NAME

prctl − operations on a process

SYNOPSIS

#include <sys/prctl.h>

int prctl(int *option***, unsigned long** *arg2***, unsigned long** *arg3***,**
unsigned long *arg4***, unsigned long** *arg5***);**

DESCRIPTION

prctl() is called with a first argument describing what to do (with values defined in <*linux/prctl.h*>), and further arguments with a significance depending on the first one. The first argument can be:

PR_CAPBSET_READ (since Linux 2.6.25)
Return (as the function result) 1 if the capability specified in *arg2* is in the calling thread's capability bounding set, or 0 if it is not. (The capability constants are defined in <*linux/capability.h*>.) The capability bounding set dictates whether the process can receive the capability through a file's permitted capability set on a subsequent call to **execve**(2).

If the capability specified in *arg2* is not valid, then the call fails with the error **EINVAL**.

PR_CAPBSET_DROP (since Linux 2.6.25)
If the calling thread has the **CAP_SETPCAP** capability, then drop the capability specified by *arg2* from the calling thread's capability bounding set. Any children of the calling thread will inherit the newly reduced bounding set.

The call fails with the error: **EPERM** if the calling thread does not have the **CAP_SETPCAP**; **EINVAL** if *arg2* does not represent a valid capability; or **EINVAL** if file capabilities are not enabled in the kernel, in which case bounding sets are not supported.

PR_SET_DUMPABLE (since Linux 2.3.20)
Set the state of the flag determining whether core dumps are produced for this process upon delivery of a signal whose default behavior is to produce a core dump. (Normally this flag is set for a process by default, but it is cleared when a set-user-ID or set-group-ID program is executed and also by various system calls that manipulate process UIDs and GIDs). In kernels up to and including 2.6.12, *arg2* must be either 0 (process is not dumpable) or 1 (process is dumpable). Between kernels 2.6.13 and 2.6.17, the value 2 was also permitted, which caused any binary which normally would not be dumped to be dumped readable by root only; for security reasons, this feature has been removed. (See also the description of */proc/sys/fs/suid_dumpable* in **proc**(5).)

PR_GET_DUMPABLE (since Linux 2.3.20)
Return (as the function result) the current state of the calling process's dumpable flag.

PR_SET_ENDIAN (since Linux 2.6.18, PowerPC only)
Set the endian-ness of the calling process to the value given in *arg2*, which should be one of the following: **PR_ENDIAN_BIG**, **PR_ENDIAN_LITTLE**, or **PR_ENDIAN_PPC_LITTLE** (PowerPC pseudo little endian).

PR_GET_ENDIAN (since Linux 2.6.18, PowerPC only)
Return the endian-ness of the calling process, in the location pointed to by *(int *) arg2*.

PR_SET_FPEMU (since Linux 2.4.18, 2.5.9, only on ia64)
Set floating-point emulation control bits to *arg2*. Pass **PR_FPEMU_NOPRINT** to silently emulate fp operations accesses, or **PR_FPEMU_SIGFPE** to not emulate fp operations and send **SIGFPE** instead.

PR_GET_FPEMU (since Linux 2.4.18, 2.5.9, only on ia64)
Return floating-point emulation control bits, in the location pointed to by *(int *) arg2*.

PR_SET_FPEXC (since Linux 2.4.21, 2.5.32, only on PowerPC)
Set floating-point exception mode to *arg2*. Pass **PR_FP_EXC_SW_ENABLE** to use FPEXC for FP exception enables, **PR_FP_EXC_DIV** for floating-point divide by zero, **PR_FP_EXC_OVF** for floating-point overflow, **PR_FP_EXC_UND** for floating-point underflow, **PR_FP_EXC_RES** for floating-point inexact result, **PR_FP_EXC_INV** for floating-point invalid operation, **PR_FP_EXC_DISABLED** for FP exceptions disabled, **PR_FP_EXC_NONRECOV** for async non-recoverable exception mode, **PR_FP_EXC_ASYNC** for async recoverable exception mode, **PR_FP_EXC_PRECISE** for precise exception mode.

PR_GET_FPEXC (since Linux 2.4.21, 2.5.32, only on PowerPC)
Return floating-point exception mode, in the location pointed to by *(int *) arg2*.

PR_SET_KEEPCAPS (since Linux 2.2.18)
Set the state of the thread's "keep capabilities" flag, which determines whether the threads's effective and permitted capability sets are cleared when a change is made to the threads's user IDs such that the threads's real UID, effective UID, and saved set-user-ID all become non-zero when at least one of them previously had the value 0. (By default, these credential sets are cleared). *arg2* must be either 0 (capabilities are cleared) or 1 (capabilities are kept). This value will be reset to 0 on subsequent calls to **execve**(2).

PR_GET_KEEPCAPS (since Linux 2.2.18)
Return (as the function result) the current state of the calling threads's "keep capabilities" flag.

PR_SET_NAME (since Linux 2.6.9)
Set the process name for the calling process, using the value in the location pointed to by *(char *) arg2*. The name can be up to 16 bytes long, and should be null terminated if it contains fewer bytes.

PR_GET_NAME (since Linux 2.6.11)
Return the process name for the calling process, in the buffer pointed to by *(char *) arg2*. The buffer should allow space for up to 16 bytes; the returned string will be null terminated if it is shorter than that.

