectors
driver	driver version
geometry	physical and logical geometry
identify	in hexadecimal
media	media type
model	manufacturer's model number
settings	drive settings
smart_thresholds	in hexadecimal
smart_values	in hexadecimal

The **hdparm**(8) utility provides access to this information in a friendly format.

/proc/interrupts
This is used to record the number of interrupts per CPU per IO device. Since Linux 2.6.24, for the i386 and x86_64 architectures, at least, this also includes interrupts internal to the system (that is, not associated with a device as such), such as NMI (nonmaskable interrupt), LOC (local timer interrupt), and for SMP systems, TLB (TLB flush interrupt), RES (rescheduling interrupt), CAL (remote function call interrupt), and possibly others. Very easy to read formatting, done in ASCII.

/proc/iomem
I/O memory map in Linux 2.4.

/proc/ioports
This is a list of currently registered Input-Output port regions that are in use.

/proc/kallsyms (since Linux 2.5.71)
This holds the kernel exported symbol definitions used by the **modules**(X) tools to dynamically link and bind loadable modules. In Linux 2.5.47 and earlier, a similar file with slightly different syntax was named ksyms.

/proc/kcore
This file represents the physical memory of the system and is stored in the ELF core file format. With this pseudo-file, and an unstripped kernel (/usr/src/linux/vmlinux) binary, GDB can be used to examine the current state of any kernel data structures.

The total length of the file is the size of physical memory (RAM) plus 4KB.

/proc/kmsg
This file can be used instead of the **syslog**(2) system call to read kernel messages. A process must have superuser privileges to read this file, and only one process should read this file. This file should not be read if a syslog process is running which uses the **syslog**(2) system call facility to log kernel messages.

Information in this file is retrieved with the **dmesg**(1) program.

/proc/ksyms (Linux 1.1.23-2.5.47)
See /proc/kallsyms.

/proc/loadavg
The first three fields in this file are load average figures giving the number of jobs in the run queue (state R) or waiting for disk I/O (state D) averaged over 1, 5, and 15 minutes. They

are the same as the load average numbers given by **uptime**(1) and other programs. The fourth field consists of two numbers separated by a slash (/). The first of these is the number of currently runnable kernel scheduling entities (processes, threads). The value after the slash is the number of kernel scheduling entities that currently exist on the system. The fifth field is the PID of the process that was most recently created on the system.

/proc/locks
This file shows current file locks (**flock**(2) and **fcntl**(2)) and leases (**fcntl**(2)).

/proc/malloc (only up to and including Linux 2.2)
This file is present only if **CONFIG_DEBUG_MALLOC** was defined during compilation.

/proc/meminfo
This file reports statistics about memory usage on the system. It is used by **free**(1) to report the amount of free and used memory (both physical and swap) on the system as well as the shared memory and buffers used by the kernel. Each line of the file consists of a parameter name, followed by a colon, the value of the parameter, and an option unit of measurement (e.g., "kB"). The list below describes the parameter names and the format specifier required to read the field value. Except as noted below, all of the fields have been present since at least Linux 2.6.0. Some fileds are displayed only if the kernel was configured with various options; those dependencies are noted in the list.

MemTotal %lu
Total usable RAM (i.e. physical RAM minus a few reserved bits and the kernel binary code).

MemFree %lu
The sum of LowFree+HighFree.

Buffers %lu
Relatively temporary storage for raw disk blocks that shouldn't get tremendously large (20MB or so).

Cached %lu
In-memory cache for files read from the disk (the page cache). Doesn't include SwapCached.

SwapCached %lu
Memory that once was swapped out, is swapped back in but still also is in the swap file. (If memory pressure is high, these pages don't need to be swapped out again because they are already in the swap file. This saves I/O.)

Active %lu
Memory that has been used more recently and usually not reclaimed unless absolutely necessary.

Inactive %lu
Memory which has been less recently used. It is more eligible to be reclaimed for other purposes.

Active(anon) %lu (since Linux 2.6.28)
[To be documented.]

Inactive(anon) %lu (since Linux 2.6.28)
[To be documented.]

Active(file) %lu (since Linux 2.6.28)
[To be documented.]

Inactive(file) %lu (since Linux 2.6.28)
[To be documented.]

Unevictable %lu (since Linux 2.6.28)
(From Linux 2.6.28 to 2.6.30, **CONFIG_UNEVICTABLE_LRU** was required.) [To be documented.]

Mlocked %lu (since Linux 2.6.28)
(From Linux 2.6.28 to 2.6.30, **CONFIG_UNEVICTABLE_LRU** was required.) [To be documented.]

HighTotal %lu
(Starting with Linux 2.6.19, **CONFIG_HIGHMEM** is required.) Total amount of highmem. Highmem is all memory above ~860MB of physical memory. Highmem areas are for use by user-space programs, or for the page cache. The kernel must use tricks to access this memory, making it slower to access than lowmem.

HighFree %lu
(Starting with Linux 2.6.19, **CONFIG_HIGHMEM** is required.) Amount of free highmem.

LowTotal %lu
(Starting with Linux 2.6.19, **CONFIG_HIGHMEM** is required.) Total amount of lowmem. Lowmem is memory which can be used for everything that highmem can be used for, but it is also available for the kernel's use for its own data structures. Among many other things, it is where everything from Slab is allocated. Bad things happen when you're out of lowmem.

LowFree %lu
(Starting with Linux 2.6.19, **CONFIG_HIGHMEM** is required.) Amount of free lowmem.

MmapCopy %lu (since Linux 2.6.29)
(**CONFIG_MMU** is required.) [To be documented.]

SwapTotal %lu
Total amount of swap space available.

SwapFree %lu
Amount of swap space that is currently unused.

Dirty %lu
Memory which is waiting to get written back to the disk.

Writeback %lu
Memory which is actively being written back to the disk.

AnonPages %lu (since Linux 2.6.18)
Non-file backed pages mapped into user-space page tables.

Mapped %lu
Files which have been mmaped, such as libraries.

Shmem %lu (since Linux 2.6.32)
[To be documented.]

Slab %lu
In-kernel data structures cache.

SReclaimable %lu (since Linux 2.6.19)
Part of Slab, that might be reclaimed, such as caches.

