



Linux Ubuntu 26.04 Manual Pages on command 'ioprio_set.2'

\$ man ioprio_set.2

ioprio_set(2) System Calls Manual ioprio_set(2)

NAME

ioprio_get, ioprio_set - get/set I/O scheduling class and priority

LIBRARY

Standard C library (libc, -lc)

SYNOPSIS

```
#include <linux/ioprio.h>  /* Definition of IOPRIO_* constants */
```

```
#include <sys/syscall.h>   /* Definition of SYS_* constants */
```

```
#include <unistd.h>
```

```
int syscall(SYS_ioprio_get, int which, int who);
```

```
int syscall(SYS_ioprio_set, int which, int who, int ioprio);
```

Note: glibc provides no wrappers for these system calls, necessitating the use of `syscall(2)`.

DESCRIPTION

The `ioprio_get()` and `ioprio_set()` system calls get and set the I/O scheduling class and priority of one or more threads.

The `which` and `who` arguments identify the thread(s) on which the system calls operate. The `which` argument determines how `who` is interpreted, and has one of the following values:

IOPRIO_WHO_PROCESS

`who` is a process ID or thread ID identifying a single process or thread. If `who` is 0, then operate on the calling thread.

IOPRIO_WHO_PGRP

`who` is a process group ID identifying all the members of a process group. If `who` is

0, then operate on the process group of which the caller is a member.

IOPRIO_WHO_USER

who is a user ID identifying all of the processes that have a matching real UID.

If which is specified as IOPRIO_WHO_PGRP or IOPRIO_WHO_USER when calling ioprio_get(), and more than one process matches who, then the returned priority will be the highest one found among all of the matching processes. One priority is said to be higher than another one if it belongs to a higher priority class (IOPRIO_CLASS_RT is the highest priority class; IOPRIO_CLASS_IDLE is the lowest) or if it belongs to the same priority class as the other process but has a higher priority level (a lower priority number means a higher priority level).

The ioprio argument given to ioprio_set() is a bit mask that specifies both the scheduling class and the priority to be assigned to the target process(es). The following macros are used for assembling and dissecting ioprio values:

IOPRIO_PRIO_VALUE(class, data)

Given a scheduling class and priority (data), this macro combines the two values to produce an ioprio value, which is returned as the result of the macro.

IOPRIO_PRIO_CLASS(mask)

Given mask (an ioprio value), this macro returns its I/O class component, that is, one of the values IOPRIO_CLASS_RT, IOPRIO_CLASS_BE, or IOPRIO_CLASS_IDLE.

IOPRIO_PRIO_DATA(mask)

Given mask (an ioprio value), this macro returns its priority (data) component.

See the NOTES section for more information on scheduling classes and priorities, as well as the meaning of specifying ioprio as 0.

I/O priorities are supported for reads and for synchronous (O_DIRECT, O_SYNC) writes. I/O priorities are not supported for asynchronous writes because they are issued outside the context of the program dirtying the memory, and thus program-specific priorities do not apply.

RETURN VALUE

On success, ioprio_get() returns the ioprio value of the process with highest I/O priority of any of the processes that match the criteria specified in which and who. On error, -1 is returned, and errno is set to indicate the error.

On success, ioprio_set() returns 0. On error, -1 is returned, and errno is set to indicate the error.

ERRORS

EINVAL Invalid value for which or ioprio. Refer to the NOTES section for available scheduler classes and priority levels for ioprio.

EPERM The calling process does not have the privilege needed to assign this ioprio to the specified process(es). See the NOTES section for more information on required privileges for ioprio_set().

ESRCH No process(es) could be found that matched the specification in which and who.

STANDARDS

Linux.

HISTORY

Linux 2.6.13.

NOTES

Two or more processes or threads can share an I/O context. This will be the case when clone(2) was called with the CLONE_IO flag. However, by default, the distinct threads of a process will not share the same I/O context. This means that if you want to change the I/O priority of all threads in a process, you may need to call ioprio_set() on each of the threads. The thread ID that you would need for this operation is the one that is returned by getpid(2) or clone(2).

