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 'clock_getres.3p' command

$ man clock_getres.3p

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

clock_getres, clock_gettime, clock_settime ? clock and timer functions

SYNOPSIS

```
#include <time.h>
int clock_getres(clockid_t clock_id, struct timespec *res);
int clock_gettime(clockid_t clock_id, struct timespec *tp);
int clock_settime(clockid_t clock_id, const struct timespec *tp);
```

DESCRIPTION

The clock_getres() function shall return the resolution of any clock. Clock resolutions are implementation-defined and cannot be set by a process. If the argument res is not NULL, the resolution of the speci? fied clock shall be stored in the location pointed to by res. If res is NULL, the clock resolution is not returned. If the time argument of clock_settime() is not a multiple of res, then the value is truncated to a multiple of res.

The clock_gettime() function shall return the current value tp for the specified clock, clock_id.

The clock_settime() function shall set the specified clock, clock_id, to the value specified by tp. Time values that are between two consec? utive non-negative integer multiples of the resolution of the specified clock shall be truncated down to the smaller multiple of the resolu? tion.

A clock may be system-wide (that is, visible to all processes) or per-process (measuring time that is meaningful only within a process). All implementations shall support a clock_id of CLOCK_REALTIME as defined in <time.h>. This clock represents the clock measuring real time for the system. For this clock, the values returned by clock_gettime() and specified by clock_settime() represent the amount of time (in seconds and nanoseconds) since the Epoch. An implementation may also support additional clocks. The interpretation of time values for these clocks is unspecified.

If the value of the CLOCK_REALTIME clock is set via clock_settime(), the new value of the clock shall be used to determine the time of expi? ration for absolute time services based upon the CLOCK_REALTIME clock. This applies to the time at which armed absolute timers expire. If the absolute time requested at the invocation of such a time service is be? fore the new value of the clock, the time service shall expire immedi? ately as if the clock had reached the requested time normally.

Setting the value of the CLOCK_REALTIME clock via clock_settime() shall have no effect on threads that are blocked waiting for a relative time service based upon this clock, including the nanosleep() function; nor on the expiration of relative timers based upon this clock. Conse? quently, these time services shall expire when the requested relative interval elapses, independently of the new or old value of the clock.

If the Monotonic Clock option is supported, all implementations shall support a clock_id of CLOCK_MONOTONIC defined in <time.h>. This clock represents the monotonic clock for the system. For this clock, the value returned by clock_gettime() represents the amount of time (in seconds and nanoseconds) since an unspecified point in the past (for example, system start-up time, or the Epoch). This point does not

change after system start-up time. The value of the CLOCK_MONOTONIC clock cannot be set via clock_settime(). This function shall fail if it is invoked with a clock_id argument of CLOCK_MONOTONIC.

The effect of setting a clock via clock_settime() on armed per-process timers associated with a clock other than CLOCK_REALTIME is implementa? tion-defined.

If the value of the CLOCK_REALTIME clock is set via clock_settime(), the new value of the clock shall be used to determine the time at which the system shall awaken a thread blocked on an absolute clock_nanosleep() call based upon the CLOCK_REALTIME clock. If the ab? solute time requested at the invocation of such a time service is be? fore the new value of the clock, the call shall return immediately as if the clock had reached the requested time normally.

Setting the value of the CLOCK_REALTIME clock via clock_settime() shall have no effect on any thread that is blocked on a relative clock_nanosleep() call. Consequently, the call shall return when the requested relative interval elapses, independently of the new or old value of the clock.

Appropriate privileges to set a particular clock are implementation-de? fined.

If _POSIX_CPUTIME is defined, implementations shall support clock ID values obtained by invoking clock_getcpuclockid(), which represent the CPU-time clock of a given process. Implementations shall also support the special clockid_t value CLOCK_PROCESS_CPUTIME_ID, which represents the CPU-time clock of the calling process when invoking one of the clock_*() or timer_*() functions. For these clock IDs, the values re? turned by clock_gettime() and specified by clock_settime() represent the amount of execution time of the process associated with the clock.

Changing the value of a CPU-time clock via clock_settime() shall have no effect on the behavior of the sporadic server scheduling policy (see Scheduling Policies).

If _POSIX_THREAD_CPUTIME is defined, implementations shall support clock ID values obtained by invoking pthread_getcpuclockid(), which

represent the CPU-time clock of a given thread. Implementations shall also support the special clockid_t value CLOCK_THREAD_CPUTIME_ID, which represents the CPU-time clock of the calling thread when invoking one of the clock_*() or timer_*() functions. For these clock IDs, the val? ues returned by clock_gettime() and specified by clock_settime() shall represent the amount of execution time of the thread associated with the clock. Changing the value of a CPU-time clock via clock_settime() shall have no effect on the behavior of the sporadic server scheduling policy (see Scheduling Policies).

RETURN VALUE

A return value of 0 shall indicate that the call succeeded. A return value of -1 shall indicate that an error occurred, and errno shall be set to indicate the error.

ERRORS

The clock_getres(), clock_gettime(), and clock_settime() functions shall fail if:

EINVAL The clock_id argument does not specify a known clock.

The clock_gettime() function shall fail if:

EOVERFLOW

The number of seconds will not fit in an object of type time_t.

The clock_settime() function shall fail if:

EINVAL The tp argument to clock_settime() is outside the range for the given clock ID.

EINVAL The tp argument specified a nanosecond value less than zero or greater than or equal to 1000 million.

EINVAL The value of the clock_id argument is CLOCK_MONOTONIC.

The clock_settime() function may fail if:

EPERM The requesting process does not have appropriate privileges to set the specified clock.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Note that the absolute value of the monotonic clock is meaningless (be?
cause its origin is arbitrary), and thus there is no need to set it.
Furthermore, realtime applications can rely on the fact that the value
of this clock is never set and, therefore, that time intervals measured
with this clock will not be affected by calls to clock_settime().

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Scheduling Policies, clock_getcpuclockid(), clock_nanosleep(), ctime(),
mq_receive(), mq_send(), nanosleep(), pthread_mutex_timedlock(),
sem_timedwait(), time(), timer_create(), timer_getoverrun()
The Base Definitions volume of POSIX.1?2017, <time.h>

COPYRIGHT