SUnreclaim %lu (since Linux 2.6.19)
Part of Slab, that cannot be reclaimed on memory pressure.

KernelStack %lu (since Linux 2.6.32)
Amount of memory allocated to kernel stacks.

PageTables %lu (since Linux 2.6.18)
Amount of memory dedicated to the lowest level of page tables.

Quicklists %lu (since Linux 2.6.27)
(**CONFIG_QUICKLIST** is required.) [To be documented.]

NFS_Unstable %lu (since Linux 2.6.18)
NFS pages sent to the server, but not yet committed to stable storage.

Bounce %lu (since Linux 2.6.18)
Memory used for block device "bounce buffers".

WritebackTmp %lu (since Linux 2.6.26)
Memory used by FUSE for temporary writeback buffers.

CommitLimit %lu (since Linux 2.6.10)
Based on the overcommit ratio ('vm.overcommit_ratio'),

this is the total amount of memory currently available to be allocated on the system. This limit is adhered to only if strict overcommit accounting is enabled (mode 2 in /proc/sys/vm/overcommit_ratio). The CommitLimit is calculated using the following formula:

CommitLimit = (overcommit_ratio * Physical RAM) + Swap

For example, on a system with 1GB of physical RAM and 7GB of swap with a overcommit_ratio of 30, this formula yields a CommitLimit of 7.3GB. For more details, see the memory overcommit documentation in the kernel source file Documentation/vm/overcommit-accounting.

Committed_AS %lu
The amount of memory presently allocated on the system. The committed memory is a sum of all of the memory which has been allocated by processes, even if it has not been "used" by them as of yet. A process which allocates 1GB of memory (using **malloc**(3) or similar), but touches only 300MB of that memory will show up as using only 300MB of memory even if it has the address space allocated for the entire 1GB. This 1GB is memory which has been "committed" to by the VM and can be used at any time by the allocating application. With strict overcommit enabled on the system (mode 2 /proc/sys/vm/overcommit_memory), allocations which would exceed the CommitLimit (detailed above) will not be permitted. This is useful if one needs to guarantee that processes will not fail due to lack of memory once that memory has been successfully allocated.

VmallocTotal %lu
Total size of vmalloc memory area.

VmallocUsed %lu
Amount of vmalloc area which is used.

VmallocChunk %lu
Largest contiguous block of vmalloc area which is free.

HardwareCorrupted %lu (since Linux 2.6.32)
(**CONFIG_MEMORY_FAILURE** is required.) [To be documented.]

AnonHugePages %lu (since Linux 2.6.38)
(**CONFIG_TRANSPARENT_HUGEPAGE** is required.) Non-file backed huge pages mapped into user-space page tables.

HugePages_Total %lu
(**CONFIG_HUGETLB_PAGE** is required.) The size of the pool of huge pages.

HugePages_Free %lu
(**CONFIG_HUGETLB_PAGE** is required.) The number of huge pages in the pool that are not yet allocated.

HugePages_Rsvd %lu (since Linux 2.6.17)
(**CONFIG_HUGETLB_PAGE** is required.) This is the number of huge pages for which a commitment to allocate from the pool has been made, but no allocation has yet been made. These reserved huge pages guarantee that an application will be able to allocate a huge page from the pool of huge pages at fault time.

HugePages_Surp %lu (since Linux 2.6.24)
(**CONFIG_HUGETLB_PAGE** is required.) This is the number of huge pages in the pool above the value in /proc/sys/vm/nr_hugepages. The maximum number of surplus huge pages is controlled by /proc/sys/vm/nr_overcommit_hugepages.

Hugepagesize %lu
(**CONFIG_HUGETLB_PAGE** is required.) The size of huge pages.

/proc/modules
A text list of the modules that have been loaded by the system. See also **lsmod**(8).

/proc/mounts

Before kernel 2.4.19, this file was a list of all the filesystems currently mounted on the system. With the introduction of per-process mount namespaces in Linux 2.4.19, this file became a link to /proc/self/mounts, which lists the mount points of the process's own mount namespace. The format of this file is documented in **fstab**(5).

/proc/mtrr
Memory Type Range Registers. See the Linux kernel source file Documentation/mtrr.txt for details.

/proc/net
various net pseudo-files, all of which give the status of some part of the networking layer. These files contain ASCII structures and are, therefore, readable with **cat**(1). However, the standard **netstat**(8) suite provides much cleaner access to these files.

/proc/net/arp
This holds an ASCII readable dump of the kernel ARP table used for address resolutions. It will show both dynamically learned and preprogrammed ARP entries. The format is:

```
IP address       HW type     Flags       HW address            Mask     Device
192.168.0.50     0x1         0x2         00:50:BF:25:68:F3     *        eth0
192.168.0.250    0x1         0xc         00:00:00:00:00:00     *        eth0
```

Here "IP address" is the IPv4 address of the machine and the "HW type" is the hardware type of the address from RFC 826. The flags are the internal flags of the ARP structure (as defined in /usr/include/linux/if_arp.h) and the "HW address" is the data link layer mapping for that IP address if it is known.

/proc/net/dev
The dev pseudo-file contains network device status information. This gives the number of received and sent packets, the number of errors and collisions and other basic statistics. These are used by the **ifconfig**(8) program to report device status. The format is:

```
Inter-|   Receive                                                |  Transmit
 face |bytes    packets errs drop fifo frame compressed multicast|bytes    packets errs drop fifo colls carrie
    lo: 2776770   11307    0    0    0     0          0         0  2776770   11307    0    0    0     0       (
  eth0: 1215645    2751    0    0    0     0          0         0  1782404    4324    0    0    0   427       (
  ppp0: 1622270    5552    1    0    0     0          0         0   354130    5669    0    0    0     0       (
  tap0:    7714      81    0    0    0     0          0         0     7714      81    0    0    0     0       (
```

/proc/net/dev_mcast
Defined in /usr/src/linux/net/core/dev_mcast.c:

```
indx interface_name  dmi_u dmi_g dmi_address
2    eth0            1     0     01005e000001
3    eth1            1     0     01005e000001
4    eth2            1     0     01005e000001
```

/proc/net/igmp
Internet Group Management Protocol. Defined in /usr/src/linux/net/core/igmp.c.

