NAME

timer_getoverrun, **timer_gettime**, **timer_settime** - per-process timers (REALTIME)

LIBRARY

POSIX Real-time Library (librt, -lrt)

SYNOPSIS

#include <time.h>

int
timer_getoverrun(*timer_t timerid*);

int
timer_gettime(*timer_t timerid, struct itimerspec *value*);

int
timer_settime(*timer_t timerid, int flags, const struct itimerspec *restrict value,*
*struct itimerspec *restrict ovalue*);

DESCRIPTION

The **timer_gettime**() system call stores the amount of time until the specified timer, *timerid*, expires and the reload value of the timer into the space pointed to by the *value* argument. The *it_value* member of this structure contains the amount of time before the timer expires, or zero if the timer is disarmed. This value is returned as the interval until timer expiration, even if the timer was armed with absolute time. The *it_interval* member of *value* contains the reload value last set by **timer_settime**().

The **timer_settime**() system call sets the time until the next expiration of the timer specified by *timerid* from the *it_value* member of the *value* argument and arms the timer if the *it_value* member of *value* is non-zero. If the specified timer was already armed when **timer_settime**() is called, this call resets the time until next expiration to the value specified. If the *it_value* member of *value* is zero, the timer is disarmed. If the timer is disarmed, then pending signal is removed.

If the flag TIMER_ABSTIME is not set in the argument *flags*, **timer_settime**() behaves as if the time until next expiration is set to be equal to the interval specified by the *it_value* member of *value*. That is, the timer expires in *it_value* nanoseconds from when the call is made. If the flag TIMER_ABSTIME is set in the argument *flags*, **timer_settime**() behaves as if the time until next expiration is set to be equal to the difference between the absolute time specified by the it_value member of value and the current value of the clock associated with *timerid*. That is, the timer expires when the clock reaches the value specified by the *it_value* member of *value*. If the specified time has already passed, the system call succeeds and the expiration notification is made.

The reload value of the timer is set to the value specified by the *it_interval* member of *value*. When a timer is armed with a non-zero *it_interval*, a periodic (or repetitive) timer is specified.

Time values that are between two consecutive non-negative integer multiples of the resolution of the specified timer are rounded up to the larger multiple of the resolution. Quantization error will not cause the timer to expire earlier than the rounded time value.

If the argument *ovalue* is not NULL, the **timer_settime**() system call stores, in the location referenced by *ovalue*, a value representing the previous amount of time before the timer would have expired, or zero if the timer was disarmed, together with the previous timer reload value. Timers do not expire before their scheduled time.

Only a single signal is queued to the process for a given timer at any point in time. When a timer for which a signal is still pending expires, no signal is queued, and a timer overrun will occur. When a timer expiration signal is accepted by a process, the **timer_getoverrun**() system call returns the timer expiration overrun count for the specified timer. The overrun count returned contains the number of extra timer expirations that occurred between the time the signal was generated (queued) and when it was accepted, up to but not including an maximum of {DELAYTIMER_MAX}. If the number of such extra expirations is greater than or equal to {DELAYTIMER_MAX}, then the overrun count is set to {DELAYTIMER_MAX}. The value returned by **timer_getoverrun**() applies to the most recent expiration signal acceptance for the timer. If no expiration signal has been delivered for the timer, the return value of **timer_getoverrun**() is unspecified.

RETURN VALUES

If the **timer_getoverrun**() system call succeeds, it returns the timer expiration overrun count as explained above. Otherwise the value -1 is returned, and the global variable *errno* is set to indicate the error.

The **timer_gettime**() and **timer_settime**() functions return the value 0 if successful; otherwise the value -1 is returned and the global variable *errno* is set to indicate the error.

ERRORS

The **timer_settime**() system call will fail if:

[EINVAL] A *value* structure specified a nanosecond value less than zero or greater than or equal to 1000 million, and the *it_value* member of that structure did not specify zero seconds and nanoseconds.

These system calls may fail if:

[EINVAL] The *timerid* argument does not correspond to an ID returned by **timer_create**() but

not yet deleted by **timer_delete**().

The **timer_settime**() system call may fail if:

[EINVAL] The *it_interval* member of *value* is not zero and the timer was created with notification by creation of a new thread (*sigev_sigev_notify* was SIGEV_THREAD) and a fixed stack address has been set in the thread attribute pointed to by *sigev_notify_attributes*.

The **timer_gettime**() and **timer_settime**() system calls may fail if:

[EFAULT] Any arguments point outside the allocated address space or there is a memory protection fault.

SEE ALSO

clock_getres(2), timer_create(2), siginfo(3)

STANDARDS

The **timer_getoverrun**(), **timer_gettime**(), and **timer_settime**() system calls conform to IEEE Std 1003.1-2004 ("POSIX.1").

HISTORY

Support for POSIX per-process timer first appeared in FreeBSD 7.0.