Xenomai Cobalt interface

2.99.0

Generated by Doxygen 1.7.1

Thu Jan 5 2012 10:14:34

Contents

Chapter 1

Deprecated List

Global rtdm_device::open_rt Only use non-real-time open handler in new drivers.

Global rtdm_device::socket_rt Only use non-real-time socket creation handler in new drivers.

Global rtdm_operations::close_rt Only use non-real-time close handler in new drivers.

Global rtdm_task_sleep_until(nanosecs_abs_t wakeup_time) Use rtdm_task_sleep_abs instead!

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 5

Module Documentation

5.1 Thread state flags.

Bits reporting permanent or transient states of thread.

Defines

- #define XNSUSP 0x00000001
  Suspended.
- #define XNPEND 0x00000002
  Sleep-wait for a resource.
- #define XNDELAY 0x00000004
  Delayed.
- #define XNREADY 0x00000008
  Linked to the ready queue.
- #define XNDORMANT 0x00000010
  Not started yet or killed.
- #define XNZOMBIE 0x00000020
  Zombie thread in deletion process.
- #define XNSTARTED 0x00000080
  Thread has been started.
- #define XNMAPPED 0x00000100
  Mapped to a regular Linux task (shadow only).
- #define XNRELAX 0x00000200
  Relaxed shadow thread (blocking bit).
- #define XNMIGRATE 0x00000400

Thread is currently migrating to another CPU.

- #define XNHELD 0x00000800

  Thread is held to process emergency.

- #define XNBOOST 0x00001000

  Undergoes a PIP boost.

- #define XNDEBUG 0x00002000

  Hit a debugger breakpoint (shadow only).

- #define XNLOCK 0x00004000

  Holds the scheduler lock (i.e.

- #define XNRRB 0x00008000

  Undergoes a round-robin scheduling.

- #define XNASDI 0x00010000

  ASR are disabled.

- #define XNDEFCAN 0x00020000

  Deferred cancelability mode (self-set only).

- #define XNTRAPSW 0x00040000

  Trap execution mode switches.

- #define XNFPU 0x00080000

  Thread uses FPU.

- #define XNSHADOW 0x00100000

  Shadow thread.

- #define XNROOT 0x00200000

  Root thread (that is, Linux/IDLE).

- #define XNOTHER 0x00400000

  Non real-time shadow (prio=0).

5.1.1 Detailed Description

Bits reporting permanent or transient states of thread.

5.1.2 Define Documentation

5.1.2.1 #define XNHELD 0x00000800

Thread is held to process emergency.

Referenced by xnpod_resume_thread(), and xnpod_suspend_thread().

5.1.2.2 #define XNLOCK 0x00004000

Holds the scheduler lock (i.e.

not preemptible)

Referenced by __xnpod_reset_thread(), pthread_set_mode_np(), xnpod_set_thread_mode(), and xnpod_welcome_thread().

5.1.2.3 #define XNMIGRATE 0x00000400

Thread is currently migrating to another CPU.

Referenced by xnpod_delete_thread().

5.1.2.4 #define XNPEND 0x00000002

Sleep-wait for a resource.

Referenced by xnpod_delete_thread(), xnpod_resume_thread(), xnpod_unblock_thread(), xnsynch_acquire(), xnsynch_flush(), xnsynch_forget_sleeper(), xnsynch_sleep_on(), xnsynch_wakeup_one_sleeper(), and xnsynch_wakeup_this_sleeper().

5.1.2.5 #define XNREADY 0x00000008

Linked to the ready queue.

Referenced by xnpod_delete_thread(), xnpod_resume_thread(), and xnpod_suspend_thread().

5.1.2.6 #define XNSUSP 0x00000001

Suspended.

Referenced by __xnpod_reset_thread(), xnpod_handle_exception(), xnpod_init_thread(), xnpod_start_thread(), and xnpod_suspend_thread().

5.2 Thread information flags.

Bits reporting events notified to the thread.

Defines

- #define XNTIMEO 0x00000001

  Woken up due to a timeout condition.

- #define XNRMID 0x00000002

  Pending on a removed resource.

- #define XNBREAK 0x00000004

  Forcibly awaken from a wait state.

- #define XNKICKED 0x00000008

  Forced out of primary mode (shadow only).

- #define XNWAKEN 0x00000010

  Thread waken up upon resource availability.

- #define XNROBBED 0x00000020

  Robbed from resource ownership.

- #define XNAFFSET 0x00000040

  CPU affinity changed from primary mode.

- #define XNPRIOSET 0x00000080

  Priority changed from primary mode.

- #define XNABORT 0x00000100

  Thread is being aborted.

- #define XNCANPND 0x00000200

  Cancellation request is pending.

- #define XNSWREP 0x00000400

  Mode switch already reported.

5.2.1 Detailed Description

Bits reporting events notified to the thread.

5.3 HAL.

Generic Adeos-based hardware abstraction layer.

Files

- file hal.c

  Adeos-based Real-Time Abstraction Layer for ARM.

- file hal.c

  Adeos-based Real-Time Abstraction Layer for the Blackfin architecture.

- file hal.c

  Generic Real-Time HAL.

- file hal.c

  Adeos-based Real-Time Abstraction Layer for the NIOS2 architecture.

- file hal.c

  Adeos-based Real-Time Abstraction Layer for PowerPC.

- file hal.c

  Adeos-based Real-Time Abstraction Layer for the SuperH architecture.

- file hal.c

  Adeos-based Real-Time Abstraction Layer for x86.

- file smi.c

  SMI workaround for x86.

Functions

- int rthal_apc_alloc (const char *name, void(*handler)(void *cookie), void *cookie)

  Allocate an APC slot.

- void rthal_apc_free (int apc)

  Releases an APC slot.

5.3.1 Detailed Description

Generic Adeos-based hardware abstraction layer.

5.3.2 Function Documentation

5.3.2.1 int rthal_apc_alloc (const char * *name*, void(*)(void *cookie) *handler*, void * *cookie*)

Allocate an APC slot.

APC is the acronym for Asynchronous Procedure Call, a mean by which activities from the Xenomai domain can schedule deferred invocations of handlers to be run into the Linux domain, as soon as possible when the Linux kernel gets back in control. Up to BITS_PER_LONG APC slots can be active at any point in time. APC support is built upon Adeos's virtual interrupt support.

The HAL guarantees that any Linux kernel service which would be callable from a regular Linux interrupt handler is also available to APC handlers.

Parameters

name is a symbolic name identifying the APC which will get reported through the /proc/xenomai/apc interface. Passing NULL to create an anonymous APC is allowed.

handler The address of the fault handler to call upon exception condition. The handle will be passed the *cookie* value unmodified.

cookie A user-defined opaque cookie the HAL will pass to the APC handler as its sole argument.

Returns

an valid APC id. is returned upon success, or a negative error code otherwise:

- -EINVAL is returned if *handler* is invalid.
- -EBUSY is returned if no more APC slots are available.

Environments:

This service can be called from:

- Linux domain context.

5.3.2.2 int rthal_apc_free (int *apc*)

Releases an APC slot.

This service deallocates an APC slot obtained by rthal_apc_alloc().

Parameters

apc The APC id. to release, as returned by a successful call to the rthal_apc_alloc() service.

Environments:

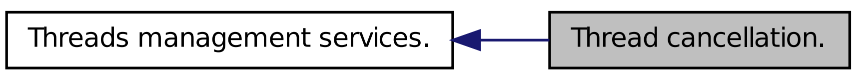
This service can be called from:

- Any domain context.

5.4 Thread cancellation.

Thread cancellation.

Collaboration diagram for Thread cancellation.:



Functions

- int pthread_cancel (pthread_t thread)
  Cancel a thread.

- void pthread_cleanup_push (cleanup_routine_t *routine, void *arg)
  Register a cleanup handler to be executed at the time of cancellation.
- void pthread_cleanup_pop (int execute)
  Unregister the last registered cleanup handler.
- int pthread_setcanceltype (int type, int *oldtype_ptr)
  Set cancelability type of the current thread.
- int pthread_setcancelstate (int state, int *oldstate_ptr)
  Set cancelability state of the current thread.

5.4.1 Detailed Description

Thread cancellation. Cancellation is the mechanism by which a thread can terminate the execution of a Xenomai POSIX skin thread (created with pthread_create()). More precisely, a thread can send a cancellation request to a Xenomai POSIX skin thread and depending on its cancelability type (see pthread_setcanceltype()) and state (see pthread_setcancelstate()), the target thread can then either ignore the request, honor it immediately, or defer it till it reaches a cancellation point. When threads are first created by pthread_create(), they always defer cancellation requests.

When a thread eventually honors a cancellation request, it behaves as if *pthread_exit(PTHREAD_-CANCELED)* was called. All cleanup handlers are executed in reverse order, finalization functions for thread-specific data are called, and finally the thread stops executing. If the canceled thread was joinable, the return value PTHREAD_CANCELED is provided to whichever thread calls pthread_join() on it. See pthread_exit() for more information.

Cancellation points are the points where the thread checks for pending cancellation requests and performs them. The POSIX threads functions pthread_join(), pthread_cond_wait(), pthread_-cond_timedwait(), pthread_testcancel(), sem_wait(), sem_timedwait(), sigwait(), sigwaitinfo() and sigtimedwait() are cancellation points.

See also

`Specification.`

5.4.2 Function Documentation

5.4.2.1 int pthread_cancel (pthread_t *thread*)

Cancel a thread.

This service sends a cancellation request to the thread *thread* and returns immediately. Depending on the target thread cancelability state (see pthread_setcancelstate()) and type (see pthread_-setcanceltype()), its termination is either immediate, deferred or ignored.

When the cancellation request is handled and before the thread is terminated, the cancellation cleanup handlers (registered with the pthread_cleanup_push() service) are called, then the thread-specific data destructor functions (registered with pthread_key_create()).

Returns

0 on success;
an error number if:

- ESRCH, the thread *thread* was not found.

See also

`Specification.`

References xnpod_schedule(), and xnpod_unblock_thread().

5.4.2.2 void pthread_cleanup_pop (int *execute*)

Unregister the last registered cleanup handler.

If the calling thread is a Xenomai POSIX skin thread (i.e. created with pthread_create()), this service unregisters the last routine which was registered with pthread_cleanup_push() and call it if *execute* is not null.

If the caller context is invalid (not a Xenomai POSIX skin thread), this service has no effect.

This service may be called at any place, but for maximal portability, should only called in the same lexical scope as the matching call to pthread_cleanup_push().

Parameters

execute if non zero, the last registered cleanup handler should be executed before it is unregistered.

Valid contexts:

- Xenomai POSIX skin kernel-space thread,
- Xenomai POSIX skin user-space thread (switches to primary mode).

See also

`Specification.`

5.4.2.3 void pthread_cleanup_push (cleanup_routine_t * *routine*, void * *arg*)

Register a cleanup handler to be executed at the time of cancellation.

This service registers the given *routine* to be executed a the time of cancellation of the calling thread, if this thread is a Xenomai POSIX skin thread (i.e. created with the pthread_create() service). If the caller context is invalid (not a Xenomai POSIX skin thread), this service has no effect.

If allocation from the system heap fails (because the system heap size is to small), this service fails silently.

The routines registered with this service get called in LIFO order when the calling thread calls pthread_exit() or is canceled, or when it calls the pthread_cleanup_pop() service with a non null argument.

Parameters

routine the cleanup routine to be registered;

arg the argument associated with this routine.

Valid contexts:

- Xenomai POSIX skin kernel-space thread,
- Xenomai POSIX skin user-space thread (switches to primary mode).

See also

Specification.

5.4.2.4 int pthread_setcancelstate (int *state*, int * *oldstate_ptr*)

Set cancelability state of the current thread.

This service atomically set the cancelability state of the calling thread and returns its previous value at the address *oldstate_ptr*, if the calling thread is a Xenomai POSIX skin thread (i.e. created with the pthread_create service).

The cancelability state of a POSIX thread may be:

- PTHREAD_CANCEL_ENABLE, meaning that cancellation requests will be handled if received;
- PTHREAD_CANCEL_DISABLE, meaning that cancellation requests will not be handled if received.

Parameters

state new cancelability state of the calling thread;

oldstate_ptr address where the old cancelability state will be stored on success.

Returns

0 on success;
an error number if:

- EINVAL, *state* is not a valid cancelability state;
- EPERM, the caller context is invalid.

Valid contexts:

- Xenomai POSIX skin kernel-space thread,
- Xenomai POSIX skin user-space thread (switches to primary mode).

See also

Specification.

5.4.2.5 int pthread_setcanceltype (int *type*, int * *oldtype_ptr*)

Set cancelability type of the current thread.

This service atomically sets the cancelability type of the calling thread, and return its previous value at the address *oldtype_ptr*, if this thread is a Xenomai POSIX skin thread (i.e. was created with the pthread_create() service).

The cancelability type of a POSIX thread may be:

- PTHREAD_CANCEL_DEFERRED, meaning that cancellation requests are only handled in services which are cancellation points;
- PTHREAD_CANCEL_ASYNCHRONOUS, meaning that cancellation requests are handled as soon as they are sent.

Parameters

type new cancelability type of the calling thread;

oldtype_ptr address where the old cancelability type will be stored on success.

Returns

0 on success;
an error number if:

- EINVAL, *type* is not a valid cancelability type;
- EPERM, the caller context is invalid.

Valid contexts:

- Xenomai POSIX skin kernel-space thread,
- Xenomai POSIX skin user-space thread (switches to primary mode).

See also

`Specification.`

5.5 Clocks and timers services.

Clocks and timers services.

Functions

- static int do_clock_host_realtime (struct timespec *tp)

  Read the host-synchronised realtime clock.

- static int timer_create (clockid_t clockid, const struct sigevent *__restrict__ evp, timer_t *__restrict__ timerid)

  Create a timer object.

- static int timer_settime (timer_t timerid, int flags, const struct itimerspec *__restrict__ value, struct itimerspec *__restrict__ ovalue)

  Start or stop a timer.

- static int timer_gettime (timer_t timerid, struct itimerspec *value)

  Get timer next expiration date and reload value.

5.5.1 Detailed Description

Clocks and timers services. Xenomai POSIX skin supports two clocks:

CLOCK_REALTIME maps to the nucleus system clock, keeping time as the amount of time since the Epoch, with a resolution of one nanosecond.

CLOCK_MONOTONIC maps to an architecture-dependent high resolution counter, so is suitable for measuring short time intervals. However, when used for sleeping (with clock_nanosleep()), the CLOCK_MONOTONIC clock has a resolution of one nanosecond, like the CLOCK_REALTIME clock.

CLOCK_MONOTONIC_RAW is Linux-specific, and provides monotonic time values from a hardware timer which is not adjusted by NTP. This is strictly equivalent to CLOCK_MONOTONIC with Xenomai, which is not NTP adjusted either.

Timer objects may be created with the timer_create() service using either of the two clocks. The resolution of these timers is one nanosecond, as is the case for clock_nanosleep().

See also

`Specification.`

5.5.2 Function Documentation

5.5.2.1 static int do_clock_host_realtime (struct timespec * *tp*) `[static]`

Read the host-synchronised realtime clock.

Obtain the current time with NTP corrections from the Linux domain

Parameters

tp pointer to a struct timespec

Return values

0 on success;

-1 if no suitable NTP-corrected clocksource is availabel

See also

`Specification.`

5.5.2.2 static int timer_create (clockid_t *clockid*, const struct sigevent *__restrict__ *evp*, timer_t *__restrict__ *timerid*) `[inline, static]`

Create a timer object.

This service creates a time object using the clock *clockid*.

If *evp* is not *NULL*, it describes the notification mechanism used on timer expiration. Only notification via signal delivery is supported (member *sigev_notify* of *evp* set to *SIGEV_SIGNAL*). The signal will be sent to the thread starting the timer with the timer_settime() service. If *evp* is *NULL*, the SIGALRM signal will be used.

Note that signals sent to user-space threads will cause them to switch to secondary mode.

If this service succeeds, an identifier for the created timer is returned at the address *timerid*. The timer is unarmed until started with the timer_settime() service.

Parameters

clockid clock used as a timing base;

evp description of the asynchronous notification to occur when the timer expires;

timerid address where the identifier of the created timer will be stored on success.

Return values

0 on success;

-1 with *errno* set if:

- EINVAL, the clock *clockid* is invalid;
- EINVAL, the member *sigev_notify* of the **sigevent** structure at the address *evp* is not SIGEV_SIGNAL;
- EINVAL, the member *sigev_signo* of the **sigevent** structure is an invalid signal number;
- EAGAIN, the maximum number of timers was exceeded, recompile with a larger value.

See also

`Specification.`

References sem_getvalue(), and xntimer_init().

5.5.2.3 static int timer_gettime (timer_t *timerid*, struct itimerspec * *value*) [inline, static]

Get timer next expiration date and reload value.

This service stores, at the address *value*, the expiration date (member *it_value*) and reload value (member *it_interval*) of the timer *timerid*. The values are returned as time intervals, and as multiples of the system clock tick duration (see note in section Clocks and timers services for details on the duration of the system clock tick). If the timer was not started, the returned members *it_value* and *it_interval* of *value* are zero.

Parameters

timerid timer identifier;

value address where the timer expiration date and reload value are stored on success.

Return values

0 on success;

-1 with *errno* set if:

- EINVAL, *timerid* is invalid;
- EPERM, the timer *timerid* does not belong to the current process.

See also

`Specification.`

5.5.2.4 static int timer_settime (timer_t *timerid*, int *flags*, const struct itimerspec *__restrict__ *value*, struct itimerspec *__restrict__ *ovalue*) [inline, static]

Start or stop a timer.

This service sets a timer expiration date and reload value of the timer *timerid*. If *ovalue* is not *NULL*, the current expiration date and reload value are stored at the address *ovalue* as with timer_gettime().

If the member *it_value* of the **itimerspec** structure at *value* is zero, the timer is stopped, otherwise the timer is started. If the member *it_interval* is not zero, the timer is periodic. The current thread must be a POSIX skin thread (created with pthread_create()) and will be notified via signal of timer expirations. Note that these notifications will cause user-space threads to switch to secondary mode.

When starting the timer, if *flags* is TIMER_ABSTIME, the expiration value is interpreted as an absolute date of the clock passed to the timer_create() service. Otherwise, the expiration value is interpreted as a time interval.

Expiration date and reload value are rounded to an integer count of nanoseconds.

Parameters

timerid identifier of the timer to be started or stopped;

flags one of 0 or TIMER_ABSTIME;

value address where the specified timer expiration date and reload value are read;

ovalue address where the specified timer previous expiration date and reload value are stored if not *NULL*.

Return values

0 on success;

-1 with *errno* set if:

- EPERM, the caller context is invalid;
- EINVAL, the specified timer identifier, expiration date or reload value is invalid;
- EPERM, the timer *timerid* does not belong to the current process.

Valid contexts:

- Xenomai kernel-space POSIX skin thread,
- kernel-space thread cancellation cleanup routine,
- Xenomai POSIX skin user-space thread (switches to primary mode),
- user-space thread cancellation cleanup routine.

See also

`Specification.`

References xntimer_start(), and xntimer_stop().

5.6 Condition variables services.

Condition variables services.

Functions

- static int pthread_cond_init (struct __shadow_cond *cnd, const pthread_condattr_t *attr)
  Initialize a condition variable.
- static int pthread_cond_destroy (struct __shadow_cond *cnd)
  Destroy a condition variable.
- static int pthread_condattr_init (pthread_condattr_t *attr)
  Initialize a condition variable attributes object.
- static int pthread_condattr_destroy (pthread_condattr_t *attr)
  Destroy a condition variable attributes object.
- static int pthread_condattr_getclock (const pthread_condattr_t *attr, clockid_t *clk_id)
  Get the clock selection attribute from a condition variable attributes object.
- static int pthread_condattr_setclock (pthread_condattr_t *attr, clockid_t clk_id)
  Set the clock selection attribute of a condition variable attributes object.
- static int pthread_condattr_getpshared (const pthread_condattr_t *attr, int *pshared)
  Get the process-shared attribute from a condition variable attributes object.
- static int pthread_condattr_setpshared (pthread_condattr_t *attr, int pshared)
  Set the process-shared attribute of a condition variable attributes object.

5.6.1 Detailed Description

Condition variables services. A condition variable is a synchronization object that allows threads to suspend execution until some predicate on shared data is satisfied. The basic operations on conditions are: signal the condition (when the predicate becomes true), and wait for the condition, suspending the thread execution until another thread signals the condition.

A condition variable must always be associated with a mutex, to avoid the race condition where a thread prepares to wait on a condition variable and another thread signals the condition just before the first thread actually waits on it.

Before it can be used, a condition variable has to be initialized with pthread_cond_init(). An attribute object, which reference may be passed to this service, allows to select the features of the created condition variable, namely the *clock* used by the pthread_cond_timedwait() service (*CLOCK_REALTIME* is used by default), and whether it may be shared between several processes (it may not be shared by default, see pthread_condattr_setpshared()).

Note that only pthread_cond_init() may be used to initialize a condition variable, using the static initializer *PTHREAD_COND_INITIALIZER* is not supported.

5.6.2 Function Documentation

5.6.2.1 static int pthread_cond_destroy (struct __shadow_cond * *cnd*) `[inline, static]`

Destroy a condition variable.

This service destroys the condition variable *cnd*, if no thread is currently blocked on it. The condition variable becomes invalid for all condition variable services (they all return the EINVAL error) except pthread_cond_init().

Parameters

cnd the condition variable to be destroyed.

Returns

0 on succes,
an error number if:

- EINVAL, the condition variable *cnd* is invalid;
- EPERM, the condition variable is not process-shared and does not belong to the current process;
- EBUSY, some thread is currently using the condition variable.

See also

Specification.

5.6.2.2 static int pthread_cond_init (struct __shadow_cond * *cnd*, const pthread_condattr_t * *attr*) [inline, static]

Initialize a condition variable.

This service initializes the condition variable *cnd*, using the condition variable attributes object *attr*. If *attr* is *NULL* or this service is called from user-space, default attributes are used (see pthread_condattr_init()).

Parameters

cnd the condition variable to be initialized;

attr the condition variable attributes object.

Returns

0 on succes,
an error number if:

- EINVAL, the condition variable attributes object *attr* is invalid or uninitialized;
- EBUSY, the condition variable *cnd* was already initialized;
- ENOMEM, insufficient memory exists in the system heap to initialize the condition variable, increase CONFIG_XENO_OPT_SYS_HEAPSZ.

See also

Specification.

References xnheap_alloc(), xnheap_free(), and xnsynch_init().

5.6.2.3 static int pthread_condattr_destroy (pthread_condattr_t * *attr*) [inline, static]

Destroy a condition variable attributes object.

This service destroys the condition variable attributes object *attr*. The object becomes invalid for all condition variable services (they all return EINVAL) except pthread_condattr_init().

Parameters

attr the initialized mutex attributes object to be destroyed.

Returns

0 on success;
an error number if:

- EINVAL, the mutex attributes object *attr* is invalid.

See also

`Specification.`

5.6.2.4 static int pthread_condattr_getclock (const pthread_condattr_t * *attr*, clockid_t * *clk_id*) [inline, static]

Get the clock selection attribute from a condition variable attributes object.

This service stores, at the address *clk_id*, the value of the *clock* attribute in the condition variable attributes object *attr*.

See pthread_cond_timedwait() documentation for a description of the effect of this attribute on a condition variable. The clock ID returned is *CLOCK_REALTIME*, *CLOCK_MONOTONIC* or *CLOCK_MONOTONIC_RAW*.

Parameters

attr an initialized condition variable attributes object,

clk_id address where the *clock* attribute value will be stored on success.

Returns

0 on success,
an error number if:

- EINVAL, the attribute object *attr* is invalid.

See also

`Specification.`

5.6.2.5 static int pthread_condattr_getpshared (const pthread_condattr_t * *attr*, int * *pshared*) [inline, static]

Get the process-shared attribute from a condition variable attributes object.

This service stores, at the address *pshared*, the value of the *pshared* attribute in the condition variable attributes object *attr*.

The *pshared* attribute may only be one of *PTHREAD_PROCESS_PRIVATE* or *PTHREAD_-PROCESS_SHARED*. See pthread_condattr_setpshared() for the meaning of these two constants.

Parameters

attr an initialized condition variable attributes object.

pshared address where the value of the *pshared* attribute will be stored on success.

Returns

0 on success,
an error number if:

- EINVAL, the *pshared* address is invalid;
- EINVAL, the condition variable attributes object *attr* is invalid.

See also

`Specification.`

5.6.2.6 static int pthread_condattr_init (pthread_condattr_t * *attr*) `[inline, static]`

Initialize a condition variable attributes object.

This services initializes the condition variable attributes object *attr* with default values for all attributes. Default value are:

- for the *clock* attribute, *CLOCK_REALTIME*;
- for the *pshared* attribute *PTHREAD_PROCESS_PRIVATE*.

If this service is called specifying a condition variable attributes object that was already initialized, the attributes object is reinitialized.

Parameters

attr the condition variable attributes object to be initialized.

Returns

0 on success;
an error number if:

- ENOMEM, the condition variable attribute object pointer *attr* is *NULL*.

See also

`Specification.`

5.6.2.7 static int pthread_condattr_setclock (pthread_condattr_t * *attr*, clockid_t *clk_id*) `[inline, static]`

Set the clock selection attribute of a condition variable attributes object.

This service set the *clock* attribute of the condition variable attributes object *attr*.

See pthread_cond_timedwait() documentation for a description of the effect of this attribute on a condition variable.

Parameters

attr an initialized condition variable attributes object,

clk_id value of the *clock* attribute, may be *CLOCK_REALTIME*, *CLOCK_MONOTONIC* or *CLOCK_MONOTONIC_RAW*.

Returns

0 on success,
an error number if:

- EINVAL, the condition variable attributes object *attr* is invalid;
- EINVAL, the value of *clk_id* is invalid for the *clock* attribute.

See also

`Specification.`

5.6.2.8 static int pthread_condattr_setpshared (pthread_condattr_t * *attr*, int *pshared*) `[inline, static]`

Set the process-shared attribute of a condition variable attributes object.

This service set the *pshared* attribute of the condition variable attributes object *attr*.

Parameters

attr an initialized condition variable attributes object.

pshared value of the *pshared* attribute, may be one of:

- PTHREAD_PROCESS_PRIVATE, meaning that a condition variable created with the attributes object *attr* will only be accessible by threads within the same process as the thread that initialized the condition variable;
- PTHREAD_PROCESS_SHARED, meaning that a condition variable created with the attributes object *attr* will be accessible by any thread that has access to the memory where the condition variable is allocated.

Returns

0 on success,
an error status if:

- EINVAL, the condition variable attributes object *attr* is invalid;
- EINVAL, the value of *pshared* is invalid.

See also

`Specification.`

5.7 POSIX skin.

Xenomai POSIX skin is an implementation of a small subset of the Single Unix specification over Xenomai generic RTOS core.

Xenomai POSIX skin is an implementation of a small subset of the Single Unix specification over Xenomai generic RTOS core. The following table gives equivalence between Alchemy services and Cobalt services.

Alchemy services	Cobalt services
alchemy_alarm	Clocks and timers services.
alchemy_cond	Condition variables services.
alchemy_event	no direct equivalence, see Condition variables services.
alchemy_heap	no direct equivalence
alchemy_mutex	Mutex services.
alchemy_pipe	no direct equivalence, see Message queues services.
alchemy_queue	Message queues services.
alchemy_sem	Semaphores services.
alchemy_task	Threads management services.
alchemy_timer	Clocks and timers services.

5.8 Message queues services.

Message queues services.

Functions

- static mqd_t mq_open (const char *name, int oflags,...)

  Open a message queue.

- static int mq_close (mqd_t fd)

  Close a message queue.

- static int mq_unlink (const char *name)

  Unlink a message queue.

- static int mq_getattr (mqd_t fd, struct mq_attr *attr)

  Get the attributes object of a message queue.

- static int mq_setattr (mqd_t fd, const struct mq_attr *__restrict__ attr, struct mq_attr *__-restrict__ oattr)

  Set flags of a message queue.

5.8.1 Detailed Description

Message queues services. A message queue allow exchanging data between real-time threads. For a POSIX message queue, maximum message length and maximum number of messages are fixed when it is created with mq_open().

5.8.2 Function Documentation

5.8.2.1 static int mq_close (mqd_t *fd*) [inline, static]

Close a message queue.

This service closes the message queue descriptor *fd*. The message queue is destroyed only when all open descriptors are closed, and when unlinked with a call to the mq_unlink() service.

Parameters

fd message queue descriptor.

Return values

0 on success;

-1 with *errno* set if:

- EBADF, *fd* is an invalid message queue descriptor;
- EPERM, the caller context is invalid.

Valid contexts:

- kernel module initialization or cleanup routine;
- kernel-space cancellation cleanup routine;
- user-space thread (Xenomai threads switch to secondary mode);
- user-space cancellation cleanup routine.

See also

`Specification.`

5.8.2.2 static int mq_getattr (mqd_t *fd*, struct mq_attr * *attr*) [inline, static]

Get the attributes object of a message queue.

This service stores, at the address *attr*, the attributes of the messages queue descriptor *fd*.

The following attributes are set:

- *mq_flags*, flags of the message queue descriptor *fd*;
- *mq_maxmsg*, maximum number of messages in the message queue;
- *mq_msgsize*, maximum message size;
- *mq_curmsgs*, number of messages currently in the queue.

Parameters

fd message queue descriptor;

attr address where the message queue attributes will be stored on success.

Return values

0 on success;

-1 with *errno* set if:

- EBADF, *fd* is not a valid descriptor.

See also

`Specification.`

5.8.2.3 static mqd_t mq_open (const char * *name*, int *oflags*, ...) [static]

Open a message queue.

This service establishes a connection between the message queue named *name* and the calling context (kernel-space as a whole, or user-space process).

One of the following values should be set in *oflags:*

- O_RDONLY, meaning that the returned queue descriptor may only be used for receiving messages;
- O_WRONLY, meaning that the returned queue descriptor may only be used for sending messages;
- O_RDWR, meaning that the returned queue descriptor may be used for both sending and receiving messages.

If no message queue named *name* exists, and *oflags* has the *O_CREAT* bit set, the message queue is created by this function, taking two more arguments:

- a *mode* argument, of type **mode_t**, currently ignored;
- an *attr* argument, pointer to an **mq_attr** structure, specifying the attributes of the new message queue.

If *oflags* has the two bits *O_CREAT* and *O_EXCL* set and the message queue alread exists, this service fails.

If the O_NONBLOCK bit is set in *oflags*, the mq_send(), mq_receive(), mq_timedsend() and mq_-timedreceive() services return *-1* with *errno* set to EAGAIN instead of blocking their caller.

The following arguments of the **mq_attr** structure at the address *attr* are used when creating a message queue:

- *mq_maxmsg* is the maximum number of messages in the queue (128 by default);
- *mq_msgsize* is the maximum size of each message (128 by default).

name may be any arbitrary string, in which slashes have no particular meaning. However, for portability, using a name which starts with a slash and contains no other slash is recommended.

Parameters

name name of the message queue to open;

oflags flags.

