proc(5) — Linux manual page

NAME | DESCRIPTION | NOTES | SEE ALSO | COLOPHON

NAME top

proc - process information pseudo-filesystem

DESCRIPTION top

The **proc** filesystem is a pseudo-filesystem which provides an interface to kernel data structures. It is commonly mounted at */proc*. Typically, it is mounted automatically by the system, but it can also be mounted manually using a command such as:

```
mount -t proc proc /proc
```

Most of the files in the **proc** filesystem are read-only, but some files are writable, allowing kernel variables to be changed.

Mount options

The **proc** filesystem supports the following mount options:

hidepid=*n* (since Linux 3.3)
This option controls who can access the information in */proc/[pid]* directories. The argument, *n*, is one of the following values:

0 Everybody may access all */proc/[pid]* directories. This is the traditional behavior, and the default if this mount option is not specified.

1 Users may not access files and subdirectories inside any */proc/[pid]* directories but their own (the */proc/[pid]* directories themselves remain visible). Sensitive files such as */proc/[pid]/cmdline* and */proc/[pid]/status* are now protected against other users. This makes it impossible to learn whether any user is running a specific program (so long as the program doesn't otherwise reveal itself by its behavior).

2 As for mode 1, but in addition the */proc/[pid]* directories belonging to other users become invisible. This means that */proc/[pid]* entries can no longer be used to discover the PIDs on the system. This doesn't hide the fact that a process with a specific PID value exists (it can be learned by other means, for example, by "kill -0 $PID"), but it hides a process's UID and GID, which could otherwise be learned by employing

stat(2) on a */proc/[pid]* directory. This greatly complicates an attacker's task of gathering information about running processes (e.g., discovering whether some daemon is running with elevated privileges, whether another user is running some sensitive program, whether other users are running any program at all, and so on).

gid=*gid* (since Linux 3.3)
Specifies the ID of a group whose members are authorized to learn process information otherwise prohibited by **hidepid** (i.e., users in this group behave as though */proc* was mounted with *hidepid=0*). This group should be used instead of approaches such as putting nonroot users into the sudoers(5) file.

Overview

Underneath */proc*, there are the following general groups of files and subdirectories:

/proc/[pid] subdirectories
Each one of these subdirectories contains files and subdirectories exposing information about the process with the corresponding process ID.

Underneath each of the */proc/[pid]* directories, a *task* subdirectory contains subdirectories of the form *task/[tid]*, which contain corresponding information about each of the threads in the process, where *tid* is the kernel thread ID of the thread.

The */proc/[pid]* subdirectories are visible when iterating through */proc* with getdents(2) (and thus are visible when one uses ls(1) to view the contents of */proc*).

/proc/[tid] subdirectories
Each one of these subdirectories contains files and subdirectories exposing information about the thread with the corresponding thread ID. The contents of these directories are the same as the corresponding */proc/[pid]/task/[tid]* directories.

The */proc/[tid]* subdirectories are *not* visible when iterating through */proc* with getdents(2) (and thus are *not* visible when one uses ls(1) to view the contents of */proc*).

/proc/self
When a process accesses this magic symbolic link, it resolves to the process's own */proc/[pid]* directory.

/proc/thread-self
When a thread accesses this magic symbolic link, it resolves to the process's own */proc/self/task/[tid]* directory.

*/proc/[a-z]**
Various other files and subdirectories under */proc* expose system-wide information.

All of the above are described in more detail below.

Files and directories

The following list provides details of 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 */proc/[pid]* subdirectory contains the pseudo-files and directories described below.

  The files inside each */proc/[pid]* directory are normally owned by the effective user and effective group ID of the process. However, as a security measure, the ownership is made *root:root* if the process's "dumpable" attribute is set to a value other than 1.

  Before Linux 4.11, *root:root* meant the "global" root user ID and group ID (i.e., UID 0 and GID 0 in the initial user namespace). Since Linux 4.11, if the process is in a noninitial user namespace that has a valid mapping for user (group) ID 0 inside the namespace, then the user (group) ownership of the files under */proc/[pid]* is instead made the same as the root user (group) ID of the namespace. This means that inside a container, things work as expected for the container "root" user.

