NAME

msleep, **msleep_sbt**, **msleep_spin**, **msleep_spin_sbt**, **pause**, **pause_sig**, **pause_sbt**, **tsleep**, **tsleep_sbt**, **wakeup**, **wakeup_one**, **wakeup_any** - wait for events

SYNOPSIS

#include <sys/param.h>
#include <sys/systm.h>
#include <sys/proc.h>

int
msleep(*const void *chan, struct mtx *mtx, int priority, const char *wmesg, int timo*);

int
msleep_sbt(*const void *chan, struct mtx *mtx, int priority, const char *wmesg, sbintime_t sbt, sbintime_t pr, int flags*);

int
msleep_spin(*const void *chan, struct mtx *mtx, const char *wmesg, int timo*);

int
msleep_spin_sbt(*const void *chan, struct mtx *mtx, const char *wmesg, sbintime_t sbt, sbintime_t pr, int flags*);

int
pause(*const char *wmesg, int timo*);

int
pause_sig(*const char *wmesg, int timo*);

int
pause_sbt(*const char *wmesg, sbintime_t sbt, sbintime_t pr, int flags*);

int
tsleep(*const void *chan, int priority, const char *wmesg, int timo*);

int
tsleep_sbt(*const void *chan, int priority, const char *wmesg, sbintime_t sbt, sbintime_t pr, int flags*);

void
wakeup(*const void *chan*);

void
wakeup_one(*const void *chan*);

void
wakeup_any(*const void *chan*);

DESCRIPTION

The functions **tsleep**(), **msleep**(), **msleep_spin**(), **pause**(), **pause_sig**(), **pause_sbt**(), **wakeup**(), **wakeup_one**(), and **wakeup_any**() handle event-based thread blocking. If a thread must wait for an external event, it is put to sleep by **tsleep**(), **msleep**(), **msleep_spin**(), **pause**(), **pause_sig**(), or **pause_sbt**(). Threads may also wait using one of the locking primitive sleep routines mtx_sleep(9), rw_sleep(9), or sx_sleep(9).

The parameter *chan* is an arbitrary address that uniquely identifies the event on which the thread is being put to sleep. All threads sleeping on a single *chan* are woken up later by **wakeup**(), often called from inside an interrupt routine, to indicate that the resource the thread was blocking on is available now.

The parameter *priority* specifies a new priority for the thread as well as some optional flags. If the new priority is not 0, then the thread will be made runnable with the specified *priority* when it resumes. PZERO should never be used, as it is for compatibility only. A new priority of 0 means to use the thread's current priority when it is made runnable again.

If *priority* includes the PCATCH flag, pending signals are allowed to interrupt the sleep, otherwise pending signals are ignored during the sleep. If PCATCH is set and a signal becomes pending, ERESTART is returned if the current system call should be restarted if possible, and EINTR is returned if the system call should be interrupted by the signal (return EINTR).

The parameter *wmesg* is a string describing the sleep condition for tools like ps(1). Due to the limited space of those programs to display arbitrary strings, this message should not be longer than 6 characters.

The parameter *timo* specifies a timeout for the sleep. If *timo* is not 0, then the thread will sleep for at most *timo* / *hz* seconds. If the timeout expires, then the sleep function will return EWOULDBLOCK.

msleep_sbt(), **msleep_spin_sbt**(), **pause_sbt**() and **tsleep_sbt**() functions take *sbt* parameter instead of *timo*. It allows the caller to specify relative or absolute wakeup time with higher resolution in form of *sbintime_t*. The parameter *pr* allows the caller to specify wanted absolute event precision. The parameter *flags* allows the caller to pass additional **callout_reset_sbt**() flags.

Several of the sleep functions including **msleep**(), **msleep_spin**(), and the locking primitive sleep routines specify an additional lock parameter. The lock will be released before sleeping and reacquired

before the sleep routine returns. If *priority* includes the PDROP flag, then the lock will not be reacquired before returning. The lock is used to ensure that a condition can be checked atomically, and that the current thread can be suspended without missing a change to the condition, or an associated wakeup. In addition, all of the sleep routines will fully drop the *Giant* mutex (even if recursed) while the thread is suspended and will reacquire the *Giant* mutex before the function returns. Note that the *Giant* mutex may be specified as the lock to drop. In that case, however, the PDROP flag is not allowed.

