NAME

clock_nanosleep − high-resolution sleep with specifiable clock

LIBRARY

Standard C library (*libc*, *−lc*), since glibc 2.17

Before glibc 2.17, Real-time library (*librt*, *−lrt*)

SYNOPSIS

```
#include <time.h>

int clock_nanosleep(clockid_t clockid, int flags,
                    const struct timespec *t,
                    struct timespec *_Nullable remain);
```

Feature Test Macro Requirements for glibc (see **feature_test_macros**(7)):

clock_nanosleep():
_POSIX_C_SOURCE >= 200112L

DESCRIPTION

Like **nanosleep**(2), **clock_nanosleep**() allows the calling thread to sleep for an interval specified with nanosecond precision. It differs in allowing the caller to select the clock against which the sleep interval is to be measured, and in allowing the sleep interval to be specified as either an absolute or a relative value.

The time values passed to and returned by this call are specified using **timespec**(3) structures.

The *clockid* argument specifies the clock against which the sleep interval is to be measured. This argument can have one of the following values:

CLOCK_REALTIME
A settable system-wide real-time clock.

CLOCK_TAI (since Linux 3.10)
A system-wide clock derived from wall-clock time but counting leap seconds.

CLOCK_MONOTONIC
A nonsettable, monotonically increasing clock that measures time since some unspecified point in the past that does not change after system startup.

CLOCK_BOOTTIME (since Linux 2.6.39)
Identical to **CLOCK_MONOTONIC**, except that it also includes any time that the system is suspended.

CLOCK_PROCESS_CPUTIME_ID
A settable per-process clock that measures CPU time consumed by all threads in the process.

See **clock_getres**(2) for further details on these clocks. In addition, the CPU clock IDs returned by **clock_getcpuclockid**(3) and **pthread_getcpuclockid**(3) can also be passed in *clockid*.

If *flags* is 0, then the value specified in *t* is interpreted as an interval relative to the current value of the clock specified by *clockid*.

If *flags* is **TIMER_ABSTIME**, then *t* is interpreted as an absolute time as measured by the clock, *clockid*. If *t* is less than or equal to the current value of the clock, then **clock_nanosleep**() returns immediately without suspending the calling thread.

clock_nanosleep() suspends the execution of the calling thread until either at least the time specified by *t* has elapsed, or a signal is delivered that causes a signal handler to be called or that terminates the process.

If the call is interrupted by a signal handler, **clock_nanosleep**() fails with the error **EINTR**. In addition, if *remain* is not NULL, and *flags* was not **TIMER_ABSTIME**, it returns the remaining unslept time in *remain*. This value can then be used to call **clock_nanosleep**() again and complete a (relative) sleep.

RETURN VALUE

On successfully sleeping for the requested interval, **clock_nanosleep**() returns 0. If the call is interrupted by a signal handler or encounters an error, then it returns one of the positive error number listed in ERRORS.

ERRORS

EFAULT
t or *remain* specified an invalid address.

EINTR
The sleep was interrupted by a signal handler; see **signal**(7).

EINVAL
The value in the *tv_nsec* field was not in the range [0, 999999999] or *tv_sec* was negative.

EINVAL
clockid was invalid. (**CLOCK_THREAD_CPUTIME_ID** is not a permitted value for *clockid*.)

ENOTSUP
The kernel does not support sleeping against this *clockid*.

STANDARDS

POSIX.1-2024.

HISTORY

POSIX.1-2001. Linux 2.6, glibc 2.1.

NOTES

If the interval specified in *t* is not an exact multiple of the granularity underlying clock (see **time**(7)), then the interval will be rounded up to the next multiple. Furthermore, after the sleep completes, there may still be a delay before the CPU becomes free to once again execute the calling thread.

Using an absolute timer is useful for preventing timer drift problems of the type described in **nanosleep**(2). (Such problems are exacerbated in programs that try to restart a relative sleep that is repeatedly interrupted by signals.) To perform a relative sleep that avoids these problems, call **clock_gettime**(2) for the desired clock, add the desired interval to the returned time value, and then call **clock_nanosleep**() with the **TIMER_ABSTIME** flag.

clock_nanosleep() is never restarted after being interrupted by a signal handler, regardless of the use of the **sigaction**(2) **SA_RESTART** flag.

The *remain* argument is unused, and unnecessary, when *flags* is **TIMER_ABSTIME**. (An absolute sleep can be restarted using the same *t* argument.)

POSIX.1 specifies that **clock_nanosleep**() has no effect on signals dispositions or the signal mask.

POSIX.1 specifies that after changing the value of the **CLOCK_REALTIME** clock via **clock_settime**(2), the new clock value shall be used to determine the time at which a thread blocked on an absolute **clock_nanosleep**() will wake up; if the new clock value falls past the end of the sleep interval, then the **clock_nanosleep**() call will return immediately.

POSIX.1 specifies that changing the value of the **CLOCK_REALTIME** clock via **clock_settime**(2) shall have no effect on a thread that is blocked on a relative **clock_nanosleep**().

SEE ALSO

clock_getres(2), **nanosleep**(2), **restart_syscall**(2), **timer_create**(2), **sleep**(3), **timespec**(3), **usleep**(3), **time**(7)