Returns

a message queue descriptor on success;
-1 with *errno* set if:

- ENAMETOOLONG, the length of the *name* argument exceeds 64 characters;
- EEXIST, the bits *O_CREAT* and *O_EXCL* were set in *oflags* and the message queue already exists;
- ENOENT, the bit *O_CREAT* is not set in *oflags* and the message queue does not exist;

- ENOSPC, allocation of system memory failed, or insufficient memory exists in the system heap to create the queue, try increasing CONFIG_XENO_OPT_SYS_HEAPSZ;
- EPERM, attempting to create a message queue from an invalid context;
- EINVAL, the *attr* argument is invalid;
- EMFILE, too many descriptors are currently open.

Valid contexts:

When creating a message queue, only the following contexts are valid:

- kernel module initialization or cleanup routine;
- user-space thread (Xenomai threads switch to secondary mode).

See also

Specification.

5.8.2.4 static int mq_setattr (mqd_t *fd*, const struct mq_attr *__restrict__ *attr*, struct mq_attr *__restrict__ *oattr*) [inline, static]

Set flags of a message queue.

This service sets the flags of the *fd* descriptor to the value of the member *mq_flags* of the **mq_attr** structure pointed to by *attr*.

The previous value of the message queue attributes are stored at the address *oattr* if it is not *NULL*.

Only setting or clearing the O_NONBLOCK flag has an effect.

Parameters

fd message queue descriptor;

attr pointer to new attributes (only *mq_flags* is used);

oattr if not *NULL*, address where previous message queue attributes will be stored on success.

Return values

0 on success;

-1 with *errno* set if:

- EBADF, *fd* is not a valid message queue descriptor.

See also

Specification.

5.8.2.5 static int mq_unlink (const char * *name*) [inline, static]

Unlink a message queue.

This service unlinks the message queue named *name*. The message queue is not destroyed until all queue descriptors obtained with the mq_open() service are closed with the mq_close() service. However, after a call to this service, the unlinked queue may no longer be reached with the mq_open() service.

Parameters

name name of the message queue to be unlinked.

Return values

0 on success;

-1 with *errno* set if:

- EPERM, the caller context is invalid;
- ENAMETOOLONG, the length of the *name* argument exceeds 64 characters;
- ENOENT, the message queue does not exist.

Valid contexts:

- kernel module initialization or cleanup routine;
- kernel-space cancellation cleanup routine;
- user-space thread (Xenomai threads switch to secondary mode);
- user-space cancellation cleanup routine.

See also

Specification.

5.9 Mutex services.

Mutex services.

Functions

- static int pthread_mutexattr_init (pthread_mutexattr_t *attr)

  Initialize a mutex attributes object.

- static int pthread_mutexattr_destroy (pthread_mutexattr_t *attr)

  Destroy a mutex attributes object.

- static int pthread_mutexattr_gettype (const pthread_mutexattr_t *attr, int *type)

  Get the mutex type attribute from a mutex attributes object.

- static int pthread_mutexattr_settype (pthread_mutexattr_t *attr, int type)

  Set the mutex type attribute of a mutex attributes object.

- static int pthread_mutexattr_getprotocol (const pthread_mutexattr_t *attr, int *proto)

  Get the protocol attribute from a mutex attributes object.

- static int pthread_mutexattr_setprotocol (pthread_mutexattr_t *attr, int proto)

  Set the protocol attribute of a mutex attributes object.

- static int pthread_mutexattr_getpshared (const pthread_mutexattr_t *attr, int *pshared)

  Get the process-shared attribute of a mutex attributes object.

- static int pthread_mutexattr_setpshared (pthread_mutexattr_t *attr, int pshared)
  Set the process-shared attribute of a mutex attributes object.

5.9.1 Detailed Description

Mutex services. A mutex is a MUTual EXclusion device, and is useful for protecting shared data structures from concurrent modifications, and implementing critical sections and monitors.

A mutex has two possible states: unlocked (not owned by any thread), and locked (owned by one thread). A mutex can never be owned by two different threads simultaneously. A thread attempting to lock a mutex that is already locked by another thread is suspended until the owning thread unlocks the mutex first.

Before it can be used, a mutex has to be initialized with pthread_mutex_init(). An attribute object, which reference may be passed to this service, allows to select the features of the created mutex, namely its *type* (see pthread_mutexattr_settype()), the priority *protocol* it uses (see pthread_mutexattr_setprotocol()) and whether it may be shared between several processes (see pthread_mutexattr_setpshared()).

By default, Xenomai POSIX skin mutexes are of the normal type, use no priority protocol and may not be shared between several processes.

Note that only pthread_mutex_init() may be used to initialize a mutex, using the static initializer *PTHREAD_MUTEX_INITIALIZER* is not supported.

5.9.2 Function Documentation

5.9.2.1 static int pthread_mutexattr_destroy (pthread_mutexattr_t * *attr*) `[inline, static]`

Destroy a mutex attributes object.

This service destroys the mutex attributes object *attr*. The object becomes invalid for all mutex services (they all return EINVAL) except pthread_mutexattr_init().

Parameters

attr the initialized mutex attributes object to be destroyed.

Returns

0 on success;
an error number if:

- EINVAL, the mutex attributes object *attr* is invalid.

See also

`Specification.`

5.9.2.2 static int pthread_mutexattr_getprotocol (const pthread_mutexattr_t * *attr*, int * *proto*) `[inline, static]`

Get the protocol attribute from a mutex attributes object.

This service stores, at the address *proto*, the value of the *protocol* attribute in the mutex attributes object *attr*.

The *protcol* attribute may only be one of *PTHREAD_PRIO_NONE* or *PTHREAD_PRIO_INHERIT*. See pthread_mutexattr_setprotocol() for the meaning of these two constants.

Parameters

attr an initialized mutex attributes object;

proto address where the value of the *protocol* attribute will be stored on success.

Returns

0 on success,
an error number if:

- EINVAL, the *proto* address is invalid;
- EINVAL, the mutex attributes object *attr* is invalid.

See also

`Specification.`

5.9.2.3 static int pthread_mutexattr_getpshared (const pthread_mutexattr_t * *attr*, int * *pshared*) [inline, static]

Get the process-shared attribute of a mutex attributes object.

This service stores, at the address *pshared*, the value of the *pshared* attribute in the mutex attributes object *attr*.

The *pashared* attribute may only be one of *PTHREAD_PROCESS_PRIVATE* or *PTHREAD_-PROCESS_SHARED*. See pthread_mutexattr_setpshared() for the meaning of these two constants.

Parameters

attr an initialized mutex attributes object;

pshared address where the value of the *pshared* attribute will be stored on success.

Returns

0 on success;
an error number if:

- EINVAL, the *pshared* address is invalid;
- EINVAL, the mutex attributes object *attr* is invalid.

See also

`Specification.`

5.9.2.4 static int pthread_mutexattr_gettype (const pthread_mutexattr_t * *attr*, int * *type*) [inline, static]

Get the mutex type attribute from a mutex attributes object.

This service stores, at the address *type*, the value of the *type* attribute in the mutex attributes object *attr*.

See pthread_mutex_lock() and pthread_mutex_unlock() documentations for a description of the values of the *type* attribute and their effect on a mutex.

Parameters

attr an initialized mutex attributes object,

type address where the *type* attribute value will be stored on success.

Returns

0 on sucess,
an error number if:

- EINVAL, the *type* address is invalid;
- EINVAL, the mutex attributes object *attr* is invalid.

See also

Specification.

5.9.2.5 static int pthread_mutexattr_init (pthread_mutexattr_t * *attr*) [inline, static]

Initialize a mutex attributes object.

This services initializes the mutex attributes object *attr* with default values for all attributes. Default value are :

- for the *type* attribute, *PTHREAD_MUTEX_NORMAL*;
- for the *protocol* attribute, *PTHREAD_PRIO_NONE*;
- for the *pshared* attribute, *PTHREAD_PROCESS_PRIVATE*.

If this service is called specifying a mutex attributes object that was already initialized, the attributes object is reinitialized.

Parameters

attr the mutex attributes object to be initialized.

Returns

0 on success;
an error number if:

- ENOMEM, the mutex attributes object pointer *attr* is *NULL*.

See also

Specification.

5.9.2.6 static int pthread_mutexattr_setprotocol (pthread_mutexattr_t * *attr*, int *proto*) [inline, static]

Set the protocol attribute of a mutex attributes object.

This service set the *type* attribute of the mutex attributes object *attr*.

Parameters

attr an initialized mutex attributes object,

proto value of the *protocol* attribute, may be one of:

- PTHREAD_PRIO_NONE, meaning that a mutex created with the attributes object *attr* will not follow any priority protocol;
- PTHREAD_PRIO_INHERIT, meaning that a mutex created with the attributes object *attr*, will follow the priority inheritance protocol.

The value PTHREAD_PRIO_PROTECT (priority ceiling protocol) is unsupported.

Returns

0 on success,
an error number if:

- EINVAL, the mutex attributes object *attr* is invalid;
- ENOTSUP, the value of *proto* is unsupported;
- EINVAL, the value of *proto* is invalid.

See also

`Specification.`

5.9.2.7 static int pthread_mutexattr_setpshared (pthread_mutexattr_t * *attr*, int *pshared*) [inline, static]

Set the process-shared attribute of a mutex attributes object.

This service set the *pshared* attribute of the mutex attributes object *attr*.

Parameters

attr an initialized mutex attributes object.

pshared value of the *pshared* attribute, may be one of:

- PTHREAD_PROCESS_PRIVATE, meaning that a mutex created with the attributes object *attr* will only be accessible by threads within the same process as the thread that initialized the mutex;
- PTHREAD_PROCESS_SHARED, meaning that a mutex created with the attributes object *attr* will be accessible by any thread that has access to the memory where the mutex is allocated.

Returns

0 on success,
an error status if:

- EINVAL, the mutex attributes object *attr* is invalid;
- EINVAL, the value of *pshared* is invalid.

See also

Specification.

5.9.2.8 static int pthread_mutexattr_settype (pthread_mutexattr_t * *attr*, int *type*) [inline, static]

Set the mutex type attribute of a mutex attributes object.

This service set the *type* attribute of the mutex attributes object *attr*.

See pthread_mutex_lock() and pthread_mutex_unlock() documentations for a description of the values of the *type* attribute and their effect on a mutex.

The *PTHREAD_MUTEX_DEFAULT* default *type* is the same as *PTHREAD_MUTEX_NORMAL*. Note that using a Xenomai POSIX skin recursive mutex with a Xenomai POSIX skin condition variable is safe (see pthread_cond_wait() documentation).

Parameters

attr an initialized mutex attributes object,

type value of the *type* attribute.

Returns

0 on success,
an error number if:

- EINVAL, the mutex attributes object *attr* is invalid;
- EINVAL, the value of *type* is invalid for the *type* attribute.

See also

Specification.

5.10 Buffer descriptors.

Files

- file bufd.h
- file bufd.c

Functions

- static void xnbufd_map_uread (struct xnbufd *bufd, const void __user *ptr, size_t len)
  Initialize a buffer descriptor for reading from user memory.
- static void xnbufd_map_uwrite (struct xnbufd *bufd, void __user *ptr, size_t len)
  Initialize a buffer descriptor for writing to user memory.

- ssize_t xnbufd_unmap_uread (struct xnbufd *bufd)

  Finalize a buffer descriptor obtained from xnbufd_map_uread().

- ssize_t xnbufd_unmap_uwrite (struct xnbufd *bufd)

  Finalize a buffer descriptor obtained from xnbufd_map_uwrite().

- static void xnbufd_map_kread (struct xnbufd *bufd, const void *ptr, size_t len)

  Initialize a buffer descriptor for reading from kernel memory.

- static void xnbufd_map_kwrite (struct xnbufd *bufd, void *ptr, size_t len)

  Initialize a buffer descriptor for writing to kernel memory.

- ssize_t xnbufd_unmap_kread (struct xnbufd *bufd)

  Finalize a buffer descriptor obtained from xnbufd_map_kread().

- ssize_t xnbufd_unmap_kwrite (struct xnbufd *bufd)

  Finalize a buffer descriptor obtained from xnbufd_map_kwrite().

- ssize_t xnbufd_copy_to_kmem (void *ptr, struct xnbufd *bufd, size_t len)

  Copy memory covered by a buffer descriptor to kernel memory.

- ssize_t xnbufd_copy_from_kmem (struct xnbufd *bufd, void *from, size_t len)

  Copy kernel memory to the area covered by a buffer descriptor.

- void xnbufd_invalidate (struct xnbufd *bufd)

  Invalidate a buffer descriptor.

- static void xnbufd_reset (struct xnbufd *bufd)

  Reset a buffer descriptor.

5.10.1 Detailed Description

A buffer descriptor is a simple abstraction dealing with copy operations to/from memory buffers which may belong to different address spaces.

To this end, the buffer descriptor library provides a small set of copy routines which are aware of address space restrictions when moving data, and a generic container type which can hold a reference to - or cover - a particular memory area, either present in kernel space, or in any of the existing user memory contexts.

The goal of the buffer descriptor abstraction is to hide address space specifics from Xenomai services dealing with memory areas, allowing them to operate on multiple address spaces seamlessly.

The common usage patterns are as follows:

- Implementing a Xenomai syscall returning a bulk of data to the caller, which may have to be copied back to either kernel or user space:

```
[Syscall implementation]
ssize_t rt_bulk_read_inner(struct xnbufd *bufd)
{
    ssize_t ret;
    size_t len;
    void *bulk;

    bulk = get_next_readable_bulk(&len);
    ret = xnbufd_copy_from_kmem(bufd, bulk, min(bufd->b_len, len));
    free_bulk(bulk);

    ret = this_may_fail();
    if (ret)
            xnbufd_invalidate(bufd);

    return ret;
}

[Kernel wrapper for in-kernel calls]
int rt_bulk_read(void *ptr, size_t len)
{
    struct xnbufd bufd;
    ssize_t ret;

    xnbufd_map_kwrite(&bufd, ptr, len);
    ret = rt_bulk_read_inner(&bufd);
    xnbufd_unmap_kwrite(&bufd);

    return ret;
}

[Userland trampoline for user syscalls]
int __rt_bulk_read(struct pt_regs *regs)
{
    struct xnbufd bufd;
    void __user *ptr;
    ssize_t ret;
    size_t len;

    ptr = (void __user *)__xn_reg_arg1(regs);
    len = __xn_reg_arg2(regs);

    xnbufd_map_uwrite(&bufd, ptr, len);
    ret = rt_bulk_read_inner(&bufd);
    xnbufd_unmap_uwrite(&bufd);

    return ret;
}
```

- Implementing a Xenomai syscall receiving a bulk of data from the caller, which may have to be read from either kernel or user space:

```
[Syscall implementation]
ssize_t rt_bulk_write_inner(struct xnbufd *bufd)
{
    void *bulk = get_free_bulk(bufd->b_len);
    return xnbufd_copy_to_kmem(bulk, bufd, bufd->b_len);
}

[Kernel wrapper for in-kernel calls]
int rt_bulk_write(const void *ptr, size_t len)
{
    struct xnbufd bufd;
    ssize_t ret;

    xnbufd_map_kread(&bufd, ptr, len);
```

```
    ret = rt_bulk_write_inner(&bufd);
    xnbufd_unmap_kread(&bufd);

    return ret;
}

[Userland trampoline for user syscalls]
int __rt_bulk_write(struct pt_regs *regs)
{
    struct xnbufd bufd;
    void __user *ptr;
    ssize_t ret;
    size_t len;

    ptr = (void __user *)__xn_reg_arg1(regs);
    len = __xn_reg_arg2(regs);

    xnbufd_map_uread(&bufd, ptr, len);
    ret = rt_bulk_write_inner(&bufd);
    xnbufd_unmap_uread(&bufd);

    return ret;
}
```

5.10.2 Function Documentation

5.10.2.1 ssize_t xnbufd_copy_from_kmem (struct xnbufd * *bufd*, void * *from*, size_t *len*)

Copy kernel memory to the area covered by a buffer descriptor.

This routine copies *len* bytes from the kernel memory starting at *from* to the area referred to by the buffer descriptor *bufd*. xnbufd_copy_from_kmem() tracks the write offset within the destination memory internally, so that it may be called several times in a loop, until the entire memory area is stored.

The destination address space is dealt with, according to the following rules:

- if *bufd* refers to a writable kernel area (i.e. see xnbufd_map_kwrite()), the copy is immediatly and fully performed with no restriction.
- if *bufd* refers to a writable user area (i.e. see xnbufd_map_uwrite()), the copy is performed only if that area lives in the currently active address space, and only if the caller may sleep Linux-wise to process any potential page fault which may arise while writing to that memory.
- if *bufd* refers to a user area which may not be immediately written to from the current context, the copy is postponed until xnbufd_unmap_uwrite() is invoked for *ubufd*, at which point the copy will take place. In such a case, the source memory is transferred to a carry over buffer allocated internally; this operation may lead to request dynamic memory from the nucleus heap if *len* is greater than 64 bytes.

Parameters

bufd The address of the buffer descriptor covering the user memory to copy data to.

from The start address of the kernel memory to copy from.

len The length of the kernel memory to copy to *bufd*.

Returns

The number of bytes written so far to the memory area covered by *ubufd*. Otherwise,

- -ENOMEM is returned when no memory is available from the nucleus heap to allocate the carry over buffer.

Environments:

This service can be called from:

- Kernel code (including from primary mode) except Xenomai kernel-based task and interrupt service routines.

Rescheduling: may switch the caller to secondary mode if a page fault occurs while writing to the user area. For that reason, xnbufd_copy_from_kmem() may only be called from a preemptible section (Linux-wise).

Note

Holding the nklock or running real-time interrupts disabled is invalid when calling this routine, and doing so would trigger a debug assertion.

5.10.2.2 ssize_t xnbufd_copy_to_kmem (void * *to*, struct xnbufd * *bufd*, size_t *len*)

Copy memory covered by a buffer descriptor to kernel memory.

This routine copies *len* bytes from the area referred to by the buffer descriptor *bufd* to the kernel memory area *to*. xnbufd_copy_to_kmem() tracks the read offset within the source memory internally, so that it may be called several times in a loop, until the entire memory area is loaded.

The source address space is dealt with, according to the following rules:

- if *bufd* refers to readable kernel area (i.e. see xnbufd_map_kread()), the copy is immediately and fully performed with no restriction.
- if *bufd* refers to a readable user area (i.e. see xnbufd_map_uread()), the copy is performed only if that area lives in the currently active address space, and only if the caller may sleep Linux-wise to process any potential page fault which may arise while reading from that memory.
- any attempt to read from *bufd* from a non-suitable context is considered as a bug, and will raise a panic assertion when the nucleus is compiled in debug mode.

Parameters

to The start address of the kernel memory to copy to.

bufd The address of the buffer descriptor covering the user memory to copy data from.

len The length of the user memory to copy from *bufd*.

Returns

The number of bytes read so far from the memory area covered by *ubufd*. Otherwise:

- -EINVAL is returned upon attempt to read from the user area from an invalid context. This error is only returned when the debug mode is disabled; otherwise a panic assertion is raised.

Environments:

This service can be called from:

- Kernel code (including from primary mode) except Xenomai kernel-based task and interrupt service routines.

Rescheduling: may switch the caller to secondary mode if a page fault occurs while reading from the user area. For that reason, xnbufd_copy_to_kmem() may only be called from a preemptible section (Linux-wise).

Note

Holding the nklock or running real-time interrupts disabled is invalid when calling this routine, and doing so would trigger a debug assertion.

5.10.2.3 void xnbufd_invalidate (struct xnbufd * *bufd*)

Invalidate a buffer descriptor.

The buffer descriptor is invalidated, making it unusable for further copy operations. If an outstanding carry over buffer was allocated by a previous call to xnbufd_copy_from_kmem(), it is immediately freed so that no data transfer will happen when the descriptor is finalized.

The only action that may subsequently be performed on an invalidated descriptor is calling the relevant unmapping routine for it. For that reason, xnbufd_invalidate() should be invoked on the error path when data may have been transferred to the carry over buffer.

Parameters

bufd The address of the buffer descriptor to invalidate.

Environments:

This service can be called from:

- Kernel code (including from primary mode)
- Kernel-based task
- Interrupt service routine

Rescheduling: never.

5.10.2.4 void xnbufd_map_kread (struct xnbufd * *bufd*, const void * *ptr*, size_t *len*) `[inline, static]`

Initialize a buffer descriptor for reading from kernel memory.

The new buffer descriptor may be used to copy data from kernel memory. This routine should be used in pair with xnbufd_unmap_kread().

Parameters

bufd The address of the buffer descriptor which will map a *len* bytes kernel memory area, starting from *ptr*.

ptr The start of the kernel buffer to map.

len The length of the kernel buffer starting at *ptr*.

Environments:

This service can be called from:

- Kernel code (including from primary mode)
- Kernel-based task
- Interrupt service routine

Rescheduling: never.

5.10.2.5 void xnbufd_map_kwrite (struct xnbufd * *bufd*, void * *ptr*, size_t *len*) [inline, static]

Initialize a buffer descriptor for writing to kernel memory.

The new buffer descriptor may be used to copy data to kernel memory. This routine should be used in pair with xnbufd_unmap_kwrite().

Parameters

bufd The address of the buffer descriptor which will map a *len* bytes kernel memory area, starting from *ptr*.

ptr The start of the kernel buffer to map.

len The length of the kernel buffer starting at *ptr*.

Environments:

This service can be called from:

- Kernel code (including from primary mode)
- Kernel-based task
- Interrupt service routine

Rescheduling: never.

5.10.2.6 void xnbufd_map_uread (struct xnbufd * *bufd*, const void __user * *ptr*, size_t *len*) [inline, static]

Initialize a buffer descriptor for reading from user memory.

The new buffer descriptor may be used to copy data from user memory. This routine should be used in pair with xnbufd_unmap_uread().

Parameters

bufd The address of the buffer descriptor which will map a *len* bytes user memory area, starting from *ptr*. *ptr* is never dereferenced directly, since it may refer to a buffer that lives in another address space.

ptr The start of the user buffer to map.

len The length of the user buffer starting at *ptr*.

Environments:

This service can be called from:

- Kernel code (including from primary mode) except Xenomai kernel-based task and interrupt service routines.

Rescheduling: never.

5.10.2.7 void xnbufd_map_uwrite (struct xnbufd * *bufd*, void __user * *ptr*, size_t *len*) `[inline, static]`

Initialize a buffer descriptor for writing to user memory.

The new buffer descriptor may be used to copy data to user memory. This routine should be used in pair with xnbufd_unmap_uwrite().

Parameters

bufd The address of the buffer descriptor which will map a *len* bytes user memory area, starting from *ptr*. *ptr* is never dereferenced directly, since it may refer to a buffer that lives in another address space.

ptr The start of the user buffer to map.

len The length of the user buffer starting at *ptr*.

Environments:

This service can be called from:

- Kernel code (including from primary mode) except Xenomai kernel-based task and interrupt service routines.

Rescheduling: never.

5.10.2.8 void xnbufd_reset (struct xnbufd * *bufd*) `[inline, static]`

Reset a buffer descriptor.

The buffer descriptor is reset, so that all data already copied is forgotten. Any carry over buffer allocated is kept, though.

Parameters

bufd The address of the buffer descriptor to reset.

Environments:

This service can be called from:

- Kernel code (including from primary mode)
- Kernel-based task
- Interrupt service routine

Rescheduling: never.

5.10.2.9 ssize_t xnbufd_unmap_kread (struct xnbufd * *bufd*)

Finalize a buffer descriptor obtained from xnbufd_map_kread().

This routine finalizes a buffer descriptor previously initialized by a call to xnbufd_map_kread(), to read data from a kernel area.

Parameters

bufd The address of the buffer descriptor to finalize.

Returns

The number of bytes read so far from the memory area covered by *ubufd*.

Environments:

This service can be called from:

- Kernel code (including from primary mode)
- Kernel-based task
- Interrupt service routine

Rescheduling: never.

5.10.2.10 ssize_t xnbufd_unmap_kwrite (struct xnbufd * *bufd*)

Finalize a buffer descriptor obtained from xnbufd_map_kwrite().

This routine finalizes a buffer descriptor previously initialized by a call to xnbufd_map_kwrite(), to write data to a kernel area.

Parameters

bufd The address of the buffer descriptor to finalize.

Returns

The number of bytes written so far to the memory area covered by *ubufd*.

Environments:

This service can be called from:

- Kernel code (including from primary mode)
- Kernel-based task
- Interrupt service routine

Rescheduling: never.

5.10.2.11 ssize_t xnbufd_unmap_uread (struct xnbufd * *bufd*)

Finalize a buffer descriptor obtained from xnbufd_map_uread().

This routine finalizes a buffer descriptor previously initialized by a call to xnbufd_map_uread(), to read data from a user area.

Parameters

bufd The address of the buffer descriptor to finalize.

Returns

The number of bytes read so far from the memory area covered by *ubufd*.

Environments:

This service can be called from:

- Kernel code (including from primary mode) except Xenomai kernel-based task and interrupt service routines.

Rescheduling: never.

Note

Holding the nklock or running real-time interrupts disabled is invalid when calling this routine, and doing so would trigger a debug assertion.

5.10.2.12 ssize_t xnbufd_unmap_uwrite (struct xnbufd * *bufd*)

Finalize a buffer descriptor obtained from xnbufd_map_uwrite().

This routine finalizes a buffer descriptor previously initialized by a call to xnbufd_map_uwrite(), to write data to a user area.

The main action taken is to write the contents of the kernel memory area passed to xnbufd_copy_from_kmem() whenever the copy operation was postponed at that time; the carry over buffer is eventually released as needed. If xnbufd_copy_from_kmem() was allowed to copy to the destination user memory at once, then xnbufd_unmap_uwrite() leads to a no-op.

Parameters

bufd The address of the buffer descriptor to finalize.

Returns

The number of bytes written so far to the memory area covered by *ubufd*.

Environments:

This service can be called from:

- Kernel code (including from primary mode) except Xenomai kernel-based task and interrupt service routines.

Rescheduling: never.

Note

Holding the nklock or running real-time interrupts disabled is invalid when calling this routine, and doing so would trigger a debug assertion.

5.11 System clock services.

Files

- file clock.h
- file clock.c

Functions

- void xnclock_adjust (xnsticks_t delta)

  Adjust the clock time for the system.

5.11.1 Function Documentation

5.11.1.1 void xnclock_adjust (xnsticks_t *delta*)

Adjust the clock time for the system.

Xenomai tracks the current time as a monotonously increasing count of ticks since the epoch. The epoch is initially the same as the underlying machine time.

This service changes the epoch for the system by applying the specified tick delta on the wallclock offset.

Parameters

delta The adjustment of the system time expressed in ticks.

Note

This routine must be entered nklock locked, interrupts off.

Environments:

This service can be called from:

- Any kernel context.

Rescheduling: never.

5.12 Debugging services.

Files

- file debug.c

  Debug services.

5.13 Dynamic memory allocation services.

Files

- file heap.c

  Dynamic memory allocation services.

Functions

- int xnheap_init (xnheap_t *heap, void *heapaddr, u_long heapsize, u_long pagesize)

  Initialize a memory heap.

- void xnheap_set_label (xnheap_t *heap, const char *label,...)

  Set the heap's label string.

- void xnheap_destroy (xnheap_t *heap, void(*flushfn)(xnheap_t *heap, void *extaddr, u_-long extsize, void *cookie), void *cookie)

  Destroys a memory heap.

- void * xnheap_alloc (xnheap_t *heap, u_long size)

  Allocate a memory block from a memory heap.

- int xnheap_test_and_free (xnheap_t *heap, void *block, int(*ckfn)(void *block))

  Test and release a memory block to a memory heap.

- int xnheap_free (xnheap_t *heap, void *block)

  Release a memory block to a memory heap.

- int xnheap_extend (xnheap_t *heap, void *extaddr, u_long extsize)

  Extend a memory heap.

- void xnheap_schedule_free (xnheap_t *heap, void *block, xnholder_t *link)

  Schedule a memory block for release.

5.13.1 Detailed Description

Dynamic memory allocation services.

The implementation of the memory allocator follows the algorithm described in a USENIX 1988 paper called "Design of a General Purpose Memory Allocator for the 4.3BSD Unix Kernel" by Marshall K. McKusick and Michael J. Karels. You can find it at various locations on the net, including http://docs.FreeBSD.org/44doc/papers/kernmalloc.pdf. A minor variation allows this implementation to have 'extendable' heaps when needed, with multiple memory extents providing autonomous page address spaces.

The data structures hierarchy is as follows:

```
HEAP {
     block_buckets[]
     extent_queue -------+
}                        |
 V
      EXTENT #1 {
     {static header}
     page_map[npages]
     page_array[npages][pagesize]
      } -+
 |
 |
 V
      EXTENT #n {
     {static header}
     page_map[npages]
     page_array[npages][pagesize]
      }
```

5.13.2 Function Documentation

5.13.2.1 void* xnheap_alloc (xnheap_t * *heap*, u_long *size*)

Allocate a memory block from a memory heap.

Allocates a contiguous region of memory from an active memory heap. Such allocation is guaranteed to be time-bounded.

Parameters

heap The descriptor address of the heap to get memory from.

size The size in bytes of the requested block. Sizes lower or equal to the page size are rounded either to the minimum allocation size if lower than this value, or to the minimum alignment size if greater or equal to this value. In the current implementation, with MINALLOC = 8 and MINALIGN = 16, a 7 bytes request will be rounded to 8 bytes, and a 17 bytes request will be rounded to 32.

Returns

The address of the allocated region upon success, or NULL if no memory is available from the specified heap.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task

Rescheduling: never.

Referenced by pthread_cond_init(), and xnshadow_map().

5.13.2.2 void xnheap_destroy (xnheap_t * *heap*, void(*)(xnheap_t *heap, void *extaddr, u_long extsize, void *cookie) *flushfn*, void * *cookie*)

Destroys a memory heap.

Destroys a memory heap.

Parameters

heap The descriptor address of the destroyed heap.

flushfn If non-NULL, the address of a flush routine which will be called for each extent attached to the heap. This routine can be used by the calling code to further release the heap memory.

cookie If *flushfn* is non-NULL, *cookie* is an opaque pointer which will be passed unmodified to *flushfn*.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task

Rescheduling: never.

Referenced by xnpod_init(), and xnpod_shutdown().

5.13.2.3 int xnheap_extend (xnheap_t * *heap*, void * *extaddr*, u_long *extsize*)

Extend a memory heap.

Add a new extent to an existing memory heap.

Parameters

heap The descriptor address of the heap to add an extent to.

extaddr The address of the extent memory.

extsize The size of the extent memory (in bytes). In the current implementation, this size must match the one of the initial extent passed to xnheap_init().

Returns

0 is returned upon success, or -EINVAL is returned if *extsize* differs from the initial extent's size.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task

Rescheduling: never.

5.13.2.4 int xnheap_free (xnheap_t * *heap*, void * *block*)

Release a memory block to a memory heap.

Releases a memory region to the memory heap it was previously allocated from.

Parameters

heap The descriptor address of the heap to release memory to.

block The address of the region to be returned to the heap.