These system calls have an effect only when used in conjunction with an I/O scheduler that supports I/O priorities. As at kernel 2.6.17 the only such scheduler is the Completely Fair Queuing (CFQ) I/O scheduler.

If no I/O scheduler has been set for a thread, then by default the I/O priority will follow the CPU nice value (setpriority(2)). Before Linux 2.6.24, once an I/O priority had been set using ioprio_set(), there was no way to reset the I/O scheduling behavior to the default. Since Linux 2.6.24, specifying ioprio as 0 can be used to reset to the default I/O scheduling behavior.

Selecting an I/O scheduler

I/O schedulers are selected on a per-device basis via the special file /sys/block/device/queue/scheduler.

One can view the current I/O scheduler via the /sys filesystem. For example, the following command displays a list of all schedulers currently loaded in the kernel:

```
$ cat /sys/block/sda/queue/scheduler  
noop anticipatory deadline [cfq]
```

The scheduler surrounded by brackets is the one actually in use for the device (sda in the example). Setting another scheduler is done by writing the name of the new scheduler to this file. For example, the following command will set the scheduler for the sda device to cfq:

```
$ su
```

```
Password:
```

```
# echo cfq > /sys/block/sda/queue/scheduler
```

The Completely Fair Queuing (CFQ) I/O scheduler

Since version 3 (also known as CFQ Time Sliced), CFQ implements I/O nice levels similar to those of CPU scheduling. These nice levels are grouped into three scheduling classes, each one containing one or more priority levels:

IOPRIO_CLASS_RT (1)

This is the real-time I/O class. This scheduling class is given higher priority than any other class: processes from this class are given first access to the disk every time. Thus, this I/O class needs to be used with some care: one I/O real-time process can starve the entire system. Within the real-time class, there are 8 levels of class data (priority) that determine exactly how much time this process needs the disk for on each service. The highest real-time priority level is 0; the lowest is 7. In the future, this might change to be more directly mappable to performance, by passing in a desired data rate instead.

IOPRIO_CLASS_BE (2)

This is the best-effort scheduling class, which is the default for any process that hasn't set a specific I/O priority. The class data (priority) determines how much I/O bandwidth the process will get. Best-effort priority levels are analogous to CPU nice values (see `getpriority(2)`). The priority level determines a priority relative to other processes in the best-effort scheduling class. Priority levels range from 0 (highest) to 7 (lowest).

IOPRIO_CLASS_IDLE (3)

This is the idle scheduling class. Processes running at this level get I/O time only when no one else needs the disk. The idle class has no class data. Attention is required when assigning this priority class to a process, since it may become starved if higher priority processes are constantly accessing the disk.

CFQ I/O Scheduler and an example program.

Required permissions to set I/O priorities

Permission to change a process's priority is granted or denied based on two criteria:

Process ownership

An unprivileged process may set the I/O priority only for a process whose real UID matches the real or effective UID of the calling process. A process which has the CAP_SYS_NICE capability can change the priority of any process.

What is the desired priority

Attempts to set very high priorities (IOPRIO_CLASS_RT) require the CAP_SYS_ADMIN capability. Up to Linux 2.6.24 also required CAP_SYS_ADMIN to set a very low priority (IOPRIO_CLASS_IDLE), but since Linux 2.6.25, this is no longer required.

A call to `ioprio_set()` must follow both rules, or the call will fail with the error EPERM.

BUGS

glibc does not yet provide a suitable header file defining the function prototypes and macros described on this page. Suitable definitions can be found in `linux/ioprio.h`.

SEE ALSO

`ionice(1)`, `getpriority(2)`, `open(2)`, `capabilities(7)`, `cgroups(7)`

Documentation/block/ioprio.rst in the Linux kernel source tree

Linux man-pages 6.17

2026-02-08

`ioprio_set(2)`