NAME

clock_getres, clock_gettime, clock_settime − clock and time functions

SYNOPSIS

#include <time.h>

int clock_getres(clockid_t *clk_id***, struct timespec ****res***);**

int clock_gettime(clockid_t *clk_id***, struct timespec ****tp***);**

int clock_settime(clockid_t *clk_id***, const struct timespec ****tp***);**

Link with *−lrt*.

Feature Test Macro Requirements for glibc (see **feature_test_macros**(7)):

clock_getres(), **clock_gettime**(), **clock_settime**(): _POSIX_C_SOURCE >= 199309L

DESCRIPTION

The function **clock_getres**() finds the resolution (precision) of the specified clock *clk_id*, and, if *res* is non-NULL, stores it in the *struct timespec* pointed to by *res*. The resolution of clocks depends on the implementation and cannot be configured by a particular process. If the time value pointed to by the argument *tp* of **clock_settime**() is not a multiple of *res*, then it is truncated to a multiple of *res*.

The functions **clock_gettime**() and **clock_settime**() retrieve and set the time of the specified clock *clk_id*.

The *res* and *tp* arguments are *timespec* structures, as specified in <*time.h*>:

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

The *clk_id* argument is the identifier of the particular clock on which to act. A clock may be system-wide and hence visible for all processes, or per-process if it measures time only within a single process.

All implementations support the system-wide real-time clock, which is identified by **CLOCK_REALTIME**. Its time represents seconds and nanoseconds since the Epoch. When its time is changed, timers for a relative interval are unaffected, but timers for an absolute point in time are affected.

More clocks may be implemented. The interpretation of the corresponding time values and the effect on timers is unspecified.

Sufficiently recent versions of glibc and the Linux kernel support the following clocks:

CLOCK_REALTIME
System-wide real-time clock. Setting this clock requires appropriate privileges.

CLOCK_REALTIME_COARSE (since Linux 2.6.32; Linux-specific)
A faster but less precise version of **CLOCK_REALTIME**. Use when you need very fast, but not fine-grained timestamps.

CLOCK_MONOTONIC
Clock that cannot be set and represents monotonic time since some unspecified starting point.

CLOCK_MONOTONIC_COARSE (since Linux 2.6.32; Linux-specific)
A faster but less precise version of **CLOCK_MONOTONIC**. Use when you need very fast, but not fine-grained timestamps.

CLOCK_MONOTONIC_RAW (since Linux 2.6.28; Linux-specific)
Similar to **CLOCK_MONOTONIC**, but provides access to a raw hardware-based time that is not subject to NTP adjustments.

CLOCK_PROCESS_CPUTIME_ID
High-resolution per-process timer from the CPU.

CLOCK_THREAD_CPUTIME_ID
Thread-specific CPU-time clock.

RETURN VALUE

clock_gettime(), **clock_settime**() and **clock_getres**() return 0 for success, or −1 for failure (in which case *errno* is set appropriately).

ERRORS

EFAULT
tp points outside the accessible address space.

EINVAL
The *clk_id* specified is not supported on this system.

EPERM
clock_settime() does not have permission to set the clock indicated.

CONFORMING TO

SUSv2, POSIX.1-2001.

AVAILABILITY

On POSIX systems on which these functions are available, the symbol **_POSIX_TIMERS** is defined in <*unistd.h*> to a value greater than 0. The symbols **_POSIX_MONOTONIC_CLOCK**, **_POSIX_CPUTIME**, **_POSIX_THREAD_CPUTIME** indicate that **CLOCK_MONOTONIC**, **CLOCK_PROCESS_CPUTIME_ID**, **CLOCK_THREAD_CPUTIME_ID** are available. (See also **sysconf**(3).)

NOTES

Note for SMP systems

The **CLOCK_PROCESS_CPUTIME_ID** and **CLOCK_THREAD_CPUTIME_ID** clocks are realized on many platforms using timers from the CPUs (TSC on i386, AR.ITC on Itanium). These registers may differ between CPUs and as a consequence these clocks may return **bogus results** if a process is migrated to another CPU.

If the CPUs in an SMP system have different clock sources then there is no way to maintain a correlation between the timer registers since each CPU will run at a slightly different frequency. If that is the case then *clock_getcpuclockid(0)* will return **ENOENT** to signify this condition. The two clocks will then only be useful if it can be ensured that a process stays on a certain CPU.

The processors in an SMP system do not start all at exactly the same time and therefore the timer registers are typically running at an offset. Some architectures include code that attempts to limit these offsets on bootup. However, the code cannot guarantee to accurately tune the offsets. Glibc contains no provisions to deal with these offsets (unlike the Linux Kernel). Typically these offsets are small and therefore the effects may be negligible in most cases.

BUGS

According to POSIX.1-2001, the **CLOCK_PROCESS_CPUTIME_ID** and **CLOCK_THREAD_CPUTIME_ID** clocks should be settable using **clock_settime**(). However, the clocks currently are not settable.

SEE ALSO

date(1), **adjtimex**(2), **gettimeofday**(2), **settimeofday**(2), **time**(2), **clock_getcpuclockid**(3), **ctime**(3), **ftime**(3), **pthread_getcpuclockid**(3), **sysconf**(3), **time**(7)

COLOPHON

This page is part of release 3.22 of the Linux *man-pages* project. A description of the project, and information about reporting bugs, can be found at http://www.kernel.org/doc/man-pages/.