Returns

0 is returned upon success, or one of the following error codes:

- -EFAULT is returned whenever the memory address is outside the heap address space.
- -EINVAL is returned whenever the memory address does not represent a valid block.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task

Rescheduling: never.

References xnheap_test_and_free().

Referenced by pthread_cond_init().

5.13.2.5 int xnheap_init (xnheap_t * *heap*, void * *heapaddr*, u_long *heapsize*, u_long *pagesize*)

Initialize a memory heap.

Initializes a memory heap suitable for time-bounded allocation requests of dynamic memory.

Parameters

heap The address of a heap descriptor which will be used to store the allocation data. This descriptor must always be valid while the heap is active therefore it must be allocated in permanent memory.

heapaddr The address of the heap storage area. All allocations will be made from the given area in time-bounded mode. Since additional extents can be added to a heap, this parameter is also known as the "initial extent".

heapsize The size in bytes of the initial extent pointed at by *heapaddr*. *heapsize* must be a multiple of pagesize and lower than 16 Mbytes. *heapsize* must be large enough to contain a dynamically-sized internal header. The following formula gives the size of this header:

H = heapsize, P=pagesize, M=sizeof(struct pagemap), E=sizeof(xnextent_t)

hdrsize = ((H - E) * M) / (M + 1)

This value is then aligned on the next 16-byte boundary. The routine xnheap_overhead() computes the corrected heap size according to the previous formula.

Parameters

pagesize The size in bytes of the fundamental memory page which will be used to subdivide the heap internally. Choosing the right page size is important regarding performance and memory fragmentation issues, so it might be a good idea to take a look at `http://docs.FreeBSD.org/44doc/papers/kernmalloc.pdf` to pick the best one for your needs. In the current implementation, pagesize must be a power of two in the range [8 .. 32768] inclusive.

Returns

0 is returned upon success, or one of the following error codes:

- -EINVAL is returned whenever a parameter is invalid.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task

Rescheduling: never.

Referenced by xnpod_init().

5.13.2.6 void xnheap_schedule_free (xnheap_t * *heap*, void * *block*, xnholder_t * *link*)

Schedule a memory block for release.

This routine records a block for later release by xnheap_finalize_free(). This service is useful to lazily free blocks of heap memory when immediate release is not an option, e.g. when active references are still pending on the object for a short time after the call. xnheap_finalize_free() is expected to be eventually called by the client code at some point in the future when actually freeing the idle objects is deemed safe.

Parameters

heap The descriptor address of the heap to release memory to.

block The address of the region to be returned to the heap.

link The address of a link member, likely but not necessarily within the released object, which will be used by the heap manager to hold the block in the queue of idle objects.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task

Rescheduling: never.

5.13.2.7 void xnheap_set_label (xnheap_t * *heap*, const char * *label*, ...)

Set the heap's label string.

Set the heap label that will be used in statistic outputs.

Parameters

heap The address of a heap descriptor.

label Label string displayed in statistic outputs. This parameter can be a format string, in which case succeeding parameters will be used to resolve the final label.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task

Rescheduling: never.

Referenced by xnpod_init().

5.13.2.8 int xnheap_test_and_free (xnheap_t * *heap*, void * *block*, int(*)(void *block) *ckfn*)

Test and release a memory block to a memory heap.

Releases a memory region to the memory heap it was previously allocated from. Before the actual release is performed, an optional user-defined can be invoked to check for additional criteria with respect to the request consistency.

Parameters

heap The descriptor address of the heap to release memory to.

block The address of the region to be returned to the heap.

ckfn The address of a user-supplied verification routine which is to be called after the memory address specified by *block* has been checked for validity. The routine is expected to proceed to further consistency checks, and either return zero upon success, or non-zero upon error. In the latter case, the release process is aborted, and *ckfn's* return value is passed back to the caller of this service as its error return code. *ckfn* must not trigger the rescheduling procedure either directly or indirectly.

Returns

0 is returned upon success, or -EINVAL is returned whenever the block is not a valid region of the specified heap. Additional return codes can also be defined locally by the *ckfn* routine.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task

Rescheduling: never.

Referenced by xnheap_free().

5.14 Interrupt management.

Files

- file intr.c

  Interrupt management.

Functions

- int xnintr_init (xnintr_t *intr, const char *name, unsigned irq, xnisr_t isr, xniack_t iack, xnflags_t flags)

  Initialize an interrupt object.

- int xnintr_destroy (xnintr_t *intr)
  Destroy an interrupt object.
- int xnintr_attach (xnintr_t *intr, void *cookie)
  Attach an interrupt object.
- int xnintr_detach (xnintr_t *intr)
  Detach an interrupt object.
- void xnintr_enable (xnintr_t *intr)
  Enable an interrupt object.
- void xnintr_disable (xnintr_t *intr)
  Disable an interrupt object.
- void xnintr_affinity (xnintr_t *intr, xnarch_cpumask_t cpumask)
  Set interrupt's processor affinity.

5.14.1 Detailed Description

Interrupt management.

5.14.2 Function Documentation

5.14.2.1 void xnintr_affinity (xnintr_t * *intr*, xnarch_cpumask_t *cpumask*)

Set interrupt's processor affinity.

Causes the IRQ associated with the interrupt object *intr* to be received only on processors which bits are set in *cpumask*.

Parameters

intr The descriptor address of the interrupt object which affinity is to be changed.

cpumask The new processor affinity of the interrupt object.

Note

Depending on architectures, setting more than one bit in *cpumask* could be meaningless.

5.14.2.2 int xnintr_attach (xnintr_t * *intr*, void * *cookie*)

Attach an interrupt object.

Attach an interrupt object previously initialized by xnintr_init(). After this operation is completed, all IRQs received from the corresponding interrupt channel are directed to the object's ISR.

Parameters

intr The descriptor address of the interrupt object to attach.

cookie A user-defined opaque value which is stored into the interrupt object descriptor for further retrieval by the ISR/ISR handlers.

Returns

0 is returned on success. Otherwise:

- -EINVAL is returned if a low-level error occurred while attaching the interrupt.
- -EBUSY is returned if the interrupt object was already attached.

Note

The caller **must not** hold nklock when invoking this service, this would cause deadlocks.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task

Rescheduling: never.

Note

Attaching an interrupt resets the tracked number of receipts to zero.

Referenced by rtdm_irq_request().

5.14.2.3 int xnintr_destroy (xnintr_t * *intr*)

Destroy an interrupt object.

Destroys an interrupt object previously initialized by xnintr_init(). The interrupt object is automatically detached by a call to xnintr_detach(). No more IRQs will be dispatched by this object after this service has returned.

Parameters

intr The descriptor address of the interrupt object to destroy.

Returns

0 is returned on success. Otherwise, -EINVAL is returned if an error occurred while detaching the interrupt (see xnintr_detach()).

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task

Rescheduling: never.

References xnintr_detach().

5.14.2.4 int xnintr_detach (xnintr_t * *intr*)

Detach an interrupt object.

Detach an interrupt object previously attached by xnintr_attach(). After this operation is completed, no more IRQs are directed to the object's ISR, but the interrupt object itself remains valid. A detached interrupt object can be attached again by a subsequent call to xnintr_attach().

Parameters

intr The descriptor address of the interrupt object to detach.

Returns

0 is returned on success. Otherwise:

- -EINVAL is returned if a low-level error occurred while detaching the interrupt, or if the interrupt object was not attached. In both cases, no action is performed.

Note

The caller **must not** hold nklock when invoking this service, this would cause deadlocks.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task

Rescheduling: never.

Referenced by xnintr_destroy().

5.14.2.5 void xnintr_disable (xnintr_t * *intr*)

Disable an interrupt object.

Disables the hardware interrupt line associated with an interrupt object. This operation invalidates further interrupt requests from the given source until the IRQ line is re-enabled anew.

Parameters

intr The descriptor address of the interrupt object to disable.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task

Rescheduling: never.

5.14.2.6 void xnintr_enable (xnintr_t * *intr*)

Enable an interrupt object.

Enables the hardware interrupt line associated with an interrupt object. Over real-time control layers which mask and acknowledge IRQs, this operation is necessary to revalidate the interrupt channel so that more interrupts can be notified.

Parameters

intr The descriptor address of the interrupt object to enable.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task

Rescheduling: never.

Referenced by rtdm_irq_request().

5.14.2.7 int xnintr_init (xnintr_t * *intr,* const char * *name,* unsigned *irq,* xnisr_t *isr,* xniack_t *iack,* xnflags_t *flags*)

Initialize an interrupt object.

Associates an interrupt object with an IRQ line.

When an interrupt occurs on the given *irq* line, the ISR is fired in order to deal with the hardware event. The interrupt service code may call any non-suspensive service from the nucleus.

Upon receipt of an IRQ, the ISR is immediately called on behalf of the interrupted stack context, the rescheduling procedure is locked, and the interrupt source is masked at hardware level. The status value returned by the ISR is then checked for the following values:

- XN_ISR_HANDLED indicates that the interrupt request has been fulfilled by the ISR.
- XN_ISR_NONE indicates the opposite to XN_ISR_HANDLED. The ISR must always return this value when it determines that the interrupt request has not been issued by the dedicated hardware device.

In addition, one of the following bits may be set by the ISR :

NOTE: use these bits with care and only when you do understand their effect on the system. The ISR is not encouraged to use these bits in case it shares the IRQ line with other ISRs in the real-time domain.

- XN_ISR_NOENABLE causes the nucleus to ask the real-time control layer _not_ to re-enable the IRQ line (read the following section). ipipe_end_irq() must be called to re-enable the IRQ line later.

- XN_ISR_PROPAGATE tells the nucleus to require the real-time control layer to forward the IRQ. For instance, this would cause the Adeos control layer to propagate the interrupt down the interrupt pipeline to other Adeos domains, such as Linux. This is the regular way to share interrupts between the nucleus and the host system. In effect, XN_ISR_-PROPAGATE implies XN_ISR_NOENABLE since it would make no sense to re-enable the interrupt channel before the next domain down the pipeline has had a chance to process the propagated interrupt.

The nucleus re-enables the IRQ line by default. Over some real-time control layers which mask and acknowledge IRQs, this operation is necessary to revalidate the interrupt channel so that more interrupts can be notified.

A count of interrupt receipts is tracked into the interrupt descriptor, and reset to zero each time the interrupt object is attached. Since this count could wrap around, it should be used as an indication of interrupt activity only.

Parameters

intr The address of a interrupt object descriptor the nucleus will use to store the object-specific data. This descriptor must always be valid while the object is active therefore it must be allocated in permanent memory.

name An ASCII string standing for the symbolic name of the interrupt object or NULL ("<unknown>" will be applied then).

irq The hardware interrupt channel associated with the interrupt object. This value is architecture-dependent. An interrupt object must then be attached to the hardware interrupt vector using the xnintr_attach() service for the associated IRQs to be directed to this object.

isr The address of a valid low-level interrupt service routine if this parameter is non-zero. This handler will be called each time the corresponding IRQ is delivered on behalf of an interrupt context. When called, the ISR is passed the descriptor address of the interrupt object.

iack The address of an optional interrupt acknowledge routine, aimed at replacing the default one. Only very specific situations actually require to override the default setting for this parameter, like having to acknowledge non-standard PIC hardware. *iack* should return a non-zero value to indicate that the interrupt has been properly acknowledged. If *iack* is NULL, the default routine will be used instead.

flags A set of creation flags affecting the operation. The valid flags are:

- XN_ISR_SHARED enables IRQ-sharing with other interrupt objects.

- XN_ISR_EDGE is an additional flag need to be set together with XN_ISR_SHARED to enable IRQ-sharing of edge-triggered interrupts.

Returns

0 is returned on success. Otherwise, -EINVAL is returned if *irq* is not a valid interrupt number.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code

- Kernel-based task

Rescheduling: never.

Referenced by rtdm_irq_request(), and xnpod_enable_timesource().

5.15 Lightweight key-to-object mapping service

Files

- file map.h
- file map.c

Functions

- xnmap_t * xnmap_create (int nkeys, int reserve, int offset)

  Create a map.

- void xnmap_delete (xnmap_t *map)

  Delete a map.

- int xnmap_enter (xnmap_t *map, int key, void *objaddr)

  Index an object into a map.

- int xnmap_remove (xnmap_t *map, int key)

  Remove an object reference from a map.

- static void * xnmap_fetch_nocheck (xnmap_t *map, int key)

  Search an object into a map - unchecked form.

- static void * xnmap_fetch (xnmap_t *map, int key)

  Search an object into a map.

5.15.1 Detailed Description

A map is a simple indexing structure which associates unique integer keys with pointers to objects. The current implementation supports reservation, for naming/indexing the real-time objects skins create, either on a fixed, user-provided integer (i.e. a reserved key value), or by drawing the next available key internally if the caller did not specify any fixed key. For instance, in some given map, the key space ranging from 0 to 255 could be reserved for fixed keys, whilst the range from 256 to 511 could be available for drawing free keys dynamically.

A maximum of 1024 unique keys per map is supported on 32bit machines.

(This implementation should not be confused with C++ STL maps, which are dynamically expandable and allow arbitrary key types; Xenomai maps don't).

5.15.2 Function Documentation

5.15.2.1 xnmap_t * xnmap_create (int *nkeys*, int *reserve*, int *offset*)

Create a map.

Allocates a new map with the specified addressing capabilities. The memory is obtained from the Xenomai system heap.

Parameters

nkeys The maximum number of unique keys the map will be able to hold. This value cannot exceed the static limit represented by XNMAP_MAX_KEYS, and must be a power of two.

reserve The number of keys which should be kept for reservation within the index space. Reserving a key means to specify a valid key to the xnmap_enter() service, which will then attempt to register this exact key, instead of drawing the next available key from the unreserved index space. When reservation is in effect, the unreserved index space will hold key values greater than *reserve*, keeping the low key values for the reserved space. For instance, passing *reserve* = 32 would cause the index range [0 .. 31] to be kept for reserved keys. When non-zero, *reserve* is rounded to the next multiple of BITS_PER_LONG. If *reserve* is zero no reservation will be available from the map.

offset The lowest key value xnmap_enter() will return to the caller. Key values will be in the range [0 + offset .. *nkeys* + offset - 1]. Negative offsets are valid.

Returns

the address of the new map is returned on success; otherwise, NULL is returned if *nkeys* is invalid.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task

Rescheduling: never.

5.15.2.2 void xnmap_delete (xnmap_t * *map*)

Delete a map.

Deletes a map, freeing any associated memory back to the Xenomai system heap.

Parameters

map The address of the map to delete.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task

Rescheduling: never.

5.15.2.3 int xnmap_enter (xnmap_t * *map*, int *key*, void * *objaddr*)

Index an object into a map.

Insert a new object into the given map.

Parameters

map The address of the map to insert into.

key The key to index the object on. If this key is within the valid index range [0 - offset .. nkeys - offset - 1], then an attempt to reserve this exact key is made. If *key* has an out-of-range value lower or equal to 0 - offset - 1, then an attempt is made to draw a free key from the unreserved index space.

objaddr The address of the object to index on the key. This value will be returned by a successful call to xnmap_fetch() with the same key.

Returns

a valid key is returned on success, either *key* if reserved, or the next free key. Otherwise:

- -EEXIST is returned upon attempt to reserve a busy key.
- -ENOSPC when no more free key is available.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task

Rescheduling: never.

5.15.2.4 void xnmap_fetch (xnmap_t * *map*, int *key*) [inline, static]

Search an object into a map.

Retrieve an object reference from the given map by its index key.

Parameters

map The address of the map to retrieve from.

key The key to be searched for in the map index.

Returns

The indexed object address is returned on success, otherwise NULL is returned when *key* is invalid or no object is currently indexed on it.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task

Rescheduling: never.

5.15.2.5 void xnmap_fetch_nocheck (xnmap_t * *map*, int *key*) [inline, static]

Search an object into a map - unchecked form.

Retrieve an object reference from the given map by its index key, but does not perform any sanity check on the provided key.

Parameters

map The address of the map to retrieve from.

key The key to be searched for in the map index.

Returns

The indexed object address is returned on success, otherwise NULL is returned when no object is currently indexed on *key*.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task

Rescheduling: never.

5.15.2.6 int xnmap_remove (xnmap_t * *map*, int *key*)

Remove an object reference from a map.

Removes an object reference from the given map, releasing the associated key.

Parameters

map The address of the map to remove from.

key The key the object reference to be removed is indexed on.

Returns

0 is returned on success. Otherwise:

- -ESRCH is returned if *key* is invalid.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task

Rescheduling: never.

5.16 Real-time pod services.

Data Structures

- struct xnpod

  Real-time pod descriptor.

Files

- file pod.h

  Real-time pod interface header.

- file pod.c

  Real-time pod services.

Functions

- void __xnpod_reset_thread (struct xnthread *thread)

  Reset the thread.

- int xnpod_init (void)

  Initialize the core pod.

- int xnpod_enable_timesource (void)

  Activate the core time source.

- void xnpod_disable_timesource (void)

  Stop the core time source.

- void xnpod_shutdown (int xtype)

  Shutdown the current pod.

- int xnpod_init_thread (struct xnthread *thread, const struct xnthread_init_attr *attr, struct xnsched_class *sched_class, const union xnsched_policy_param *sched_param)

  Initialize a new thread.

- int xnpod_start_thread (xnthread_t *thread, const struct xnthread_start_attr *attr)

  Initial start of a newly created thread.

- void xnpod_stop_thread (xnthread_t *thread)

  Stop a thread.

- void xnpod_delete_thread (xnthread_t *thread)

  Delete a thread.

- void xnpod_abort_thread (xnthread_t *thread)

  Abort a thread.

- xnflags_t xnpod_set_thread_mode (xnthread_t *thread, xnflags_t clrmask, xnflags_t setmask)

  Change a thread's control mode.

- void xnpod_suspend_thread (xnthread_t *thread, xnflags_t mask, xnticks_t timeout, xntmode_t timeout_mode, struct xnsynch *wchan)

  Suspend a thread.

- void xnpod_resume_thread (xnthread_t *thread, xnflags_t mask)

  Resume a thread.

- int xnpod_unblock_thread (xnthread_t *thread)

  Unblock a thread.

- int xnpod_set_thread_schedparam (struct xnthread *thread, struct xnsched_class *sched_class, const union xnsched_policy_param *sched_param)

  Change the base scheduling parameters of a thread.

- int xnpod_migrate_thread (int cpu)

  Migrate the current thread.

- void xnpod_dispatch_signals (void)

  Deliver pending asynchronous signals to the running thread.

- static void xnpod_schedule (void)

  Rescheduling procedure entry point.

- int xnpod_handle_exception (struct ipipe_trap_data *d)

  Exception handler.

- int xnpod_set_thread_periodic (xnthread_t *thread, xnticks_t idate, xntmode_t timeout_mode, xnticks_t period)

  Make a thread periodic.

- int xnpod_wait_thread_period (unsigned long *overruns_r)

  Wait for the next periodic release point.

- int xnpod_set_thread_tslice (struct xnthread *thread, xnticks_t quantum)

  Set thread time-slicing information.

- int xnpod_add_hook (int type, void(*routine)(xnthread_t *))

  Install a nucleus hook.

- int xnpod_remove_hook (int type, void(*routine)(xnthread_t *))

  Remove a nucleus hook.

- void xnpod_welcome_thread (xnthread_t *thread, int imask)

  Thread prologue.

5.16.1 Detailed Description

Real-time pod services.

5.16.2 Function Documentation

5.16.2.1 void __xnpod_reset_thread (struct xnthread * *thread*)

Reset the thread.

For internal use only.

This internal routine resets the state of a thread so that it can be subsequently stopped or restarted.

References XNLOCK, xnpod_resume_thread(), xnpod_unblock_thread(), XNSUSP, and xnsynch_release_all_ownerships().

Referenced by xnpod_stop_thread().

5.16.2.2 void xnpod_abort_thread (xnthread_t * *thread*)

Abort a thread.

Unconditionally terminates a thread and releases all the nucleus resources it currently holds, regardless of whether the target thread is currently active in kernel or user-space. xnpod_abort_thread() should be reserved for use by skin cleanup routines; xnpod_delete_thread() should be preferred as the common method for removing threads from a running system.

Parameters

thread The descriptor address of the terminated thread.

This service forces a call to xnpod_delete_thread() for the target thread.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task

Rescheduling: possible if the current thread self-deletes.

References XNABORT, XNDORMANT, xnpod_delete_thread(), and xnpod_suspend_thread().

5.16.2.3 int xnpod_add_hook (int *type*, void(*)(xnthread_t *) *routine*)

Install a nucleus hook.

The nucleus allows to register user-defined routines which get called whenever a specific scheduling event occurs. Multiple hooks can be chained for a single event type, and get called on a FIFO basis.

The scheduling is locked while a hook is executing.

Parameters

type Defines the kind of hook to install:

- XNHOOK_THREAD_START: The user-defined routine will be called on behalf of the starter thread whenever a new thread starts. The descriptor address of the started thread is passed to the routine.
- XNHOOK_THREAD_DELETE: The user-defined routine will be called on behalf of the deletor thread whenever a thread is deleted. The descriptor address of the deleted thread is passed to the routine.
- XNHOOK_THREAD_SWITCH: The user-defined routine will be called on behalf of the resuming thread whenever a context switch takes place. The descriptor address of the thread which has been switched out is passed to the routine.

Parameters

routine The address of the user-supplied routine to call.

Returns

0 is returned on success. Otherwise, one of the following error codes indicates the cause of the failure:

- -EINVAL is returned if type is incorrect.
- -ENOMEM is returned if not enough memory is available from the system heap to add the new hook.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task

Rescheduling: never.

5.16.2.4 void xnpod_delete_thread (xnthread_t * *thread*)

Delete a thread.

Terminates a thread and releases all the nucleus resources it currently holds. A thread exists in the system since xnpod_init_thread() has been called to create it, so this service must be called in order to destroy it afterwards.

Parameters

thread The descriptor address of the terminated thread.

The target thread's resources may not be immediately removed if this is an active shadow thread running in user-space. In such a case, the mated Linux task is sent a termination signal instead, and the actual deletion is deferred until the task exit event is called.

The DELETE hooks are called on behalf of the calling context (if any). The information stored in the thread control block remains valid until all hooks have been called.

Self-terminating a thread is allowed. In such a case, this service does not return to the caller.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task

Rescheduling: possible if the current thread self-deletes.

References xnsched::curr, xnsched::status, XNABORT, XNCANPND, XNDEFCAN, XNDORMANT, XNMIGRATE, XNPEND, xnpod_schedule(), xnpod_unblock_thread(), XNREADY, XNROOT, xnselector_destroy(), xnsynch_forget_sleeper(), xnsynch_release_all_ownerships(), xntimer_destroy(), and XNZOMBIE.

Referenced by rtdm_task_init(), xnpod_abort_thread(), and xnpod_shutdown().

5.16.2.5 void xnpod_disable_timesource (void)

Stop the core time source.

Releases the hardware timer, and deactivates the system clock.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- User-space task in secondary mode

Rescheduling: never.

References xntimer_freeze().

Referenced by xnpod_shutdown().

5.16.2.6 void xnpod_dispatch_signals (void)

Deliver pending asynchronous signals to the running thread.

For internal use only.

This internal routine checks for the presence of asynchronous signals directed to the running thread, and attempts to start the asynchronous service routine (ASR) if any. Called with nklock locked, interrupts off.

References XNASDI.

Referenced by xnpod_welcome_thread(), and xnshadow_harden().

5.16.2.7 int xnpod_enable_timesource (void)

Activate the core time source.

On every architecture, Xenomai directly manages a hardware timer clocked in one-shot mode, to support any number of software timers internally. Timings are always specified as a count of nanoseconds.

The xnpod_enable_timesource() service configures the hardware timer chip. Because Xenomai most often interposes on the system timer used by the Linux kernel, a software timer may be started to relay periodic ticks to the host kernel if needed.

Returns

0 is returned on success. Otherwise:

- -ENODEV is returned if a failure occurred while configuring the hardware timer.
- -ENOSYS is returned if no active pod exists.

Environments:

This service can be called from:

- Regular Linux kernel context.

Rescheduling: never.

References xnsched::htimer, xnintr_init(), and xntimer_start().

Referenced by xnpod_init().

5.16.2.8 int xnpod_handle_exception (struct ipipe_trap_data * *d*)

Exception handler.

This is the handler which is called whenever an exception/fault is caught over the primary domain.

Parameters

d A pointer to the trap information block received from the pipeline core.

References xnpod_suspend_thread(), xnshadow_relax(), and XNSUSP.

5.16.2.9 int xnpod_init (void)

Initialize the core pod.

Initializes the core interface pod which can subsequently be used to start real-time activities. Once the core pod is active, real-time skins can be stacked over. There can only be a single core pod active in the host environment.

Returns

0 is returned on success. Otherwise:

- -ENOMEM is returned if the memory manager fails to initialize.

Environments:

This service can be called from:

- Kernel module initialization code

References xnpod::refcnt, xnsched::rootcb, xnpod::sched, xnpod::status, xnpod::tdeleteq, xnpod::threadq, xnpod::timerlck, xnpod::tstartq, xnpod::tswitchq, xnheap_destroy(), xnheap_init(), xnheap_set_label(), xnpod_enable_timesource(), and xnpod_shutdown().

5.16.2.10 int xnpod_init_thread (struct xnthread * *thread*, const struct xnthread_init_attr * *attr*, struct xnsched_class * *sched_class*, const union xnsched_policy_param * *sched_param*)

Initialize a new thread.

Initializes a new thread attached to the active pod. The thread is left in an innocuous state until it is actually started by xnpod_start_thread().

Parameters

thread The address of a thread descriptor the nucleus will use to store the thread-specific data. This descriptor must always be valid while the thread is active therefore it must be allocated in permanent memory.

Warning

Some architectures may require the descriptor to be properly aligned in memory; this is an additional reason for descriptors not to be laid in the program stack where alignement constraints might not always be satisfied.

Parameters

attr A pointer to an attribute block describing the initial properties of the new thread. Members of this structure are defined as follows:

- name: An ASCII string standing for the symbolic name of the thread. This name is copied to a safe place into the thread descriptor. This name might be used in various situations by the nucleus for issuing human-readable diagnostic messages, so it is usually a good idea to provide a sensible value here. NULL is fine though and means "anonymous".

- flags: A set of creation flags affecting the operation. The following flags can be part of this bitmask, each of them affecting the nucleus behaviour regarding the created thread:

- XNSUSP creates the thread in a suspended state. In such a case, the thread will have to be explicitly resumed using the xnpod_resume_thread() service for its execution to actually begin, additionally to issuing xnpod_start_thread() for it. This flag can also be specified when invoking xnpod_start_thread() as a starting mode.

- XNFPU (enable FPU) tells the nucleus that the new thread will use the floating-point unit. In such a case, the nucleus will handle the FPU context save/restore ops upon thread switches at the expense of a few additional cycles per context switch. By default, a thread is not expected to use the FPU. This flag is simply ignored when the nucleus runs on behalf of a userspace-based real-time control layer since the FPU management is always active if present.

- stacksize: The size of the stack (in bytes) for the new thread. If zero is passed, the nucleus will use a reasonable pre-defined size depending on the underlying real-time control layer.

- ops: A pointer to a structure defining the class-level operations available for this thread. Fields from this structure must have been set appropriately by the caller.

Parameters

sched_class The initial scheduling class the new thread should be assigned to.

sched_param The initial scheduling parameters to set for the new thread; *sched_param* must be valid within the context of *sched_class*.

Returns

0 is returned on success. Otherwise, one of the following error codes indicates the cause of the failure:

- -EINVAL is returned if *attr->flags* has invalid bits set.
- -ENOMEM is returned if not enough memory is available from the system heap to create the new thread's stack.

Side-effect: This routine does not call the rescheduling procedure.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task

Rescheduling: never.

References XNDORMANT, XNFPU, xnpod_suspend_thread(), XNSHADOW, and XNSUSP.

Referenced by pthread_create(), and rtdm_task_init().

5.16.2.11 int xnpod_migrate_thread (int *cpu*)

Migrate the current thread.

This call makes the current thread migrate to another CPU if its affinity allows it.

Parameters

cpu The destination CPU.

Return values

0 if the thread could migrate ;

-EPERM if the calling context is asynchronous, or the current thread affinity forbids this migration ;

-EBUSY if the scheduler is locked.

References xnpod_schedule().

5.16.2.12 int xnpod_remove_hook (int *type*, void(*)(xnthread_t *) *routine*)

Remove a nucleus hook.

This service removes a nucleus hook previously registered using xnpod_add_hook().

Parameters

type Defines the kind of hook to remove among XNHOOK_THREAD_START, XNHOOK_-THREAD_DELETE and XNHOOK_THREAD_SWITCH.

routine The address of the user-supplied routine to remove.

Returns

0 is returned on success. Otherwise, -EINVAL is returned if type is incorrect or if the routine has never been registered before.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task

Rescheduling: never.

5.16.2.13 void xnpod_resume_thread (xnthread_t * *thread*, xnflags_t *mask*)

Resume a thread.

Resumes the execution of a thread previously suspended by one or more calls to xnpod_suspend_-thread(). This call removes a suspensive condition affecting the target thread. When all suspensive conditions are gone, the thread is left in a READY state at which point it becomes eligible anew for scheduling.

Parameters

thread The descriptor address of the resumed thread.

mask The suspension mask specifying the suspensive condition to remove from the thread's wait mask. Possible values usable by the caller are:

- XNSUSP. This flag removes the explicit suspension condition. This condition might be additive to the XNPEND condition.
- XNDELAY. This flag removes the counted delay wait condition.
- XNPEND. This flag removes the resource wait condition. If a watchdog is armed, it is automatically disarmed by this call. Unlike the two previous conditions, only the current thread can set this condition for itself, i.e. no thread can force another one to pend on a resource.

When the thread is eventually resumed by one or more calls to xnpod_resume_thread(), the caller of xnpod_suspend_thread() in the awakened thread that suspended itself should check for the following bits in its own information mask to determine what caused its wake up:

- XNRMID means that the caller must assume that the pended synchronization object has been destroyed (see xnsynch_flush()).
- XNTIMEO means that the delay elapsed, or the watchdog went off before the corresponding synchronization object was signaled.
- XNBREAK means that the wait has been forcibly broken by a call to xnpod_unblock_-thread().

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task

Rescheduling: never.

References XNDELAY, XNHELD, XNPEND, XNREADY, xnsynch_forget_sleeper(), and xntimer_-stop().

Referenced by __xnpod_reset_thread(), xnpod_start_thread(), xnpod_unblock_thread(), xnsynch_flush(), xnsynch_wakeup_one_sleeper(), and xnsynch_wakeup_this_sleeper().

5.16.2.14 void xnpod_schedule (void) [inline, static]

Rescheduling procedure entry point.

This is the central rescheduling routine which should be called to validate and apply changes which have previously been made to the nucleus scheduling state, such as suspending, resuming or changing the priority of threads. This call first determines if a thread switch should take place, and performs it as needed. xnpod_schedule() schedules out the current thread if:

- the current thread is now blocked or deleted.
- a runnable thread from a higher priority scheduling class is waiting for the CPU.
- the current thread does not lead the runnable threads from its own scheduling class (e.g. round-robin in the RT class).

