Full credit is given to the above companies including the OS that this PDF file was generated!

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'timer_getoverrun.3p' command

$ man timer_getoverrun.3p

TIMER_GETOVERRUN(3P) POSIX Programmer's Manual TIMER_GETOVERRUN(3P)

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

timer_getoverrun, timer_gettime, timer_settime ? per-process timers

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() function shall store 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 shall contain the amount of time before the timer ex? pires, or zero if the timer is disarmed. This value is returned as the interval until timer expiration, even if the timer was armed with abso? lute time. The it_interval member of value shall contain the reload

value last set by timer_settime().

The timer_settime() function shall set the time until the next expira? tion of the timer specified by timerid from the it_value member of the value argument and arm 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 shall reset the time until next expiration to the value specified. If the it_value member of value is zero, the timer shall be disarmed. The effect of disarming or resetting a timer with pending expiration notifications is unspecified.

If the flag TIMER_ABSTIME is not set in the argument flags, timer_set? time() shall behave 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 shall expire in it_value nanoseconds from when the call is made. If the flag TIMER_ABSTIME is set in the argument flags, timer_settime() shall behave 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 as? sociated with timerid. That is, the timer shall expire when the clock reaches the value specified by the it_value member of value. If the specified time has already passed, the function shall succeed and the expiration notification shall be made.

The reload value of the timer shall be 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 mul? tiples of the resolution of the specified timer shall be rounded up to the larger multiple of the resolution. Quantization error shall not cause the timer to expire earlier than the rounded time value.

If the argument ovalue is not NULL, the timer_settime() function shall store, 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 shall not expire before their scheduled time.

Only a single signal shall be 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 shall be queued, and a timer overrun shall occur. When a timer expiration signal is delivered to or accepted by a process, the timer_getoverrun() function shall return the timer expira? tion 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 delivered or accepted, up to but not including an implementation-defined maximum of {DELAYTIMER_MAX}. If the number of such extra expirations is greater than or equal to {DELAYTIMER_MAX}, then the overrun count shall be set to {DELAYTIMER_MAX}. The value returned by timer_getoverrun() shall apply to the most recent expiration signal delivery or acceptance for the timer. If no expiration signal has been delivered for the timer, the return value of timer_getoverrun() is unspecified.

The behavior is undefined if the value specified by the timerid argu? ment to timer_getoverrun(), timer_gettime(), or timer_settime() does not correspond to a timer ID returned by timer_create() but not yet deleted by timer_delete().

RETURN VALUE

If the timer_getoverrun() function succeeds, it shall return the timer expiration overrun count as explained above.

If the timer_gettime() or timer_settime() functions succeed, a value of 0 shall be returned.

If an error occurs for any of these functions, the value -1 shall be returned, and errno set to indicate the error.

ERRORS

The timer_settime() function shall 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.

The timer_settime() function 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_no? tify_attributes.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Using fixed stack addresses is problematic when timer expiration is signaled by the creation of a new thread. Since it cannot be assumed that the thread created for one expiration is finished before the next expiration of the timer, it could happen that two threads use the same memory as a stack at the same time. This is invalid and produces unde? fined results.

RATIONALE

Practical clocks tick at a finite rate, with rates of 100 hertz and 1000 hertz being common. The inverse of this tick rate is the clock resolution, also called the clock granularity, which in either case is expressed as a time duration, being 10 milliseconds and 1 millisecond respectively for these common rates. The granularity of practical clocks implies that if one reads a given clock twice in rapid succes? sion, one may get the same time value twice; and that timers must wait for the next clock tick after the theoretical expiration time, to en? sure that a timer never returns too soon. Note also that the granular? ity of the clock may be significantly coarser than the resolution of the data format used to set and get time and interval values. Also note that some implementations may choose to adjust time and/or interval values to exactly match the ticks of the underlying clock.

This volume of POSIX.1?2017 defines functions that allow an application to determine the implementation-supported resolution for the clocks and requires an implementation to document the resolution supported for timers and nanosleep() if they differ from the supported clock resolu? tion. This is more of a procurement issue than a runtime application

issue.

If an implementation detects that the value specified by the timerid argument to timer_getoverrun(), timer_gettime(), or timer_settime() does not correspond to a timer ID returned by timer_create() but not yet deleted by timer_delete(), it is recommended that the function should fail and report an [EINVAL] error.

FUTURE DIRECTIONS

None.

SEE ALSO

clock_getres(), timer_create()

The Base Definitions volume of POSIX.1?2017, <time.h>