NAME

init_sleepqueues, **sleepq_abort**, **sleepq_add**, **sleepq_alloc**, **sleepq_broadcast**, **sleepq_free**, **sleepq_lock**, **sleepq_lookup**, **sleepq_release**, **sleepq_remove**, **sleepq_signal**, **sleepq_set_timeout**, **sleepq_set_timeout_sbt**, **sleepq_sleepcnt**, **sleepq_timedwait**, **sleepq_timedwait_sig**, **sleepq_type**, **sleepq_wait**, **sleepq_wait_sig** - manage the queues of sleeping threads

SYNOPSIS

#include <sys/param.h>
#include <sys/sleepqueue.h>

void
init_sleepqueues(*void*);

int
sleepq_abort(*struct thread *td*);

void
sleepq_add(*const void *wchan*, *struct lock_object *lock*, *const char *wmesg*, *int flags*, *int queue*);

*struct sleepqueue **
sleepq_alloc(*void*);

int
sleepq_broadcast(*const void *wchan*, *int flags*, *int pri*, *int queue*);

void
sleepq_free(*struct sleepqueue *sq*);

*struct sleepqueue **
sleepq_lookup(*const void *wchan*);

void
sleepq_lock(*const void *wchan*);

void
sleepq_release(*const void *wchan*);

void
sleepq_remove(*struct thread *td*, *const void *wchan*);

int
sleepq_signal(*const void *wchan, int flags, int pri, int queue*);

void
sleepq_set_timeout(*const void *wchan, int timo*);

void
sleepq_set_timeout_sbt(*const void *wchan, sbintime_t sbt, sbintime_t pr, int flags*);

u_int
sleepq_sleepcnt(*const void *wchan, int queue*);

int
sleepq_timedwait(*const void *wchan, int pri*);

int
sleepq_timedwait_sig(*const void *wchan, int pri*);

int
sleepq_type(*const void *wchan*);

void
sleepq_wait(*const void *wchan, int pri*);

int
sleepq_wait_sig(*const void *wchan, int pri*);

DESCRIPTION

Sleep queues provide a mechanism for suspending execution of a thread until some condition is met. Each queue is associated with a specific wait channel when it is active, and only one queue may be associated with a wait channel at any given point in time. The implementation of each wait channel splits its sleepqueue into 2 sub-queues in order to enable some optimizations on threads' wakeups. An active queue holds a list of threads that are blocked on the associated wait channel. Threads that are not blocked on a wait channel have an associated inactive sleep queue. When a thread blocks on a wait channel it donates its inactive sleep queue to the wait channel. When a thread is resumed, the wait channel that it was blocked on gives it an inactive sleep queue for later use.

The **sleepq_alloc**() function allocates an inactive sleep queue and is used to assign a sleep queue to a thread during thread creation. The **sleepq_free**() function frees the resources associated with an inactive sleep queue and is used to free a queue during thread destruction.

Active sleep queues are stored in a hash table hashed on the addresses pointed to by wait channels. Each bucket in the hash table contains a sleep queue chain. A sleep queue chain contains a spin mutex and a list of sleep queues that hash to that specific chain. Active sleep queues are protected by their chain's spin mutex. The **init_sleepqueues**() function initializes the hash table of sleep queue chains.

The **sleepq_lock**() function locks the sleep queue chain associated with wait channel *wchan*.

The **sleepq_lookup**() returns a pointer to the currently active sleep queue for that wait channel associated with *wchan* or NULL if there is no active sleep queue associated with argument *wchan*. It requires the sleep queue chain associated with *wchan* to have been locked by a prior call to **sleepq_lock**().

The **sleepq_release**() function unlocks the sleep queue chain associated with **wchan**() and is primarily useful when aborting a pending sleep request before one of the wait functions is called.

The **sleepq_add**() function places the current thread on the sleep queue associated with the wait channel *wchan*. The sleep queue chain associated with argument *wchan* must be locked by a prior call to **sleepq_lock**() when this function is called. If a lock is specified via the *lock* argument, and if the kernel was compiled with **options INVARIANTS**, then the sleep queue code will perform extra checks to ensure that the lock is used by all threads sleeping on *wchan*. The *wmesg* parameter should be a short description of *wchan*. The *flags* parameter is a bitmask consisting of the type of sleep queue being slept on and zero or more optional flags. The *queue* parameter specifies the sub-queue, in which the contending thread will be inserted.