The nucleus implements a lazy rescheduling scheme so that most of the services affecting the threads state MUST be followed by a call to the rescheduling procedure for the new scheduling state to be applied. In other words, multiple changes on the scheduler state can be done in a row, waking threads up, blocking others, without being immediately translated into the corresponding context switches, like it would be necessary would it appear that a higher priority thread than the current one became runnable for instance. When all changes have been applied, the rescheduling procedure is then called to consider those changes, and possibly replace the current thread by another one.

As a notable exception to the previous principle however, every action which ends up suspending or deleting the current thread begets an immediate call to the rescheduling procedure on behalf of the service causing the state transition. For instance, self-suspension, self-destruction, or sleeping on a synchronization object automatically leads to a call to the rescheduling procedure, therefore the caller does not need to explicitly issue xnpod_schedule() after such operations.

The rescheduling procedure always leads to a null-effect if it is called on behalf of an ISR or callout. Any outstanding scheduler lock held by the outgoing thread will be restored when the thread is scheduled back in.

Calling this procedure with no applicable context switch pending is harmless and simply leads to a null-effect.

Side-effects:

- If an asynchronous service routine exists, the pending asynchronous signals are delivered to a resuming thread or on behalf of the caller before it returns from the procedure if no context switch has taken place. This behaviour can be disabled by setting the XNASDI flag in the thread's status mask by calling xnpod_set_thread_mode().

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine, although this leads to a no-op.
- Kernel-based task
- User-space task

Note

The switch hooks are called on behalf of the resuming thread.

References xnsched::lflags, and xnsched::status.

Referenced by pthread_cancel(), pthread_set_mode_np(), pthread_setschedparam(), pthread_-setschedparam_ex(), rtdm_event_signal(), rtdm_sem_up(), xnpod_delete_thread(), xnpod_-migrate_thread(), xnpod_shutdown(), xnpod_start_thread(), xnpod_stop_thread(), xnpod_-suspend_thread(), xnregistry_enter(), xnregistry_put(), xnselect_bind(), and xnselect_destroy().

5.16.2.15 xnflags_t xnpod_set_thread_mode (xnthread_t * *thread*, xnflags_t *clrmask*, xnflags_t *setmask*)

Change a thread's control mode.

Change the control mode of a given thread. The control mode affects the behaviour of the nucleus regarding the specified thread.

Parameters

thread The descriptor address of the affected thread.

clrmask Clears the corresponding bits from the control field before setmask is applied. The scheduler lock held by the current thread can be forcibly released by passing the XNLOCK bit in this mask. In this case, the lock nesting count is also reset to zero.

setmask The new thread mode. The following flags can be part of this bitmask, each of them affecting the nucleus behaviour regarding the thread:

- XNLOCK causes the thread to lock the scheduler. The target thread will have to call the xnpod_unlock_sched() service to unlock the scheduler or clear the XNLOCK bit forcibly using this service. A non-preemptible thread may still block, in which case, the lock is reasserted when the thread is scheduled back in.

- XNASDI disables the asynchronous signal handling for this thread. See xnpod_schedule() for more on this.

Environments:

This service can be called from:

- Kernel-based task
- User-space task in primary mode.

Rescheduling: never, therefore, the caller should reschedule if XNLOCK has been passed into *clrmask*.

References XNLOCK.

Referenced by pthread_set_mode_np().

5.16.2.16 int xnpod_set_thread_periodic (xnthread_t * *thread*, xnticks_t *idate*, xntmode_t *timeout_mode*, xnticks_t *period*)

Make a thread periodic.

Make a thread periodic by programming its first release point and its period in the processor time line. Subsequent calls to xnpod_wait_thread_period() will delay the thread until the next periodic release point in the processor timeline is reached.

Parameters

- ***thread*** The descriptor address of the affected thread. This thread is immediately delayed until the first periodic release point is reached.
- ***idate*** The initial (absolute) date of the first release point, expressed in nanoseconds. The affected thread will be delayed by the first call to xnpod_wait_thread_period() until this point is reached. If *idate* is equal to XN_INFINITE, the current system date is used, and no initial delay takes place. In the latter case, *timeout_mode* is not considered and can have any valid value.
- ***timeout_mode*** The mode of the *idate* parameter. It can either be set to XN_ABSOLUTE or XN_REALTIME with *idate* different from XN_INFINITE (see also xntimer_start()).
- ***period*** The period of the thread, expressed in nanoseconds. As a side-effect, passing XN_-INFINITE attempts to stop the thread's periodic timer; in the latter case, the routine always exits succesfully, regardless of the previous state of this timer.

Returns

0 is returned upon success. Otherwise:

- -ETIMEDOUT is returned *idate* is different from XN_INFINITE and represents a date in the past.
- -EINVAL is returned if *period* is different from XN_INFINITE but shorter than the scheduling latency value for the target system, as available from /proc/xenomai/latency. -EINVAL is also returned if *timeout_mode* is not compatible with *idate*, such as XN_RELATIVE with *idate* different from XN_INFINITE.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task

- User-space task

Rescheduling: none.

References xntimer_start(), and xntimer_stop().

Referenced by pthread_make_periodic_np(), and rtdm_task_init().

5.16.2.17 int xnpod_set_thread_schedparam (struct xnthread * *thread*, struct xnsched_class * *sched_class*, const union xnsched_policy_param * *sched_param*)

Change the base scheduling parameters of a thread.

Changes the base scheduling policy and paramaters of a thread. If the thread is currently blocked, waiting in priority-pending mode (XNSYNCH_PRIO) for a synchronization object to be signaled, the nucleus will attempt to reorder the object's wait queue so that it reflects the new sleeper's priority, unless the XNSYNCH_DREORD flag has been set for the pended object.

Parameters

thread The descriptor address of the affected thread.

sched_class The new scheduling class the thread should be assigned to.

sched_param The scheduling parameters to set for the thread; *sched_param* must be valid within the context of *sched_class*.

It is absolutely required to use this service to change a thread priority, in order to have all the needed housekeeping chores correctly performed. i.e. Do *not* call xnsched_set_policy() directly or worse, change the thread.cprio field by hand in any case.

Returns

0 is returned on success. Otherwise, a negative error code indicates the cause of a failure that happened in the scheduling class implementation for *sched_class*. Invalid parameters passed into *sched_param* are common causes of error.

Side-effects:

- This service does not call the rescheduling procedure but may affect the state of the runnable queue for the previous and new scheduling classes.

- Assigning the same scheduling class and parameters to a running or ready thread moves it to the end of the runnable queue, thus causing a manual round-robin.

- If the thread is a user-space shadow, this call propagates the request to the mated Linux task.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine

- Kernel-based task
- User-space task

Rescheduling: never.

Referenced by pthread_setschedparam(), and pthread_setschedparam_ex().

5.16.2.18 int xnpod_set_thread_tslice (struct xnthread * *thread*, xnticks_t *quantum*)

Set thread time-slicing information.

Update the time-slicing information for a given thread. This service enables or disables round-robin scheduling for the thread, depending on the value of *quantum*. By default, times-slicing is disabled for a new thread initialized by a call to xnpod_init_thread().

Parameters

thread The descriptor address of the affected thread.

quantum The time quantum assigned to the thread expressed in nanoseconds. If *quantum* is different from XN_INFINITE, the time-slice for the thread is set to that value and its current time credit is refilled (i.e. the thread is given a full time-slice to run next). Otherwise, if *quantum* equals XN_INFINITE, time-slicing is stopped for that thread.

Returns

0 is returned upon success. Otherwise:

- -EINVAL is returned if *quantum* is not XN_INFINITE, and the base scheduling class of the target thread does not support time-slicing.

Environments:

This service can be called from:

- Any kernel context.

Rescheduling: never.

References XNRRB, xntimer_start(), and xntimer_stop().

Referenced by pthread_create(), pthread_setschedparam(), and pthread_setschedparam_ex().

5.16.2.19 void xnpod_shutdown (int *xtype*)

Shutdown the current pod.

Forcibly shutdowns the active pod. All existing nucleus threads (but the root one) are terminated, and the system heap is freed.

Parameters

xtype An exit code passed to the host environment who started the nucleus. Zero is always interpreted as a successful return.

The nucleus never calls this routine directly. Skins should provide their own shutdown handlers which end up calling xnpod_shutdown() after their own housekeeping chores have been carried out.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code

Rescheduling: never.

References xnheap_destroy(), xnpod_delete_thread(), xnpod_disable_timesource(), xnpod_-schedule(), and XNROOT.

Referenced by xnpod_init().

5.16.2.20 int xnpod_start_thread (xnthread_t * *thread*, const struct xnthread_start_attr * *attr*)

Initial start of a newly created thread.

Starts a (newly) created thread, scheduling it for the first time. This call releases the target thread from the XNDORMANT state. This service also sets the initial mode and interrupt mask for the new thread.

Parameters

thread The descriptor address of the affected thread which must have been previously initialized by the xnpod_init_thread() service.

attr A pointer to an attribute block describing the execution properties of the new thread. Members of this structure are defined as follows:

- mode: The initial thread mode. The following flags can be part of this bitmask, each of them affecting the nucleus behaviour regarding the started thread:

- XNLOCK causes the thread to lock the scheduler when it starts. The target thread will have to call the xnpod_unlock_sched() service to unlock the scheduler. A non-preemptible thread may still block, in which case, the lock is reasserted when the thread is scheduled back in.

- XNASDI disables the asynchronous signal handling for this thread. See xnpod_schedule() for more on this.

- XNSUSP makes the thread start in a suspended state. In such a case, the thread will have to be explicitly resumed using the xnpod_resume_thread() service for its execution to actually begin.

- imask: The interrupt mask that should be asserted when the thread starts. The processor interrupt state will be set to the given value when the thread starts running. The interpretation of this value might be different across real-time layers, but a non-zero value should always mark an interrupt masking in effect (e.g. local_irq_disable()). Conversely, a zero value should always mark a fully preemptible state regarding interrupts (e.g. local_irq_enable()).

- affinity: The processor affinity of this thread. Passing XNPOD_ALL_CPUS or an empty affinity set means "any cpu".
- entry: The address of the thread's body routine. In other words, it is the thread entry point.
- cookie: A user-defined opaque cookie the nucleus will pass to the emerging thread as the sole argument of its entry point.

The START hooks are called on behalf of the calling context (if any).

Return values

0 if *thread* could be started ;

-EBUSY if *thread* was not dormant or stopped ;

-EINVAL if the value of *attr->affinity* is invalid.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task

Rescheduling: possible.

References XNDORMANT, xnpod_resume_thread(), xnpod_schedule(), XNSHADOW, XNSTARTED, and XNSUSP.

Referenced by rtdm_task_init(), and xnshadow_map().

5.16.2.21 void xnpod_stop_thread (xnthread_t * *thread*)

Stop a thread.

Stop a previously started thread. The thread is put back into the dormant state; however, it is not deleted from the system.

Parameters

thread The descriptor address of the affected thread which must have been previously started by the xnpod_start_thread() service.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task

Rescheduling: possible.

References __xnpod_reset_thread(), XNDORMANT, xnpod_schedule(), xnpod_suspend_thread(), and XNROOT.

5.16.2.22 void xnpod_suspend_thread (xnthread_t * *thread*, xnflags_t *mask*, xnticks_t *timeout*, xntmode_t *timeout_mode*, struct xnsynch * *wchan*)

Suspend a thread.

Suspends the execution of a thread according to a given suspensive condition. This thread will not be eligible for scheduling until it all the pending suspensive conditions set by this service are removed by one or more calls to xnpod_resume_thread().

Parameters

thread The descriptor address of the suspended thread.

mask The suspension mask specifying the suspensive condition to add to the thread's wait mask. Possible values usable by the caller are:

- XNSUSP. This flag forcibly suspends a thread, regardless of any resource to wait for. A reverse call to xnpod_resume_thread() specifying the XNSUSP bit must be issued to remove this condition, which is cumulative with other suspension bits.*wchan* should be NULL when using this suspending mode.

- XNDELAY. This flags denotes a counted delay wait (in ticks) which duration is defined by the value of the timeout parameter.

- XNPEND. This flag denotes a wait for a synchronization object to be signaled. The wchan argument must points to this object. A timeout value can be passed to bound the wait. This suspending mode should not be used directly by the client interface, but rather through the xnsynch_sleep_on() call.

Parameters

timeout The timeout which may be used to limit the time the thread pends on a resource. This value is a wait time given in nanoseconds. It can either be relative, absolute monotonic, or absolute adjustable depending on *timeout_mode*. Passing XN_INFINITE **and** setting *timeout_mode* to XN_RELATIVE specifies an unbounded wait. All other values are used to initialize a watchdog timer. If the current operation mode of the system timer is oneshot and *timeout* elapses before xnpod_suspend_thread() has completed, then the target thread will not be suspended, and this routine leads to a null effect.

timeout_mode The mode of the *timeout* parameter. It can either be set to XN_RELATIVE, XN_ABSOLUTE, or XN_REALTIME (see also xntimer_start()).

wchan The address of a pended resource. This parameter is used internally by the synchronization object implementation code to specify on which object the suspended thread pends. NULL is a legitimate value when this parameter does not apply to the current suspending mode (e.g. XNSUSP).

Note

If the target thread is a shadow which has received a Linux-originated signal, then this service immediately exits without suspending the thread, but raises the XNBREAK condition in its information mask.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task

Rescheduling: possible if the current thread suspends itself.

References xnsched::curr, XNBREAK, XNDELAY, XNDORMANT, XNHELD, XNKICKED, xnpod_schedule(), XNREADY, XNRELAX, XNRMID, XNROBBED, XNROOT, XNSHADOW, XNSUSP, xnsynch_forget_sleeper(), XNTIMEO, xntimer_start(), and XNWAKEN.

Referenced by xnpod_abort_thread(), xnpod_handle_exception(), xnpod_init_thread(), xnpod_-stop_thread(), xnpod_wait_thread_period(), xnshadow_map(), xnshadow_relax(), xnsynch_-acquire(), and xnsynch_sleep_on().

5.16.2.23 int xnpod_unblock_thread (xnthread_t * *thread*)

Unblock a thread.

Breaks the thread out of any wait it is currently in. This call removes the XNDELAY and XNPEND suspensive conditions previously put by xnpod_suspend_thread() on the target thread. If all suspensive conditions are gone, the thread is left in a READY state at which point it becomes eligible anew for scheduling.

Parameters

thread The descriptor address of the unblocked thread.

This call neither releases the thread from the XNSUSP, XNRELAX, XNDORMANT or XNHELD suspensive conditions.

When the thread resumes execution, the XNBREAK bit is set in the unblocked thread's information mask. Unblocking a non-blocked thread is perfectly harmless.

Returns

non-zero is returned if the thread was actually unblocked from a pending wait state, 0 otherwise.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task

Rescheduling: never.

References XNBREAK, XNDELAY, XNPEND, and xnpod_resume_thread().

Referenced by __xnpod_reset_thread(), pthread_cancel(), and xnpod_delete_thread().

5.16.2.24 int xnpod_wait_thread_period (unsigned long * *overruns_r*)

Wait for the next periodic release point.

Make the current thread wait for the next periodic release point in the processor time line.

Parameters

overruns_r If non-NULL, *overruns_r* must be a pointer to a memory location which will be written with the count of pending overruns. This value is copied only when xnpod_wait_thread_period() returns -ETIMEDOUT or success; the memory location remains unmodified otherwise. If NULL, this count will never be copied back.

Returns

0 is returned upon success; if *overruns_r* is valid, zero is copied to the pointed memory location. Otherwise:

- -EWOULDBLOCK is returned if xnpod_set_thread_periodic() has not previously been called for the calling thread.
- -EINTR is returned if xnpod_unblock_thread() has been called for the waiting thread before the next periodic release point has been reached. In this case, the overrun counter is reset too.
- -ETIMEDOUT is returned if the timer has overrun, which indicates that one or more previous release points have been missed by the calling thread. If *overruns_r* is valid, the count of pending overruns is copied to the pointed memory location.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task

Rescheduling: always, unless the current release point has already been reached. In the latter case, the current thread immediately returns from this service without being delayed.

References XNBREAK, XNDELAY, xnpod_suspend_thread(), and xntimer_get_overruns().

5.16.2.25 void xnpod_welcome_thread (xnthread_t * *thread*, int *imask*)

Thread prologue.

For internal use only.

This internal routine is called on behalf of a (re)starting thread's prologue before the user entry point is invoked. This call is reserved for internal housekeeping chores and cannot be inlined.

Entered with nklock locked, irqs off.

References XNLOCK, and xnpod_dispatch_signals().

5.17 Registry services.

Files

- file registry.h

  This file is part of the Xenomai project.

- file registry.c

  This file is part of the Xenomai project.

Functions

- int xnregistry_enter (const char *key, void *objaddr, xnhandle_t *phandle, struct xnpnode *pnode)

  Register a real-time object.

- int xnregistry_bind (const char *key, xnticks_t timeout, int timeout_mode, xnhandle_t *phandle)

  Bind to a real-time object.

- int xnregistry_remove (xnhandle_t handle)

  Forcibly unregister a real-time object.

- int xnregistry_remove_safe (xnhandle_t handle, xnticks_t timeout)

  Unregister an idle real-time object.

- void * xnregistry_get (xnhandle_t handle)

  Find and lock a real-time object into the registry.

- u_long xnregistry_put (xnhandle_t handle)

  Unlock a real-time object from the registry.

- void * xnregistry_fetch (xnhandle_t handle)

  Find a real-time object into the registry.

5.17.1 Detailed Description

The registry provides a mean to index real-time object descriptors created by Xenomai skins on unique alphanumeric keys. When labeled this way, a real-time object is globally exported; it can be searched for, and its descriptor returned to the caller for further use; the latter operation is called a "binding". When no object has been registered under the given name yet, the registry can be asked to set up a rendez-vous, blocking the caller until the object is eventually registered.

5.17.2 Function Documentation

5.17.2.1 int xnregistry_bind (const char * *key*, xnticks_t *timeout*, int *timeout_mode*, xnhandle_t * *phandle*)

Bind to a real-time object.

This service retrieves the registry handle of a given object identified by its key. Unless otherwise specified, this service will block the caller if the object is not registered yet, waiting for such registration to occur.

Parameters

key A valid NULL-terminated string which identifies the object to bind to.

timeout The timeout which may be used to limit the time the thread wait for the object to be registered. This value is a wait time given as a count of nanoseconds. It can either be relative, absolute monotonic (XN_ABSOLUTE), or absolute adjustable (XN_REALTIME) depending on *timeout_mode*. Passing XN_INFINITE **and** setting *timeout_mode* to XN_-RELATIVE specifies an unbounded wait. Passing XN_NONBLOCK causes the service to return immediately without waiting if the object is not registered on entry. All other values are used as a wait limit.

timeout_mode The mode of the *timeout* parameter. It can either be set to XN_RELATIVE, XN_ABSOLUTE, or XN_REALTIME (see also xntimer_start()).

phandle A pointer to a memory location which will be written upon success with the generic handle defined by the registry for the retrieved object. Contents of this memory is undefined upon failure.

Returns

0 is returned upon success. Otherwise:

- -EINVAL is returned if *key* is NULL.
- -EINTR is returned if xnpod_unblock_thread() has been called for the waiting thread before the retrieval has completed.
- -EWOULDBLOCK is returned if *timeout* is equal to XN_NONBLOCK and the searched object is not registered on entry. As a special exception, this error is also returned if this service should block, but was called from a context which cannot sleep (e.g. interrupt, non-realtime or scheduler locked).
- -ETIMEDOUT is returned if the object cannot be retrieved within the specified amount of time.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine only if *timeout* is equal to XN_NONBLOCK.
- Kernel-based thread.

Rescheduling: always unless the request is immediately satisfied or *timeout* specifies a non-blocking operation.

References XNBREAK, xnsynch_sleep_on(), and XNTIMEO.

5.17.2.2 int xnregistry_enter (const char * *key*, void * *objaddr*, xnhandle_t * *phandle*, struct xnpnode * *pnode*)

Register a real-time object.

This service allocates a new registry slot for an associated object, and indexes it by an alphanumeric key for later retrieval.

Parameters

- ***key*** A valid NULL-terminated string by which the object will be indexed and later retrieved in the registry. Since it is assumed that such key is stored into the registered object, it will *not* be copied but only kept by reference in the registry. Pass an empty string if the object shall only occupy a registry slot for handle-based lookups.
- ***objaddr*** An opaque pointer to the object to index by *key*.
- ***phandle*** A pointer to a generic handle defined by the registry which will uniquely identify the indexed object, until the latter is unregistered using the xnregistry_remove() service.
- ***pnode*** A pointer to an optional /proc node class descriptor. This structure provides the information needed to export all objects from the given class through the /proc filesystem, under the /proc/xenomai/registry entry. Passing NULL indicates that no /proc support is available for the newly registered object.

Returns

0 is returned upon success. Otherwise:

- -EINVAL is returned if *objaddr* are NULL, or if *key* constains an invalid '/' character.
- -ENOMEM is returned if the system fails to get enough dynamic memory from the global real-time heap in order to register the object.
- -EEXIST is returned if the *key* is already in use.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based thread

Rescheduling: possible.

References xnpod_schedule(), and xnsynch_init().

5.17.2.3 void* xnregistry_fetch (xnhandle_t *handle*)

Find a real-time object into the registry.

This service retrieves an object from its handle into the registry and returns the memory address of its descriptor.

Parameters

handle The generic handle of the object to fetch. If XNOBJECT_SELF is passed, the object is the calling Xenomai thread.

Returns

The memory address of the object's descriptor is returned on success. Otherwise, NULL is returned if *handle* does not reference a registered object, or if *handle* is equal to XNOBJECT_-SELF but the current context is not a real-time thread.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine only if *handle* is different from XNOBJECT_SELF.
- Kernel-based thread

Rescheduling: never.

5.17.2.4 void* xnregistry_get (xnhandle_t *handle*)

Find and lock a real-time object into the registry.

This service retrieves an object from its handle into the registry and prevents it removal atomically. A locking count is tracked, so that xnregistry_get() and xnregistry_put() must be used in pair.

Parameters

handle The generic handle of the object to find and lock. If XNOBJECT_SELF is passed, the object is the calling Xenomai thread.

Returns

The memory address of the object's descriptor is returned on success. Otherwise, NULL is returned if *handle* does not reference a registered object, or if *handle* is equal to XNOBJECT_-SELF but the current context is not a real-time thread.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine only if *handle* is different from XNOBJECT_SELF.
- Kernel-based thread.

Rescheduling: never.

5.17.2.5 u_long xnregistry_put (xnhandle_t *handle*)

Unlock a real-time object from the registry.

This service decrements the lock count of a registered object previously locked by a call to xnregistry_get(). The object is actually unlocked from the registry when the locking count falls down to zero, thus waking up any thread currently blocked on xnregistry_remove() for unregistering it.

Parameters

handle The generic handle of the object to unlock. If XNOBJECT_SELF is passed, the object is the calling Xenomai thread.

Returns

The decremented lock count is returned upon success. Zero is also returned if *handle* does not reference a registered object, or if *handle* is equal to XNOBJECT_SELF but the current context is not a real-time thread.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine only if *handle* is different from XNOBJECT_SELF.
- Kernel-based thread

Rescheduling: possible if the lock count falls down to zero and some thread is currently waiting for the object to be unlocked.

References xnpod_schedule(), and xnsynch_flush().

5.17.2.6 int xnregistry_remove (xnhandle_t *handle*)

Forcibly unregister a real-time object.

This service forcibly removes an object from the registry. The removal is performed regardless of the current object's locking status.

Parameters

handle The generic handle of the object to remove.

Returns

0 is returned upon success. Otherwise:

- -ESRCH is returned if *handle* does not reference a registered object.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based thread

Rescheduling: never.

Referenced by xnregistry_remove_safe().

5.17.2.7 int xnregistry_remove_safe (xnhandle_t *handle*, xnticks_t *timeout*)

Unregister an idle real-time object.

This service removes an object from the registry. The caller might sleep as a result of waiting for the target object to be unlocked prior to the removal (see xnregistry_put()).

Parameters

handle The generic handle of the object to remove.

timeout If the object is locked on entry, *param* gives the number of nanoseconds to wait for the unlocking to occur. Passing XN_INFINITE causes the caller to block indefinitely until the object is unlocked. Passing XN_NONBLOCK causes the service to return immediately without waiting if the object is locked on entry.

Returns

0 is returned upon success. Otherwise:

- -ESRCH is returned if *handle* does not reference a registered object.
- -EWOULDBLOCK is returned if *timeout* is equal to XN_NONBLOCK and the object is locked on entry.
- -EBUSY is returned if *handle* refers to a locked object and the caller could not sleep until it is unlocked.
- -ETIMEDOUT is returned if the object cannot be removed within the specified amount of time.
- -EINTR is returned if xnpod_unblock_thread() has been called for the calling thread waiting for the object to be unlocked.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine only if *timeout* is equal to XN_NONBLOCK.
- Kernel-based thread.

Rescheduling: possible if the object to remove is currently locked and the calling context can sleep.

References XNBREAK, xnregistry_remove(), xnsynch_sleep_on(), and XNTIMEO.

5.18 File descriptors events multiplexing services.

Files

- file select.h

  file descriptors events multiplexing header.

- file select.c

  file descriptors events multiplexing.

Functions

- void xnselect_init (struct xnselect *select_block)

  Initialize a struct xnselect structure.

- int xnselect_bind (struct xnselect *select_block, struct xnselect_binding *binding, struct xnselector *selector, unsigned type, unsigned index, unsigned state)

  Bind a file descriptor (represented by its xnselect structure) to a selector block.

- static int xnselect_signal (struct xnselect *select_block, unsigned state)

  Signal a file descriptor state change.

- void xnselect_destroy (struct xnselect *select_block)

  Destroy the xnselect structure associated with a file descriptor.

- int xnselector_init (struct xnselector *selector)

  Initialize a selector structure.

- int xnselect (struct xnselector *selector, fd_set *out_fds[XNSELECT_MAX_TYPES], fd_set *in_fds[XNSELECT_MAX_TYPES], int nfds, xnticks_t timeout, xntmode_t timeout_mode)

  Check the state of a number of file descriptors, wait for a state change if no descriptor is ready.

- void xnselector_destroy (struct xnselector *selector)

  Destroy a selector block.

5.18.1 Detailed Description

File descriptors events multiplexing services.

This module implements the services needed for implementing the posix "select" service, or any other events multiplexing services.

Following the implementation of the posix select service, this module defines three types of events:

- *XNSELECT_READ* meaning that a file descriptor is ready for reading;
- *XNSELECT_WRITE* meaning that a file descriptor is ready for writing;

- *XNSELECT_EXCEPT* meaning that a file descriptor received an exceptional event.

It works by defining two structures:

- a *struct xnselect* structure, which should be added to every file descriptor for every event type (read, write, or except);
- a *struct xnselector* structure, the selection structure, passed by the thread calling the xnselect service, where this service does all its housekeeping.

5.18.2 Function Documentation

5.18.2.1 int xnselect (struct xnselector * *selector*, fd_set * *out_fds[XNSELECT_MAX_TYPES]*, fd_set * *in_fds[XNSELECT_MAX_TYPES]*, int *nfds*, xnticks_t *timeout*, xntmode_t *timeout_mode*)

Check the state of a number of file descriptors, wait for a state change if no descriptor is ready.

Parameters

selector structure to check for pending events

out_fds The set of descriptors with pending events if a strictly positive number is returned, or the set of descriptors not yet bound if -ECHRNG is returned;

in_fds the set of descriptors which events should be checked

nfds the highest-numbered descriptor in any of the *in_fds* sets, plus 1;

timeout the timeout, whose meaning depends on *timeout_mode*, note that xnselect() pass *timeout* and *timeout_mode* unchanged to xnsynch_sleep_on, so passing a relative value different from XN_INFINITE as a timeout with *timeout_mode* set to XN_RELATIVE, will cause a longer sleep than expected if the sleep is interrupted.

timeout_mode the mode of *timeout*.

Return values

-EINVAL if *nfds* is negative;

-ECHRNG if some of the descriptors passed in *in_fds* have not yet been registered with xnselect_bind(), *out_fds* contains the set of such descriptors;

-EINTR if *xnselect* was interrupted while waiting;

0 in case of timeout.

the number of file descriptors having received an event.

References XNBREAK, xnsynch_sleep_on(), and XNTIMEO.

5.18.2.2 int xnselect_bind (struct xnselect * *select_block*, struct xnselect_binding * *binding*, struct xnselector * *selector*, unsigned *type*, unsigned *index*, unsigned *state*)

Bind a file descriptor (represented by its *xnselect* structure) to a selector block.

Parameters

select_block pointer to the *struct xnselect* to be bound;

binding pointer to a newly allocated (using xnmalloc) *struct xnselect_binding*;

selector pointer to the selector structure;

type type of events (*XNSELECT_READ*, *XNSELECT_WRITE*, or *XNSELECT_EXCEPT*);

index index of the file descriptor (represented by *select_block*) in the bit fields used by the *selector* structure;

state current state of the file descriptor>.

select_block must have been initialized with xnselect_init(), the *xnselector* structure must have been initialized with xnselector_init(), *binding* may be uninitialized.

This service must be called with nklock locked, irqs off. For this reason, the *binding* parameter must have been allocated by the caller outside the locking section.

Return values

-EINVAL if *type* or *index* is invalid;

0 otherwise.

References xnpod_schedule().

Referenced by rtdm_event_select_bind(), and rtdm_sem_select_bind().

5.18.2.3 void xnselect_destroy (struct xnselect * *select_block*)

Destroy the *xnselect* structure associated with a file descriptor.

Any binding with a *xnselector* block is destroyed.

Parameters

select_block pointer to the *xnselect* structure associated with a file descriptor

References xnpod_schedule().

5.18.2.4 void xnselect_init (struct xnselect * *select_block*)

Initialize a *struct xnselect* structure.

This service must be called to initialize a *struct xnselect* structure before it is bound to a selector by the means of xnselect_bind().

Parameters

select_block pointer to the xnselect structure to be initialized

Referenced by rtdm_event_init(), and rtdm_sem_init().

5.18.2.5 static int xnselect_signal (struct xnselect * *select_block*, unsigned *state*) [inline, static]

Signal a file descriptor state change.

Parameters

select_block pointer to an *xnselect* structure representing the file descriptor whose state changed;

state new value of the state.

Return values

1 if rescheduling is needed;

0 otherwise.

Referenced by rtdm_event_clear(), rtdm_event_signal(), rtdm_event_timedwait(), rtdm_sem_-timeddown(), and rtdm_sem_up().

5.18.2.6 void xnselector_destroy (struct xnselector * *selector*)

Destroy a selector block.

All bindings with file descriptor are destroyed.

Parameters

selector the selector block to be destroyed

Referenced by xnpod_delete_thread().

5.18.2.7 int xnselector_init (struct xnselector * *selector*)

Initialize a selector structure.

Parameters

selector The selector structure to be initialized.

Return values

0

References xnsynch_init().

5.19 Real-time shadow services.

Files

- file shadow.c

  Real-time shadow services.

Functions

- int xnshadow_harden (void)

    Migrate a Linux task to the Xenomai domain.

