MyWebUniversity
python
Rocky Linux
Manual Pages
FPDF Library
PDF generator
FPDF
Full credit is given to the above companies including the OS that this PDF file was generated!

Rocky Enterprise Linux 9.2 Manual Pages on command 'sched_getscheduler.2'

$ man sched_getscheduler.2

SCHED_SETSCHEDULER(2)   Linux Programmer's Manual   SCHED_SETSCHEDULER(2)

NAME

sched_setscheduler, sched_getscheduler - set and get scheduling pol?

icy/parameters

SYNOPSIS

```
#include <sched.h>

int sched_setscheduler(pid_t pid, int policy,

                       const struct sched_param *param);

int sched_getscheduler(pid_t pid);
```

DESCRIPTION

The sched_setscheduler() system call sets both the scheduling policy

and parameters for the thread whose ID is specified in pid. If pid

equals zero, the scheduling policy and parameters of the calling thread

will be set.

The scheduling parameters are specified in the param argument, which is

a pointer to a structure of the following form:

```
    struct sched_param {

        ...
        int sched_priority;
        ...
    };
```

In the current implementation, the structure contains only one field, sched_priority. The interpretation of param depends on the selected policy.

Currently, Linux supports the following "normal" (i.e., non-real-time) scheduling policies as values that may be specified in policy:

SCHED_OTHER   the standard round-robin time-sharing policy;

SCHED_BATCH   for "batch" style execution of processes; and

SCHED_IDLE   for running very low priority background jobs.

For each of the above policies, param->sched_priority must be 0.

Various "real-time" policies are also supported, for special time-crit? ical applications that need precise control over the way in which runnable threads are selected for execution. For the rules governing when a process may use these policies, see sched(7). The real-time policies that may be specified in policy are:

SCHED_FIFO   a first-in, first-out policy; and

SCHED_RR   a round-robin policy.

For each of the above policies, param->sched_priority specifies a scheduling priority for the thread. This is a number in the range re? turned by calling sched_get_priority_min(2) and sched_get_prior? ity_max(2) with the specified policy. On Linux, these system calls re? turn, respectively, 1 and 99.

Since Linux 2.6.32, the SCHED_RESET_ON_FORK flag can be ORed in policy when calling sched_setscheduler(). As a result of including this flag, children created by fork(2) do not inherit privileged scheduling poli? cies. See sched(7) for details.

sched_getscheduler() returns the current scheduling policy of the thread identified by pid. If pid equals zero, the policy of the call? ing thread will be retrieved.

RETURN VALUE

On success, sched_setscheduler() returns zero. On success, sched_getscheduler() returns the policy for the thread (a nonnegative

integer). On error, both calls return -1, and errno is set appropri?

ately.

ERRORS

EINVAL Invalid arguments: pid is negative or param is NULL.

EINVAL (sched_setscheduler()) policy is not one of the recognized poli?

cies.

EINVAL (sched_setscheduler()) param does not make sense for the speci?

fied policy.

EPERM The calling thread does not have appropriate privileges.

ESRCH The thread whose ID is pid could not be found.

CONFORMING TO

POSIX.1-2001, POSIX.1-2008 (but see BUGS below). The SCHED_BATCH and

SCHED_IDLE policies are Linux-specific.

NOTES

Further details of the semantics of all of the above "normal" and

"real-time" scheduling policies can be found in the sched(7) manual

page. That page also describes an additional policy, SCHED_DEADLINE,

which is settable only via sched_setattr(2).

POSIX systems on which sched_setscheduler() and sched_getscheduler()

are available define _POSIX_PRIORITY_SCHEDULING in <unistd.h>.

POSIX.1 does not detail the permissions that an unprivileged thread re?

quires in order to call sched_setscheduler(), and details vary across

systems. For example, the Solaris 7 manual page says that the real or

effective user ID of the caller must match the real user ID or the save

set-user-ID of the target.

The scheduling policy and parameters are in fact per-thread attributes

on Linux. The value returned from a call to gettid(2) can be passed in

the argument pid. Specifying pid as 0 will operate on the attributes

of the calling thread, and passing the value returned from a call to

getpid(2) will operate on the attributes of the main thread of the

thread group.   (If you are using the POSIX threads API, then use

pthread_setschedparam(3),   pthread_getschedparam(3),   and pthread_setschedprio(3), instead of the sched_*(2) system calls.)

BUGS

POSIX.1 says that on success, sched_setscheduler() should return the previous scheduling policy. Linux sched_setscheduler() does not con? form to this requirement, since it always returns 0 on success.

SEE ALSO

chrt(1), nice(2), sched_get_priority_max(2), sched_get_priority_min(2), sched_getaffinity(2), sched_getattr(2), sched_getparam(2), sched_rr_get_interval(2), sched_setaffinity(2), sched_setattr(2), sched_setparam(2), sched_yield(2), setpriority(2), capabilities(7), cpuset(7), sched(7)

COLOPHON

This page is part of release 5.10 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/.