/proc/net/rarp
This file uses the same format as the arp file and contains the current reverse mapping database used to provide **rarp**(8) reverse address lookup services. If RARP is not configured into the kernel, this file will not be present.

/proc/net/raw
Holds a dump of the RAW socket table. Much of the information is not of use apart from debugging. The "sl" value is the kernel hash slot for the socket, the "local_address" is the local address and protocol number pair. "St" is the internal status of the socket. The "tx_queue" and "rx_queue" are the outgoing and incoming data queue in terms of kernel memory usage. The "tr", "tm->when", and "rexmits" fields are not used by RAW. The "uid" field holds the effective UID of the creator of the socket.

/proc/net/snmp
This file holds the ASCII data needed for the IP, ICMP, TCP, and UDP management information bases for an SNMP agent.

/proc/net/tcp
Holds a dump of the TCP socket table. Much of the information

is not of use apart from debugging. The "sl" value is the kernel hash slot for the socket, the "local_address" is the local address and port number pair. The "rem_address" is the remote address and port number pair (if connected). "St" is the internal status of the socket. The "tx_queue" and "rx_queue" are the outgoing and incoming data queue in terms of kernel memory usage. The "tr", "tm->when", and "rexmits" fields hold internal information of the kernel socket state and are only useful for debugging. The "uid" field holds the effective UID of the creator of the socket.

/proc/net/udp
Holds a dump of the UDP socket table. Much of the information is not of use apart from debugging. The "sl" value is the kernel hash slot for the socket, the "local_address" is the local address and port number pair. The "rem_address" is the remote address and port number pair (if connected). "St" is the internal status of the socket. The "tx_queue" and "rx_queue" are the outgoing and incoming data queue in terms of kernel memory usage. The "tr", "tm->when", and "rexmits" fields are not used by UDP. The "uid" field holds the effective UID of the creator of the socket. The format is:

```
sl  local_address rem_address   st tx_queue rx_queue tr rexmits  tm->when uid
 1: 01642C89:0201 0C642C89:03FF 01 00000000:00000001 01:000071BA 00000000 0
 1: 00000000:0801 00000000:0000 0A 00000000:00000000 00:00000000 6F000100 0
 1: 00000000:0201 00000000:0000 0A 00000000:00000000 00:00000000 00000000 0
```

/proc/net/unix
Lists the UNIX domain sockets present within the system and their status. The format is:

```
Num RefCount Protocol Flags    Type St Path
 0: 00000002 00000000 00000000 0001 03
 1: 00000001 00000000 00010000 0001 01 /dev/printer
```

Here "Num" is the kernel table slot number, "RefCount" is the number of users of the socket, "Protocol" is currently always 0, "Flags" represent the internal kernel flags holding the status of the socket. Currently, type is always "1" (UNIX domain datagram sockets are not yet supported in the kernel). "St" is the internal state of the socket and Path is the bound path (if any) of the socket.

/proc/partitions
Contains the major and minor numbers of each partition as well as the number of 1024-byte blocks and the partition name.

/proc/pci
This is a listing of all PCI devices found during kernel initialization and their configuration.

This file has been deprecated in favor of a new /proc interface for PCI (/proc/bus/pci). It became optional in Linux 2.2 (available with **CONFIG_PCI_OLD_PROC** set at kernel compilation). It became once more nonoptionally enabled in Linux 2.4. Next, it was deprecated in Linux 2.6 (still available with **CONFIG_PCI_LEGACY_PROC** set), and finally removed altogether since Linux 2.6.17.

/proc/profile (since Linux 2.4)
This file is present only if the kernel was booted with the profile=1 command-line option. It exposes kernel profiling information in a binary format for use by **readprofile**(1). Writing (e.g., an empty string) to this file resets the profiling counters; on some architectures, writing a binary integer "profiling multiplier" of size sizeof(int) sets the profiling interrupt frequency.

/proc/scsi
A directory with the scsi mid-level pseudo-file and various SCSI low-level driver directories, which contain a file for each SCSI host in this system, all of which give the status of some part of the SCSI IO subsystem. These files contain ASCII structures and are, therefore, readable with **cat**(1).

You can also write to some of the files to reconfigure the subsystem or switch certain features on or off.

/proc/scsi/scsi
This is a listing of all SCSI devices known to the kernel. The listing is similar to the one seen during bootup. scsi

currently supports only the add-single-device command which allows root to add a hotplugged device to the list of known devices.

The command

```
echo 'scsi add-single-device 1 0 5 0' > /proc/scsi/scsi
```

will cause host scsi1 to scan on SCSI channel 0 for a device on ID 5 LUN 0. If there is already a device known on this address or the address is invalid, an error will be returned.

/proc/scsi/[drivername]
[drivername] can currently be NCR53c7xx, aha152x, aha1542, aha1740, aic7xxx, buslogic, eata_dma, eata_pio, fdomain, in2000, pas16, qlogic, scsi_debug, seagate, t128, u15-24f, ultrastore, or wd7000. These directories show up for all drivers that registered at least one SCSI HBA. Every directory contains one file per registered host. Every host-file is named after the number the host was assigned during initialization.

Reading these files will usually show driver and host configuration, statistics, etc.

Writing to these files allows different things on different hosts. For example, with the latency and nolatency commands, root can switch on and off command latency measurement code in the eata_dma driver. With the lockup and unlock commands, root can control bus lockups simulated by the scsi_debug driver.

/proc/self
This directory refers to the process accessing the /proc filesystem, and is identical to the /proc directory named by the process ID of the same process.

/proc/slabinfo
Information about kernel caches. Since Linux 2.6.16 this file is present only if the **CONFIG_SLAB** kernel configuration option is enabled. The columns in /proc/slabinfo are:

```
cache-name
num-active-objs
total-objs
object-size
num-active-slabs
total-slabs
num-pages-per-slab
```

See **slabinfo**(5) for details.