- void xnshadow_relax (int notify, int reason)

    Switch a shadow thread back to the Linux domain.

- int xnshadow_map (xnthread_t *thread, xncompletion_t __user *u_completion, unsigned long __user *u_window_offset)

    Create a shadow thread context.

- xnshadow_ppd_t * xnshadow_ppd_get (unsigned int muxid)

    Return the per-process data attached to the calling process.

5.19.1 Detailed Description

Real-time shadow services.

5.19.2 Function Documentation

5.19.2.1 int xnshadow_harden (void)

Migrate a Linux task to the Xenomai domain.

For internal use only.

This service causes the transition of "current" from the Linux domain to Xenomai. The shadow will resume in the Xenomai domain as returning from schedule().

Environments:

This service can be called from:

- User-space thread operating in secondary (i.e. relaxed) mode.

Rescheduling: always.

References XNDEBUG, xnpod_dispatch_signals(), XNRELAX, and xnshadow_relax().

Referenced by xnshadow_map().

5.19.2.2 int xnshadow_map (xnthread_t * *thread*, xncompletion_t __user * *u_completion*, unsigned long __user * *u_window_offset*)

Create a shadow thread context.

For internal use only.

This call maps a nucleus thread to the "current" Linux task. The priority and scheduling class of the underlying Linux task are not affected; it is assumed that the interface library did set them appropriately before issuing the shadow mapping request.

Parameters

thread The descriptor address of the new shadow thread to be mapped to "current". This descriptor must have been previously initialized by a call to xnpod_init_thread().

u_completion is the address of an optional completion descriptor aimed at synchronizing our parent thread with us. If non-NULL, the information xnshadow_map() will store into the completion block will be later used to wake up the parent thread when the current shadow has been initialized. In the latter case, the new shadow thread is left in a dormant state (XNDORMANT) after its creation, leading to the suspension of "current" in the Linux domain, only processing signals. Otherwise, the shadow thread is immediately started and "current" immediately resumes in the Xenomai domain from this service.

u_window_offset will receive the offset of the per-thread "u_window" structure in the process shared heap associated to *thread*. This structure reflects thread state information visible from userland through a shared memory window.

Returns

0 is returned on success. Otherwise:

- -ERESTARTSYS is returned if the current Linux task has received a signal, thus preventing the final migration to the Xenomai domain (i.e. in order to process the signal in the Linux domain). This error should not be considered as fatal.

- -EPERM is returned if the shadow thread has been killed before the current task had a chance to return to the caller. In such a case, the real-time mapping operation has failed globally, and no Xenomai resource remains attached to it.

- -EINVAL is returned if the thread control block does not bear the XNSHADOW bit.

- -EBUSY is returned if either the current Linux task or the associated shadow thread is already involved in a shadow mapping.

Environments:

This service can be called from:

- Regular user-space process.

Rescheduling: always.

References xnheap_alloc(), XNMAPPED, XNOTHER, xnpod_start_thread(), xnpod_suspend_-thread(), XNPRIOSET, XNRELAX, XNSHADOW, and xnshadow_harden().

5.19.2.3 xnshadow_ppd_t* xnshadow_ppd_get (unsigned int *muxid*)

Return the per-process data attached to the calling process.

This service returns the per-process data attached to the calling process for the skin whose muxid is *muxid*. It must be called with nklock locked, irqs off.

See xnshadow_register_interface() documentation for information on the way to attach a per-process data to a process.

Parameters

muxid the skin muxid.

Returns

the per-process data if the current context is a user-space process;
NULL otherwise.

5.19.2.4 void xnshadow_relax (int *notify,* int *reason*)

Switch a shadow thread back to the Linux domain.

For internal use only.

This service yields the control of the running shadow back to Linux. This is obtained by suspending the shadow and scheduling a wake up call for the mated user task inside the Linux domain. The Linux task will resume on return from xnpod_suspend_thread() on behalf of the root thread.

Parameters

notify A boolean flag indicating whether threads monitored from secondary mode switches should be sent a SIGDEBUG signal. For instance, some internal operations like task exit should not trigger such signal.

reason The reason to report along with the SIGDEBUG signal.

Environments:

This service can be called from:

- User-space thread operating in primary (i.e. harden) mode.

Rescheduling: always.

Note

"current" is valid here since the shadow runs with the properties of the Linux task.

References XNAFFSET, xnpod_suspend_thread(), XNPRIOSET, XNRELAX, XNROOT, and XNTRAPSW.

Referenced by xnpod_handle_exception(), and xnshadow_harden().

5.20 Thread synchronization services.

Files

- file synch.c

  Thread synchronization services.

Functions

- void xnsynch_init (struct xnsynch *synch, xnflags_t flags, xnarch_atomic_t *fastlock)

  Initialize a synchronization object.

- xnflags_t xnsynch_sleep_on (struct xnsynch *synch, xnticks_t timeout, xntmode_t timeout_-mode)

  Sleep on an ownerless synchronization object.

- struct xnthread * xnsynch_wakeup_one_sleeper (struct xnsynch *synch)

  Give the resource ownership to the next waiting thread.

- struct xnpholder * xnsynch_wakeup_this_sleeper (struct xnsynch *synch, struct xnpholder *holder)

  Give the resource ownership to a given waiting thread.

- xnflags_t xnsynch_acquire (struct xnsynch *synch, xnticks_t timeout, xntmode_t timeout_-mode)

  Acquire the ownership of a synchronization object.

- static void xnsynch_clear_boost (struct xnsynch *synch, struct xnthread *owner)

  Clear the priority boost.

- void xnsynch_requeue_sleeper (struct xnthread *thread)

  Change a sleeper's priority.

- struct xnthread * xnsynch_peek_pendq (struct xnsynch *synch)

  Access the thread leading a synch object wait queue.

- int xnsynch_flush (struct xnsynch *synch, xnflags_t reason)

  Unblock all waiters pending on a resource.

- void xnsynch_forget_sleeper (struct xnthread *thread)

  Abort a wait for a resource.

- void xnsynch_release_all_ownerships (struct xnthread *thread)

  Release all ownerships.

- static struct xnthread * xnsynch_release (struct xnsynch *synch, struct xnthread *thread)

  Give the resource ownership to the next waiting thread.

5.20.1 Detailed Description

Thread synchronization services.

5.20.2 Function Documentation

5.20.2.1 xnflags_t xnsynch_acquire (struct xnsynch * *synch*, xnticks_t *timeout*, xntmode_t *timeout_mode*)

Acquire the ownership of a synchronization object.

This service should be called by upper interfaces wanting the current thread to acquire the ownership of the given resource. If the resource is already assigned to a thread, the caller is suspended.

This service must be used only with synchronization objects that track ownership (XNSYNCH_-OWNER set.

Parameters

synch The descriptor address of the synchronization object to acquire.

timeout The timeout which may be used to limit the time the thread pends on the resource. This value is a wait time given as a count of nanoseconds. It can either be relative, absolute monotonic, or absolute adjustable depending on *timeout_mode*. Passing XN_-INFINITE **and** setting *mode* to XN_RELATIVE specifies an unbounded wait. All other values are used to initialize a watchdog timer.

timeout_mode The mode of the *timeout* parameter. It can either be set to XN_RELATIVE, XN_ABSOLUTE, or XN_REALTIME (see also xntimer_start()).

Returns

A bitmask which may include zero or one information bit among XNRMID, XNTIMEO and XNBREAK, which should be tested by the caller, for detecting respectively: object deletion, timeout or signal/unblock conditions which might have happened while waiting.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task

Rescheduling: possible.

References XNBOOST, XNBREAK, XNOTHER, XNPEND, xnpod_suspend_thread(), XNRMID, XNROBBED, XNTIMEO, and XNWAKEN.

Referenced by rtdm_mutex_timedlock().

5.20.2.2 void xnsynch_clear_boost (struct xnsynch * *synch*, struct xnthread * *owner*) [static]

Clear the priority boost.

For internal use only.

This service is called internally whenever a synchronization object is not claimed anymore by sleepers to reset the object owner's priority to its initial level.

Parameters

synch The descriptor address of the synchronization object.

owner The descriptor address of the thread which currently owns the synchronization object.

Note

This routine must be entered nklock locked, interrupts off.

References XNBOOST.

Referenced by xnsynch_flush(), and xnsynch_forget_sleeper().

5.20.2.3 int xnsynch_flush (struct xnsynch * *synch*, xnflags_t *reason*)

Unblock all waiters pending on a resource.

This service atomically releases all threads which currently sleep on a given resource.

This service should be called by upper interfaces under circumstances requiring that the pending queue of a given resource is cleared, such as before the resource is deleted.

Parameters

synch The descriptor address of the synchronization object to be flushed.

reason Some flags to set in the information mask of every unblocked thread. Zero is an acceptable value. The following bits are pre-defined by the nucleus:

- XNRMID should be set to indicate that the synchronization object is about to be destroyed (see xnpod_resume_thread()).

- XNBREAK should be set to indicate that the wait has been forcibly interrupted (see xnpod_-unblock_thread()).

Returns

XNSYNCH_RESCHED is returned if at least one thread is unblocked, which means the caller should invoke xnpod_schedule() for applying the new scheduling state. Otherwise, XNSYNCH_DONE is returned.

Side-effects:

- The effective priority of the previous resource owner might be lowered to its base priority value as a consequence of the priority inheritance boost being cleared.

- The synchronization object is no more owned by any thread.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task

Rescheduling: never.

References XNPEND, xnpod_resume_thread(), and xnsynch_clear_boost().

Referenced by rtdm_event_signal(), and xnregistry_put().

5.20.2.4 void xnsynch_forget_sleeper (struct xnthread * *thread*)

Abort a wait for a resource.

For internal use only.

Performs all the necessary housekeeping chores to stop a thread from waiting on a given synchronization object.

Parameters

thread The descriptor address of the affected thread.

When the trace support is enabled (i.e. MVM), the idle state is posted to the synchronization object's state diagram (if any) whenever no thread remains blocked on it. The real-time interfaces must ensure that such condition (i.e. EMPTY/IDLE) is mapped to state #0.

Note

This routine must be entered nklock locked, interrupts off.

References XNPEND, and xnsynch_clear_boost().

Referenced by xnpod_delete_thread(), xnpod_resume_thread(), and xnpod_suspend_thread().

5.20.2.5 void xnsynch_init (struct xnsynch * *synch*, xnflags_t *flags*, xnarch_atomic_t * *fastlock*)

Initialize a synchronization object.

Initializes a new specialized object which can subsequently be used to synchronize real-time activities. The Xenomai nucleus provides a basic synchronization object which can be used to build higher resource objects. Nucleus threads can wait for and signal such objects in order to synchronize their activities.

This object has built-in support for priority inheritance.

Parameters

synch The address of a synchronization object descriptor the nucleus will use to store the object-specific data. This descriptor must always be valid while the object is active therefore it must be allocated in permanent memory.

flags A set of creation flags affecting the operation. The valid flags are:

- XNSYNCH_PRIO causes the threads waiting for the resource to pend in priority order. Otherwise, FIFO ordering is used (XNSYNCH_FIFO).

- XNSYNCH_OWNER indicates that the synchronization object shall track its owning thread (required if XNSYNCH_PIP is selected). Note that setting this flag implies the use xnsynch_acquire and xnsynch_release instead of xnsynch_sleep_on and xnsynch_wakeup_one_sleeper/xnsynch_wakeup_this_sleeper.

- XNSYNCH_PIP causes the priority inheritance mechanism to be automatically activated when a priority inversion is detected among threads using this object. Otherwise, no priority inheritance takes place upon priority inversion (XNSYNCH_NOPIP).

- XNSYNCH_DREORD (Disable REORDering) tells the nucleus that the wait queue should not be reordered whenever the priority of a blocked thread it holds is changed. If this flag is not specified, changing the priority of a blocked thread using xnpod_set_thread_schedparam() will cause this object's wait queue to be reordered according to the new priority level, provided the synchronization object makes the waiters wait by priority order on the awaited resource (XNSYNCH_PRIO).

Parameters

fastlock Address of the fast lock word to be associated with the synchronization object. If NULL is passed or XNSYNCH_OWNER is not set, fast-lock support is disabled.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task

Rescheduling: never.

Referenced by pthread_cond_init(), pthread_create(), rtdm_event_init(), rtdm_mutex_init(), rtdm_sem_init(), xnregistry_enter(), and xnselector_init().

5.20.2.6 struct xnthread* xnsynch_peek_pendq (struct xnsynch * *synch*) [read]

Access the thread leading a synch object wait queue.

This services returns the descriptor address of to the thread leading a synchronization object wait queue.

Parameters

synch The descriptor address of the target synchronization object.

Returns

The descriptor address of the unblocked thread.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task

Rescheduling: never.

5.20.2.7 struct xnthread * xnsynch_release (struct xnsynch * *synch*, struct xnthread * *owner*) [static, read]

Give the resource ownership to the next waiting thread.

This service releases the ownership of the given synchronization object. The thread which is currently leading the object's pending list, if any, is unblocked from its pending state. However, no reschedule is performed.

This service must be used only with synchronization objects that track ownership (XNSYNCH_-OWNER set).

Parameters

synch The descriptor address of the synchronization object whose ownership is changed.

owner The descriptor address of the current owner.

Returns

The descriptor address of the unblocked thread.

Side-effects:

- The effective priority of the previous resource owner might be lowered to its base priority value as a consequence of the priority inheritance boost being cleared.
- The synchronization object ownership is transfered to the unblocked thread.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task

Rescheduling: never.

References XNOTHER.

Referenced by xnsynch_release_all_ownerships().

5.20.2.8 void xnsynch_release_all_ownerships (struct xnthread * *thread*)

Release all ownerships.

For internal use only.

This call is used internally to release all the ownerships obtained by a thread on synchronization objects. This routine must be entered interrupts off.

Parameters

thread The descriptor address of the affected thread.

Note

This routine must be entered nklock locked, interrupts off.

References xnsynch_release().

Referenced by __xnpod_reset_thread(), and xnpod_delete_thread().

5.20.2.9 void xnsynch_requeue_sleeper (struct xnthread * *thread*)

Change a sleeper's priority.

For internal use only.

This service is used by the PIP code to update the pending priority of a sleeping thread.

Parameters

thread The descriptor address of the affected thread.

Note

This routine must be entered nklock locked, interrupts off.

References XNBOOST.

5.20.2.10 xnflags_t xnsynch_sleep_on (struct xnsynch * *synch*, xnticks_t *timeout*, xntmode_t *timeout_mode*)

Sleep on an ownerless synchronization object.

Makes the calling thread sleep on the specified synchronization object, waiting for it to be signaled.

This service should be called by upper interfaces wanting the current thread to pend on the given resource. It must not be used with synchronization objects that are supposed to track ownership (XNSYNCH_OWNER).

Parameters

synch The descriptor address of the synchronization object to sleep on.

timeout The timeout which may be used to limit the time the thread pends on the resource. This value is a wait time given as a count of nanoseconds. It can either be relative, absolute monotonic, or absolute adjustable depending on *timeout_mode*. Passing XN_-INFINITE **and** setting *mode* to XN_RELATIVE specifies an unbounded wait. All other values are used to initialize a watchdog timer.

timeout_mode The mode of the *timeout* parameter. It can either be set to XN_RELATIVE, XN_ABSOLUTE, or XN_REALTIME (see also xntimer_start()).

Returns

A bitmask which may include zero or one information bit among XNRMID, XNTIMEO and XNBREAK, which should be tested by the caller, for detecting respectively: object deletion, timeout or signal/unblock conditions which might have happened while waiting.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task

Rescheduling: always.

References XNBREAK, XNPEND, xnpod_suspend_thread(), XNRMID, and XNTIMEO.

Referenced by rtdm_event_timedwait(), rtdm_sem_timeddown(), xnregistry_bind(), xnregistry_-remove_safe(), and xnselect().

5.20.2.11 struct xnthread* xnsynch_wakeup_one_sleeper (struct xnsynch * *synch*) `[read]`

Give the resource ownership to the next waiting thread.

This service wakes up the thread which is currently leading the synchronization object's pending list. The sleeping thread is unblocked from its pending state, but no reschedule is performed.

This service should be called by upper interfaces wanting to signal the given resource so that a single waiter is resumed. It must not be used with synchronization objects that are supposed to track ownership (XNSYNCH_OWNER not set).

Parameters

synch The descriptor address of the synchronization object whose ownership is changed.

Returns

The descriptor address of the unblocked thread.

Side-effects:

- The effective priority of the previous resource owner might be lowered to its base priority value as a consequence of the priority inheritance boost being cleared.
- The synchronization object ownership is transfered to the unblocked thread.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task

Rescheduling: never.

References XNPEND, and xnpod_resume_thread().

Referenced by rtdm_sem_up().

5.20.2.12 struct xnpholder* xnsynch_wakeup_this_sleeper (struct xnsynch * *synch*, struct xnpholder * *holder*) [read]

Give the resource ownership to a given waiting thread.

This service wakes up a specific thread which is currently pending on the given synchronization object. The sleeping thread is unblocked from its pending state, but no reschedule is performed.

This service should be called by upper interfaces wanting to signal the given resource so that a specific waiter is resumed. It must not be used with synchronization objects that are supposed to track ownership (XNSYNCH_OWNER not set).

Parameters

synch The descriptor address of the synchronization object whose ownership is changed.

holder The link holder address of the thread to unblock (&thread->plink) which MUST be currently linked to the synchronization object's pending queue (i.e. synch->pendq).

Returns

The link address of the unblocked thread in the synchronization object's pending queue.

Side-effects:

- The effective priority of the previous resource owner might be lowered to its base priority value as a consequence of the priority inheritance boost being cleared.
- The synchronization object ownership is transfered to the unblocked thread.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task

Rescheduling: never.

References XNPEND, and xnpod_resume_thread().

5.21 Timer services.

Files

- file timer.h
- file timer.c

Functions

- int xntimer_start (xntimer_t *timer, xnticks_t value, xnticks_t interval, xntmode_t mode)

  Arm a timer.

- xnticks_t xntimer_get_date (xntimer_t *timer)

  Return the absolute expiration date.

- xnticks_t xntimer_get_timeout (xntimer_t *timer)

  Return the relative expiration date.

- xnticks_t xntimer_get_interval (xntimer_t *timer)

  Return the timer interval value.

- static void xntimer_stop (xntimer_t *timer)

  Disarm a timer.

- void xntimer_tick (void)

  Process a timer tick.

- void xntimer_init (xntimer_t *timer, void(*handler)(xntimer_t *timer))

  Initialize a timer object.

- void xntimer_destroy (xntimer_t *timer)

  Release a timer object.

- unsigned long xntimer_get_overruns (xntimer_t *timer, xnticks_t now)

  Get the count of overruns for the last tick.

- void xntimer_freeze (void)

  Freeze all timers (from every time bases).

5.21.1 Detailed Description

The Xenomai timer facility always operate the timer hardware in oneshot mode, regardless of the time base in effect. Periodic timing is obtained through a software emulation, using cascading timers.

The timer object stores time as a count of CPU ticks (e.g. TSC values).

5.21.2 Function Documentation

5.21.2.1 void xntimer_destroy (xntimer_t * *timer*)

Release a timer object.

Destroys a timer. After it has been destroyed, all resources associated with the timer have been released. The timer is automatically deactivated before deletion if active on entry.

Parameters

timer The address of a valid timer descriptor.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task

Rescheduling: never.

References xntimer_stop().

Referenced by rtdm_timer_destroy(), and xnpod_delete_thread().

5.21.2.2 void xntimer_freeze (void)

Freeze all timers (from every time bases).

For internal use only.

This routine deactivates all active timers atomically.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task

Rescheduling: never.

References xnsched::status.

Referenced by xnpod_disable_timesource().

5.21.2.3 xnticks_t xntimer_get_date (xntimer_t * *timer*)

Return the absolute expiration date.

Return the next expiration date of a timer as an absolute count of nanoseconds.

Parameters

timer The address of a valid timer descriptor.

Returns

The expiration date in nanoseconds. The special value XN_INFINITE is returned if *timer* is currently disabled.

Environments:

This service can be called from:

- Any kernel context.

Rescheduling: never.

5.21.2.4 xnticks_t xntimer_get_interval (xntimer_t * *timer*)

Return the timer interval value.

Return the timer interval value in nanoseconds.

Parameters

timer The address of a valid timer descriptor.

Returns

The duration of a period in nanoseconds. The special value XN_INFINITE is returned if *timer* is currently disabled or one shot.

Environments:

This service can be called from:

- Any kernel context.

Rescheduling: never.

5.21.2.5 unsigned long xntimer_get_overruns (xntimer_t * *timer*, xnticks_t *now*)

Get the count of overruns for the last tick.

This service returns the count of pending overruns for the last tick of a given timer, as measured by the difference between the expected expiry date of the timer and the date *now* passed as argument.

Parameters

timer The address of a valid timer descriptor.

now current date (in the monotonic time base)

Returns

the number of overruns of *timer* at date *now*

Referenced by xnpod_wait_thread_period().

5.21.2.6 xnticks_t xntimer_get_timeout (xntimer_t * *timer*)

Return the relative expiration date.

This call returns the count of nanoseconds remaining until the timer expires.

Parameters

timer The address of a valid timer descriptor.

Returns

The count of nanoseconds until expiry. The special value XN_INFINITE is returned if *timer* is currently disabled. It might happen that the timer expires when this service runs (even if the associated handler has not been fired yet); in such a case, 1 is returned.

Environments:

This service can be called from:

- Any kernel context.

Rescheduling: never.

5.21.2.7 void xntimer_init (xntimer_t * *timer,* void(*)(xntimer_t *timer) *handler*)

Initialize a timer object.

Creates a timer. When created, a timer is left disarmed; it must be started using xntimer_start() in order to be activated.

Parameters

timer The address of a timer descriptor the nucleus will use to store the object-specific data. This descriptor must always be valid while the object is active therefore it must be allocated in permanent memory.

handler The routine to call upon expiration of the timer.

There is no limitation on the number of timers which can be created/active concurrently.

Environments:

This service can be called from:

- Any kernel context.

Rescheduling: never.

Referenced by timer_create().

5.21.2.8 int xntimer_start (xntimer_t * *timer*, xnticks_t *value*, xnticks_t *interval*, xntmode_t *mode*)

Arm a timer.

Activates a timer so that the associated timeout handler will be fired after each expiration time. A timer can be either periodic or one-shot, depending on the reload value passed to this routine. The given timer must have been previously initialized.

Parameters

timer The address of a valid timer descriptor.

value The date of the initial timer shot, expressed in nanoseconds.

interval The reload value of the timer. It is a periodic interval value to be used for reprogramming the next timer shot, expressed in nanoseconds. If *interval* is equal to XN_INFINITE, the timer will not be reloaded after it has expired.

mode The timer mode. It can be XN_RELATIVE if *value* shall be interpreted as a relative date, XN_ABSOLUTE for an absolute date based on the monotonic clock of the related time base (as returned my xnclock_read_monotonic()), or XN_REALTIME if the absolute date is based on the adjustable real-time clock (obtained from xnclock_read()).

Returns

0 is returned upon success, or -ETIMEDOUT if an absolute date in the past has been given.

Environments:

This service can be called from:

- Any kernel context.

Rescheduling: never.

Note

Must be called with nklock held, IRQs off.

Referenced by rtdm_timer_start(), timer_settime(), xnpod_enable_timesource(), xnpod_set_-thread_periodic(), xnpod_set_thread_tslice(), and xnpod_suspend_thread().

5.21.2.9 int xntimer_stop (xntimer_t * *timer*) [inline, static]

Disarm a timer.

This service deactivates a timer previously armed using xntimer_start(). Once disarmed, the timer can be subsequently re-armed using the latter service.

Parameters

timer The address of a valid timer descriptor.

Environments:

This service can be called from:

- Any kernel context.

Rescheduling: never.

Note

Must be called with nklock held, IRQs off.

Referenced by rtdm_timer_stop(), timer_settime(), xnpod_resume_thread(), xnpod_set_thread_-periodic(), xnpod_set_thread_tslice(), and xntimer_destroy().

5.21.2.10 void xntimer_tick (void)

Process a timer tick.

For internal use only.

This routine informs all active timers that the clock has been updated by processing the outstanding timer list. Elapsed timer actions will be fired.

Environments:

This service can be called from:

- Interrupt service routine, nklock locked, interrupts off

Rescheduling: never.

References xnsched::htimer, xnsched::lflags, and xnsched::status.

5.22 Virtual file services

Data Structures

- struct xnvfile_lock_ops
  Vfile locking operations.
- struct xnvfile_regular_ops
  Regular vfile operation descriptor.
- struct xnvfile_regular_iterator
  Regular vfile iterator.
- struct xnvfile_snapshot_ops
  Snapshot vfile operation descriptor.
- struct xnvfile_rev_tag
  Snapshot revision tag.
- struct xnvfile_snapshot

Snapshot vfile descriptor.

- struct xnvfile_snapshot_iterator
  Snapshot-driven vfile iterator.

Files

- file vfile.h
  This file is part of the Xenomai project.

Functions

- int xnvfile_init_snapshot (const char *name, struct xnvfile_snapshot *vfile, struct xnvfile_directory *parent)
  Initialize a snapshot-driven vfile.
- int xnvfile_init_regular (const char *name, struct xnvfile_regular *vfile, struct xnvfile_directory *parent)
  Initialize a regular vfile.
- int xnvfile_init_dir (const char *name, struct xnvfile_directory *vdir, struct xnvfile_directory *parent)
  Initialize a virtual directory entry.
- int xnvfile_init_link (const char *from, const char *to, struct xnvfile_link *vlink, struct xnvfile_directory *parent)
  Initialize a virtual link entry.
- void xnvfile_destroy (struct xnvfile *vfile)
  Removes a virtual file entry.
- ssize_t xnvfile_get_blob (struct xnvfile_input *input, void *data, size_t size)
  Read in a data bulk written to the vfile.
- ssize_t xnvfile_get_string (struct xnvfile_input *input, char *s, size_t maxlen)
  Read in a C-string written to the vfile.
- ssize_t xnvfile_get_integer (struct xnvfile_input *input, long *valp)
  Evaluate the string written to the vfile as a long integer.

Variables

- struct xnvfile_directory nkvfroot
  Xenomai vfile root directory.
- struct xnvfile_directory nkvfroot
  Xenomai vfile root directory.

5.22.1 Detailed Description

Virtual files provide a mean to export Xenomai object states to user-space, based on common kernel interfaces. This encapsulation is aimed at:

- supporting consistent collection of very large record-based output, without encurring latency peaks for undergoing real-time activities.
- in the future, hiding discrepancies between linux kernel releases, regarding the proper way to export kernel object states to userland, either via the /proc interface or by any other mean.

This virtual file implementation offers record-based read support based on seq_files, single-buffer write support, directory and link handling, all visible from the /proc namespace.

The vfile support exposes four filesystem object types:

- snapshot-driven file (struct xnvfile_snapshot). This is commonly used to export real-time object states via the /proc filesystem. To minimize the latency involved in protecting the vfile routines from changes applied by real-time code on such objects, a snapshot of the data to output is first taken under proper locking, before the collected data is formatted and sent out in a lockless manner.

Because a large number of records may have to be output, the data collection phase is not strictly atomic as a whole, but only protected at record level. The vfile implementation can be notified of updates to the underlying data set, and restart the collection from scratch until the snapshot is fully consistent.

- regular sequential file (struct xnvfile_regular). This is basically an encapsulated sequential file object as available from the host kernel (i.e. seq_file), with a few additional features to make it more handy in a Xenomai environment, like implicit locking support and shortened declaration for simplest, single-record output.
- virtual link (struct xnvfile_link). This is a symbolic link feature integrated with the vfile semantics. The link target is computed dynamically at creation time from a user-given helper routine.
- virtual directory (struct xnvfile_directory). A directory object, which can be used to create a hierarchy for ordering a set of vfile objects.

5.22.2 Function Documentation

5.22.2.1 void xnvfile_destroy (struct xnvfile * *vfile*)

Removes a virtual file entry.

Parameters

vfile A pointer to the virtual file descriptor to remove.

References nkvfroot.

5.22.2.2 ssize_t xnvfile_get_blob (struct xnvfile_input * *input*, void * *data*, size_t *size*)

Read in a data bulk written to the vfile.

When writing to a vfile, the associated store() handler from the snapshot-driven vfile or regular vfile is called, with a single argument describing the input data. xnvfile_get_blob() retrieves this data as an untyped binary blob, and copies it back to the caller's buffer.

Parameters

input A pointer to the input descriptor passed to the store() handler.

data The address of the destination buffer to copy the input data to.

size The maximum number of bytes to copy to the destination buffer. If *size* is larger than the actual data size, the input is truncated to *size*.

Returns

The number of bytes read and copied to the destination buffer upon success. Otherwise, a negative error code is returned:

- -EFAULT indicates an invalid source buffer address.

Referenced by xnvfile_get_integer(), and xnvfile_get_string().

5.22.2.3 ssize_t xnvfile_get_integer (struct xnvfile_input * *input*, long * *valp*)

Evaluate the string written to the vfile as a long integer.

When writing to a vfile, the associated store() handler from the snapshot-driven vfile or regular vfile is called, with a single argument describing the input data. xnvfile_get_integer() retrieves and interprets this data as a long integer, and copies the resulting value back to *valp*.

The long integer can be expressed in decimal, octal or hexadecimal bases depending on the prefix found.

Parameters

input A pointer to the input descriptor passed to the store() handler.

valp The address of a long integer variable to receive the value.

Returns

The number of characters read while evaluating the input as a long integer upon success. Otherwise, a negative error code is returned:

- -EINVAL indicates a parse error on the input stream; the written text cannot be evaluated as a long integer.

- -EFAULT indicates an invalid source buffer address.

References xnvfile_get_blob().

5.22.2.4 ssize_t xnvfile_get_string (struct xnvfile_input * *input*, char * *s*, size_t *maxlen*)

Read in a C-string written to the vfile.

When writing to a vfile, the associated store() handler from the snapshot-driven vfile or regular vfile is called, with a single argument describing the input data. xnvfile_get_string() retrieves this data as a null-terminated character string, and copies it back to the caller's buffer.

Parameters

input A pointer to the input descriptor passed to the store() handler.

s The address of the destination string buffer to copy the input data to.

maxlen The maximum number of bytes to copy to the destination buffer, including the ending null character. If *maxlen* is larger than the actual string length, the input is truncated to *maxlen*.

Returns

The number of characters read and copied to the destination buffer upon success. Otherwise, a negative error code is returned:

- -EFAULT indicates an invalid source buffer address.

References xnvfile_get_blob().

5.22.2.5 int xnvfile_init_dir (const char * *name*, struct xnvfile_directory * *vdir*, struct xnvfile_directory * *parent*)

Initialize a virtual directory entry.

Parameters

name The name which should appear in the pseudo-filesystem, identifying the vdir entry.

vdir A pointer to the virtual directory descriptor to initialize.

parent A pointer to a virtual directory descriptor standing for the parent directory of the new vdir. If NULL, the /proc root directory will be used. /proc/xenomai is mapped on the globally available *nkvfroot* vdir.

Returns

0 is returned on success. Otherwise:

- -ENOMEM is returned if the virtual directory entry cannot be created in the /proc hierarchy.

