NAME

timer_settime, timer_gettime, timer_getoverrun – high-resolution timer operations

SYNOPSIS

cc [*flag* ...] *file* ... **–lposix4** [*library* ...]

#include <time.h>

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

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

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

```
struct itimerspec       {
      struct timespec   it_interval;   /* timer period */
      struct timespec   it_value;      /* timer expiration */
};

struct timespec         {
      time_t            tv_sec;        /* seconds */
      long              tv_nsec;       /* and nanoseconds */
};
```

MT-LEVEL

Async-Signal-Safe

DESCRIPTION

If *value*->**it_value** is non-zero, **timer_settime( )** arms the timer, *timerid,* to next expire after the time designated by *value*->**it_value**. Upon expiration, an application-specified notification (see **timer_create**(3R)) or the default signal, **SIGALRM**, is queued for the calling LWP. If *timerid* was already armed when **timer_settime( )** is called, this call resets the time until the next expiration to the value of *value*->**it_value**. If *value*->**it_value** is zero, then the timer is disarmed.

value->**it_value** may be expressed as either an absolute or relative time. If *flags* is set to **TIMER_RELTIME**, then the timer will initially expire relative to when the call is made. If *flags* is set to **TIMER_ABSTIME**, then the initial expiration will be relative to 00:00 Universal Coordinated Time, January 1, 1970. If the specified (absolute) time has already passed, **timer_settime( )** succeeds and the expiration notification is made.

If *value*->**it_interval** is non-zero, then *timerid,* will be a “"periodic” timer, to be reloaded to expire every *value*->**it_interval** seconds (nanoseconds). Otherwise, if *value*->**it_interval** is zero and *value*->**it_value** is non-zero, then *timerid* is a “one-shot” timer, which will expire only at the time specified by *value*->**it_value**.

If *ovalue* is not **NULL**, and timer *timerid* had previously been used, then **timer_settime( )** will store the remaining time until the previous timer expires in *ovalue*->**it_value,** and the previous reload interval in *ovalue*->**it_interval.** (If the previous timer was disarmed, *ovalue*->**it_value** will be set to zero). The values stored in *ovalue* by **timer_settime( )** are the same values that would have been returned by a call to **timer_gettime(** *timerid,***...).**

timer_gettime( ) stores the amount of time until the specified timer, *timerid,* expires into *value*->**it_value,** and the timer’s reload value into *value*->**it_interval.**

Only a single signal can be queued to the LWP for a given timer at any point in time. When a timer, for which a signal is still pending expires, (from a previous interval), no signal will be queued, and a "timer overrun count" will be incremented. When a timer expiration signal is delivered to an LWP, **timer_overrun( )** may be used to determine the timer expiration overrun count for the specified timer. The overrun count returned contains the number of extra timer expirations which occurred between the time the signal was generated (queued) and when it was delivered, up to but not including a 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 delivery for the timer.

RETURN VALUES

timer_settime( ), and **timer_gettime( )** return **0** upon success. If **timer_getoverrun( )** succeeds, the number of extra timer expirations which occurred between the time the signal was queued and when it was delivered is returned. If these functions fail, they return **-1** and set **errno** to indicate the error condition.

ERRORS

EINVAL *timerid* does not correspond to a timer returned by **timer_create**(3R).

The timer, *timerid,* had already been deleted by **timer_delete**(3R).

A *value* structure specified a nanosecond value less than zero or greater than or equal to 1,000,000,000.

ENOSYS **timer_settime( ), timer_gettime( ),** or **timer_getoverrun( )** is not supported by this implementation.

SEE ALSO

clock_settime(3R), **timer_create**(3R), **timer_delete**(3R)