/proc/stat
kernel/system statistics. Varies with architecture. Common entries include:

cpu 3357 0 4313 1362393
The amount of time, measured in units of USER_HZ (1/100ths of a second on most architectures, use sysconf(_SC_CLK_TCK) to obtain the right value), that the system spent in various states:

user (1) Time spent in user mode.

nice (2) Time spent in user mode with low priority (nice).

system (3) Time spent in system mode.

idle (4) Time spent in the idle task. This value should be USER_HZ times the second entry in the /proc/uptime pseudo-file.

iowait (since Linux 2.5.41)
(5) Time waiting for I/O to complete.

irq (since Linux 2.6.0-test4)
(6) Time servicing interrupts.

softirq (since Linux 2.6.0-test4)
(7) Time servicing softirqs.

steal (since Linux 2.6.11)

(8) Stolen time, which is the time spent in other operating systems when running in a virtualized environment

guest (since Linux 2.6.24)
(9) Time spent running a virtual CPU for guest operating systems under the control of the Linux kernel.

guest_nice (since Linux 2.6.33)
(10) Time spent running a niced guest (virtual CPU for guest operating systems under the control of the Linux kernel).

page 5741 1808
The number of pages the system paged in and the number that were paged out (from disk).

swap 1 0
The number of swap pages that have been brought in and out.

intr 1462898
This line shows counts of interrupts serviced since boot time, for each of the possible system interrupts. The first column is the total of all interrupts serviced; each subsequent column is the total for a particular interrupt.

disk_io: (2,0):(31,30,5764,1,2) (3,0):...
(major,disk_idx):(noinfo, read_io_ops, blks_read, write_io_ops, blks_written)
(Linux 2.4 only)

ctxt 115315
The number of context switches that the system underwent.

btime 769041601
boot time, in seconds since the Epoch, 1970-01-01 00:00:00 +0000 (UTC).

processes 86031
Number of forks since boot.

procs_running 6
Number of processes in runnable state. (Linux 2.5.45 onward.)

procs_blocked 2
Number of processes blocked waiting for I/O to complete. (Linux 2.5.45 onward.)

/proc/swaps
Swap areas in use. See also **swapon**(8).

/proc/sys
This directory (present since 1.3.57) contains a number of files and subdirectories corresponding to kernel variables. These variables can be read and sometimes modified using the /proc filesystem, and the (deprecated) **sysctl**(2) system call.

/proc/sys/abi (since Linux 2.4.10)
This directory may contain files with application binary information. See the Linux kernel source file Documentation/sysctl/abi.txt for more information.

/proc/sys/debug
This directory may be empty.

/proc/sys/dev
This directory contains device-specific information (e.g., dev/cdrom/info). On some systems, it may be empty.

/proc/sys/fs
This directory contains the files and subdirectories for kernel variables related to filesystems.

/proc/sys/fs/binfmt_misc
Documentation for files in this directory can be found in the Linux kernel sources in Documentation/binfmt_misc.txt.

/proc/sys/fs/dentry-state (since Linux 2.2)
This file contains information about the status of the directory cache (dcache). The file contains six numbers, nr_dentry, nr_unused, age_limit (age in seconds), want_pages (pages requested by system) and two dummy values.

* nr_dentry is the number of allocated dentries (dcache entries). This field is unused in Linux 2.2.

* nr_unused is the number of unused dentries.

* age_limit is the age in seconds after which dcache entries can be reclaimed when memory is short.

* want_pages is nonzero when the kernel has called shrink_dcache_pages() and the dcache isn't pruned yet.

/proc/sys/fs/dir-notify-enable
This file can be used to disable or enable the dnotify interface described in **fcntl**(2) on a system-wide basis. A value of 0 in this file disables the interface, and a value of 1 enables it.

/proc/sys/fs/dquot-max
This file shows the maximum number of cached disk quota entries. On some (2.4) systems, it is not present. If the number of free cached disk quota entries is very low and you have some awesome number of simultaneous system users, you might want to raise the limit.

/proc/sys/fs/dquot-nr
This file shows the number of allocated disk quota entries and the number of free disk quota entries.

/proc/sys/fs/epoll (since Linux 2.6.28)
This directory contains the file max_user_watches, which can be used to limit the amount of kernel memory consumed by the epoll interface. For further details, see **epoll**(7).

/proc/sys/fs/file-max
This file defines a system-wide limit on the number of open files for all processes. (See also **setrlimit**(2), which can be used by a process to set the per-process limit, **RLIMIT_NOFILE**, on the number of files it may open.) If you get lots of error messages in the kernel log about running out of file handles (look for "VFS: file-max limit <number> reached"), try increasing this value:

```
echo 100000 > /proc/sys/fs/file-max
```

The kernel constant **NR_OPEN** imposes an upper limit on the value that may be placed in file-max.

Privileged processes (**CAP_SYS_ADMIN**) can override the file-max limit.

/proc/sys/fs/file-nr
This (read-only) file contains three numbers: the number of allocated file handles (i.e., the number of files presently opened); the number of free file handles; and the maximum number of file handles (i.e., the same value as /proc/sys/fs/file-max). If the number of allocated file handles is close to the maximum, you should consider increasing the maximum. Before Linux 2.6, the kernel allocated file handles dynamically, but it didn't free them again. Instead the free file handles were kept in a list for reallocation; the "free file handles" value indicates the size of that list. A large number of free file handles indicates that there was a past peak in the usage of open file handles. Since Linux 2.6, the kernel does deallocate freed file handles, and the "free file handles" value is always zero.

/proc/sys/fs/inode-max (only present until Linux 2.2)
This file contains the maximum number of in-memory inodes. This value should be 3-4 times larger than the value in file-max, since stdin, stdout and network sockets also need an inode to handle them. When you regularly run out of inodes, you need to increase this value.

Starting with Linux 2.4, there is no longer a static limit on the number of inodes, and this file is removed.

/proc/sys/fs/inode-nr

This file contains the first two values from inode-state.

/proc/sys/fs/inode-state
This file contains seven numbers: nr_inodes, nr_free_inodes, preshrink, and four dummy values (always zero).

nr_inodes is the number of inodes the system has allocated. nr_free_inodes represents the number of free inodes.

preshrink is nonzero when the nr_inodes > inode-max and the system needs to prune the inode list instead of allocating more; since Linux 2.4, this field is a dummy value (always zero).

/proc/sys/fs/inotify (since Linux 2.6.13)
This directory contains files max_queued_events, max_user_instances, and max_user_watches, that can be used to limit the amount of kernel memory consumed by the inotify interface. For further details, see **inotify**(7).

