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_create.3p' command

$ man timer_create.3p

TIMER_CREATE(3P) POSIX Programmer's Manual TIMER_CREATE(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_create ? create a per-process timer

SYNOPSIS

```
#include <signal.h>
#include <time.h>
int timer_create(clockid_t clockid, struct sigevent *restrict evp,
    timer_t *restrict timerid);
```

DESCRIPTION

The timer_create() function shall create a per-process timer using the specified clock, clock_id, as the timing base. The timer_create() func? tion shall return, in the location referenced by timerid, a timer ID of type timer_t used to identify the timer in timer requests. This timer ID shall be unique within the calling process until the timer is deleted. The particular clock, clock_id, is defined in <time.h>. The timer whose ID is returned shall be in a disarmed state upon return from timer_create().

The evp argument, if non-NULL, points to a sigevent structure. This

structure, allocated by the application, defines the asynchronous noti?
fication to occur as specified in Section 2.4.1, Signal Generation and
Delivery when the timer expires. If the evp argument is NULL, the ef?
fect is as if the evp argument pointed to a sigevent structure with the
sigev_notify member having the value SIGEV_SIGNAL, the sigev_signo hav?
ing a default signal number, and the sigev_value member having the
value of the timer ID.

Each implementation shall define a set of clocks that can be used as
timing bases for per-process timers. All implementations shall support
a clock_id of CLOCK_REALTIME. If the Monotonic Clock option is sup?
ported, implementations shall support a clock_id of CLOCK_MONOTONIC.

Per-process timers shall not be inherited by a child process across a
fork() and shall be disarmed and deleted by an exec.

If _POSIX_CPUTIME is defined, implementations shall support clock_id
values representing the CPU-time clock of the calling process.

If _POSIX_THREAD_CPUTIME is defined, implementations shall support
clock_id values representing the CPU-time clock of the calling thread.

It is implementation-defined whether a timer_create() function will
succeed if the value defined by clock_id corresponds to the CPU-time
clock of a process or thread different from the process or thread in?
voking the function.

If evp->sigev_sigev_notify is SIGEV_THREAD and sev->sigev_notify_at?
tributes is not NULL, if the attribute pointed to by sev->sigev_no?
tify_attributes has a thread stack address specified by a call to
pthread_attr_setstack(), the results are unspecified if the signal is
generated more than once.

RETURN VALUE

If the call succeeds, timer_create() shall return zero and update the
location referenced by timerid to a timer_t, which can be passed to the
per-process timer calls. If an error occurs, the function shall return
a value of -1 and set errno to indicate the error. The value of timerid
is undefined if an error occurs.

ERRORS

The timer_create() function shall fail if:

EAGAIN The system lacks sufficient signal queuing resources to honor the request.

EAGAIN The calling process has already created all of the timers it is allowed by this implementation.

EINVAL The specified clock ID is not defined.

ENOTSUP

The implementation does not support the creation of a timer at? tached to the CPU-time clock that is specified by clock_id and associated with a process or thread different from the process or thread invoking timer_create().

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

If a timer is created which has evp->sigev_sigev_notify set to SIGEV_THREAD and the attribute pointed to by evp->sigev_notify_at? tributes has a thread stack address specified by a call to pthread_attr_setstack(), the memory dedicated as a thread stack cannot be recovered. The reason for this is that the threads created in re? sponse to a timer expiration are created detached, or in an unspecified way if the thread attribute's detachstate is PTHREAD_CREATE_JOINABLE. In neither case is it valid to call pthread_join(), which makes it im? possible to determine the lifetime of the created thread which thus means the stack memory cannot be reused.

RATIONALE

Periodic Timer Overrun and Resource Allocation

The specified timer facilities may deliver realtime signals (that is, queued signals) on implementations that support this option. Since re? altime applications cannot afford to lose notifications of asynchronous events, like timer expirations or asynchronous I/O completions, it must be possible to ensure that sufficient resources exist to deliver the signal when the event occurs. In general, this is not a difficulty be?

cause there is a one-to-one correspondence between a request and a sub?
sequent signal generation. If the request cannot allocate the signal
delivery resources, it can fail the call with an [EAGAIN] error.
Periodic timers are a special case. A single request can generate an
unspecified number of signals. This is not a problem if the requesting
process can service the signals as fast as they are generated, thus
making the signal delivery resources available for delivery of subse?
quent periodic timer expiration signals. But, in general, this cannot
be assured?processing of periodic timer signals may ``overrun''; that
is, subsequent periodic timer expirations may occur before the cur?
rently pending signal has been delivered.
Also, for signals, according to the POSIX.1?1990 standard, if subse?
quent occurrences of a pending signal are generated, it is implementa?
tion-defined whether a signal is delivered for each occurrence. This is
not adequate for some realtime applications. So a mechanism is required
to allow applications to detect how many timer expirations were delayed
without requiring an indefinite amount of system resources to store the
delayed expirations.
The specified facilities provide for an overrun count. The overrun
count is defined as the number of extra timer expirations that occurred
between the time a timer expiration signal is generated and the time
the signal is delivered. The signal-catching function, if it is con?
cerned with overruns, can retrieve this count on entry. With this
method, a periodic timer only needs one ``signal queuing resource''
that can be allocated at the time of the timer_create() function call.
A function is defined to retrieve the overrun count so that an applica?
tion need not allocate static storage to contain the count, and an im?
plementation need not update this storage asynchronously on timer expi?
rations. But, for some high-frequency periodic applications, the over?
head of an additional system call on each timer expiration may be pro?
hibitive. The functions, as defined, permit an implementation to main?
tain the overrun count in user space, associated with the timerid. The
timer_getoverrun() function can then be implemented as a macro that

uses the timerid argument (which may just be a pointer to a user space structure containing the counter) to locate the overrun count with no system call overhead. Other implementations, less concerned with this class of applications, can avoid the asynchronous update of user space by maintaining the count in a system structure at the cost of the extra system call to obtain it.

Timer Expiration Signal Parameters

The Realtime Signals Extension option supports an application-specific datum that is delivered to the extended signal handler. This value is explicitly specified by the application, along with the signal number to be delivered, in a sigevent structure. The type of the application-defined value can be either an integer constant or a pointer. This ex? plicit specification of the value, as opposed to always sending the timer ID, was selected based on existing practice.

It is common practice for realtime applications (on non-POSIX systems or realtime extended POSIX systems) to use the parameters of event han? dlers as the case label of a switch statement or as a pointer to an ap? plication-defined data structure. Since timer_ids are dynamically allo? cated by the timer_create() function, they can be used for neither of these functions without additional application overhead in the signal handler; for example, to search an array of saved timer IDs to asso? ciate the ID with a constant or application data structure.

FUTURE DIRECTIONS

None.

SEE ALSO

clock_getres(), timer_delete(), timer_getoverrun()

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

event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.ker?nel.org/doc/man-pages/reporting_bugs.html .