5.22.2.6 int xnvfile_init_link (const char * *from*, const char * *to*, struct xnvfile_link * *vlink*, struct xnvfile_directory * *parent*)

Initialize a virtual link entry.

Parameters

from The name which should appear in the pseudo-filesystem, identifying the vlink entry.

to The target file name which should be referred to symbolically by *name*.

vlink A pointer to the virtual link descriptor to initialize.

parent A pointer to a virtual directory descriptor standing for the parent directory of the new vlink. If NULL, the /proc root directory will be used. /proc/xenomai is mapped on the globally available *nkvfroot* vdir.

Returns

0 is returned on success. Otherwise:

- -ENOMEM is returned if the virtual link entry cannot be created in the /proc hierarchy.

5.22.2.7 int xnvfile_init_regular (const char * *name*, struct xnvfile_regular * *vfile*, struct xnvfile_directory * *parent*)

Initialize a regular vfile.

Parameters

name The name which should appear in the pseudo-filesystem, identifying the vfile entry.

vfile A pointer to a vfile descriptor to initialize from. The following fields in this structure should be filled in prior to call this routine:

- .privsz is the size (in bytes) of the private data area to be reserved in the vfile iterator. A NULL value indicates that no private area should be reserved.

- entry.lockops is a pointer to a locking descriptor", defining the lock and unlock operations for the vfile. This pointer may be left to NULL, in which case no locking will be applied.

- .ops is a pointer to an operation descriptor.

Parameters

parent A pointer to a virtual directory descriptor; the vfile entry will be created into this directory. If NULL, the /proc root directory will be used. /proc/xenomai is mapped on the globally available *nkvfroot* vdir.

Returns

0 is returned on success. Otherwise:

- -ENOMEM is returned if the virtual file entry cannot be created in the /proc hierarchy.

5.22.2.8 int xnvfile_init_snapshot (const char * *name*, struct xnvfile_snapshot * *vfile*, struct xnvfile_directory * *parent*)

Initialize a snapshot-driven vfile.

Parameters

name The name which should appear in the pseudo-filesystem, identifying the vfile entry.

vfile A pointer to a vfile descriptor to initialize from. The following fields in this structure should be filled in prior to call this routine:

- .privsz is the size (in bytes) of the private data area to be reserved in the vfile iterator. A NULL value indicates that no private area should be reserved.

- .datasz is the size (in bytes) of a single record to be collected by the next() handler from the operation descriptor.

- .tag is a pointer to a mandatory vfile revision tag structure (struct xnvfile_rev_tag). This tag will be monitored for changes by the vfile core while collecting data to output, so that any update detected will cause the current snapshot data to be dropped, and the collection to restart from the beginning. To this end, any change to the data which may be part of the collected records, should also invoke xnvfile_touch() on the associated tag.

- entry.lockops is a pointer to a locking descriptor", defining the lock and unlock operations for the vfile. This pointer may be left to NULL, in which case the operations on the nucleus lock (i.e. nklock) will be used internally around calls to data collection handlers (see operation descriptor).

- .ops is a pointer to an operation descriptor.

Parameters

parent A pointer to a virtual directory descriptor; the vfile entry will be created into this directory. If NULL, the /proc root directory will be used. /proc/xenomai is mapped on the globally available *nkvfroot* vdir.

Returns

0 is returned on success. Otherwise:

- -ENOMEM is returned if the virtual file entry cannot be created in the /proc hierarchy.

References xnvfile_snapshot_ops::store.

5.22.3 Variable Documentation

5.22.3.1 struct xnvfile_directory nkvfroot

Xenomai vfile root directory.

This vdir maps the /proc/xenomai directory. It can be used to create a hierarchy of Xenomai-related vfiles under this root.

Referenced by xnvfile_destroy().

5.22.3.2 struct xnvfile_directory nkvfroot

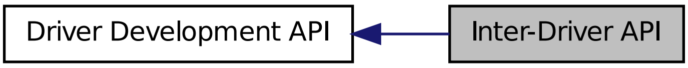
Xenomai vfile root directory.

This vdir maps the /proc/xenomai directory. It can be used to create a hierarchy of Xenomai-related vfiles under this root.

Referenced by xnvfile_destroy().

5.23 Inter-Driver API

Collaboration diagram for Inter-Driver API:



Functions

- struct rtdm_dev_context * rtdm_context_get (int fd)

  Retrieve and lock a device context.

- int rtdm_select_bind (int fd, rtdm_selector_t *selector, enum rtdm_selecttype type, unsigned fd_index)

  Bind a selector to specified event types of a given file descriptor.

- void rtdm_context_lock (struct rtdm_dev_context *context)

  Increment context reference counter.

- void rtdm_context_unlock (struct rtdm_dev_context *context)

  Decrement context reference counter.

- void rtdm_context_put (struct rtdm_dev_context *context)

  Release a device context obtained via rtdm_context_get().

- int rtdm_open (const char *path, int oflag,...)

  Open a device.

- int rtdm_socket (int protocol_family, int socket_type, int protocol)

  Create a socket.

- int rtdm_close (int fd)

  Close a device or socket.

- int rtdm_ioctl (int fd, int request,...)

  Issue an IOCTL.

- ssize_t rtdm_read (int fd, void *buf, size_t nbyte)

  Read from device.

- ssize_t rtdm_write (int fd, const void *buf, size_t nbyte)

  Write to device.

- ssize_t rtdm_recvmsg (int fd, struct msghdr *msg, int flags)

  Receive message from socket.

- ssize_t rtdm_recvfrom (int fd, void *buf, size_t len, int flags, struct sockaddr *from, socklen_t *fromlen)

  Receive message from socket.

- ssize_t rtdm_recv (int fd, void *buf, size_t len, int flags)

  Receive message from socket.

- ssize_t rtdm_sendmsg (int fd, const struct msghdr *msg, int flags)

  Transmit message to socket.

- ssize_t rtdm_sendto (int fd, const void *buf, size_t len, int flags, const struct sockaddr *to, socklen_t tolen)

  Transmit message to socket.

- ssize_t rtdm_send (int fd, const void *buf, size_t len, int flags)

  Transmit message to socket.

- int rtdm_bind (int fd, const struct sockaddr *my_addr, socklen_t addrlen)

  Bind to local address.

- int rtdm_connect (int fd, const struct sockaddr *serv_addr, socklen_t addrlen)

  Connect to remote address.

- int rtdm_listen (int fd, int backlog)

  Listen for incomming connection requests.

- int rtdm_accept (int fd, struct sockaddr *addr, socklen_t *addrlen)

  Accept a connection requests.

- int rtdm_shutdown (int fd, int how)

  Shut down parts of a connection.

- int rtdm_getsockopt (int fd, int level, int optname, void *optval, socklen_t *optlen)

  Get socket option.

- int rtdm_setsockopt (int fd, int level, int optname, const void *optval, socklen_t optlen)

  Set socket option.

- int rtdm_getsockname (int fd, struct sockaddr *name, socklen_t *namelen)

  Get local socket address.

- int rtdm_getpeername (int fd, struct sockaddr *name, socklen_t *namelen)

  Get socket destination address.

5.23.1 Function Documentation

5.23.1.1 int rtdm_accept (int *fd*, struct sockaddr * *addr*, socklen_t * *addrlen*)

Accept a connection requests.

Refer to rt_dev_accept() for parameters and return values

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

5.23.1.2 int rtdm_bind (int *fd*, const struct sockaddr * *my_addr*, socklen_t *addrlen*)

Bind to local address.

Refer to rt_dev_bind() for parameters and return values

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

5.23.1.3 int rtdm_close (int *fd*)

Close a device or socket.

Refer to rt_dev_close() for parameters and return values

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

5.23.1.4 int rtdm_connect (int *fd*, const struct sockaddr * *serv_addr*, socklen_t *addrlen*)

Connect to remote address.

Refer to rt_dev_connect() for parameters and return values

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

5.23.1.5 struct rtdm_dev_context* rtdm_context_get (int *fd*) `[read]`

Retrieve and lock a device context.

Parameters

`[in]` *fd* File descriptor

Returns

Pointer to associated device context, or NULL on error

Note

The device context has to be unlocked using rtdm_context_put() when it is no longer referenced.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

References rtdm_dev_context::close_lock_count.

Referenced by rtdm_select_bind().

5.23.1.6 void rtdm_context_lock (struct rtdm_dev_context * *context*)

Increment context reference counter.

Parameters

`[in]` ***context*** Device context

Note

rtdm_context_get() automatically increments the lock counter. You only need to call this function in special scenarios, e.g. when keeping additional references to the context structure that have different lifetimes. Only use rtdm_context_lock() on contexts that are currently locked via an earlier rtdm_context_get()/rtdm_contex_lock() or while running a device operation handler.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.23.1.7 void rtdm_context_put (struct rtdm_dev_context * *context*)

Release a device context obtained via rtdm_context_get().

Parameters

`[in]` *context* Device context

Note

Every successful call to rtdm_context_get() must be matched by a rtdm_context_put() invocation.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.23.1.8 void rtdm_context_unlock (struct rtdm_dev_context * *context*)

Decrement context reference counter.

Parameters

`[in]` *context* Device context

Note

Every call to rtdm_context_locked() must be matched by a rtdm_context_unlock() invocation.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

Referenced by rtdm_select_bind().

5.23.1.9 int rtdm_getpeername (int *fd*, struct sockaddr * *name*, socklen_t * *namelen*)

Get socket destination address.

Refer to rt_dev_getpeername() for parameters and return values

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

5.23.1.10 int rtdm_getsockname (int *fd*, struct sockaddr * *name*, socklen_t * *namelen*)

Get local socket address.

Refer to rt_dev_getsockname() for parameters and return values

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

5.23.1.11 int rtdm_getsockopt (int *fd*, int *level*, int *optname*, void * *optval*, socklen_t * *optlen*)

Get socket option.

Refer to rt_dev_getsockopt() for parameters and return values

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

5.23.1.12 int rtdm_ioctl (int *fd*, int *request*, ...)

Issue an IOCTL.

Refer to rt_dev_ioctl() for parameters and return values

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

5.23.1.13 int rtdm_listen (int *fd*, int *backlog*)

Listen for incomming connection requests.

Refer to rt_dev_listen() for parameters and return values

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

5.23.1.14 int rtdm_open (const char * *path*, int *oflag*, ...)

Open a device.

Refer to rt_dev_open() for parameters and return values

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

5.23.1.15 ssize_t rtdm_read (int *fd*, void * *buf*, size_t *nbyte*)

Read from device.

Refer to rt_dev_read() for parameters and return values

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

5.23.1.16 ssize_t rtdm_recv (int *fd*, void * *buf*, size_t *len*, int *flags*)

Receive message from socket.

Refer to rt_dev_recv() for parameters and return values

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

5.23.1.17 ssize_t rtdm_recvfrom (int *fd*, void * *buf*, size_t *len*, int *flags*, struct sockaddr * *from*, socklen_t * *fromlen*)

Receive message from socket.

Refer to rt_dev_recvfrom() for parameters and return values

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

5.23.1.18 ssize_t rtdm_recvmsg (int *fd*, struct msghdr * *msg*, int *flags*)

Receive message from socket.

Refer to rt_dev_recvmsg() for parameters and return values

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

5.23.1.19 int rtdm_select_bind (int *fd*, rtdm_selector_t * *selector*, enum rtdm_selecttype *type*, unsigned *fd_index*)

Bind a selector to specified event types of a given file descriptor.

For internal use only.

This function is invoked by higher RTOS layers implementing select-like services. It shall not be called directly by RTDM drivers.

Parameters

`[in]` ***fd*** File descriptor to bind to

`[in,out]` ***selector*** Selector object that shall be bound to the given event

`[in]` ***type*** Event type the caller is interested in

`[in]` ***fd_index*** Index in the file descriptor set of the caller

Returns

0 on success, otherwise:

- -EBADF is returned if the file descriptor *fd* cannot be resolved.
- -EINVAL is returned if *type* or *fd_index* are invalid.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

References rtdm_dev_context::ops, rtdm_context_get(), rtdm_context_unlock(), and rtdm_-operations::select_bind.

5.23.1.20 ssize_t rtdm_send (int *fd*, const void * *buf*, size_t *len*, int *flags*)

Transmit message to socket.

Refer to rt_dev_send() for parameters and return values

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

5.23.1.21 ssize_t rtdm_sendmsg (int *fd*, const struct msghdr * *msg*, int *flags*)

Transmit message to socket.

Refer to rt_dev_sendmsg() for parameters and return values

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

5.23.1.22 ssize_t rtdm_sendto (int *fd*, const void * *buf*, size_t *len*, int *flags*, const struct sockaddr * *to*, socklen_t *tolen*)

Transmit message to socket.

Refer to rt_dev_sendto() for parameters and return values

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

5.23.1.23 int rtdm_setsockopt (int *fd*, int *level*, int *optname*, const void * *optval*, socklen_t *optlen*)

Set socket option.

Refer to rt_dev_setsockopt() for parameters and return values

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

5.23.1.24 int rtdm_shutdown (int *fd*, int *how*)

Shut down parts of a connection.

Refer to rt_dev_shutdown() for parameters and return values

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

5.23.1.25 int rtdm_socket (int *protocol_family*, int *socket_type*, int *protocol*)

Create a socket.

Refer to rt_dev_socket() for parameters and return values

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

5.23.1.26 ssize_t rtdm_write (int *fd,* const void * *buf,* size_t *nbyte*)

Write to device.

Refer to rt_dev_write() for parameters and return values

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

5.24 Device Registration Services

Collaboration diagram for Device Registration Services:

Data Structures

- struct rtdm_operations

  Device operations.

- struct rtdm_dev_context

  Device context.

- struct rtdm_device

  RTDM device.

Modules

- Synchronisation Services

Functions

- int rtdm_dev_register (struct rtdm_device *device)

  Register a RTDM device.

- int rtdm_dev_unregister (struct rtdm_device *device, unsigned int poll_delay)

  Unregisters a RTDM device.

- static void * rtdm_context_to_private (struct rtdm_dev_context *context)

  Locate the driver private area associated to a device context structure.

- static struct rtdm_dev_context * rtdm_private_to_context (void *dev_private)

    Locate a device context structure from its driver private area.

Operation Handler Prototypes

- typedef int(* rtdm_open_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info, int oflag)

    Named device open handler.

- typedef int(* rtdm_socket_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info, int protocol)

    Socket creation handler for protocol devices.

- typedef int(* rtdm_close_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info)

    Close handler.

- typedef int(* rtdm_ioctl_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info, unsigned int request, void __user *arg)

    IOCTL handler.

- typedef int(* rtdm_select_bind_handler_t)(struct rtdm_dev_context *context, rtdm_selector_t *selector, enum rtdm_selecttype type, unsigned fd_index)

    Select binding handler.

- typedef ssize_t(* rtdm_read_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info, void *buf, size_t nbyte)

    Read handler.

- typedef ssize_t(* rtdm_write_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info, const void *buf, size_t nbyte)

    Write handler.

- typedef ssize_t(* rtdm_recvmsg_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info, struct msghdr *msg, int flags)

    Receive message handler.

- typedef ssize_t(* rtdm_sendmsg_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info, const struct msghdr *msg, int flags)

    Transmit message handler.

Device Flags

Static flags describing a RTDM device

- #define RTDM_EXCLUSIVE 0x0001

If set, only a single instance of the device can be requested by an application.

- #define RTDM_NAMED_DEVICE 0x0010

  If set, the device is addressed via a clear-text name.

- #define RTDM_PROTOCOL_DEVICE 0x0020

  If set, the device is addressed via a combination of protocol ID and socket type.

- #define RTDM_DEVICE_TYPE_MASK 0x00F0

  Mask selecting the device type.

Context Flags

Dynamic flags describing the state of an open RTDM device (bit numbers)

- #define RTDM_CREATED_IN_NRT 0

  Set by RTDM if the device instance was created in non-real-time context.

- #define RTDM_CLOSING 1

  Set by RTDM when the device is being closed.

- #define RTDM_USER_CONTEXT_FLAG 8

  Lowest bit number the driver developer can use freely.

Driver Versioning

Current revisions of RTDM structures, encoding of driver versions. See API Versioning for the interface revision.

- #define RTDM_DEVICE_STRUCT_VER 5

  Version of struct rtdm_device.

- #define RTDM_CONTEXT_STRUCT_VER 3

  Version of struct rtdm_dev_context.

- #define RTDM_SECURE_DEVICE 0x80000000

  Flag indicating a secure variant of RTDM (not supported here).

- #define RTDM_DRIVER_VER(major, minor, patch) (((major & 0xFF) << 16) | ((minor & 0xFF) << 8) | (patch & 0xFF))

  Version code constructor for driver revisions.

- #define RTDM_DRIVER_MAJOR_VER(ver) (((ver) >> 16) & 0xFF)

  Get major version number from driver revision code.

- #define RTDM_DRIVER_MINOR_VER(ver) (((ver) >> 8) & 0xFF)

  Get minor version number from driver revision code.

- #define RTDM_DRIVER_PATCH_VER(ver) ((ver) & 0xFF)

    Get patch version number from driver revision code.

5.24.1 Define Documentation

5.24.1.1 #define RTDM_CLOSING 1

Set by RTDM when the device is being closed.

5.24.1.2 #define RTDM_CREATED_IN_NRT 0

Set by RTDM if the device instance was created in non-real-time context.

5.24.1.3 #define RTDM_DEVICE_TYPE_MASK 0x00F0

Mask selecting the device type.

Referenced by rtdm_dev_register(), and rtdm_dev_unregister().

5.24.1.4 #define RTDM_EXCLUSIVE 0x0001

If set, only a single instance of the device can be requested by an application.

Referenced by rtdm_dev_register().

5.24.1.5 #define RTDM_NAMED_DEVICE 0x0010

If set, the device is addressed via a clear-text name.

Referenced by rtdm_dev_register(), and rtdm_dev_unregister().

5.24.1.6 #define RTDM_PROTOCOL_DEVICE 0x0020

If set, the device is addressed via a combination of protocol ID and socket type.

Referenced by rtdm_dev_register().

5.24.2 Typedef Documentation

5.24.2.1 typedef int(* rtdm_close_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info)

Close handler.

Parameters

`[in]` *context* Context structure associated with opened device instance `[in]` *user_info* Opaque pointer to information about user mode caller, NULL if kernel mode or deferred user mode call

Returns

0 on success. On failure return either -ENOSYS, to request that this handler be called again from the opposite realtime/non-realtime context, -EAGAIN to request a recall after a grace period, or a valid negative error code according to IEEE Std 1003.1.

Note

Drivers must be prepared for that case that the close handler is invoked more than once per open context (even if the handler already completed an earlier run successfully). The driver has to avoid releasing resources twice as well as returning false errors on successive close invocations.

See also

`close()` in IEEE Std 1003.1, http://www.opengroup.org/onlinepubs/009695399

5.24.2.2 typedef int(* rtdm_ioctl_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info, unsigned int request, void __user *arg)

IOCTL handler.

Parameters

`[in]` *context* Context structure associated with opened device instance

`[in]` *user_info* Opaque pointer to information about user mode caller, NULL if kernel mode call

`[in]` *request* Request number as passed by the user

`[in,out]` *arg* Request argument as passed by the user

Returns

A positive value or 0 on success. On failure return either -ENOSYS, to request that the function be called again from the opposite realtime/non-realtime context, or another negative error code.

See also

`ioctl()` in IEEE Std 1003.1, http://www.opengroup.org/onlinepubs/009695399

5.24.2.3 typedef int(* rtdm_open_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info, int oflag)

Named device open handler.

Parameters

`[in]` *context* Context structure associated with opened device instance

`[in]` *user_info* Opaque pointer to information about user mode caller, NULL if kernel mode call `[in]` *oflag* Open flags as passed by the user

Returns

0 on success. On failure return either -ENOSYS, to request that this handler be called again from the opposite realtime/non-realtime context, or another negative error code.

See also

`open()` in IEEE Std 1003.1, http://www.opengroup.org/onlinepubs/009695399

5.24.2.4 typedef ssize_t(* rtdm_read_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info, void *buf, size_t nbyte)

Read handler.

Parameters

- `[in]` *context* Context structure associated with opened device instance
- `[in]` *user_info* Opaque pointer to information about user mode caller, NULL if kernel mode call
- `[out]` *buf* Input buffer as passed by the user
- `[in]` *nbyte* Number of bytes the user requests to read

Returns

On success, the number of bytes read. On failure return either -ENOSYS, to request that this handler be called again from the opposite realtime/non-realtime context, or another negative error code.

See also

`read()` in IEEE Std 1003.1, http://www.opengroup.org/onlinepubs/009695399

5.24.2.5 typedef ssize_t(* rtdm_recvmsg_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info, struct msghdr *msg, int flags)

Receive message handler.

Parameters

- `[in]` *context* Context structure associated with opened device instance
- `[in]` *user_info* Opaque pointer to information about user mode caller, NULL if kernel mode call
- `[in,out]` *msg* Message descriptor as passed by the user, automatically mirrored to safe kernel memory in case of user mode call
- `[in]` *flags* Message flags as passed by the user

Returns

On success, the number of bytes received. On failure return either -ENOSYS, to request that this handler be called again from the opposite realtime/non-realtime context, or another negative error code.

See also

recvmsg() in IEEE Std 1003.1, http://www.opengroup.org/onlinepubs/009695399

5.24.2.6 typedef int(* rtdm_select_bind_handler_t)(struct rtdm_dev_context *context, rtdm_selector_t *selector, enum rtdm_selecttype type, unsigned fd_index)

Select binding handler.

Parameters

[in] *context* Context structure associated with opened device instance

[in,out] *selector* Object that shall be bound to the given event

[in] *type* Event type the selector is interested in

[in] *fd_index* Opaque value, to be passed to rtdm_event_select_bind or rtdm_sem_select_-bind unmodfied

Returns

0 on success. On failure return either -ENOSYS, to request that this handler be called again from the opposite realtime/non-realtime context, or another negative error code.

5.24.2.7 typedef ssize_t(* rtdm_sendmsg_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info, const struct msghdr *msg, int flags)

Transmit message handler.

Parameters

[in] *context* Context structure associated with opened device instance

[in] *user_info* Opaque pointer to information about user mode caller, NULL if kernel mode call

[in] *msg* Message descriptor as passed by the user, automatically mirrored to safe kernel memory in case of user mode call

[in] *flags* Message flags as passed by the user

Returns

On success, the number of bytes transmitted. On failure return either -ENOSYS, to request that this handler be called again from the opposite realtime/non-realtime context, or another negative error code.

See also

sendmsg() in IEEE Std 1003.1, http://www.opengroup.org/onlinepubs/009695399

5.24.2.8 typedef int(* rtdm_socket_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info, int protocol)

Socket creation handler for protocol devices.

Parameters

`[in]` *context* Context structure associated with opened device instance

`[in]` *user_info* Opaque pointer to information about user mode caller, NULL if kernel mode call

`[in]` *protocol* Protocol number as passed by the user

Returns

0 on success. On failure return either -ENOSYS, to request that this handler be called again from the opposite realtime/non-realtime context, or another negative error code.

See also

`socket()` in IEEE Std 1003.1, http://www.opengroup.org/onlinepubs/009695399

5.24.2.9 typedef ssize_t(* rtdm_write_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info, const void *buf, size_t nbyte)

Write handler.

Parameters

`[in]` *context* Context structure associated with opened device instance

`[in]` *user_info* Opaque pointer to information about user mode caller, NULL if kernel mode call

`[in]` *buf* Output buffer as passed by the user

`[in]` *nbyte* Number of bytes the user requests to write

Returns

On success, the number of bytes written. On failure return either -ENOSYS, to request that this handler be called again from the opposite realtime/non-realtime context, or another negative error code.

See also

`write()` in IEEE Std 1003.1, http://www.opengroup.org/onlinepubs/009695399

5.24.3 Function Documentation

5.24.3.1 static void* rtdm_context_to_private (struct rtdm_dev_context * *context*) `[inline, static]`

Locate the driver private area associated to a device context structure.

Parameters

`[in]` ***context*** Context structure associated with opened device instance

Returns

The address of the private driver area associated to *context*.

References rtdm_dev_context::dev_private.

5.24.3.2 int rtdm_dev_register (struct rtdm_device * *device*)

Register a RTDM device.

Parameters

`[in]` ***device*** Pointer to structure describing the new device.

Returns

0 is returned upon success. Otherwise:

- -EINVAL is returned if the device structure contains invalid entries. Check kernel log in this case.
- -ENOMEM is returned if the context for an exclusive device cannot be allocated.
- -EEXIST is returned if the specified device name of protocol ID is already in use.
- -EAGAIN is returned if some /proc entry cannot be created.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code

Rescheduling: never.

References rtdm_operations::close_nrt, rtdm_operations::close_rt, rtdm_device::context_-size, rtdm_device::device_class, rtdm_device::device_flags, rtdm_device::device_name, rtdm_device::device_sub_class, rtdm_device::driver_version, rtdm_device::open_rt, rtdm_-device::ops, rtdm_device::proc_name, rtdm_device::profile_version, rtdm_device::protocol_-family, rtdm_device::reserved, RTDM_DEVICE_STRUCT_VER, RTDM_DEVICE_TYPE_-MASK, RTDM_EXCLUSIVE, RTDM_NAMED_DEVICE, RTDM_PROTOCOL_DEVICE, rtdm_operations::select_bind, rtdm_device::socket_rt, rtdm_device::socket_type, and rtdm_-device::struct_version.

5.24.3.3 int rtdm_dev_unregister (struct rtdm_device * *device*, unsigned int *poll_delay*)

Unregisters a RTDM device.

Parameters

`[in]` ***device*** Pointer to structure describing the device to be unregistered.

`[in]` ***poll_delay*** Polling delay in milliseconds to check repeatedly for open instances of *device*, or 0 for non-blocking mode.

Returns

0 is returned upon success. Otherwise:

- -ENODEV is returned if the device was not registered.
- -EAGAIN is returned if the device is busy with open instances and 0 has been passed for *poll_delay*.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code

Rescheduling: never.

References rtdm_device::device_flags, rtdm_device::device_name, rtdm_device::protocol_family, rtdm_device::reserved, RTDM_DEVICE_TYPE_MASK, RTDM_NAMED_DEVICE, and rtdm_-device::socket_type.

5.24.3.4 static struct rtdm_dev_context* rtdm_private_to_context (void * *dev_private*) `[static, read]`

Locate a device context structure from its driver private area.

Parameters

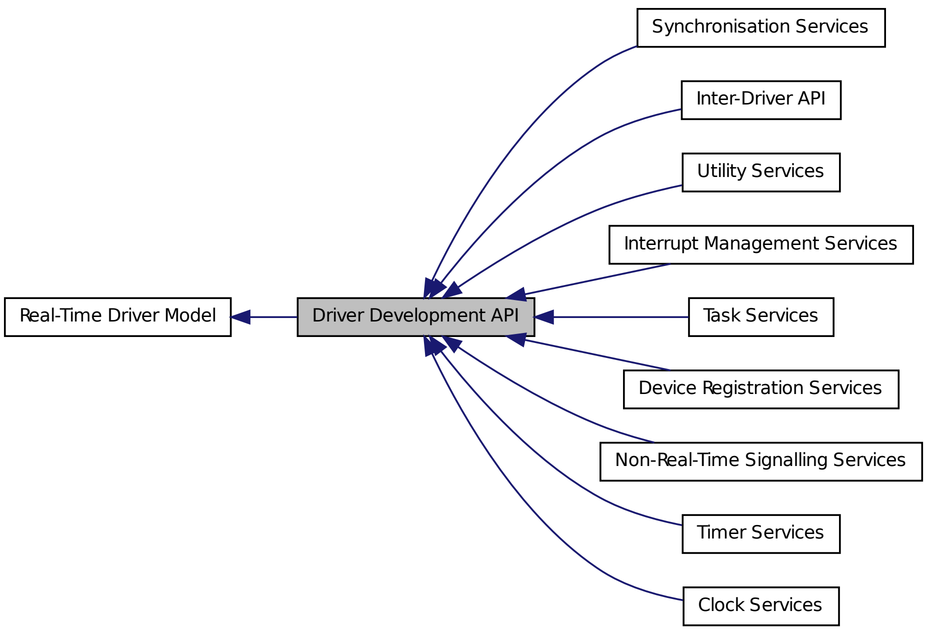
`[in]` ***dev_private*** Address of a private context area

Returns

The address of the device context structure defining *dev_private*.

5.25 Driver Development API

Collaboration diagram for Driver Development API:



Modules

- Inter-Driver API
- Device Registration Services
- Clock Services
- Task Services
- Timer Services
- Synchronisation Services
- Interrupt Management Services
- Non-Real-Time Signalling Services
- Utility Services

Files

- file rtdm_driver.h

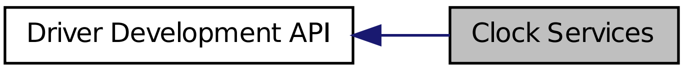
    Real-Time Driver Model for Xenomai, driver API header.

5.25.1 Detailed Description

This is the lower interface of RTDM provided to device drivers, currently limited to kernel-space. Real-time drivers should only use functions of this interface in order to remain portable.

5.26 Clock Services

Collaboration diagram for Clock Services:



Functions

- nanosecs_abs_t rtdm_clock_read (void)
  Get system time.
- nanosecs_abs_t rtdm_clock_read_monotonic (void)
  Get monotonic time.

5.26.1 Function Documentation

5.26.1.1 nanosecs_abs_t rtdm_clock_read (void)

Get system time.

Returns

The system time in nanoseconds is returned

Note

The resolution of this service depends on the system timer. In particular, if the system timer is running in periodic mode, the return value will be limited to multiples of the timer tick period.
The system timer may have to be started to obtain valid results. Whether this happens automatically (as on Xenomai) or is controlled by the application depends on the RTDM host environment.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.26.1.2 nanosecs_abs_t rtdm_clock_read_monotonic (void)

Get monotonic time.

Returns

The monotonic time in nanoseconds is returned

Note

The resolution of this service depends on the system timer. In particular, if the system timer is running in periodic mode, the return value will be limited to multiples of the timer tick period.
The system timer may have to be started to obtain valid results. Whether this happens automatically (as on Xenomai) or is controlled by the application depends on the RTDM host environment.

Environments:

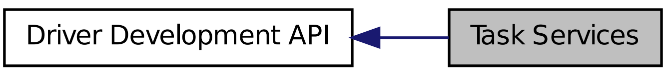
This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.27 Task Services

Collaboration diagram for Task Services:



Typedefs

- typedef void(* rtdm_task_proc_t)(void *arg)

  Real-time task procedure.

Functions

- int rtdm_task_init (rtdm_task_t *task, const char *name, rtdm_task_proc_t task_proc, void *arg, int priority, nanosecs_rel_t period)

  Intialise and start a real-time task.

- void rtdm_task_destroy (rtdm_task_t *task)

  Destroy a real-time task.

- void rtdm_task_set_priority (rtdm_task_t *task, int priority)

  Adjust real-time task priority.

- int rtdm_task_set_period (rtdm_task_t *task, nanosecs_rel_t period)

  Adjust real-time task period.

- int rtdm_task_wait_period (void)

  Wait on next real-time task period.

