NAME

getpriority, setpriority - get/set program scheduling priority

SYNOPSIS

#include <sys/time.h>
#include <sys/resource.h>

int getpriority(int *which***, id_t** *who***);**
int setpriority(int *which***, id_t** *who***, int** *prio***);**

DESCRIPTION

The scheduling priority of the process, process group, or user, as indicated by *which* and *who* is obtained with the **getpriority**() call and set with the **setpriority**() call. The process attribute dealt with by these system calls is the same attribute (also known as the "nice" value) that is dealt with by nice(2).

The value *which* is one of **PRIO_PROCESS**, **PRIO_PGRP**, or **PRIO_USER**, and *who* is interpreted relative to *which* (a process identifier for **PRIO_PROCESS**, process group identifier for **PRIO_PGRP**, and a user ID for **PRIO_USER**). A zero value for *who* denotes (respectively) the calling process, the process group of the calling process, or the real user ID of the calling process.

The *prio* argument is a value in the range -20 to 19 (but see NOTES below). with -20 being the highest priority and 19 being the lowest priority. Attempts to set a priority outside this range are silently clamped to the range. The default priority is 0; lower values give a process a higher scheduling priority.

The **getpriority**() call returns the highest priority (lowest numerical value) enjoyed by any of the specified processes. The**setpriority**() call sets the priorities of all of the specified processes to the specified v alue.

Traditionally, only a privileged process could lower the nice value (i.e., set a higher priority). However, since Linux 2.6.12, an unprivileged process can decrease the nice value of a target process that has a suitable **RLIMIT_NICE** soft limit; see getrlimit(2) for details.

RETURN VALUE

On success, **getpriority**() returns the calling thread's nice value, which may be a negative number. On error, it returns -1 and sets *errno* to indicate the cause of the error. Since a successful call to**getpriority**() can legitimately return the value -1, it is necessary to clear the external variable *errno* prior to the call, then check it afterward to determine if -1 is an error or a legitimate value.

setpriority() returns 0 on success. On error, it returns -1 and sets *errno* to indicate the cause of the error.

ERRORS

EINVAL
which was not one of **PRIO_PROCESS**, **PRIO_PGRP**, or **PRIO_USER**.

ESRCH
No process was located using the *which* and *who* values specified.

In addition to the errors indicated above, **setpriority**() may fail if:

EACCES
The caller attempted to set a lower nice value (i.e., a higher process priority), but did not have the required privilege (on Linux: did not have the **CAP_SYS_NICE** capability).

EPERM
A process was located, but its effective user ID did not match either the effective or the real user ID of the caller, and was not privileged (on Linux: did not have the **CAP_SYS_NICE** capability). But see NOTES below.

CONFORMING TO

POSIX.1-2001, POSIX.1-2008, SVr4, 4.4BSD (these interfaces first appeared in 4.2BSD).

NOTES

For further details on the nice value, see sched(7).

Note: the addition of the "autogroup" feature in Linux 2.6.38 means that the nice value no longer has its

traditional effect in many circumstances. For details, see sched(7).

A child created by fork(2) inherits its parent's nice value. The nice value is preserved across execve(2).

The details on the condition for **EPERM** depend on the system. The above description is what POSIX.1-2001 says, and seems to be followed on all System V-like systems. Linux kernels before 2.6.12 required the real or effective user ID of the caller to match the real user of the process *who* (instead of its effective user ID). Linux 2.6.12 and later require the effective user ID of the caller to match the real or effective user ID of the process *who*. All BSD-like systems (SunOS 4.1.3, Ultrix 4.2, 4.3BSD, FreeBSD 4.3, OpenBSD-2.5, ...) behave in the same manner as Linux 2.6.12 and later.

Including <*sys/time.h*> is not required these days, but increases portability. (Indeed,<*sys/r esource.h*> defines the *rusage* structure with fields of type *struct timeval* defined in <*sys/time.h*>.)

C library/kernel differences

Within the kernel, nice values are actually represented using the range 40..1 (since negative numbers are error codes) and these are the values employed by the **setpriority**() and **getpriority**() system calls. The glibc wrapper functions for these system calls handle the translations between the user-land and kernel representations of the nice value according to the formula *unice = 20 - knice*. (Thus, the kernel's 40..1 range corresponds to the range -20..19 as seen by user space.)

BUGS

According to POSIX, the nice value is a per-process setting. However, under the current Linux/NPTL implementation of POSIX threads, the nice value is a per-thread attribute: different threads in the same process can have different nice values. Portable applications should avoid relying on the Linux behavior, which may be made standards conformant in the future.

SEE ALSO

nice(1), renice(1), fork(2), capabilities(7), sched(7)

Documentation/scheduler/sched-nice-design.txt in the Linux kernel source tree (since Linux 2.6.23)

COLOPHON

This page is part of release 4.10 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/.