To avoid lost wakeups, either a lock should be used to protect against races, or a timeout should be specified to place an upper bound on the delay due to a lost wakeup. As a result, the **tsleep**() function should only be invoked with a timeout of 0 when the *Giant* mutex is held.

The **msleep**() function requires that *mtx* reference a default, i.e. non-spin, mutex. Its use is deprecated in favor of mtx_sleep(9) which provides identical behavior.

The **msleep_spin**() function requires that *mtx* reference a spin mutex. The **msleep_spin**() function does not accept a *priority* parameter and thus does not support changing the current thread's priority, the PDROP flag, or catching signals via the PCATCH flag.

The **pause**() function is a wrapper around **tsleep**() that suspends execution of the current thread for the indicated timeout. The thread can not be awakened early by signals or calls to **wakeup**(), **wakeup_one**() or **wakeup_any**(). The **pause_sig**() function is a variant of **pause**() which can be awakened early by signals.

The **wakeup_one**() function makes the first highest priority thread in the queue that is sleeping on the parameter *chan* runnable. This reduces the load when a large number of threads are sleeping on the same address, but only one of them can actually do any useful work when made runnable.

Due to the way it works, the **wakeup_one**() function requires that only related threads sleep on a specific *chan* address. It is the programmer's responsibility to choose a unique *chan* value. The older **wakeup**() function did not require this, though it was never good practice for threads to share a *chan* value. When converting from **wakeup**() to **wakeup_one**(), pay particular attention to ensure that no other threads wait on the same *chan*.

The **wakeup_any**() function is similar to **wakeup_one**(), except that it makes runnable last thread on the queue (sleeping less), ignoring fairness. It can be used when threads sleeping on the *chan* are known to be identical and there is no reason to be fair.

If the timeout given by *timo* or *sbt* is based on an absolute real-time clock value, then the thread should copy the global *rtc_generation* into its *td_rtcgen* member before reading the RTC. If the real-time clock is adjusted, these functions will set *td_rtcgen* to zero and return zero. The caller should reconsider its

orientation with the new RTC value.

RETURN VALUES

When awakened by a call to **wakeup**() or **wakeup_one**(), if a signal is pending and PCATCH is specified, a non-zero error code is returned. If the thread is awakened by a call to **wakeup**() or **wakeup_one**(), the **msleep**(), **msleep_spin**(), **tsleep**(), and locking primitive sleep functions return 0. Zero can also be returned when the real-time clock is adjusted; see above regarding *td_rtcgen*. Otherwise, a non-zero error code is returned.

ERRORS

msleep(), **msleep_spin**(), **tsleep**(), and the locking primitive sleep functions will fail if:

[EINTR] The PCATCH flag was specified, a signal was caught, and the system call should be interrupted.

[ERESTART] The PCATCH flag was specified, a signal was caught, and the system call should be restarted.

[EWOULDBLOCK] A non-zero timeout was specified and the timeout expired.

SEE ALSO

ps(1), callout(9), locking(9), malloc(9), mi_switch(9), mtx_sleep(9), rw_sleep(9), sx_sleep(9)

HISTORY

The functions **sleep**() and **wakeup**() were present in Version 1 AT&T UNIX. They were probably also present in the preceding PDP-7 version of UNIX. They were the basic process synchronization model.

The **tsleep**() function appeared in 4.4BSD and added the parameters *wmesg* and *timo*. The **sleep**() function was removed in FreeBSD 2.2. The **wakeup_one**() function appeared in FreeBSD 2.2. The **msleep**() function appeared in FreeBSD 5.0, and the **msleep_spin**() function appeared in FreeBSD 6.2. The **pause**() function appeared in FreeBSD 7.0. The **pause_sig**() function appeared in FreeBSD 12.0.

AUTHORS

This manual page was written by Jörg Wunsch <*joerg@FreeBSD.org*>.