NAME

sched_setattr, sched_getattr − set and get scheduling policy and attributes

LIBRARY

Standard C library (*libc*, *−lc*)

SYNOPSIS

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

int syscall(SYS_sched_setattr, pid_t pid, struct sched_attr *attr,
        unsigned int flags);
int syscall(SYS_sched_getattr, pid_t pid, struct sched_attr *attr,
        unsigned int size, unsigned int flags);
```

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

DESCRIPTION

sched_setattr()

The **sched_setattr**() system call sets the scheduling policy and associated attributes for the thread whose ID is specified in *pid*. If*pid* equals zero, the scheduling polic y and attributes of the calling thread will be set.

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.

Various "real-time" policies are also supported, for special time-critical 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.

Linux also provides the following policy:

SCHED_DEADLINE
a deadline scheduling policy; see **sched**(7) for details.

The *attr* argument is a pointer to a structure that defines the new scheduling policy and attributes for the specified thread. This structure has the following form:

```
struct sched_attr {
    u32 size;              /* Size of this structure */
    u32 sched_policy;      /* Policy (SCHED_*) */
    u64 sched_flags;       /* Flags */
    s32 sched_nice;        /* Nice value (SCHED_OTHER,
                              SCHED_BATCH) */
    u32 sched_priority;    /* Static priority (SCHED_FIFO,
                              SCHED_RR) */
    /* Remaining fields are for SCHED_DEADLINE */
    u64 sched_runtime;
    u64 sched_deadline;
    u64 sched_period;
};
```

The fields of the *sched_attr* structure are as follows:

size This field should be set to the size of the structure in bytes, as in *sizeof(struct sched_attr)*. If the provided structure is smaller than the kernel structure, any additional fields are assumed to be '0'. If the provided structure is larger than the kernel structure, the kernel verifies that all additional fields are 0; if they are not, **sched_setattr**() fails with the error **E2BIG** and updates *size* to contain the size of the kernel structure.

The above behavior when the size of the user-space *sched_attr* structure does not match the size of the kernel structure allows for future extensibility of the interface. Malformed applications that pass oversize structures won't break in the future if the size of the kernel *sched_attr* structure is increased. In the future, it could also allow applications that know about a larger user-space *sched_attr* structure to determine whether they are running on an older kernel that does not support the larger structure.

sched_policy
This field specifies the scheduling policy, as one of the **SCHED_*** values listed above.

sched_flags
This field contains zero or more of the following flags that are ORed together to control scheduling behavior:

SCHED_FLAG_RESET_ON_FORK
Children created by **fork**(2) do not inherit privileged scheduling policies. See **sched**(7) for details.

SCHED_FLAG_RECLAIM (since Linux 4.13)
This flag allows a **SCHED_DEADLINE** thread to reclaim bandwidth unused by other real-time threads.

SCHED_FLAG_DL_OVERRUN (since Linux 4.16)
This flag allows an application to get informed about run-time overruns in **SCHED_DEADLINE** threads. Such overruns may be caused by (for example) coarse execution time accounting or incorrect parameter assignment. Notification takes the form of a **SIGXCPU** signal which is generated on each overrun.

This **SIGXCPU** signal is *process-directed* (see **signal**(7)) rather than thread-directed. This is probably a bug. On the one hand,**sched_setattr**() is being used to set a per -thread attribute. On the other hand, if the process-directed signal is delivered to a thread inside the process other than the one that had a run-time overrun, the application has no way of knowing which thread overran.

sched_nice
This field specifies the nice value to be set when specifying *sched_policy* as **SCHED_OTHER** or **SCHED_BATCH**. The nice value is a number in the range −20 (high priority) to +19 (low priority); see **sched**(7).

sched_priority
This field specifies the static priority to be set when specifying *sched_policy* as **SCHED_FIFO** or **SCHED_RR**. The allowed range of priorities for these policies can be determined using **sched_get_priority_min**(2) and **sched_get_priority_max**(2). For other policies, this field must be specified as 0.

sched_runtime
