NAME

getitimer, setitimer – get or set value of interval timer

SYNOPSIS

#include <sys/time.h>

int getitimer(int *which***, struct itimerval** **value***);**

int setitimer(int *which***, const struct itimerval** **value***, struct itimerval** **ovalue***);**

MT-LEVEL

MT-Safe

DESCRIPTION

The system provides each process with four interval timers, defined in **sys/time.h**. The **getitimer( )** function stores the current value of the timer specified by *which* into the structure pointed to by *value*. The **setitimer( )** call sets the value of the timer specified by *which* to the value specified in the structure pointed to by *value*, and if *ovalue* is not **NULL**, stores the previous value of the timer in the structure pointed to by *ovalue*.

A timer value is defined by the **itimerval** structure (see **gettimeofday**(3C) for the definition of **timeval**), which includes the following members:

```
struct timeval    it_interval;    /* timer interval */
struct timeval    it_value;       /* current value */
```

it_value indicates the time to the next timer expiration. **it_interval** specifies a value to be used in reloading **it_value** when the timer expires. Setting **it_value** to zero disables a timer, regardless of the value of **it_interval**. Setting **it_interval** to zero disables a timer after its next expiration (assuming **it_value** is non-zero).

Time values smaller than the resolution of the system clock are rounded up to the resolution of the system clock, except for **ITIMER_REALPROF**, whose values are rounded up to the resolution of the profiling clock.

The four timers are:

ITIMER_REAL

Decrements in real time. A **SIGALRM** signal is delivered when this timer expires.

In the current and previous releases, when **setitimer(ITIMER_REAL, . . .)** is called in a multi-thread process linked with **-lthread** (Solaris threads) or **-lpthread** (POSIX threads), the resulting **SIGALRM** is sent to the bound thread that called **setitimer( )**, i.e. **setitimer( )** has a per-thread semantic when called from a bound thread. This semantic will become obsolete in a future release. The semantic will move to a per-process semantic, i.e. the resulting **SIGALRM** will be sent to the process. The **SIGALRM** so generated is not maskable on this bound thread via any signal masking function, **pthread_sigmask**(3T), **thr_sigsetmask**(3T), or **sigprocmask**(2). This is a bug that will not be fixed, since the per-thread semantic will be discontinued in the next release.

Also, calling this routine from an unbound thread is not guaranteed to work as in the case of bound threads. The resulting **SIGALRM** may be sent to some other thread (see **alarm**(2)). This is a bug and will not be fixed since the per-thread semantic is going to be discontinued.

Calling **setitimer(ITIMER_REAL, . . .)** from a process linked with **-lpthread** (POSIX threads) has the same behavior as Solaris threads described above, where a Solaris bound thread is the same as a POSIX thread in system scheduling scope and a Solaris unbound thread is the same as a POSIX thread in local scheduling scope.

Hence, for multi-threaded (Solaris or POSIX) programs in the current and previous releases, the only reliable way to use the **ITIMER_REAL** flag is to call it from a bound thread which does not mask **SIGALRM** and to expect the **SIGALRM** to be delivered to this bound thread.

The current working of this flag is not being improved since some applications might depend on the current (slightly broken) semantic. When this semantic is discontinued in the future, it will be replaced with a per-process semantic, i.e. using this flag from any thread, bound or unbound, will result in the **SIGALRM** being sent to the process.

New MT applications should not use this flag, and should use **alarm**(2) instead.

ITIMER_VIRTUAL Decrements in process virtual time. It runs only when the process is executing. A **SIGVTALRM** signal is delivered when it expires. (For multi-threaded programs see ''Warnings'' section below).

ITIMER_PROF Decrements both in process virtual time and when the system is running on behalf of the process. It is designed to be used by interpreters in statistically profiling the execution of interpreted programs. Each time the **ITIMER_PROF** timer expires, the **SIGPROF** signal is delivered. Because this signal may interrupt in-progress functions, programs using this timer must be prepared to restart interrupted functions. (For multi-threaded programs see ''Warnings'' section below).

ITIMER_REALPROF Decrements in real time. It is designed to be used for real-time profiling of multithreaded programs. Each time the **ITIMER_REALPROF** timer expires, one counter in a set of counters maintained by the system for each lightweight process (lwp) is incremented. The counter corresponds to the state of the lwp at the time of the timer tick. All lwps executing in user mode when the timer expires are interrupted into system mode. When each lwp resumes execution in user mode, if any of the elements in its set of counters are non-zero, the **SIGPROF** signal is delivered to the lwp. The **SIGPROF** signal is delivered before any other signal except **SIGKILL.** This signal does not interrupt any in-progress function. A **siginfo** structure, defined in **sys/siginfo.h**, is associated with the delivery of the **SIGPROF** signal, and includes the following members:

```
si_tstamp;      /* high resolution timestamp */
si_syscall;     /* current syscall */
si_nsysarg;     /* number of syscall arguments */
si_sysarg[ ];   /* actual syscall arguments */
si_fault;       /* last fault type */
si_faddr;       /* last fault address */
si_mstate[ ];   /* ticks in each microstate */
```

The enumeration of microstates (indices into **si_mstate)** is defined in **sys/msacct.h**. (For multi-threaded programs see ''Warnings'' section below).

RETURN VALUES

If the calls succeed, a value of 0 is returned. If an error occurs, the value −1 is returned, and an error code is placed in the global variable **errno.**

ERRORS

getitimer( ) and **setitimer( )** will fail if:

EINVAL The specified number of seconds is greater than 100,000,000, the number of microseconds is greater than or equal to 1,000,000, or the *which* parameter is unrecognized.

setitimer( ) will fail if:

EACCES An unbound Solaris thread or a POSIX thread in local scheduling scope, with a flag other than **ITIMER_REAL,** called **setitimer( )**.

SEE ALSO

alarm(2), **sigprocmask**(2), **gettimeofday**(3C), **sysconf**(3C), **pthread_attr_setscope**(3T), **pthread_sigmask**(3T)

WARNINGS

All flags to **setitimer( )**, other than **ITIMER_REAL,** only behave as documented with ''bound'' threads. Additionally, their ability to mask the signal only works with bound threads. If the call is made using one of these flags from an unbound thread, the system call returns **-1** and sets **errno** to **EACCES.**

These behaviors are the same for bound or unbound POSIX threads. (A POSIX thread with system-wide scope, created by the call

pthread_attr_setscope(&attr**, PTHREAD_SCOPE_SYSTEM);**

is equivalent to a Solaris bound thread. A POSIX thread with local process scope, created by the call

pthread_attr_setscope(&attr**, PTHREAD_SCOPE_PROCESS);**

is equivalent to a Solaris unbound thread.

NOTES

The microseconds field should not be equal to or greater than one second.

setitimer( ) is independent of the **alarm( )** function.

Do not use **setitimer(ITIMER_REAL)** with the **sleep( )** routine. A **sleep( )** wipes out knowledge of the user signal handler for **SIGALRM.**

ITIMER_PROF and **ITIMER_REALPROF** deliver the same signal and have different semantics. They cannot be used together.

The granularity of the resolution of alarm time is platform-dependent.