  The process's "dumpable" attribute may change for the following reasons:

  * The attribute was explicitly set via the prctl(2) **PR_SET_DUMPABLE** operation.

  * The attribute was reset to the value in the file */proc/sys/fs/suid_dumpable* (described below), for the reasons described in prctl(2).

  Resetting the "dumpable" attribute to 1 reverts the ownership of the */proc/[pid]/** files to the process's effective UID and GID. Note, however, that if the effective UID or GID is subsequently modified, then the "dumpable" attribute may be reset, as described in prctl(2). Therefore, it may be desirable to reset the "dumpable" attribute *after* making any desired changes to the process's effective UID or GID.

/proc/[pid]/cmdline
: This read-only file 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.

  If, after an execve(2), the process modifies its *argv* strings, those changes will show up here. This is not the same thing as modifying the *argv* array.

  Furthermore, a process may change the memory location that this file refers via prctl(2) operations such as **PR_SET_MM_ARG_START**.

  Think of this file as the command line that the process wants you to see.

/proc/[pid]/comm (since Linux 2.6.33)
This file exposes the process's *comm* value—that is, the command name associated with the process. Different threads in the same process may have different *comm* values, accessible via */proc/[pid]/task/[tid]/comm*. A thread may modify its *comm* value, or that of any of other thread in the same thread group (see the discussion of **CLONE_THREAD** in clone(2)), by writing to the file */proc/self/task/[tid]/comm*. Strings longer than **TASK_COMM_LEN** (16) characters (including the terminating null byte) are silently truncated.

This file provides a superset of the prctl(2) **PR_SET_NAME** and **PR_GET_NAME** operations, and is employed by pthread_setname_np(3) when used to rename threads other than the caller. The value in this file is used for the *%e* specifier in */proc/sys/kernel/core_pattern*; see core(5).

/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; 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)).

Permission to dereference or read (readlink(2)) this symbolic link is governed by a ptrace access mode **PTRACE_MODE_READ_FSCREDS** check; see ptrace(2).

/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 that is being run by process [pid]. If the pathname has been unlinked, the symbolic link will contain the string '(deleted)' appended to the original pathname. 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)).

Permission to dereference or read (readlink(2)) this symbolic link is governed by a ptrace access mode **PTRACE_MODE_READ_FSCREDS** check; see ptrace(2).

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, and so on.

  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 bpf(2), epoll_create(2), eventfd(2), inotify_init(2), perf_event_open(2), signalfd(2), timerfd_create(2), and userfaultfd(2)), the entry will be a symbolic link with contents of the form

      anon_inode:<file-type>

  In many cases (but not all), 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 take a filename as a command-line argument, but don't take input from standard input if no argument is supplied, and programs that write to a file named as a command-line argument, but don't send their output to standard output if no argument is supplied, can nevertheless be made to use standard input or standard output by using */proc/[pid]/fd* files as command-line arguments. 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 ...
```

Permission to dereference or read (readlink(2)) the symbolic links in this directory is governed by a ptrace access mode **PTRACE_MODE_READ_FSCREDS** check; see ptrace(2).

Note that for file descriptors referring to inodes (pipes and sockets, see above), those inodes still have permission bits and ownership information distinct from those of the */proc/[pid]/fd* entry, and that the owner may differ from the user and group IDs of the process. An unprivileged process may lack permissions to open them, as in this example:

```
$ echo test | sudo -u nobody cat
test
$ echo test | sudo -u nobody cat /proc/self/fd/0
cat: /proc/self/fd/0: Permission denied
```

File descriptor 0 refers to the pipe created by the shell and owned by that shell's user, which is not *nobody*, so **cat** does not have permission to create a new file descriptor to read from that inode, even though it can still read from its existing file descriptor 0.

/proc/[pid]/fdinfo/ (since Linux 2.6.22)
This is a subdirectory containing one entry for each file which the process has open, named by its file descriptor. The files in this directory are readable only by the owner of the process. The contents of each file can be read to obtain information about the corresponding file descriptor. The content depends on the type of file referred to by the corresponding file descriptor.

For regular files and directories, we see something like:

```
$ cat /proc/12015/fdinfo/4
pos:    1000
flags:  01002002
mnt_id: 21
```

The fields are as follows:

pos This is a decimal number showing the file offset.

flags This is an octal number that displays the file access mode and file status flags (see open(2)). If the close-on-exec file descriptor flag is set, then *flags* will also include the value **O_CLOEXEC**.

Before Linux 3.1, this field incorrectly displayed the setting of **O_CLOEXEC** at the time the file was opened, rather than the current setting of the close-on-exec flag.

mnt_id This field, present since Linux 3.15, is the ID of the mount containing this file. See the description of */proc/[pid]/mountinfo*.

For eventfd file descriptors (see eventfd(2)), we see

(since Linux 3.8) the following fields:

```
pos: 0
flags:    02
mnt_id:   10
eventfd-count:               40
```

eventfd-count is the current value of the eventfd counter, in hexadecimal.

For epoll file descriptors (see epoll(7)), we see (since Linux 3.8) the following fields:

```
pos: 0
flags:    02
mnt_id:   10
tfd:        9 events:       19 data: 74253d2500000009
tfd:        7 events:       19 data: 74253d2500000007
```

Each of the lines beginning *tfd* describes one of the file descriptors being monitored via the epoll file descriptor (see epoll_ctl(2) for some details). The *tfd* field is the number of the file descriptor. The *events* field is a hexadecimal mask of the events being monitored for this file descriptor. The *data* field is the data value associated with this file descriptor.

For signalfd file descriptors (see signalfd(2)), we see (since Linux 3.8) the following fields:

```
pos: 0
flags:    02
mnt_id:   10
sigmask:  0000000000000006
```

sigmask is the hexadecimal mask of signals that are accepted via this signalfd file descriptor. (In this example, bits 2 and 3 are set, corresponding to the signals **SIGINT** and **SIGQUIT**; see signal(7).)

For inotify file descriptors (see inotify(7)), we see (since Linux 3.8) the following fields:

```
pos: 0
flags:    00
mnt_id:   11
inotify wd:2 ino:7ef82a sdev:800001 mask:800afff ignored_mask:0
fhandle-bytes:8 fhandle-type:1 f_handle:2af87e00220ffd73
inotify wd:1 ino:192627 sdev:800001 mask:800afff ignored_mask:0
fhandle-bytes:8 fhandle-type:1 f_handle:27261900802dfd73
```

Each of the lines beginning with "inotify" displays information about one file or directory that is being monitored. The fields in this line are as follows:

wd A watch descriptor number (in decimal).

ino The inode number of the target file (in hexadecimal).

sdev The ID of the device where the target file resides (in hexadecimal).

mask The mask of events being monitored for the target file (in hexadecimal).

If the kernel was built with exportfs support, the path to the target file is exposed as a file handle, via three hexadecimal fields: *fhandle-bytes*, *fhandle-type*, and *f_handle*.

For fanotify file descriptors (see fanotify(7)), we see (since Linux 3.8) the following fields:

```
pos: 0
flags:    02
mnt_id:   11
fanotify flags:0 event-flags:88002
fanotify ino:19264f sdev:800001 mflags:0 mask:1 ignored_mask:0 fhandle-
bytes:8 fhandle-type:1 f_handle:4f261900a82dfd73
```

The fourth line displays information defined when the fanotify group was created via fanotify_init(2):

flags The *flags* argument given to fanotify_init(2) (expressed in hexadecimal).

event-flags
The *event_f_flags* argument given to fanotify_init(2) (expressed in hexadecimal).

Each additional line shown in the file contains information about one of the marks in the fanotify group. Most of these fields are as for inotify, except:

mflags The flags associated with the mark (expressed in hexadecimal).

mask The events mask for this mark (expressed in hexadecimal).

ignored_mask
The mask of events that are ignored for this mark (expressed in hexadecimal).

For details on these fields, see fanotify_mark(2).

For timerfd file descriptors (see **timerfd**(2)), we see (since Linux 3.17) the following fields:

```
pos:    0
flags:  02004002
mnt_id: 13
clockid: 0
ticks: 0
settime flags: 03
it_value: (7695568592, 640020877)
it_interval: (0, 0)
```

clockid
This is the numeric value of the clock ID (corresponding to one of the **CLOCK_*** constants defined via *<time.h>*) that is used to mark the progress of the timer (in this example, 0 is **CLOCK_REALTIME**).

ticks This is the number of timer expirations that have occurred, (i.e., the value that read(2) on it would return).

settime flags
This field lists the flags with which the timerfd was last armed (see timerfd_settime(2)), in octal (in this example, both **TFD_TIMER_ABSTIME** and **TFD_TIMER_CANCEL_ON_SET** are set).

it_value
This field contains the amount of time until the timer will next expire, expressed in seconds and nanoseconds. This is always expressed as a relative value, regardless of whether the timer was created using the **TFD_TIMER_ABSTIME** flag.

it_interval
This field contains the interval of the timer, in seconds and nanoseconds. (The *it_value* and *it_interval* fields contain the values that timerfd_gettime(2) on this file descriptor would return.)

/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).

Permission to access this file is governed by a ptrace access mode **PTRACE_MODE_ATTACH_FSCREDS** check; see ptrace(2).

/proc/[pid]/mountinfo (since Linux 2.6.26)
This file contains information about mounts in the process's mount namespace (see mount_namespaces(7)). It supplies various information (e.g., propagation state, root of mount for bind mounts, identifier for each mount and its parent) that is missing from the (older) */proc/[pid]/mounts* file, and fixes various other problems with that file (e.g., nonextensibility, failure to distinguish per-mount versus per-superblock options).

The file 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: a unique ID for the mount (may be reused after umount(2)).

(2) parent ID: the ID of the parent mount (or of self for the root of this mount namespace's mount tree).

If a new mount is stacked on top of a previous existing mount (so that it hides the existing mount) at pathname P, then the parent of the new mount is the previous mount at that location. Thus, when looking at all the mounts stacked at a particular location, the top-most mount is the one that is not

the parent of any other mount at the same location. (Note, however, that this top-most mount will be accessible only if the longest path subprefix of P that is a mount point is not itself hidden by a stacked mount.)

If the parent mount lies outside the process's root directory (see chroot(2)), the ID shown here won't have a corresponding record in *mountinfo* whose mount ID (field 1) matches this parent mount ID (because mounts that lie outside the process's root directory are not shown in *mountinfo*). As a special case of this point, the process's root mount may have a parent mount (for the initramfs filesystem) that lies outside the process's root directory, and an entry for that mount will not appear in *mountinfo*.

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

(4) root: the pathname of the directory in the filesystem which forms the root of this mount.

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

(6) mount options: per-mount options (see mount(2)).

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

(8) separator: the end of the optional fields is marked by a single hyphen.

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

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

(11) super options: per-superblock options (see mount(2)).

Currently, the possible optional fields are *shared*, *master*, *propagate_from*, and *unbindable*. See mount_namespaces(7) for a description of these fields. Parsers should ignore all unrecognized optional fields.

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 file lists all the filesystems currently mounted in the process's mount namespace (see mount_namespaces(7)). 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 having an exceptional condition, and poll(2) and epoll_wait(2) mark the file as having a priority event (**POLLPRI**). (Before Linux 2.6.30, a change in this file was indicated by the file descriptor being

marked as readable for select(2), and being marked as having an error condition for poll(2) and epoll_wait(2).)

/proc/[pid]/mountstats (since Linux 2.6.17)
This file exports information (statistics, configuration information) about the mounts in the process's mount namespace (see mount_namespaces(7)). Lines in this file have the form:

```
device /dev/sda7 mounted on /home with fstype ext3 [stats]
(       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]/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
Umask:  0022
State:  S (sleeping)
Tgid:   17248
Ngid:   0
Pid:    17248
PPid:   17200
TracerPid:      0
Uid:    1000    1000    1000    1000
Gid:    100     100     100     100
FDSize: 256
Groups: 16 33 100
NStgid: 17248
NSpid:  17248
NSpgid: 17248
NSsid:  17200
VmPeak:     131168 kB
VmSize:     131168 kB
VmLck:           0 kB
VmPin:           0 kB
VmHWM:       13484 kB
VmRSS:       13484 kB
RssAnon:     10264 kB
RssFile:      3220 kB
RssShmem:        0 kB
VmData:      10332 kB
VmStk:         136 kB
VmExe:         992 kB
VmLib:        2104 kB
VmPTE:          76 kB
```

```
    VmPMD:          12 kB
    VmSwap:          0 kB
    HugetlbPages:          0 kB        # 4.4
    CoreDumping:   0                       # 4.15
    Threads:        1
    SigQ:   0/3067
    SigPnd: 0000000000000000
    ShdPnd: 0000000000000000
    SigBlk: 0000000000010000
    SigIgn: 0000000000384004
    SigCgt: 000000004b813efb
    CapInh: 0000000000000000
    CapPrm: 0000000000000000
    CapEff: 0000000000000000
    CapBnd: ffffffffffffffff
    CapAmb:   0000000000000000
    NoNewPrivs:     0
    Seccomp:        0
    Speculation_Store_Bypass:       vulnerable
    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. Strings longer than **TASK_COMM_LEN** (16) characters (including the terminating null byte) are silently truncated.

Umask Process umask, expressed in octal with a leading zero; see umask(2). (Since Linux 4.7.)

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).

Ngid NUMA group ID (0 if none; since Linux 3.13).

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.

NStgid Thread group ID (i.e., PID) in each of the PID namespaces of which *[pid]* is a member. The

leftmost entry shows the value with respect to the PID namespace of the process that mounted this procfs (or the root namespace if mounted by the kernel), followed by the value in successively nested inner namespaces. (Since Linux 4.1.)

NSpid Thread ID in each of the PID namespaces of which *[pid]* is a member. The fields are ordered as for *NStgid*. (Since Linux 4.1.)

NSpgid Process group ID in each of the PID namespaces of which *[pid]* is a member. The fields are ordered as for *NStgid*. (Since Linux 4.1.)

NSsid descendant namespace session ID hierarchy Session ID in each of the PID namespaces of which *[pid]* is a member. The fields are ordered as for *NStgid*. (Since Linux 4.1.)

VmPeak Peak virtual memory size.

VmSize Virtual memory size.

VmLck Locked memory size (see mlock(2)).

VmPin Pinned memory size (since Linux 3.2). These are pages that can't be moved because something needs to directly access physical memory.

VmHWM Peak resident set size ("high water mark"). This value is inaccurate; see */proc/[pid]/statm* above.

VmRSS Resident set size. Note that the value here is the sum of *RssAnon*, *RssFile*, and *RssShmem*. This value is inaccurate; see */proc/[pid]/statm* above.

RssAnon
Size of resident anonymous memory. (since Linux 4.5). This value is inaccurate; see */proc/[pid]/statm* above.

RssFile
Size of resident file mappings. (since Linux 4.5). This value is inaccurate; see */proc/[pid]/statm* above.

RssShmem
Size of resident shared memory (includes System V shared memory, mappings from tmpfs(5), and shared anonymous mappings). (since Linux 4.5).

VmData, *VmStk*, *VmExe*
Size of data, stack, and text segments. This value is inaccurate; see */proc/[pid]/statm* above.

VmLib Shared library code size.

VmPTE Page table entries size (since Linux 2.6.10).

VmPMD Size of second-level page tables (added in Linux 4.0; removed in Linux 4.15).

VmSwap Swapped-out virtual memory size by anonymous

private pages; shmem swap usage is not included (since Linux 2.6.34). This value is inaccurate; see */proc/[pid]/statm* above.

HugetlbPages
Size of hugetlb memory portions (since Linux 4.4).

CoreDumping
Contains the value 1 if the process is currently dumping core, and 0 if it is not (since Linux 4.15). This information can be used by a monitoring process to avoid killing a process that is currently dumping core, which could result in a corrupted core dump file.