This field specifies the "Runtime" parameter for deadline scheduling. The value is expressed in nanoseconds. This field, and the next two fields, are used only for **SCHED_DEADLINE** scheduling; for further details, see **sched**(7).

sched_deadline
This field specifies the "Deadline" parameter for deadline scheduling. The value is expressed in nanoseconds.

sched_period
This field specifies the "Period" parameter for deadline scheduling. The value is expressed in nanoseconds.

The *flags* argument is provided to allow for future extensions to the interface; in the current implementation it must be specified as 0.

sched_getattr()
The **sched_getattr**() system call fetches the scheduling policy and the associated attributes for the thread whose ID is specified in *pid*. If*pid* equals zero, the scheduling polic y and attributes of the calling thread will be retrieved.

The *size* argument should be set to the size of the *sched_attr* structure as known to user space. The value must be at least as large as the size of the initially published *sched_attr* structure, or the call fails with the error **EINVAL**.

The retrieved scheduling attributes are placed in the fields of the *sched_attr* structure pointed to by *attr*. The kernel sets *attr.size* to the size of its *sched_attr* structure.

If the caller-provided *attr* buffer is larger than the kernel's *sched_attr* structure, the additional bytes in the user-space structure are not touched. If the caller-provided structure is smaller than the kernel *sched_attr* structure, the kernel will silently not return any values which would be stored outside the provided space. As with **sched_setattr**(), these semantics allow for future extensibility of the interface.

The *flags* argument is provided to allow for future extensions to the interface; in the current implementation it must be specified as 0.

RETURN VALUE
On success, **sched_setattr**() and **sched_getattr**() return 0. On error, −1 is returned, and *errno* is set to indicate the error.

ERRORS
sched_getattr() and **sched_setattr**() can both fail for the following reasons:

EINVAL
attr is NULL; or *pid* is negative; or *fla gs* is not zero.

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

In addition, **sched_getattr**() can fail for the following reasons:

E2BIG The buffer specified by *size* and *attr* is too small.

EINVAL
size is invalid; that is, it is smaller than the initial version of the *sched_attr* structure (48 bytes) or larger than the system page size.

In addition, **sched_setattr**() can fail for the following reasons:

E2BIG The buffer specified by *size* and *attr* is larger than the kernel structure, and one or more of the excess bytes is nonzero.

EBUSY
SCHED_DEADLINE admission control failure, see **sched**(7).

EINVAL
attr.sched_policy is not one of the recognized policies; *attr.sched_flags* contains a flag other than **SCHED_FLAG_RESET_ON_FORK**; or *attr.sched_priority* is invalid; or *attr.sched_policy* is **SCHED_DEADLINE** and the deadline scheduling parameters in *attr* are invalid.

EPERM
The caller does not have appropriate privileges.

EPERM
The CPU affinity mask of the thread specified by *pid* does not include all CPUs in the system (see **sched_setaffinity**(2)).

VERSIONS
These system calls first appeared in Linux 3.14.

STANDARDS
These system calls are nonstandard Linux extensions.

NOTES
glibc does not provide wrappers for these system calls; call them using **syscall**(2).

sched_setattr() provides a superset of the functionality of **sched_setscheduler**(2), **sched_setparam**(2), **nice**(2), and (other than the ability to set the priority of all processes belonging to a specified user or all processes in a specified group) **setpriority**(2). Analogously, **sched_getattr**() provides a superset of the functionality of **sched_getscheduler**(2), **sched_getparam**(2), and (partially) **getpriority**(2).

BUGS
In Linux versions up to 3.15, **sched_setattr**() failed with the error **EFAULT** instead of **E2BIG** for the case described in ERRORS.

Up to Linux 5.3, **sched_getattr**() failed with the error **EFBIG** if the in-kernel *sched_attr* structure was larger than the *size* passed by user space.

SEE ALSO
chrt(1), **nice**(2), **sched_get_priority_max**(2), **sched_get_priority_min**(2), **sched_getaffinity**(2), **sched_getparam**(2), **sched_getscheduler**(2), **sched_rr_get_interval**(2), **sched_setaffinity**(2), **sched_setparam**(2), **sched_setscheduler**(2), **sched_yield**(2), **setpriority**(2), **pthread_getschedparam**(3), **pthread_setschedparam**(3), **pthread_setschedprio**(3), **capabilities**(7), **cpuset**(7), **sched**(7)