/proc/sys/fs/lease-break-time
This file specifies the grace period that the kernel grants to a process holding a file lease (**fcntl**(2)) after it has sent a signal to that process notifying it that another process is waiting to open the file. If the lease holder does not remove or downgrade the lease within this grace period, the kernel forcibly breaks the lease.

/proc/sys/fs/leases-enable
This file can be used to enable or disable file leases (**fcntl**(2)) on a system-wide basis. If this file contains the value 0, leases are disabled. A nonzero value enables leases.

/proc/sys/fs/mqueue (since Linux 2.6.6)
This directory contains files msg_max, msgsize_max, and queues_max, controlling the resources used by POSIX message queues. See **mq_overview**(7) for details.

/proc/sys/fs/overflowgid and /proc/sys/fs/overflowuid
These files allow you to change the value of the fixed UID and GID. The default is 65534. Some filesystems support only 16-bit UIDs and GIDs, although in Linux UIDs and GIDs are 32 bits. When one of these filesystems is mounted with writes enabled, any UID or GID that would exceed 65535 is translated to the overflow value before being written to disk.

/proc/sys/fs/pipe-max-size (since Linux 2.6.35)
The value in this file defines an upper limit for raising the capacity of a pipe using the **fcntl**(2) **F_SETPIPE_SZ** operation. This limit applies only to unprivileged processes. The default value for this file is 1,048,576. The value assigned to this file may be rounded upward, to reflect the value actually employed for a convenient implementation. To determine the rounded-up value, display the contents of this file after assigning a value to it. The minimum value that can be assigned to this file is the system page size.

/proc/sys/fs/protected_hardlinks (since Linux 3.6)
When the value in this file is 0, no restrictions are placed on the creation of hard links (i.e., this is the historical behaviour before Linux 3.6). When the value in this file is 1, a hard link can be created to a target file only if one of the following conditions is true:

* The caller has the **CAP_FOWNER** capability.

* The filesystem UID of the process creating the link matches the owner (UID) of the target file (as described in **credentials**(7), a process's filesystem UID is normally the same as its effective UID).

* All of the following conditions are true:

  · the target is a regular file;

  · the target file does not have its set-user-ID permission bit enabled;

  · the target file does not have both its set-group-ID and group-executable permission bits enabled; and

  · the caller has permission to read and write the target

file (either via the file's permissions mask or because it has suitable capabilities).

The default value in this file is 0. Setting the value to 1 prevents a longstanding class of security issues caused by hard-link-based time-of-check, time-of-use races, most commonly seen in world-writable directories such as /tmp. The common method of exploiting this flaw is to cross privilege boundaries when following a given hard link (i.e., a root process follows a hard link created by another user). Additionally, on systems without separated partitions, this stops unauthorized users from "pinning" vulnerable set-user-ID and set-group-ID files against being upgraded by the administrator, or linking to special files.

/proc/sys/fs/protected_symlinks (since Linux 3.6)
When the value in this file is 0, no restrictions are placed on following symbolic links (i.e., this is the historical behaviour before Linux 3.6). When the value in this file is 1, symbolic links are followed only in the following circumstances:

* the filesystem UID of the process following the link matches the owner (UID) of the symbolic link (as described in **credentials**(7), a process's filesystem UID is normally the same as its effective UID);

* the link is not in a sticky world-writable directory; or

* the symbolic link and and its parent directory have the same owner (UID)

A system call that fails to follow a symbolic link because of the above restrictions returns the error **EACCES** in errno.

The default value in this file is 0. Setting the value to 1 avoids a longstanding class of security issues based on time-of-check, time-of-use races when accessing symbolic links.

/proc/sys/fs/suid_dumpable (since Linux 2.6.13)
The value in this file determines whether core dump files are produced for set-user-ID or otherwise protected/tainted binaries. Three different integer values can be specified:

0 (default)
This provides the traditional (pre-Linux 2.6.13) behavior. A core dump will not be produced for a process which has changed credentials (by calling **seteuid**(2), **setgid**(2), or similar, or by executing a set-user-ID or set-group-ID program) or whose binary does not have read permission enabled.

1 ("debug")
All processes dump core when possible. The core dump is owned by the filesystem user ID of the dumping process and no security is applied. This is intended for system debugging situations only. Ptrace is unchecked.

2 ("suidsafe")
Any binary which normally would not be dumped (see "0" above) is dumped readable by root only. This allows the user to remove the core dump file but not to read it. For security reasons core dumps in this mode will not overwrite one another or other files. This mode is appropriate when administrators are attempting to debug problems in a normal environment.

Additionally, since Linux 3.6, /proc/sys/kernel/core_pattern must either be an absolute pathname or a pipe command, as detailed in **core**(5). Warnings will be written to the kernel log if core_pattern does not follow these rules, and no core dump will be produced.

/proc/sys/fs/super-max
This file controls the maximum number of superblocks, and thus the maximum number of mounted filesystems the kernel can have. You need increase only super-max if you need to mount more filesystems than the current value in super-max allows you to.

/proc/sys/fs/super-nr
This file contains the number of filesystems currently mounted.

/proc/sys/kernel
This directory contains files controlling a range of kernel parameters, as described below.

/proc/sys/kernel/acct
This file contains three numbers: highwater, lowwater, and frequency. If BSD-style process accounting is enabled these values control its behavior. If free space on filesystem where the log lives goes below lowwater percent accounting suspends. If free space gets above highwater percent accounting resumes. frequency determines how often the kernel checks the amount of free space (value is in seconds). Default values are 4, 2 and 30. That is, suspend accounting if 2% or less space is free; resume it if 4% or more space is free; consider information about amount of free space valid for 30 seconds.

/proc/sys/kernel/cap_last_cap (since Linux 3.2)
See **capabilities**(7).

/proc/sys/kernel/cap-bound (from Linux 2.2 to 2.6.24)
This file holds the value of the kernel capability bounding set (expressed as a signed decimal number). This set is ANDed against the capabilities permitted to a process during **execve**(2). Starting with Linux 2.6.25, the system-wide capability bounding set disappeared, and was replaced by a per-thread bounding set; see **capabilities**(7).