There are currently three types of sleep queues:

SLEEPQ_CONDVAR A sleep queue used to implement condition variables.
SLEEPQ_SLEEP A sleep queue used to implement sleep(9), wakeup(9) and wakeup_one(9).
SLEEPQ_PAUSE A sleep queue used to implement pause(9).

There are currently two optional flag:

SLEEPQ_INTERRUPTIBLE The current thread is entering an interruptible sleep.
SLEEPQ_STOP_ON_BDRY When thread is entering an interruptible sleep, do not stop it upon arrival of stop action, like SIGSTOP. Wake it up instead.

A timeout on the sleep may be specified by calling **sleepq_set_timeout**() after **sleepq_add**(). The *wchan* parameter should be the same value from the preceding call to **sleepq_add**(), and the sleep queue chain associated with *wchan* must have been locked by a prior call to **sleepq_lock**(). The *timo* parameter should specify the timeout value in ticks.

sleepq_set_timeout_sbt() function takes *sbt* argument instead of *timo*. It allows to specify relative or absolute wakeup time with higher resolution in form of *sbintime_t*. The parameter *pr* allows to specify wanted absolute event precision. The parameter *flags* allows to pass additional **callout_reset_sbt**() flags.

Once the thread is ready to suspend, one of the wait functions is called to put the current thread to sleep until it is awakened and to context switch to another thread. The **sleepq_wait**() function is used for non-interruptible sleeps that do not have a timeout. The **sleepq_timedwait**() function is used for non-interruptible sleeps that have had a timeout set via **sleepq_set_timeout**(). The **sleepq_wait_sig**() function is used for interruptible sleeps that do not have a timeout. The **sleepq_timedwait_sig**() function is used for interruptible sleeps that do have a timeout set. The *wchan* argument to all of the wait functions is the wait channel being slept on. The sleep queue chain associated with argument *wchan* needs to have been locked with a prior call to **sleepq_lock**(). The *pri* argument is used to set the priority of the thread when it is awakened. If it is set to zero, the thread's priority is left alone.

When the thread is resumed, the wait functions return a non-zero value if the thread was awakened due to an interrupt other than a signal or a timeout. If the sleep timed out, then EWOULDBLOCK is returned. If the sleep was interrupted by something other than a signal, then some other return value will be returned.

A sleeping thread is normally resumed by the **sleepq_broadcast**() and **sleepq_signal**() functions. The **sleepq_signal**() function awakens the highest priority thread sleeping on a wait channel (if SLEEPQ_UNFAIR flag is set, thread that went to sleep recently) while **sleepq_broadcast**() awakens all of the threads sleeping on a wait channel. The *wchan* argument specifics which wait channel to awaken. The *flags* argument must match the sleep queue type contained in the *flags* argument passed to **sleepq_add**() by the threads sleeping on the wait channel. If the *pri* argument does not equal -1, then each thread that is awakened will have its priority raised to *pri* if it has a lower priority. The sleep queue chain associated with argument *wchan* must be locked by a prior call to **sleepq_lock**() before calling any of these functions. The *queue* argument specifies the sub-queue, from which threads need to be woken up.

A thread in an interruptible sleep can be interrupted by another thread via the **sleepq_abort**() function. The *td* argument specifies the thread to interrupt. An individual thread can also be awakened from sleeping on a specific wait channel via the **sleepq_remove**() function. The *td* argument specifies the thread to awaken and the *wchan* argument specifies the wait channel to awaken it from. If the thread *td* is not blocked on the wait channel *wchan* then this function will not do anything, even if the thread is asleep on a different wait channel. This function should only be used if one of the other functions above is not sufficient. One possible use is waking up a specific thread from a widely shared sleep channel.

The **sleepq_sleepcnt**() function offer a simple way to retrieve the number of threads sleeping for the specified *queue*, given a *wchan*.

The **sleepq_type**() function returns the type of *wchan* associated to a sleepqueue.

The **sleepq_abort**(), **sleepq_broadcast**(), and **sleepq_signal**() functions all return a boolean value. If the return value is true, then at least one thread was resumed that is currently swapped out. The caller is responsible for awakening the scheduler process so that the resumed thread will be swapped back in. This is done by calling the **kick_proc0**() function after releasing the sleep queue chain lock via a call to **sleepq_release**().

The sleep queue interface is currently used to implement the sleep(9) and condvar(9) interfaces. Almost all other code in the kernel should use one of those interfaces rather than manipulating sleep queues directly.

SEE ALSO

callout(9), condvar(9), runqueue(9), scheduler(9), sleep(9)