## NAME

timer_getoverrun − get overrun count for a POSIX per-process timer

## LIBRARY

Real-time library (*librt*, *−lrt*)

## SYNOPSIS

**#include <time.h>**

**int timer_getoverrun(timer_t** *timerid***);**

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

**timer_getoverrun**():
    _POSIX_C_SOURCE >= 199309L

## DESCRIPTION

**timer_getoverrun**() returns the "overrun count" for the timer referred to by *timerid*. An application can use the overrun count to accurately calculate the number of timer expirations that would have occurred over a given time interval. Timer overruns can occur both when receiving expiration notifications via signals (**SIGEV_SIGNAL**), and via threads (**SIGEV_THREAD**).

When expiration notifications are delivered via a signal, overruns can occur as follows. Regardless of whether or not a real-time signal is used for timer notifications, the system queues at most one signal per timer. (This is the behavior specified by POSIX.1. The alternative, queuing one signal for each timer expiration, could easily result in overflowing the allowed limits for queued signals on the system.) Because of system scheduling delays, or because the signal may be temporarily blocked, there can be a delay between the time when the notification signal is generated and the time when it is delivered (e.g., caught by a signal handler) or accepted (e.g., using **sigwaitinfo**(2)). In this interval, further timer expirations may occur. The timer overrun count is the number of additional timer expirations that occurred between the time when the signal was generated and when it was delivered or accepted.

Timer overruns can also occur when expiration notifications are delivered via invocation of a thread, since there may be an arbitrary delay between an expiration of the timer and the invocation of the notification thread, and in that delay interval, additional timer expirations may occur.

## RETURN VALUE

On success, **timer_getoverrun**() returns the overrun count of the specified timer; this count may be 0 if no overruns have occurred. On failure, −1 is returned, and *errno* is set to indicate the error.

## ERRORS

**EINVAL**
    *timerid* is not a valid timer ID.

## VERSIONS

This system call is available since Linux 2.6.

## STANDARDS

POSIX.1-2001, POSIX.1-2008.

## NOTES

When timer notifications are delivered via signals (**SIGEV_SIGNAL**), on Linux it is also possible to obtain the overrun count via the *si_overrun* field of the *siginfo_t* structure (see **sigaction**(2)). This allows an application to avoid the overhead of making a system call to obtain the overrun count, but is a nonportable extension to POSIX.1.

POSIX.1 discusses timer overruns only in the context of timer notifications using signals.

## BUGS

POSIX.1 specifies that if the timer overrun count is equal to or greater than an implementation-defined maximum, **DELAYTIMER_MAX**, then **timer_getoverrun**() should return **DELAYTIMER_MAX**. However, before Linux 4.19, if the timer overrun value exceeds the maximum representable integer, the counter cycles, starting once more from low values. Since Linux 4.19,**timer_geto verrun**() returns

**DELAYTIMER_MAX** (defined as **INT_MAX** in <*limits.h*>) in this case (and the overrun value is reset to 0).

## EXAMPLES

See **timer_create**(2).

## SEE ALSO

**clock_gettime**(2), **sigaction**(2), **signalfd**(2), **sigwaitinfo**(2), **timer_create**(2), **timer_delete**(2), **timer_settime**(2), **signal**(7), **time**(7)