/proc/sys/kernel/core_pattern
See **core**(5).

/proc/sys/kernel/core_uses_pid
See **core**(5).

/proc/sys/kernel/ctrl-alt-del
This file controls the handling of Ctrl-Alt-Del from the keyboard. When the value in this file is 0, Ctrl-Alt-Del is trapped and sent to the **init**(8) program to handle a graceful restart. When the value is greater than zero, 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.

/proc/sys/kernel/dmesg_restrict (since Linux 2.6.37)
The value in this file determines who can see kernel syslog contents. A value of 0 in this file imposes no restrictions. If the value is 1, only privileged users can read the kernel syslog. (See **syslog**(2) for more details.) Since Linux 3.4, only users with the **CAP_SYS_ADMIN** capability may change the value in this file.

/proc/sys/kernel/domainname and /proc/sys/kernel/hostname
can be used to set the NIS/YP domainname and the hostname of your box in exactly the same way as the commands **domainname**(1) and **hostname**(1), that is:

```
# 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.

/proc/sys/kernel/hotplug
This file contains the path for the hotplug policy agent. The default value in this file is /sbin/hotplug.

/proc/sys/kernel/htab-reclaim
(PowerPC only) If this file is set to a nonzero value, the PowerPC htab (see kernel file

Documentation/powerpc/ppc_htab.txt) is pruned each time the system hits the idle loop.

/proc/sys/kernel/kptr_restrict (since Linux 2.6.38)
The value in this file determines whether kernel addresses are exposed via /proc files and other interfaces. A value of 0 in this file imposes no restrictions. If the value is 1, kernel pointers printed using the %pK format specifier will be replaced with zeros unless the user has the **CAP_SYSLOG** capability. If the value is 2, kernel pointers printed using the %pK format specifier will be replaced with zeros regardless of the user's capabilities. The initial default value for this file was 1, but the default was changed to 0 in Linux 2.6.39. Since Linux 3.4, only users with the **CAP_SYS_ADMIN** capability can change the value in this file.

/proc/sys/kernel/l2cr
(PowerPC only) This file contains a flag that controls the L2 cache of G3 processor boards. If 0, the cache is disabled. Enabled if nonzero.

/proc/sys/kernel/modprobe
This file contains the path for the kernel module loader. The default value is /sbin/modprobe. The file is present only if the kernel is built with the **CONFIG_MODULES** (**CONFIG_KMOD** in Linux 2.6.26 and earlier) option enabled. It is described by the Linux kernel source file Documentation/kmod.txt (present only in kernel 2.4 and earlier).

/proc/sys/kernel/modules_disabled (since Linux 2.6.31)
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. The file is present only if the kernel is built with the **CONFIG_MODULES** option enabled.

/proc/sys/kernel/msgmax
This file defines a system-wide limit specifying the maximum number of bytes in a single message written on a System V message queue.

/proc/sys/kernel/msgmni (since Linux 2.4)
This file defines the system-wide limit on the number of message queue identifiers.

/proc/sys/kernel/msgmnb
This file defines a system-wide parameter used to initialize the msg_qbytes setting for subsequently created message queues. The msg_qbytes setting specifies the maximum number of bytes that may be written to the message queue.

/proc/sys/kernel/ostype and /proc/sys/kernel/osrelease
These files give substrings of /proc/version.

/proc/sys/kernel/overflowgid and /proc/sys/kernel/overflowuid
These files duplicate the files /proc/sys/fs/overflowgid and /proc/sys/fs/overflowuid.

/proc/sys/kernel/panic
This file gives read/write access to the kernel variable panic_timeout. If this is zero, the kernel will loop on a panic; if nonzero it indicates that the kernel should autoreboot after this number of seconds. When you use the software watchdog device driver, the recommended setting is 60.

/proc/sys/kernel/panic_on_oops (since Linux 2.5.68)
This file controls the kernel's behavior when an oops or BUG is encountered. If this file contains 0, then the system tries to continue operation. If it contains 1, then the system delays a few seconds (to give klogd time to record the oops output) and then panics. If the /proc/sys/kernel/panic file is also nonzero then the machine will be rebooted.

/proc/sys/kernel/pid_max (since Linux 2.5.34)
This file specifies the value at which PIDs wrap around (i.e., the value in this file is one greater than the maximum PID). The default value for this file, 32768, results in the same range of PIDs as on earlier kernels. On 32-bit platforms, 32768 is the maximum value for pid_max. On 64-bit systems, pid_max can be set to any value up to 2^22 (**PID_MAX_LIMIT**, approximately

4 million).

/proc/sys/kernel/powersave-nap (PowerPC only)
This file contains a flag. If set, Linux-PPC will use the "nap" mode of powersaving, otherwise the "doze" mode will be used.

/proc/sys/kernel/printk
The four values in this file are console_loglevel, default_message_loglevel, minimum_console_level, and default_console_loglevel. These values influence printk() behavior when printing or logging error messages. See **syslog**(2) for more info on the different loglevels. Messages with a higher priority than console_loglevel will be printed to the console. Messages without an explicit priority will be printed with priority default_message_level. minimum_console_loglevel is the minimum (highest) value to which console_loglevel can be set. default_console_loglevel is the default value for console_loglevel.

/proc/sys/kernel/pty (since Linux 2.6.4)
This directory contains two files relating to the number of UNIX 98 pseudoterminals (see **pts**(4)) on the system.

/proc/sys/kernel/pty/max
This file defines the maximum number of pseudoterminals.

/proc/sys/kernel/pty/nr
This read-only file indicates how many pseudoterminals are currently in use.

/proc/sys/kernel/random
This directory contains various parameters controlling the operation of the file /dev/random. See **random**(4) for further information.

/proc/sys/kernel/real-root-dev
This file is documented in the Linux kernel source file Documentation/initrd.txt.

/proc/sys/kernel/reboot-cmd (Sparc only)
This file 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?

/proc/sys/kernel/rtsig-max
(Only in kernels up to and including 2.6.7; see **setrlimit**(2)) This file can be used to tune the maximum number of POSIX real-time (queued) signals that can be outstanding in the system.

/proc/sys/kernel/rtsig-nr
(Only in kernels up to and including 2.6.7.) This file shows the number POSIX real-time signals currently queued.

/proc/sys/kernel/sched_rr_timeslice_ms (since Linux 3.9)
See **sched_rr_get_interval**(2).

/proc/sys/kernel/sem (since Linux 2.4)
This file contains 4 numbers defining limits for System V IPC semaphores. These fields are, in order:

SEMMSL The maximum semaphores per semaphore set.

SEMMNS A system-wide limit on the number of semaphores in all semaphore sets.

SEMOPM The maximum number of operations that may be specified in a **semop**(2) call.

SEMMNI A system-wide limit on the maximum number of semaphore identifiers.

/proc/sys/kernel/sg-big-buff
This file shows the size of the generic SCSI device (sg) buffer. You can't tune it just yet, but you could change it at compile time by editing include/scsi/sg.h and changing the value of **SG_BIG_BUFF**. However, there shouldn't be any reason to change this value.

/proc/sys/kernel/shm_rmid_forced (since Linux 3.1)
If this file is set to 1, all System V shared memory segments will be marked for destruction as soon as the number of attached

processes falls to zero; in other words, it is no longer possible to create shared memory segments that exist independently of any attached process.

The effect is as though a **shmctl**(2) **IPC_RMID** is performed on all existing segments as well as all segments created in the future (until this file is reset to 0). Note that existing segments that are attached to no process will be immediately destroyed when this file is set to 1. Setting this option will also destroy segments that were created, but never attached, upon termination of the process that created the segment with **shmget**(2).

Setting this file to 1 provides a way of ensuring that all System V shared memory segments are counted against the resource usage and resource limits (see the description of **RLIMIT_AS** in **getrlimit**(2)) of at least one process.

Because setting this file to 1 produces behavior that is nonstandard and could also break existing applications, the default value in this file is 0. Only set this file to 1 if you have a good understanding of the semantics of the applications using System V shared memory on your system.

/proc/sys/kernel/shmall
This file contains the system-wide limit on the total number of pages of System V shared memory.

/proc/sys/kernel/shmmax
This file can be used to query and set the run-time limit on the maximum (System V IPC) 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**.

/proc/sys/kernel/shmmni (since Linux 2.4)
This file specifies the system-wide maximum number of System V shared memory segments that can be created.

/proc/sys/kernel/sysrq
This file controls the functions allowed to be invoked by the SysRq key. By default, the file contains 1 meaning that every possible SysRq request is allowed (in older kernel versions, SysRq was disabled by default, and you were required to specifically enable it at run-time, but this is not the case any more). Possible values in this file are:

```
   0 - disable sysrq completely
   1 - enable all functions of sysrq
  >1 - bit mask of allowed sysrq functions, as follows:
         2 - enable control of console logging level
         4 - enable control of keyboard (SAK, unraw)
         8 - enable debugging dumps of processes etc.
        16 - enable sync command
        32 - enable remount read-only
        64 - enable signalling of processes (term, kill, oom-kill)
       128 - allow reboot/poweroff
       256 - allow nicing of all real-time tasks
```

This file is present only if the **CONFIG_MAGIC_SYSRQ** kernel configuration option is enabled. For further details see the Linux kernel source file Documentation/sysrq.txt.

/proc/sys/kernel/version
This file contains a string like:

```
    #5 Wed Feb 25 21:49:24 MET 1998
```

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.

/proc/sys/kernel/threads-max (since Linux 2.3.11)
This file specifies the system-wide limit on the number of threads (tasks) that can be created on the system.

/proc/sys/kernel/zero-paged (PowerPC only)
This file contains a flag. When enabled (nonzero), Linux-PPC will pre-zero pages in the idle loop, possibly speeding up get_free_pages.

/proc/sys/net
This directory contains networking stuff. Explanations for some of the files under this directory can be found in **tcp**(7) and **ip**(7).

/proc/sys/net/core/somaxconn
This file defines a ceiling value for the backlog argument of **listen**(2); see the **listen**(2) manual page for details.

/proc/sys/proc
This directory may be empty.

/proc/sys/sunrpc
This directory supports Sun remote procedure call for network filesystem (NFS). On some systems, it is not present.

/proc/sys/vm
This directory contains files for memory management tuning, buffer and cache management.

/proc/sys/vm/drop_caches (since Linux 2.6.16)
Writing to this file causes the kernel to drop clean caches, dentries, and inodes from memory, causing that memory to become free. This can be useful for memory management testing and performing reproducible filesystem benchmarks. Because writing to this file causes the benefits of caching to be lost, it can degrade overall system performance.

To free pagecache, use:

```
echo 1 > /proc/sys/vm/drop_caches
```

To free dentries and inodes, use:

```
echo 2 > /proc/sys/vm/drop_caches
```

To free pagecache, dentries and inodes, use:

```
echo 3 > /proc/sys/vm/drop_caches
```

Because writing to this file is a nondestructive operation and dirty objects are not freeable, the user should run **sync**(8) first.

/proc/sys/vm/legacy_va_layout (since Linux 2.6.9)
If nonzero, this disables the new 32-bit memory-mapping layout; the kernel will use the legacy (2.4) layout for all processes.

/proc/sys/vm/memory_failure_early_kill (since Linux 2.6.32)
Control how to kill processes when an uncorrected memory error (typically a 2-bit error in a memory module) that cannot be handled by the kernel is detected in the background by hardware. In some cases (like the page still having a valid copy on disk), the kernel will handle the failure transparently without affecting any applications. But if there is no other up-to-date copy of the data, it will kill processes to prevent any data corruptions from propagating.

The file has one of the following values:

1: Kill all processes that have the corrupted-and-not-reloadable page mapped as soon as the corruption is detected. Note this is not supported for a few types of pages, like kernel internally allocated data or the swap cache, but works for the majority of user pages.

0: Only unmap the corrupted page from all processes and kill only a process that tries to access it.

The kill is performed using a **SIGBUS** signal with si_code set to **BUS_MCEERR_AO**. Processes can handle this if they want to; see **sigaction**(2) for more details.

This feature is active only on architectures/platforms with advanced machine check handling and depends on the hardware capabilities.

Applications can override the memory_failure_early_kill setting individually with the **prctl**(2) **PR_MCE_KILL** operation.

Only present if the kernel was configured with

CONFIG_MEMORY_FAILURE.

/proc/sys/vm/memory_failure_recovery (since Linux 2.6.32)
Enable memory failure recovery (when supported by the platform)

1: Attempt recovery.

0: Always panic on a memory failure.

Only present if the kernel was configured with **CONFIG_MEMORY_FAILURE.**

/proc/sys/vm/oom_dump_tasks (since Linux 2.6.25)
Enables a system-wide task dump (excluding kernel threads) to be produced when the kernel performs an OOM-killing. The dump includes the following information for each task (thread, process): thread ID, real user ID, thread group ID (process ID), virtual memory size, resident set size, the CPU that the task is scheduled on, oom_adj score (see the description of /proc/[pid]/oom_adj), and command name. This is helpful to determine why the OOM-killer was invoked and to identify the rogue task that caused it.

If this contains the value zero, this information is suppressed. On very large systems with thousands of tasks, it may not be feasible to dump the memory state information for each one. Such systems should not be forced to incur a performance penalty in OOM situations when the information may not be desired.

If this is set to nonzero, this information is shown whenever the OOM-killer actually kills a memory-hogging task.

The default value is 0.

/proc/sys/vm/oom_kill_allocating_task (since Linux 2.6.24)
This enables or disables killing the OOM-triggering task in out-of-memory situations.

If this is set to zero, the OOM-killer will scan through the entire tasklist and select a task based on heuristics to kill. This normally selects a rogue memory-hogging task that frees up a large amount of memory when killed.

If this is set to nonzero, the OOM-killer simply kills the task that triggered the out-of-memory condition. This avoids a possibly expensive tasklist scan.

If /proc/sys/vm/panic_on_oom is nonzero, it takes precedence over whatever value is used in /proc/sys/vm/oom_kill_allocating_task.

The default value is 0.

/proc/sys/vm/overcommit_memory
This file contains the kernel virtual memory accounting mode. Values are:

0: heuristic overcommit (this is the default)
1: always overcommit, never check
2: always check, never overcommit

In mode 0, calls of **mmap**(2) with **MAP_NORESERVE** are not checked, and the default check is very weak, leading to the risk of getting a process "OOM-killed". Under Linux 2.4 any nonzero value implies mode 1. In mode 2 (available since Linux 2.6), the total virtual address space on the system is limited to (SS + RAM*(r/100)), where SS is the size of the swap space, and RAM is the size of the physical memory, and r is the contents of the file /proc/sys/vm/overcommit_ratio.

/proc/sys/vm/overcommit_ratio
See the description of /proc/sys/vm/overcommit_memory.

/proc/sys/vm/panic_on_oom (since Linux 2.6.18)
This enables or disables a kernel panic in an out-of-memory situation.

If this file is set to the value 0, the kernel's OOM-killer will kill some rogue process. Usually, the OOM-killer is able to kill a rogue process and the system will survive.

If this file is set to the value 1, then the kernel normally panics when out-of-memory happens. However, if a process limits allocations to certain nodes using memory policies (**mbind**(2) **MPOL_BIND**) or cpusets (**cpuset**(7)) and those nodes reach memory exhaustion status, one process may be killed by the OOM-killer. No panic occurs in this case: because other nodes' memory may be free, this means the system as a whole may not have reached an out-of-memory situation yet.

If this file is set to the value 2, the kernel always panics when an out-of-memory condition occurs.

The default value is 0. 1 and 2 are for failover of clustering. Select either according to your policy of failover.

/proc/sys/vm/swappiness
The value in this file controls how aggressively the kernel will swap memory pages. Higher values increase aggressiveness, lower values decrease aggressiveness. The default value is 60.

/proc/sysrq-trigger (since Linux 2.4.21)
Writing a character to this file triggers the same SysRq function as typing ALT-SysRq-<character> (see the description of /proc/sys/kernel/sysrq). This file is normally writable only by root. For further details see the Linux kernel source file Documentation/sysrq.txt.

/proc/sysvipc
Subdirectory containing the pseudo-files msg, sem and shm. These files list the System V Interprocess Communication (IPC) objects (respectively: message queues, semaphores, and shared memory) that currently exist on the system, providing similar information to that available via **ipcs**(1). These files have headers and are formatted (one IPC object per line) for easy understanding. **svipc**(7) provides further background on the information shown by these files.

/proc/tty
Subdirectory containing the pseudo-files and subdirectories for tty drivers and line disciplines.

/proc/uptime
This file contains two numbers: the uptime of the system (seconds), and the amount of time spent in idle process (seconds).

/proc/version
This string identifies the kernel version that is currently running. It includes the contents of /proc/sys/kernel/ostype, /proc/sys/kernel/osrelease and /proc/sys/kernel/version. For example:

```
Linux version 1.0.9 (quinlan@phaze) #1 Sat May 14 01:51:54 EDT 1994
```

/proc/vmstat (since Linux 2.6)
This file displays various virtual memory statistics.

/proc/zoneinfo (since Linux 2.6.13)
This file display information about memory zones. This is useful for analyzing virtual memory behavior.

NOTES

Many strings (i.e., the environment and command line) are in the internal format, with subfields terminated by null bytes ('\0'), so you may find that things are more readable if you use od -c or tr "\000" "\n" to read them. Alternatively, echo `cat <file>` works well.

This manual page is incomplete, possibly inaccurate, and is the kind of thing that needs to be updated very often.

SEE ALSO

cat(1), **dmesg**(1), **find**(1), **free**(1), **ps**(1), **tr**(1), **uptime**(1), **chroot**(2), **mmap**(2), **readlink**(2), **syslog**(2), **slabinfo**(5), **hier**(7), **time**(7), **arp**(8), **hdparm**(8), **ifconfig**(8), **init**(8), **lsmod**(8), **lspci**(8), **mount**(8), **netstat**(8), **procinfo**(8), **route**(8), **sysctl**(8)

The Linux kernel source files: Documentation/filesystems/proc.txt and Documentation/sysctl/vm.txt.

COLOPHON

This page is part of release 3.54 of the Linux man-pages project. A description of the project, and information about reporting bugs, can be found at http://www.kernel.org/doc/man-pages/.