PR_SET_PDEATHSIG (since Linux 2.1.57)
Set the parent process death signal of the calling process to *arg2* (either a signal value in the range 1..maxsig, or 0 to clear). This is the signal that the calling process will get when its parent dies. This value is cleared for the child of a **fork**(2).

PR_GET_PDEATHSIG (since Linux 2.3.15)
Return the current value of the parent process death signal, in the location pointed to by *(int *) arg2*.

PR_SET_SECCOMP (since Linux 2.6.23)
Set the secure computing mode for the calling thread. In the current implementation, *arg2* must be 1. After the secure computing mode has been set to 1, the only system calls that the thread is permitted to make are **read**(2), **write**(2), **_exit**(2), and **sigreturn**(2). Other system calls result in the delivery of a **SIGKILL** signal. Secure computing mode is useful for number-crunching applications that may need to execute untrusted byte code, perhaps obtained by reading from a pipe or socket. This operation is only available if the kernel is configured with CONFIG_SECCOMP enabled.

PR_GET_SECCOMP (since Linux 2.6.23)
Return the secure computing mode of the calling thread. Not very useful for the current implementation (mode equals 1), but may be useful for other possible future modes: if the caller is not in secure computing mode, this operation returns 0; if the caller is in secure computing mode, then the **prctl**() call will cause a **SIGKILL** signal to be sent to the process. This operation is only available if the kernel is configured with CONFIG_SECCOMP enabled.

PR_SET_SECUREBITS (since Linux 2.6.26)
Set the "securebits" flags of the calling thread to the value supplied in *arg2*. See **capabilities**(7).

PR_GET_SECUREBITS (since Linux 2.6.26)
Return (as the function result) the "securebits" flags of the calling thread. See **capabilities**(7).

PR_SET_TIMING (since Linux 2.6.0-test4)
Set whether to use (normal, traditional) statistical process timing or accurate timestamp based process timing, by passing **PR_TIMING_STATISTICAL** or **PR_TIMING_TIMESTAMP** to *arg2*. **PR_TIMING_TIMESTAMP** is not currently implemented (attempting to set this mode will yield the error **EINVAL**).

PR_GET_TIMING (since Linux 2.6.0-test4)
Return (as the function result) which process timing method is currently in use.

PR_SET_TSC (since Linux 2.6.26, x86 only)
Set the state of the flag determining whether the timestamp counter can be read by the process. Pass **PR_TSC_ENABLE** to *arg2* to allow it to be read, or **PR_TSC_SIGSEGV** to generate a **SIGSEGV** when the process tries to read the timestamp counter.

PR_GET_TSC (since Linux 2.6.26, x86 only)
Return the state of the flag determining whether the timestamp counter can be read, in the location pointed to by *(int *) arg2*.

PR_SET_UNALIGN
(Only on: ia64, since Linux 2.3.48; parisc, since Linux 2.6.15; PowerPC, since Linux 2.6.18; Alpha, since Linux 2.6.22) Set unaligned access control bits to *arg2*. Pass **PR_UNALIGN_NOPRINT** to silently fix up unaligned user accesses, or **PR_UNALIGN_SIGBUS** to generate **SIGBUS** on unaligned user access.

PR_GET_UNALIGN
(see **PR_SET_UNALIGN** for information on versions and architectures) Return unaligned access control bits, in the location pointed to by *(int *) arg2*.

RETURN VALUE

On success, **PR_GET_DUMPABLE**, **PR_GET_KEEPCAPS**, **PR_CAPBSET_READ**, **PR_GET_TIMING**, **PR_GET_SECUREBITS**, and (if it returns) **PR_GET_SECCOMP** return the non-negative values described above. All other *option* values return 0 on success. On error, −1 is returned, and *errno* is set appropriately.

ERRORS

EFAULT
arg2 is an invalid address.

EINVAL
The value of *option* is not recognized.

EINVAL
arg2 is not valid value for this *option*.

EINVAL
option is **PR_SET_SECCOMP** or **PR_SET_SECCOMP**, and the kernel was not configured with **CONFIG_SECCOMP**.

EPERM
option is **PR_SET_SECUREBITS**, and the caller does not have the **CAP_SETPCAP** capability, or tried to unset a "locked" flag, or tried to set a flag whose corresponding locked flag was set (see **capabilities**(7)).

EPERM
option is **PR_SET_KEEPCAPS**, and the callers's **SECURE_KEEP_CAPS_LOCKED** flag is set (see **capabilities**(7)).

EPERM
option is **PR_CAPBSET_DROP**, and the caller does not have the **CAP_SETPCAP** capability.

VERSIONS

The **prctl**() system call was introduced in Linux 2.1.57.

CONFORMING TO

This call is Linux-specific. IRIX has a **prctl**() system call (also introduced in Linux 2.1.44 as irix_prctl on the MIPS architecture), with prototype

ptrdiff_t prctl(int *option***, int** *arg2***, int** *arg3***);**

and options to get the maximum number of processes per user, get the maximum number of processors the calling process can use, find out whether a specified process is currently blocked, get or set the maximum stack size, etc.

SEE ALSO

signal(2), **core**(5)

COLOPHON

This page is part of release 3.22 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/.