- int rtdm_task_unblock (rtdm_task_t *task)

  Activate a blocked real-time task.

- rtdm_task_t * rtdm_task_current (void)

  Get current real-time task.

- int rtdm_task_sleep (nanosecs_rel_t delay)

  Sleep a specified amount of time.

- int rtdm_task_sleep_until (nanosecs_abs_t wakeup_time)

  Sleep until a specified absolute time.

- int rtdm_task_sleep_abs (nanosecs_abs_t wakeup_time, enum rtdm_timer_mode mode)

  Sleep until a specified absolute time.

- void rtdm_task_join_nrt (rtdm_task_t *task, unsigned int poll_delay)

  Wait on a real-time task to terminate.

- void rtdm_task_busy_sleep (nanosecs_rel_t delay)

  Busy-wait a specified amount of time.

Task Priority Range

Maximum and minimum task priorities

- #define **RTDM_TASK_LOWEST_PRIORITY** XNSCHED_LOW_PRIO
- #define **RTDM_TASK_HIGHEST_PRIORITY** XNSCHED_HIGH_PRIO

Task Priority Modification

Raise or lower task priorities by one level

- #define **RTDM_TASK_RAISE_PRIORITY** (+1)
- #define **RTDM_TASK_LOWER_PRIORITY** (-1)

5.27.1 Typedef Documentation

5.27.1.1 typedef void(* rtdm_task_proc_t)(void *arg)

Real-time task procedure.

Parameters

`[in,out]` *arg* argument as passed to rtdm_task_init()

5.27.2 Function Documentation

5.27.2.1 void rtdm_task_busy_sleep (nanosecs_rel_t *delay*)

Busy-wait a specified amount of time.

Parameters

`[in]` *delay* Delay in nanoseconds. Note that a zero delay does **not** have the meaning of `RTDM_TIMEOUT_INFINITE` here.

Note

The caller must not be migratable to different CPUs while executing this service. Otherwise, the actual delay will be undefined.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine (should be avoided or kept short)
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never (except due to external interruptions).

5.27.2.2 rtdm_task_t* rtdm_task_current (void)

Get current real-time task.

Returns

Pointer to task handle

Environments:

This service can be called from:

- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.27.2.3 void rtdm_task_destroy (rtdm_task_t * *task*)

Destroy a real-time task.

Parameters

`[in,out]` ***task*** Task handle as returned by rtdm_task_init()

Note

Passing the same task handle to RTDM services after the completion of this function is not allowed.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.27.2.4 int rtdm_task_init (rtdm_task_t * *task*, const char * *name*, rtdm_task_proc_t *task_proc*, void * *arg*, int *priority*, nanosecs_rel_t *period*)

Intialise and start a real-time task.

After initialising a task, the task handle remains valid and can be passed to RTDM services until either rtdm_task_destroy() or rtdm_task_join_nrt() was invoked.

Parameters

`[in,out]` ***task*** Task handle

`[in]` ***name*** Optional task name

[in] *task_proc* Procedure to be executed by the task

[in] *arg* Custom argument passed to `task_proc()` on entry

[in] *priority* Priority of the task, see also Task Priority Range

[in] *period* Period in nanoseconds of a cyclic task, 0 for non-cyclic mode

Returns

0 on success, otherwise negative error code

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: possible.

References xnpod_delete_thread(), xnpod_init_thread(), xnpod_set_thread_periodic(), and xnpod_start_thread().

5.27.2.5 void rtdm_task_join_nrt (rtdm_task_t * *task*, unsigned int *poll_delay*)

Wait on a real-time task to terminate.

Parameters

[in,out] *task* Task handle as returned by rtdm_task_init()

[in] *poll_delay* Delay in milliseconds between periodic tests for the state of the real-time task. This parameter is ignored if the termination is internally realised without polling.

Note

Passing the same task handle to RTDM services after the completion of this function is not allowed.
This service does not trigger the termination of the targeted task. The user has to take of this, otherwise rtdm_task_join_nrt() will never return.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- User-space task (non-RT)

Rescheduling: possible.

References XNZOMBIE.

5.27.2.6 int rtdm_task_set_period (rtdm_task_t * *task*, nanosecs_rel_t *period*)

Adjust real-time task period.

Parameters

`[in,out]` ***task*** Task handle as returned by rtdm_task_init()

`[in]` ***period*** New period in nanoseconds of a cyclic task, 0 for non-cyclic mode

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: possible.

5.27.2.7 void rtdm_task_set_priority (rtdm_task_t * *task*, int *priority*)

Adjust real-time task priority.

Parameters

`[in,out]` ***task*** Task handle as returned by rtdm_task_init()

`[in]` ***priority*** New priority of the task, see also Task Priority Range

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: possible.

5.27.2.8 int rtdm_task_sleep (nanosecs_rel_t *delay*)

Sleep a specified amount of time.

Parameters

`[in]` ***delay*** Delay in nanoseconds, see RTDM_TIMEOUT_xxx for special values.

Returns

0 on success, otherwise:

- -EINTR is returned if calling task has been unblock by a signal or explicitly via rtdm_task_-unblock().
- -EPERM *may* be returned if an illegal invocation environment is detected.

Environments:

This service can be called from:

- Kernel-based task
- User-space task (RT)

Rescheduling: always.

5.27.2.9 int rtdm_task_sleep_abs (nanosecs_abs_t *wakeup_time*, enum rtdm_timer_mode *mode*)

Sleep until a specified absolute time.

Parameters

`[in]` ***wakeup_time*** Absolute timeout in nanoseconds

`[in]` ***mode*** Selects the timer mode, see RTDM_TIMERMODE_xxx for details

Returns

0 on success, otherwise:

- -EINTR is returned if calling task has been unblock by a signal or explicitly via rtdm_task_-unblock().
- -EPERM *may* be returned if an illegal invocation environment is detected.
- -EINVAL is returned if an invalid parameter was passed.

Environments:

This service can be called from:

- Kernel-based task
- User-space task (RT)

Rescheduling: always, unless the specified time already passed.

5.27.2.10 int rtdm_task_sleep_until (nanosecs_abs_t *wakeup_time*)

Sleep until a specified absolute time.

Deprecated

Use rtdm_task_sleep_abs instead!

Parameters

`[in]` *wakeup_time* Absolute timeout in nanoseconds

Returns

0 on success, otherwise:

- -EINTR is returned if calling task has been unblock by a signal or explicitly via rtdm_task_unblock().
- -EPERM *may* be returned if an illegal invocation environment is detected.

Environments:

This service can be called from:

- Kernel-based task
- User-space task (RT)

Rescheduling: always, unless the specified time already passed.

5.27.2.11 int rtdm_task_unblock (rtdm_task_t * *task*)

Activate a blocked real-time task.

Returns

Non-zero is returned if the task was actually unblocked from a pending wait state, 0 otherwise.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: possible.

5.27.2.12 int rtdm_task_wait_period (void)

Wait on next real-time task period.

Returns

0 on success, otherwise:

- -EINVAL is returned if calling task is not in periodic mode.
- -ETIMEDOUT is returned if a timer overrun occurred, which indicates that a previous release point has been missed by the calling task.

Environments:

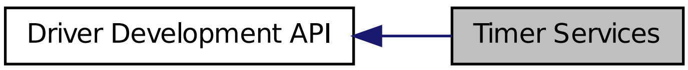
This service can be called from:

- Kernel-based task
- User-space task (RT)

Rescheduling: always, unless a timer overrun occured.

5.28 Timer Services

Collaboration diagram for Timer Services:



Typedefs

- typedef void(* rtdm_timer_handler_t)(rtdm_timer_t *timer)

  Timer handler.

Functions

- int rtdm_timer_init (rtdm_timer_t *timer, rtdm_timer_handler_t handler, const char *name)

  Initialise a timer.

- void rtdm_timer_destroy (rtdm_timer_t *timer)

  Destroy a timer.

- int rtdm_timer_start (rtdm_timer_t *timer, nanosecs_abs_t expiry, nanosecs_rel_t interval, enum rtdm_timer_mode mode)

  Start a timer.

- void rtdm_timer_stop (rtdm_timer_t *timer)

  Stop a timer.

- int rtdm_timer_start_in_handler (rtdm_timer_t *timer, nanosecs_abs_t expiry, nanosecs_-rel_t interval, enum rtdm_timer_mode mode)

  Start a timer from inside a timer handler.

- void rtdm_timer_stop_in_handler (rtdm_timer_t *timer)

  Stop a timer from inside a timer handler.

RTDM_TIMERMODE_xxx

Timer operation modes

- enum rtdm_timer_mode { RTDM_TIMERMODE_RELATIVE = XN_RELATIVE, RTDM_-TIMERMODE_ABSOLUTE = XN_ABSOLUTE, RTDM_TIMERMODE_REALTIME = XN_-REALTIME }

5.28.1 Typedef Documentation

5.28.1.1 typedef void(* rtdm_timer_handler_t)(rtdm_timer_t *timer)

Timer handler.

Parameters

`[in]` ***timer*** Timer handle as returned by rtdm_timer_init()

5.28.2 Enumeration Type Documentation

5.28.2.1 enum rtdm_timer_mode

Enumerator:

RTDM_TIMERMODE_RELATIVE Monotonic timer with relative timeout.

RTDM_TIMERMODE_ABSOLUTE Monotonic timer with absolute timeout.

RTDM_TIMERMODE_REALTIME Adjustable timer with absolute timeout.

5.28.3 Function Documentation

5.28.3.1 void rtdm_timer_destroy (rtdm_timer_t * *timer*)

Destroy a timer.

Parameters

[in,out] *timer* Timer handle as returned by rtdm_timer_init()

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

References xntimer_destroy().

5.28.3.2 int rtdm_timer_init (rtdm_timer_t * *timer*, rtdm_timer_handler_t *handler*, const char * *name*)

Initialise a timer.

Parameters

[in,out] *timer* Timer handle

[in] *handler* Handler to be called on timer expiry

[in] *name* Optional timer name

Returns

0 on success, otherwise negative error code

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.28.3.3 int rtdm_timer_start (rtdm_timer_t * *timer*, nanosecs_abs_t *expiry*, nanosecs_rel_t *interval*, enum rtdm_timer_mode *mode*)

Start a timer.

Parameters

`[in,out]` ***timer*** Timer handle as returned by rtdm_timer_init()

`[in]` ***expiry*** Firing time of the timer, `mode` defines if relative or absolute

`[in]` ***interval*** Relative reload value, > 0 if the timer shall work in periodic mode with the specific interval, 0 for one-shot timers

`[in]` ***mode*** Defines the operation mode, see RTDM_TIMERMODE_xxx for possible values

Returns

0 on success, otherwise:

- -ETIMEDOUT is returned if `expiry` describes an absolute date in the past.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

References xntimer_start().

5.28.3.4 int rtdm_timer_start_in_handler (rtdm_timer_t * *timer*, nanosecs_abs_t *expiry*, nanosecs_rel_t *interval*, enum rtdm_timer_mode *mode*)

Start a timer from inside a timer handler.

Parameters

`[in,out]` ***timer*** Timer handle as returned by rtdm_timer_init()

`[in]` ***expiry*** Firing time of the timer, `mode` defines if relative or absolute

`[in]` ***interval*** Relative reload value, > 0 if the timer shall work in periodic mode with the specific interval, 0 for one-shot timers

`[in]` ***mode*** Defines the operation mode, see RTDM_TIMERMODE_xxx for possible values

Returns

0 on success, otherwise:

- -ETIMEDOUT is returned if `expiry` describes an absolute date in the past.

Environments:

This service can be called from:

- Timer handler

Rescheduling: never.

5.28.3.5 void rtdm_timer_stop (rtdm_timer_t * *timer*)

Stop a timer.

Parameters

`[in,out]` ***timer*** Timer handle as returned by rtdm_timer_init()

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

References xntimer_stop().

5.28.3.6 void rtdm_timer_stop_in_handler (rtdm_timer_t * *timer*)

Stop a timer from inside a timer handler.

Parameters

`[in,out]` ***timer*** Timer handle as returned by rtdm_timer_init()

Environments:

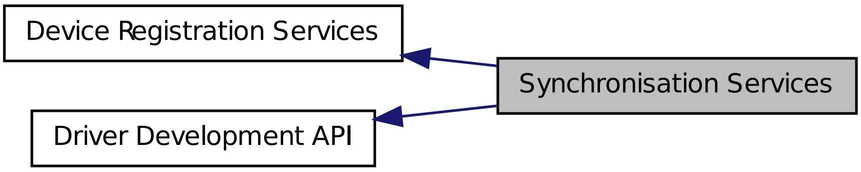
This service can be called from:

- Timer handler

Rescheduling: never.

5.29 Synchronisation Services

Collaboration diagram for Synchronisation Services:



Functions

- int rtdm_select_bind (int fd, rtdm_selector_t *selector, enum rtdm_selecttype type, unsigned fd_index)

  Bind a selector to specified event types of a given file descriptor.

RTDM_SELECTTYPE_xxx

Event types select can bind to

- enum rtdm_selecttype { RTDM_SELECTTYPE_READ = XNSELECT_READ, RTDM_SELECTTYPE_WRITE = XNSELECT_WRITE, RTDM_SELECTTYPE_EXCEPT = XNSELECT_EXCEPT }

Spinlock with Preemption Deactivation

- typedef ipipe_spinlock_t rtdm_lock_t

  Lock variable.

- typedef unsigned long rtdm_lockctx_t

  Variable to save the context while holding a lock.

- #define RTDM_LOCK_UNLOCKED IPIPE_SPIN_LOCK_UNLOCKED

  Static lock initialisation.

- #define rtdm_lock_init(lock) spin_lock_init(lock)

  Dynamic lock initialisation.

- #define rtdm_lock_get(lock) spin_lock(lock)

  Acquire lock from non-preemptible contexts.

- #define rtdm_lock_put(lock) spin_unlock(lock)
  Release lock without preemption restoration.
- #define rtdm_lock_get_irqsave(lock, context) spin_lock_irqsave(lock, context)
  Acquire lock and disable preemption.
- #define rtdm_lock_put_irqrestore(lock, context) spin_unlock_irqrestore(lock, context)
  Release lock and restore preemption state.
- #define rtdm_lock_irqsave(context) splhigh(context)
  Disable preemption locally.
- #define rtdm_lock_irqrestore(context) splexit(context)
  Restore preemption state.

Timeout Sequence Management

- void rtdm_toseq_init (rtdm_toseq_t *timeout_seq, nanosecs_rel_t timeout)
  Initialise a timeout sequence.

Event Services

- void rtdm_event_init (rtdm_event_t *event, unsigned long pending)
  Initialise an event.
- void rtdm_event_destroy (rtdm_event_t *event)
  Destroy an event.
- void rtdm_event_pulse (rtdm_event_t *event)
  Signal an event occurrence to currently listening waiters.
- void rtdm_event_signal (rtdm_event_t *event)
  Signal an event occurrence.
- int rtdm_event_wait (rtdm_event_t *event)
  Wait on event occurrence.
- int rtdm_event_timedwait (rtdm_event_t *event, nanosecs_rel_t timeout, rtdm_toseq_t *timeout_seq)
  Wait on event occurrence with timeout.
- void rtdm_event_clear (rtdm_event_t *event)
  Clear event state.
- int rtdm_event_select_bind (rtdm_event_t *event, rtdm_selector_t *selector, enum rtdm_selecttype type, unsigned fd_index)
  Bind a selector to an event.

Semaphore Services

- void rtdm_sem_init (rtdm_sem_t *sem, unsigned long value)

  Initialise a semaphore.

- void rtdm_sem_destroy (rtdm_sem_t *sem)

  Destroy a semaphore.

- int rtdm_sem_down (rtdm_sem_t *sem)

  Decrement a semaphore.

- int rtdm_sem_timeddown (rtdm_sem_t *sem, nanosecs_rel_t timeout, rtdm_toseq_t *timeout_seq)

  Decrement a semaphore with timeout.

- void rtdm_sem_up (rtdm_sem_t *sem)

  Increment a semaphore.

- int rtdm_sem_select_bind (rtdm_sem_t *sem, rtdm_selector_t *selector, enum rtdm_selecttype type, unsigned fd_index)

  Bind a selector to a semaphore.

Mutex Services

- void rtdm_mutex_init (rtdm_mutex_t *mutex)

  Initialise a mutex.

- void rtdm_mutex_destroy (rtdm_mutex_t *mutex)

  Destroy a mutex.

- void rtdm_mutex_unlock (rtdm_mutex_t *mutex)

  Release a mutex.

- int rtdm_mutex_lock (rtdm_mutex_t *mutex)

  Request a mutex.

- int rtdm_mutex_timedlock (rtdm_mutex_t *mutex, nanosecs_rel_t timeout, rtdm_toseq_t *timeout_seq)

  Request a mutex with timeout.

Global Lock across Scheduler Invocation

- #define RTDM_EXECUTE_ATOMICALLY(code_block)

  Execute code block atomically.

5.29.1 Define Documentation

5.29.1.1 #define RTDM_EXECUTE_ATOMICALLY(*code_block*)

Value:

```
{                                               \
        <ENTER_ATOMIC_SECTION>                  \
        code_block;                             \
        <LEAVE_ATOMIC_SECTION>                  \
}
```

Execute code block atomically.

Generally, it is illegal to suspend the current task by calling rtdm_task_sleep(), rtdm_event_-wait(), etc. while holding a spinlock. In contrast, this macro allows to combine several operations including a potentially rescheduling call to an atomic code block with respect to other RTDM_-EXECUTE_ATOMICALLY() blocks. The macro is a light-weight alternative for protecting code blocks via mutexes, and it can even be used to synchronise real-time and non-real-time contexts.

Parameters

code_block Commands to be executed atomically

Note

It is not allowed to leave the code block explicitly by using `break`, `return`, `goto`, etc. This would leave the global lock held during the code block execution in an inconsistent state. Moreover, do not embed complex operations into the code bock. Consider that they will be executed under preemption lock with interrupts switched-off. Also note that invocation of rescheduling calls may break the atomicity until the task gains the CPU again.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: possible, depends on functions called within *code_block*.

5.29.1.2 #define rtdm_lock_get(*lock*) spin_lock(lock)

Acquire lock from non-preemptible contexts.

Parameters

lock Address of lock variable

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.29.1.3 #define rtdm_lock_get_irqsave(*lock*, *context*) spin_lock_irqsave(lock, context)

Acquire lock and disable preemption.

Parameters

lock Address of lock variable

context name of local variable to store the context in

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.29.1.4 #define rtdm_lock_init(*lock*) spin_lock_init(lock)

Dynamic lock initialisation.

Parameters

lock Address of lock variable

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.29.1.5 #define rtdm_lock_irqrestore(*context*) splexit(context)

Restore preemption state.

Parameters

context name of local variable which stored the context

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: possible.

5.29.1.6 #define rtdm_lock_irqsave(*context*) splhigh(context)

Disable preemption locally.

Parameters

context name of local variable to store the context in

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.29.1.7 #define rtdm_lock_put(*lock*) spin_unlock(lock)

Release lock without preemption restoration.

Parameters

lock Address of lock variable

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.29.1.8 #define rtdm_lock_put_irqrestore(*lock*, *context*) spin_unlock_irqrestore(lock, context)

Release lock and restore preemption state.

Parameters

lock Address of lock variable

context name of local variable which stored the context

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: possible.

5.29.2 Enumeration Type Documentation

5.29.2.1 enum rtdm_selecttype

Enumerator:

RTDM_SELECTTYPE_READ Select input data availability events.

RTDM_SELECTTYPE_WRITE Select ouput buffer availability events.

RTDM_SELECTTYPE_EXCEPT Select exceptional events.

5.29.3 Function Documentation

5.29.3.1 void rtdm_event_clear (rtdm_event_t * *event*)

Clear event state.

Parameters

`[in,out]` *event* Event handle as returned by rtdm_event_init()

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

References xnselect_signal().

5.29.3.2 void rtdm_event_destroy (rtdm_event_t * *event*)

Destroy an event.

Parameters

`[in,out]` *event* Event handle as returned by rtdm_event_init()

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: possible.

5.29.3.3 void rtdm_event_init (rtdm_event_t * *event*, unsigned long *pending*)

Initialise an event.

Parameters

`[in,out]` *event* Event handle

`[in]` *pending* Non-zero if event shall be initialised as set, 0 otherwise

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

References xnselect_init(), and xnsynch_init().

5.29.3.4 void rtdm_event_pulse (rtdm_event_t * *event*)

Signal an event occurrence to currently listening waiters.

This function wakes up all current waiters of the given event, but it does not change the event state. Subsequently callers of rtdm_event_wait() or rtdm_event_timedwait() will therefore be blocked first.

Parameters

`[in,out]` ***event*** Event handle as returned by rtdm_event_init()

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: possible.

5.29.3.5 int rtdm_event_select_bind (rtdm_event_t * *event*, rtdm_selector_t * *selector*, enum rtdm_selecttype *type*, unsigned *fd_index*)

Bind a selector to an event.

This functions binds the given selector to an event so that the former is notified when the event state changes. Typically the select binding handler will invoke this service.

Parameters

`[in,out]` ***event*** Event handle as returned by rtdm_event_init()

`[in,out]` ***selector*** Selector as passed to the select binding handler

`[in]` ***type*** Type of the bound event as passed to the select binding handler

`[in]` ***fd_index*** File descriptor index as passed to the select binding handler

Returns

0 on success, otherwise:

- -ENOMEM is returned if there is insufficient memory to establish the dynamic binding.
- -EINVAL is returned if *type* or *fd_index* are invalid.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code

- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

References xnselect_bind().

5.29.3.6 void rtdm_event_signal (rtdm_event_t * *event*)

Signal an event occurrence.

This function sets the given event and wakes up all current waiters. If no waiter is presently registered, the next call to rtdm_event_wait() or rtdm_event_timedwait() will return immediately.

Parameters

`[in,out]` ***event*** Event handle as returned by rtdm_event_init()

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: possible.

References xnpod_schedule(), xnselect_signal(), and xnsynch_flush().

5.29.3.7 int rtdm_event_timedwait (rtdm_event_t * *event*, nanosecs_rel_t *timeout*, rtdm_toseq_t * *timeout_seq*)

Wait on event occurrence with timeout.

This function waits or tests for the occurence of the given event, taking the provided timeout into account. On successful return, the event is reset.

Parameters

`[in,out]` ***event*** Event handle as returned by rtdm_event_init()

`[in]` ***timeout*** Relative timeout in nanoseconds, see RTDM_TIMEOUT_xxx for special values

`[in,out]` ***timeout_seq*** Handle of a timeout sequence as returned by rtdm_toseq_init() or NULL

Returns

0 on success, otherwise:

- -ETIMEDOUT is returned if the if the request has not been satisfied within the specified amount of time.

- -EINTR is returned if calling task has been unblock by a signal or explicitly via rtdm_task_-unblock().
- -EIDRM is returned if *event* has been destroyed.
- -EPERM *may* be returned if an illegal invocation environment is detected.
- -EWOULDBLOCK is returned if a negative *timeout* (i.e., non-blocking operation) has been specified.

Environments:

This service can be called from:

- Kernel-based task
- User-space task (RT)

Rescheduling: possible.

References XNBREAK, XNRMID, xnselect_signal(), xnsynch_sleep_on(), and XNTIMEO.

Referenced by rtdm_event_wait().

5.29.3.8 int rtdm_event_wait (rtdm_event_t * *event*)

Wait on event occurrence.

This is the light-weight version of rtdm_event_timedwait(), implying an infinite timeout.

Parameters

`[in,out]` ***event*** Event handle as returned by rtdm_event_init()

Returns

0 on success, otherwise:

- -EINTR is returned if calling task has been unblock by a signal or explicitly via rtdm_task_-unblock().
- -EIDRM is returned if *event* has been destroyed.
- -EPERM *may* be returned if an illegal invocation environment is detected.

Environments:

This service can be called from:

- Kernel-based task
- User-space task (RT)

Rescheduling: possible.

References rtdm_event_timedwait().

5.29.3.9 void rtdm_mutex_destroy (rtdm_mutex_t * *mutex*)

Destroy a mutex.

Parameters

`[in,out]` ***mutex*** Mutex handle as returned by rtdm_mutex_init()

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: possible.

5.29.3.10 void rtdm_mutex_init (rtdm_mutex_t * *mutex*)

Initialise a mutex.

This function initalises a basic mutex with priority inversion protection. "Basic", as it does not allow a mutex owner to recursively lock the same mutex again.

Parameters

`[in,out]` ***mutex*** Mutex handle

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

References xnsynch_init().

5.29.3.11 int rtdm_mutex_lock (rtdm_mutex_t * *mutex*)

Request a mutex.

This is the light-weight version of rtdm_mutex_timedlock(), implying an infinite timeout.

Parameters

`[in,out]` ***mutex*** Mutex handle as returned by rtdm_mutex_init()

Returns

0 on success, otherwise:

- -EIDRM is returned if *mutex* has been destroyed.
- -EPERM *may* be returned if an illegal invocation environment is detected.

Environments:

This service can be called from:

- Kernel-based task
- User-space task (RT)

Rescheduling: possible.

References rtdm_mutex_timedlock().

5.29.3.12 int rtdm_mutex_timedlock (rtdm_mutex_t * *mutex*, nanosecs_rel_t *timeout*, rtdm_toseq_t * *timeout_seq*)

Request a mutex with timeout.

This function tries to acquire the given mutex. If it is not available, the caller is blocked unless non-blocking operation was selected.

Parameters

`[in,out]` ***mutex*** Mutex handle as returned by rtdm_mutex_init()

`[in]` ***timeout*** Relative timeout in nanoseconds, see RTDM_TIMEOUT_xxx for special values

`[in,out]` ***timeout_seq*** Handle of a timeout sequence as returned by rtdm_toseq_init() or NULL

Returns

0 on success, otherwise:

- -ETIMEDOUT is returned if the if the request has not been satisfied within the specified amount of time.
- -EWOULDBLOCK is returned if *timeout* is negative and the semaphore value is currently not positive.
- -EIDRM is returned if *mutex* has been destroyed.
- -EPERM *may* be returned if an illegal invocation environment is detected.

Environments:

This service can be called from:

- Kernel-based task

- User-space task (RT)

Rescheduling: possible.

References XNBREAK, XNRMID, xnsynch_acquire(), and XNTIMEO.

Referenced by rtdm_mutex_lock().

5.29.3.13 void rtdm_mutex_unlock (rtdm_mutex_t * *mutex*)

Release a mutex.

This function releases the given mutex, waking up a potential waiter which was blocked upon rtdm_mutex_lock() or rtdm_mutex_timedlock().

Parameters

`[in,out]` ***mutex*** Mutex handle as returned by rtdm_mutex_init()

Environments:

This service can be called from:

- Kernel-based task
- User-space task (RT)

Rescheduling: possible.

5.29.3.14 int rtdm_select_bind (int *fd*, rtdm_selector_t * *selector*, enum rtdm_selecttype *type*, unsigned *fd_index*)

Bind a selector to specified event types of a given file descriptor.

For internal use only.

This function is invoked by higher RTOS layers implementing select-like services. It shall not be called directly by RTDM drivers.

Parameters

`[in]` ***fd*** File descriptor to bind to

`[in,out]` ***selector*** Selector object that shall be bound to the given event

`[in]` ***type*** Event type the caller is interested in

`[in]` ***fd_index*** Index in the file descriptor set of the caller

Returns

0 on success, otherwise:

- -EBADF is returned if the file descriptor *fd* cannot be resolved.
- -EINVAL is returned if *type* or *fd_index* are invalid.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

References rtdm_dev_context::ops, rtdm_context_get(), rtdm_context_unlock(), and rtdm_-operations::select_bind.

5.29.3.15 void rtdm_sem_destroy (rtdm_sem_t * *sem*)

Destroy a semaphore.

Parameters

`[in,out]` *sem* Semaphore handle as returned by rtdm_sem_init()

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: possible.

5.29.3.16 int rtdm_sem_down (rtdm_sem_t * *sem*)

Decrement a semaphore.

This is the light-weight version of rtdm_sem_timeddown(), implying an infinite timeout.

Parameters

`[in,out]` *sem* Semaphore handle as returned by rtdm_sem_init()

Returns

0 on success, otherwise:

- -EINTR is returned if calling task has been unblock by a signal or explicitly via rtdm_task_-unblock().
- -EIDRM is returned if *sem* has been destroyed.
- -EPERM *may* be returned if an illegal invocation environment is detected.

Environments:

This service can be called from:

- Kernel-based task
- User-space task (RT)

Rescheduling: possible.

References rtdm_sem_timeddown().

5.29.3.17 void rtdm_sem_init (rtdm_sem_t * *sem*, unsigned long *value*)

Initialise a semaphore.

Parameters

`[in,out]` ***sem*** Semaphore handle

`[in]` ***value*** Initial value of the semaphore

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

References xnselect_init(), and xnsynch_init().

5.29.3.18 int rtdm_sem_select_bind (rtdm_sem_t * *sem*, rtdm_selector_t * *selector*, enum rtdm_selecttype *type*, unsigned *fd_index*)

Bind a selector to a semaphore.

This functions binds the given selector to the semaphore so that the former is notified when the semaphore state changes. Typically the select binding handler will invoke this service.

Parameters

`[in,out]` ***sem*** Semaphore handle as returned by rtdm_sem_init()

`[in,out]` ***selector*** Selector as passed to the select binding handler

`[in]` ***type*** Type of the bound event as passed to the select binding handler

`[in]` ***fd_index*** File descriptor index as passed to the select binding handler

Returns

0 on success, otherwise:

- -ENOMEM is returned if there is insufficient memory to establish the dynamic binding.

- -EINVAL is returned if *type* or *fd_index* are invalid.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

References xnselect_bind().

5.29.3.19 int rtdm_sem_timeddown (rtdm_sem_t * *sem*, nanosecs_rel_t *timeout*, rtdm_toseq_t * *timeout_seq*)

Decrement a semaphore with timeout.

This function tries to decrement the given semphore's value if it is positive on entry. If not, the caller is blocked unless non-blocking operation was selected.

Parameters

`[in,out]` ***sem*** Semaphore handle as returned by rtdm_sem_init()

`[in]` ***timeout*** Relative timeout in nanoseconds, see RTDM_TIMEOUT_xxx for special values

`[in,out]` ***timeout_seq*** Handle of a timeout sequence as returned by rtdm_toseq_init() or NULL

Returns

0 on success, otherwise:

- -ETIMEDOUT is returned if the if the request has not been satisfied within the specified amount of time.
- -EWOULDBLOCK is returned if *timeout* is negative and the semaphore value is currently not positive.
- -EINTR is returned if calling task has been unblock by a signal or explicitly via rtdm_task_-unblock().
- -EIDRM is returned if *sem* has been destroyed.
- -EPERM *may* be returned if an illegal invocation environment is detected.

Environments:

This service can be called from:

- Kernel-based task

- User-space task (RT)

Rescheduling: possible.

References sem_timedwait(), XNBREAK, XNRMID, xnselect_signal(), xnsynch_sleep_on(), and XNTIMEO.

Referenced by rtdm_sem_down().

5.29.3.20 void rtdm_sem_up (rtdm_sem_t * *sem*)

Increment a semaphore.