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*
Mask (expressed in hexadecimal) of signals pending for thread and for process as a whole (see pthreads(7) and signal(7)).

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

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

CapBnd Capability bounding set, expressed in hexadecimal (since Linux 2.6.26, see capabilities(7)).

CapAmb Ambient capability set, expressed in hexadecimal (since Linux 4.3, see capabilities(7)).

NoNewPrivs
Value of the *no_new_privs* bit (since Linux 4.10, see prctl(2)).

Seccomp
Seccomp mode of the process (since Linux 3.8, see seccomp(2)). 0 means **SECCOMP_MODE_DISABLED**; 1 means **SECCOMP_MODE_STRICT**; 2 means **SECCOMP_MODE_FILTER**. This field is provided only if the kernel was built with the **CONFIG_SECCOMP** kernel configuration option enabled.

Speculation_Store_Bypass
Speculation flaw mitigation state (since Linux 4.17, see prctl(2)).

Cpus_allowed
Hexadecimal 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_ctxt_switches, *nonvoluntary_ctxt_switches*
Number of voluntary and involuntary context switches (since Linux 2.6.23).

/proc/[pid]/syscall (since Linux 2.6.27)
This file exposes the system call number and argument registers for the system call currently being executed by the process, followed by the values of the stack pointer and program counter registers. The values of all six argument registers are exposed, although most system calls use fewer registers.

If the process is blocked, but not in a system call, then the file displays -1 in place of the system call number, followed by just the values of the stack pointer and program counter. If process is not blocked, then the file contains just the string "running".

This file is present only if the kernel was configured with **CONFIG_HAVE_ARCH_TRACEHOOK**.

Permission to access this file is governed by a ptrace access mode **PTRACE_MODE_ATTACH_FSCREDS** check; see ptrace(2).

/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/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/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
Contains subdirectories that in turn contain files with information about (certain) mounted filesystems.

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

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

cpu 10132153 290696 3084719 46828483 16683 0 25195 0 175628 0
cpu0 1393280 32966 572056 13343292 6130 0 17875 0 23933 0
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 ("cpu" line) or the specific CPU ("cpu*N*" line) 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. This value is not reliable, for the following reasons:

1. The CPU will not wait for I/O to complete; iowait is the time that a task is waiting for I/O to complete. When a CPU goes into idle state for outstanding task I/O, another task will be scheduled

on this CPU.

2. On a multi-core CPU, the task waiting for I/O to complete is not running on any CPU, so the iowait of each CPU is difficult to calculate.

3. The value in this field may *decrease* in certain conditions.

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

softirq (since Linux 2.6.0)
(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 including unnumbered architecture specific interrupts; each subsequent column is the total for that particular numbered interrupt. Unnumbered interrupts are not shown, only summed into the total.

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.)

softirq 229245889 94 60001584 13619 5175704 2471304 28 51212741 59130143 0 51240672
This line shows the number of softirq for all CPUs. The first column is the total of all softirqs and each subsequent column is the total for particular softirq. (Linux 2.6.31 onward.)

/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 in some cases modified using the */proc* filesystem, and the (deprecated) sysctl(2) system call.

String values may be terminated by either '\0' or '\n'.

Integer and long values may be written either in decimal or in hexadecimal notation (e.g., 0x3FFF). When writing multiple integer or long values, these may be separated by any of the following whitespace characters: ' ', '\t', or '\n'. Using other separators leads to the error **EINVAL**.

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

/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
```

NOTES top

Many files contain strings (e.g., the environment and command line) that are in the internal format, with subfields terminated by null bytes ('\0'). When inspecting such files, you may find that the results are more readable if you use a command of the following form to display them:

```
$ cat file | tr '\000' '\n'
```

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

SEE ALSO top

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

The Linux kernel source files: *Documentation/filesystems/proc.txt*, *Documentation/sysctl/fs.txt*, *Documentation/sysctl/kernel.txt*, *Documentation/sysctl/net.txt*, and *Documentation/sysctl/vm.txt*.

COLOPHON top

This page is part of release 5.13 of the Linux *man-pages* project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at https://www.kernel.org/doc/man-pages/.