This function increments the given semphore's value, waking up a potential waiter which was blocked upon rtdm_sem_down().

Parameters

`[in,out]` ***sem*** Semaphore handle as returned by rtdm_sem_init()

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: possible.

References xnpod_schedule(), xnselect_signal(), and xnsynch_wakeup_one_sleeper().

5.29.3.21 void rtdm_toseq_init (rtdm_toseq_t * *timeout_seq*, nanosecs_rel_t *timeout*)

Initialise a timeout sequence.

This service initialises a timeout sequence handle according to the given timeout value. Timeout sequences allow to maintain a continuous *timeout* across multiple calls of blocking synchronisation services. A typical application scenario is given below.

Parameters

`[in,out]` ***timeout_seq*** Timeout sequence handle

`[in]` ***timeout*** Relative timeout in nanoseconds, see RTDM_TIMEOUT_xxx for special values

Application Scenario:

```
int device_service_routine(...)
{
        rtdm_toseq_t timeout_seq;
        ...

        rtdm_toseq_init(&timeout_seq, timeout);
```

```
        ...
        while (received < requested) {
                ret = rtdm_event_timedwait(&data_available, timeout, &timeout_seq
      );
                if (ret < 0) // including -ETIMEDOUT
                        break;

                // receive some data
                ...
        }
        ...
}
```

Using a timeout sequence in such a scenario avoids that the user-provided relative `timeout` is restarted on every call to rtdm_event_timedwait(), potentially causing an overall delay that is larger than specified by `timeout`. Moreover, all functions supporting timeout sequences also interpret special timeout values (infinite and non-blocking), disburdening the driver developer from handling them separately.

Environments:

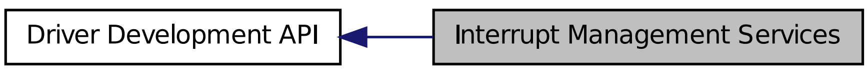
This service can be called from:

- Kernel-based task
- User-space task (RT)

Rescheduling: never.

5.30 Interrupt Management Services

Collaboration diagram for Interrupt Management Services:



Defines

- #define rtdm_irq_get_arg(irq_handle, type) ((type *)irq_handle->cookie)
  Retrieve IRQ handler argument.

Typedefs

- typedef int(* rtdm_irq_handler_t)(rtdm_irq_t *irq_handle)
  Interrupt handler.

Functions

- int rtdm_irq_request (rtdm_irq_t *irq_handle, unsigned int irq_no, rtdm_irq_handler_t handler, unsigned long flags, const char *device_name, void *arg)

  Register an interrupt handler.

- int rtdm_irq_free (rtdm_irq_t *irq_handle)

  Release an interrupt handler.

- int rtdm_irq_enable (rtdm_irq_t *irq_handle)

  Enable interrupt line.

- int rtdm_irq_disable (rtdm_irq_t *irq_handle)

  Disable interrupt line.

RTDM_IRQTYPE_xxx

Interrupt registrations flags

- #define RTDM_IRQTYPE_SHARED XN_ISR_SHARED

  Enable IRQ-sharing with other real-time drivers.

- #define RTDM_IRQTYPE_EDGE XN_ISR_EDGE

  Mark IRQ as edge-triggered, relevant for correct handling of shared edge-triggered IRQs.

RTDM_IRQ_xxx

Return flags of interrupt handlers

- #define RTDM_IRQ_NONE XN_ISR_NONE

  Unhandled interrupt.

- #define RTDM_IRQ_HANDLED XN_ISR_HANDLED

  Denote handled interrupt.

5.30.1 Define Documentation

5.30.1.1 #define rtdm_irq_get_arg(*irq_handle*, *type*) ((type *)irq_handle->cookie)

Retrieve IRQ handler argument.

Parameters

irq_handle IRQ handle

type Type of the pointer to return

Returns

The argument pointer registered on rtdm_irq_request() is returned, type-casted to the specified *type*.

Environments:

This service can be called from:

- Interrupt service routine

Rescheduling: never.

5.30.2 Typedef Documentation

5.30.2.1 typedef int(∗ rtdm_irq_handler_t)(rtdm_irq_t ∗irq_handle)

Interrupt handler.

Parameters

`[in]` ***irq_handle*** IRQ handle as returned by rtdm_irq_request()

Returns

0 or a combination of RTDM_IRQ_xxx flags

5.30.3 Function Documentation

5.30.3.1 int rtdm_irq_disable (rtdm_irq_t ∗ *irq_handle*)

Disable interrupt line.

Parameters

`[in,out]` ***irq_handle*** IRQ handle as returned by rtdm_irq_request()

Returns

0 on success, otherwise negative error code

Note

This service is for exceptional use only. Drivers should always prefer interrupt masking at device level (via corresponding control registers etc.) over masking at line level. Keep in mind that the latter is incompatible with IRQ line sharing and can also be more costly as interrupt controller access requires broader synchronization.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine

- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.30.3.2 int rtdm_irq_enable (rtdm_irq_t * *irq_handle*)

Enable interrupt line.

Parameters

`[in,out]` ***irq_handle*** IRQ handle as returned by rtdm_irq_request()

Returns

0 on success, otherwise negative error code

Note

This service is for exceptional use only. Drivers should always prefer interrupt masking at device level (via corresponding control registers etc.) over masking at line level. Keep in mind that the latter is incompatible with IRQ line sharing and can also be more costly as interrupt controller access requires broader synchronization.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: possible.

5.30.3.3 int rtdm_irq_free (rtdm_irq_t * *irq_handle*)

Release an interrupt handler.

Parameters

`[in,out]` ***irq_handle*** IRQ handle as returned by rtdm_irq_request()

Returns

0 on success, otherwise negative error code

Note

The caller is responsible for shutting down the IRQ source at device level before invoking this service. In turn, rtdm_irq_free ensures that any pending event on the given IRQ line is fully processed on return from this service.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.30.3.4 int rtdm_irq_request (rtdm_irq_t * *irq_handle*, unsigned int *irq_no*, rtdm_irq_handler_t *handler*, unsigned long *flags*, const char * *device_name*, void * *arg*)

Register an interrupt handler.

This function registers the provided handler with an IRQ line and enables the line.

Parameters

`[in,out]` ***irq_handle*** IRQ handle

`[in]` ***irq_no*** Line number of the addressed IRQ

`[in]` ***handler*** Interrupt handler

`[in]` ***flags*** Registration flags, see RTDM_IRQTYPE_xxx for details

`[in]` ***device_name*** Device name to show up in real-time IRQ lists

`[in]` ***arg*** Pointer to be passed to the interrupt handler on invocation

Returns

0 on success, otherwise:

- -EINVAL is returned if an invalid parameter was passed.
- -EBUSY is returned if the specified IRQ line is already in use.

Environments:

This service can be called from:

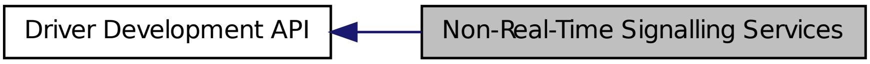
- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

References xnintr_attach(), xnintr_enable(), and xnintr_init().

5.31 Non-Real-Time Signalling Services

Collaboration diagram for Non-Real-Time Signalling Services:



Typedefs

- typedef void(* rtdm_nrtsig_handler_t)(rtdm_nrtsig_t nrt_sig, void *arg)

  Non-real-time signal handler.

Functions

- int rtdm_nrtsig_init (rtdm_nrtsig_t *nrt_sig, rtdm_nrtsig_handler_t handler, void *arg)

  Register a non-real-time signal handler.

- void rtdm_nrtsig_destroy (rtdm_nrtsig_t *nrt_sig)

  Release a non-realtime signal handler.

- void rtdm_nrtsig_pend (rtdm_nrtsig_t *nrt_sig)

  Trigger non-real-time signal.

5.31.1 Detailed Description

These services provide a mechanism to request the execution of a specified handler in non-real-time context. The triggering can safely be performed in real-time context without suffering from unknown delays. The handler execution will be deferred until the next time the real-time subsystem releases the CPU to the non-real-time part.

5.31.2 Typedef Documentation

5.31.2.1 typedef void(* rtdm_nrtsig_handler_t)(rtdm_nrtsig_t nrt_sig, void *arg)

Non-real-time signal handler.

Parameters

`[in]` ***nrt_sig*** Signal handle as returned by rtdm_nrtsig_init()

`[in]` ***arg*** Argument as passed to rtdm_nrtsig_init()

Note

The signal handler will run in soft-IRQ context of the non-real-time subsystem. Note the implications of this context, e.g. no invocation of blocking operations.

5.31.3 Function Documentation

5.31.3.1 void rtdm_nrtsig_destroy (rtdm_nrtsig_t * *nrt_sig*)

Release a non-realtime signal handler.

Parameters

`[in,out]` ***nrt_sig*** Signal handle

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.31.3.2 int rtdm_nrtsig_init (rtdm_nrtsig_t * *nrt_sig*, rtdm_nrtsig_handler_t *handler*, void * *arg*)

Register a non-real-time signal handler.

Parameters

`[in,out]` ***nrt_sig*** Signal handle

`[in]` ***handler*** Non-real-time signal handler

`[in]` ***arg*** Custom argument passed to `handler()` on each invocation

Returns

0 on success, otherwise:

- -EAGAIN is returned if no free signal slot is available.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.31.3.3 void rtdm_nrtsig_pend (rtdm_nrtsig_t * *nrt_sig*)

Trigger non-real-time signal.

Parameters

`[in,out]` ***nrt_sig*** Signal handle

Environments:

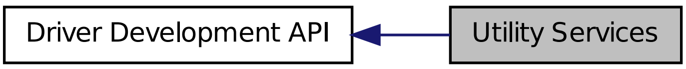
This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never in real-time context, possible in non-real-time environments.

5.32 Utility Services

Collaboration diagram for Utility Services:



Functions

- int rtdm_mmap_to_user (rtdm_user_info_t *user_info, void *src_addr, size_t len, int prot, void **pptr, struct vm_operations_struct *vm_ops, void *vm_private_data)

  Map a kernel memory range into the address space of the user.

- int rtdm_iomap_to_user (rtdm_user_info_t *user_info, phys_addr_t src_addr, size_t len, int prot, void **pptr, struct vm_operations_struct *vm_ops, void *vm_private_data)

  Map an I/O memory range into the address space of the user.

- int rtdm_munmap (rtdm_user_info_t *user_info, void *ptr, size_t len)

  Unmap a user memory range.

- void rtdm_printk (const char *format,...)

  Real-time safe message printing on kernel console.

- void * rtdm_malloc (size_t size)
  Allocate memory block in real-time context.
- void rtdm_free (void *ptr)
  Release real-time memory block.
- int rtdm_read_user_ok (rtdm_user_info_t *user_info, const void __user *ptr, size_t size)
  Check if read access to user-space memory block is safe.
- int rtdm_rw_user_ok (rtdm_user_info_t *user_info, const void __user *ptr, size_t size)
  Check if read/write access to user-space memory block is safe.
- int rtdm_copy_from_user (rtdm_user_info_t *user_info, void *dst, const void __user *src, size_t size)
  Copy user-space memory block to specified buffer.
- int rtdm_safe_copy_from_user (rtdm_user_info_t *user_info, void *dst, const void __user *src, size_t size)
  Check if read access to user-space memory block and copy it to specified buffer.
- int rtdm_copy_to_user (rtdm_user_info_t *user_info, void __user *dst, const void *src, size_t size)
  Copy specified buffer to user-space memory block.
- int rtdm_safe_copy_to_user (rtdm_user_info_t *user_info, void __user *dst, const void *src, size_t size)
  Check if read/write access to user-space memory block is safe and copy specified buffer to it.
- int rtdm_strncpy_from_user (rtdm_user_info_t *user_info, char *dst, const char __user *src, size_t count)
  Copy user-space string to specified buffer.
- int rtdm_in_rt_context (void)
  Test if running in a real-time task.
- int rtdm_rt_capable (rtdm_user_info_t *user_info)
  Test if the caller is capable of running in real-time context.

5.32.1 Function Documentation

5.32.1.1 int rtdm_copy_from_user (rtdm_user_info_t * *user_info*, void * *dst*, const void __user * *src*, size_t *size*)

Copy user-space memory block to specified buffer.

Parameters

`[in]` ***user_info*** User information pointer as passed to the invoked device operation handler `[in]` *dst* Destination buffer address

`[in]` *src* Address of the user-space memory block

`[in]` *size* Size of the memory block

Returns

0 on success, otherwise:

- -EFAULT is returned if an invalid memory area was accessed.

Note

Before invoking this service, verify via rtdm_read_user_ok() that the provided user-space address can securely be accessed.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.32.1.2 int rtdm_copy_to_user (rtdm_user_info_t * *user_info*, void __user * *dst*, const void * *src*, size_t *size*)

Copy specified buffer to user-space memory block.

Parameters

`[in]` *user_info* User information pointer as passed to the invoked device operation handler

`[in]` *dst* Address of the user-space memory block

`[in]` *src* Source buffer address

`[in]` *size* Size of the memory block

Returns

0 on success, otherwise:

- -EFAULT is returned if an invalid memory area was accessed.

Note

Before invoking this service, verify via rtdm_rw_user_ok() that the provided user-space address can securely be accessed.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.32.1.3 void rtdm_free (void * *ptr*)

Release real-time memory block.

Parameters

`[in]` *ptr* Pointer to memory block as returned by rtdm_malloc()

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine (consider the overhead!)
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.32.1.4 int rtdm_in_rt_context (void)

Test if running in a real-time task.

Returns

Non-zero is returned if the caller resides in real-time context, 0 otherwise.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.32.1.5 int rtdm_iomap_to_user (rtdm_user_info_t * *user_info*, phys_addr_t *src_addr*, size_t *len*, int *prot*, void ** *pptr*, struct vm_operations_struct * *vm_ops*, void * *vm_private_data*)

Map an I/O memory range into the address space of the user.

Parameters

- `[in]` ***user_info*** User information pointer as passed to the invoked device operation handler
- `[in]` ***src_addr*** physical I/O address to be mapped
- `[in]` ***len*** Length of the memory range
- `[in]` ***prot*** Protection flags for the user's memory range, typically either PROT_READ or PROT_READ|PROT_WRITE
- `[in,out]` ***pptr*** Address of a pointer containing the desired user address or NULL on entry and the finally assigned address on return
- `[in]` ***vm_ops*** vm_operations to be executed on the vma_area of the user memory range or NULL
- `[in]` ***vm_private_data*** Private data to be stored in the vma_area, primarily useful for vm_-operation handlers

Returns

0 on success, otherwise (most common values):

- -EINVAL is returned if an invalid start address, size, or destination address was passed.
- -ENOMEM is returned if there is insufficient free memory or the limit of memory mapping for the user process was reached.
- -EAGAIN is returned if too much memory has been already locked by the user process.
- -EPERM *may* be returned if an illegal invocation environment is detected.

Note

RTDM supports two models for unmapping the user memory range again. One is explicit unmapping via rtdm_munmap(), either performed when the user requests it via an IOCTL etc. or when the related device is closed. The other is automatic unmapping, triggered by the user invoking standard munmap() or by the termination of the related process. To track release of the mapping and therefore relinquishment of the referenced physical memory, the caller of rtdm_iomap_to_user() can pass a vm_operations_struct on invocation, defining a close handler for the vm_area. See Linux documentaion (e.g. Linux Device Drivers book) on virtual memory management for details.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- User-space task (non-RT)

Rescheduling: possible.

5.32.1.6 void* rtdm_malloc (size_t *size*)

Allocate memory block in real-time context.

Parameters

`[in]` ***size*** Requested size of the memory block

Returns

The pointer to the allocated block is returned on success, NULL otherwise.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine (consider the overhead!)
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.32.1.7 int rtdm_mmap_to_user (rtdm_user_info_t * *user_info*, void * *src_addr*, size_t *len*, int *prot*, void ** *pptr*, struct vm_operations_struct * *vm_ops*, void * *vm_private_data*)

Map a kernel memory range into the address space of the user.

Parameters

`[in]` ***user_info*** User information pointer as passed to the invoked device operation handler

`[in]` ***src_addr*** Kernel virtual address to be mapped

`[in]` ***len*** Length of the memory range

`[in]` ***prot*** Protection flags for the user's memory range, typically either PROT_READ or PROT_READ|PROT_WRITE

`[in,out]` ***pptr*** Address of a pointer containing the desired user address or NULL on entry and the finally assigned address on return

`[in]` ***vm_ops*** vm_operations to be executed on the vma_area of the user memory range or NULL

`[in]` ***vm_private_data*** Private data to be stored in the vma_area, primarily useful for vm_-operation handlers

Returns

0 on success, otherwise (most common values):

- -EINVAL is returned if an invalid start address, size, or destination address was passed.
- -ENOMEM is returned if there is insufficient free memory or the limit of memory mapping for the user process was reached.

- -EAGAIN is returned if too much memory has been already locked by the user process.
- -EPERM *may* be returned if an illegal invocation environment is detected.

Note

This service only works on memory regions allocated via kmalloc() or vmalloc(). To map physical I/O memory to user-space use rtdm_iomap_to_user() instead.
RTDM supports two models for unmapping the user memory range again. One is explicit unmapping via rtdm_munmap(), either performed when the user requests it via an IOCTL etc. or when the related device is closed. The other is automatic unmapping, triggered by the user invoking standard munmap() or by the termination of the related process. To track release of the mapping and therefore relinquishment of the referenced physical memory, the caller of rtdm_mmap_to_user() can pass a vm_operations_struct on invocation, defining a close handler for the vm_area. See Linux documentaion (e.g. Linux Device Drivers book) on virtual memory management for details.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- User-space task (non-RT)

Rescheduling: possible.

5.32.1.8 int rtdm_munmap (rtdm_user_info_t * *user_info*, void * *ptr*, size_t *len*)

Unmap a user memory range.

Parameters

`[in]` ***user_info*** User information pointer as passed to rtdm_mmap_to_user() when requesting to map the memory range

`[in]` ***ptr*** User address or the memory range

`[in]` ***len*** Length of the memory range

Returns

0 on success, otherwise:

- -EINVAL is returned if an invalid address or size was passed.
- -EPERM *may* be returned if an illegal invocation environment is detected.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- User-space task (non-RT)

Rescheduling: possible.

5.32.1.9 void rtdm_printk (const char * *format*, ...)

Real-time safe message printing on kernel console.

Parameters

`[in]` ***format*** Format string (conforming standard `printf()`)

... Arguments referred by *format*

Returns

On success, this service returns the number of characters printed. Otherwise, a negative error code is returned.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine (consider the overhead!)
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never in real-time context, possible in non-real-time environments.

5.32.1.10 int rtdm_read_user_ok (rtdm_user_info_t * *user_info*, const void __user * *ptr*, size_t *size*)

Check if read access to user-space memory block is safe.

Parameters

`[in]` ***user_info*** User information pointer as passed to the invoked device operation handler

`[in]` ***ptr*** Address of the user-provided memory block

`[in]` ***size*** Size of the memory block

Returns

Non-zero is return when it is safe to read from the specified memory block, 0 otherwise.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.32.1.11 int rtdm_rt_capable (rtdm_user_info_t * *user_info*)

Test if the caller is capable of running in real-time context.

Parameters

`[in]` ***user_info*** User information pointer as passed to the invoked device operation handler

Returns

Non-zero is returned if the caller is able to execute in real-time context (independent of its current execution mode), 0 otherwise.

Note

This function can be used by drivers that provide different implementations for the same service depending on the execution mode of the caller. If a caller requests such a service in non-real-time context but is capable of running in real-time as well, it might be appropriate for the driver to reject the request via -ENOSYS so that RTDM can switch the caller and restart the request in real-time context.

Environments:

This service can be called from:

- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.32.1.12 int rtdm_rw_user_ok (rtdm_user_info_t * *user_info*, const void __user * *ptr*, size_t *size*)

Check if read/write access to user-space memory block is safe.

Parameters

`[in]` ***user_info*** User information pointer as passed to the invoked device operation handler
`[in]` ***ptr*** Address of the user-provided memory block
`[in]` ***size*** Size of the memory block

Returns

Non-zero is return when it is safe to read from or write to the specified memory block, 0 otherwise.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.32.1.13 int rtdm_safe_copy_from_user (rtdm_user_info_t * *user_info*, void * *dst*, const void __user * *src*, size_t *size*)

Check if read access to user-space memory block and copy it to specified buffer.

Parameters

[in] ***user_info*** User information pointer as passed to the invoked device operation handler

[in] ***dst*** Destination buffer address

[in] ***src*** Address of the user-space memory block

[in] ***size*** Size of the memory block

Returns

0 on success, otherwise:

- -EFAULT is returned if an invalid memory area was accessed.

Note

This service is a combination of rtdm_read_user_ok and rtdm_copy_from_user.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.32.1.14 int rtdm_safe_copy_to_user (rtdm_user_info_t * *user_info*, void __user * *dst*, const void * *src*, size_t *size*)

Check if read/write access to user-space memory block is safe and copy specified buffer to it.

Parameters

[in] ***user_info*** User information pointer as passed to the invoked device operation handler

[in] ***dst*** Address of the user-space memory block

[in] ***src*** Source buffer address

[in] ***size*** Size of the memory block

Returns

0 on success, otherwise:

- -EFAULT is returned if an invalid memory area was accessed.

Note

This service is a combination of rtdm_rw_user_ok and rtdm_copy_to_user.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.32.1.15 int rtdm_strncpy_from_user (rtdm_user_info_t * *user_info*, char * *dst*, const char __user * *src*, size_t *count*)

Copy user-space string to specified buffer.

Parameters

`[in]` ***user_info*** User information pointer as passed to the invoked device operation handler

`[in]` ***dst*** Destination buffer address

`[in]` ***src*** Address of the user-space string

`[in]` ***count*** Maximum number of bytes to copy, including the trailing '0'

Returns

Length of the string on success (not including the trailing '0'), otherwise:

- -EFAULT is returned if an invalid memory area was accessed.

Note

This services already includes a check of the source address, calling rtdm_read_user_ok() for *src* explicitly is not required.

Environments:

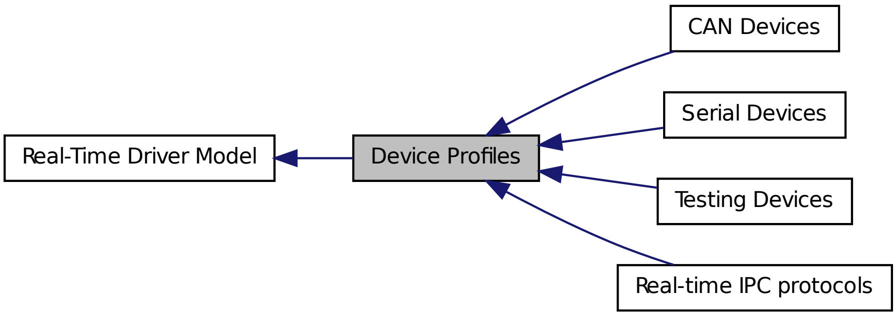
This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.33 Device Profiles

Collaboration diagram for Device Profiles:



Data Structures

- struct rtdm_device_info

    Device information.

Modules

- CAN Devices
- Real-time IPC protocols

    ***Profile Revision:** 1*

- Serial Devices
- Testing Devices

Typedefs

- typedef struct rtdm_device_info rtdm_device_info_t

    Device information.

RTDM_CLASS_xxx

Device classes

- #define **RTDM_CLASS_PARPORT** 1
- #define **RTDM_CLASS_SERIAL** 2
- #define **RTDM_CLASS_CAN** 3

- #define **RTDM_CLASS_NETWORK** 4
- #define **RTDM_CLASS_RTMAC** 5
- #define **RTDM_CLASS_TESTING** 6
- #define **RTDM_CLASS_RTIPC** 7
- #define **RTDM_CLASS_EXPERIMENTAL** 224
- #define **RTDM_CLASS_MAX** 255

Device Naming

Maximum length of device names (excluding the final null character)

- #define **RTDM_MAX_DEVNAME_LEN** 31

RTDM_PURGE_xxx_BUFFER

Flags selecting buffers to be purged

- #define **RTDM_PURGE_RX_BUFFER** 0x0001
- #define **RTDM_PURGE_TX_BUFFER** 0x0002

Common IOCTLs

The following IOCTLs are common to all device profiles.

- #define RTIOC_DEVICE_INFO _IOR(RTIOC_TYPE_COMMON, 0x00, struct rtdm_-device_info)

  Retrieve information about a device or socket.

- #define RTIOC_PURGE _IOW(RTIOC_TYPE_COMMON, 0x10, int)

  Purge internal device or socket buffers.

5.33.1 Detailed Description

Device profiles define which operation handlers a driver of a certain class has to implement, which name or protocol it has to register, which IOCTLs it has to provide, and further details. Sub-classes can be defined in order to extend a device profile with more hardware-specific functions.

5.33.2 Define Documentation

5.33.2.1 #define RTIOC_DEVICE_INFO _IOR(RTIOC_TYPE_COMMON, 0x00, struct rtdm_device_info)

Retrieve information about a device or socket.

Parameters

`[out]` *arg* Pointer to information buffer (struct rtdm_device_info)

5.33.2.2 #define RTIOC_PURGE _IOW(RTIOC_TYPE_COMMON, 0x10, int)

Purge internal device or socket buffers.

Parameters

`[in]` *arg* Purge mask, see RTDM_PURGE_xxx_BUFFER

5.34 Semaphores services.

Semaphores services.

Functions

- static int sem_destroy (struct __shadow_sem *sm)
  Destroy an unnamed semaphore.
- sem_t * sem_open (const char *name, int oflags,...)
  Open a named semaphore.
- int sem_close (struct __shadow_sem *sm)
  Close a named semaphore.
- int sem_unlink (const char *name)
  Unlink a named semaphore.
- static int sem_trywait (cobalt_sem_t *sem)
  Attempt to decrement a semaphore.
- static int sem_wait (cobalt_sem_t *sem)
  Decrement a semaphore.
- static int sem_timedwait (cobalt_sem_t *sem, const struct timespec *abs_timeout)
  Attempt, during a bounded time, to decrement a semaphore.
- static int sem_post (cobalt_sem_t *sm)
  Post a semaphore.
- int sem_getvalue (cobalt_sem_t *sem, int *value)
  Get the value of a semaphore.

5.34.1 Detailed Description

Semaphores services. Semaphores are counters for resources shared between threads. The basic operations on semaphores are: increment the counter atomically, and wait until the counter is non-null and decrement it atomically.

Semaphores have a maximum value past which they cannot be incremented. The macro *SEM_VALUE_MAX* is defined to be this maximum value.

5.34.2 Function Documentation

5.34.2.1 int sem_close (struct __shadow_sem * *sm*)

Close a named semaphore.

This service closes the semaphore *sm*. The semaphore is destroyed only when unlinked with a call to the sem_unlink() service and when each call to sem_open() matches a call to this service.

When a semaphore is destroyed, the memory it used is returned to the system heap, so that further references to this semaphore are not guaranteed to fail, as is the case for unnamed semaphores.

This service fails if *sm* is an unnamed semaphore.

Parameters

sm the semaphore to be closed.

Return values

0 on success;

-1 with *errno* set if:

- EINVAL, the semaphore *sm* is invalid or is an unnamed semaphore.

See also

`Specification.`

5.34.2.2 static int sem_destroy (struct __shadow_sem * *sm*) `[static]`

Destroy an unnamed semaphore.

This service destroys the semaphore *sm*. Threads currently blocked on *sm* are unblocked and the service they called return -1 with *errno* set to EINVAL. The semaphore is then considered invalid by all semaphore services (they all fail with *errno* set to EINVAL) except sem_init().

This service fails if *sm* is a named semaphore.

Parameters

sm the semaphore to be destroyed.

Return values

always 0 on success if SEM_WARNDEL was not mentioned via sem_init_np(). If SEM_-WARNDEL was mentioned, then a strictly positive value is returned to warn the caller if threads were pending on the semaphore, or zero otherwise.

-1 with *errno* set if:

- EINVAL, the semaphore *sm* is invalid or a named semaphore;
- EPERM, the semaphore *sm* is not process-shared and does not belong to the current process.

See also

`Specification.`

5.34.2.3 int sem_getvalue (cobalt_sem_t * *sem*, int * *value*)

Get the value of a semaphore.

This service stores at the address *value*, the current count of the semaphore *sm*. The state of the semaphore is unchanged.

If the semaphore is currently fully depleted, the value stored is zero, unless SEM_REPORT was mentioned for a non-standard semaphore (see sem_init_np()), in which case the current number of waiters is returned as the semaphore's negative value (e.g. -2 would mean the semaphore is fully depleted AND two threads are currently pending on it).

Parameters

sem a semaphore;

value address where the semaphore count will be stored on success.

Return values

0 on success;

-1 with *errno* set if:

- EINVAL, the semaphore is invalid or uninitialized;
- EPERM, the semaphore *sm* is not process-shared and does not belong to the current process.

See also

Specification.

Referenced by timer_create().

5.34.2.4 sem_t* sem_open (const char * *name*, int *oflags*, ...)

Open a named semaphore.

This service establishes a connection between the semaphore named *name* and the calling context (kernel-space as a whole, or user-space process).

If no semaphore named *name* exists and *oflags* has the *O_CREAT* bit set, the semaphore is created by this function, using two more arguments:

- a *mode* argument, of type **mode_t**, currently ignored;
- a *value* argument, of type **unsigned**, specifying the initial value of the created semaphore.

If *oflags* has the two bits *O_CREAT* and *O_EXCL* set and the semaphore already exists, this service fails.

name may be any arbitrary string, in which slashes have no particular meaning. However, for portability, using a name which starts with a slash and contains no other slash is recommended.

If sem_open() is called from the same context (kernel-space as a whole, or user-space process) several times with the same value of *name*, the same address is returned.

Parameters

name the name of the semaphore to be created;

oflags flags.

Returns

the address of the named semaphore on success;
SEM_FAILED with *errno* set if:

- ENAMETOOLONG, the length of the *name* argument exceeds 64 characters;
- EEXIST, the bits *O_CREAT* and *O_EXCL* were set in *oflags* and the named semaphore already exists;
- ENOENT, the bit *O_CREAT* is not set in *oflags* and the named semaphore does not exist;
- ENOSPC, insufficient memory exists in the system heap to create the semaphore, increase CONFIG_XENO_OPT_SYS_HEAPSZ;
- EINVAL, the *value* argument exceeds *SEM_VALUE_MAX*.

See also

`Specification.`

5.34.2.5 static int sem_post (cobalt_sem_t * *sm*) `[static]`

Post a semaphore.

This service posts the semaphore *sm*.

If no thread is currently blocked on this semaphore, its count is incremented unless "pulse" mode is enabled for it (see sem_init_np(), SEM_PULSE). If a thread is blocked on the semaphore, the thread heading the wait queue is unblocked.

Parameters

sm the semaphore to be signaled.

Return values

0 on success;

-1 with errno set if:

- EINVAL, the specified semaphore is invalid or uninitialized;
- EPERM, the semaphore *sm* is not process-shared and does not belong to the current process;
- EAGAIN, the semaphore count is *SEM_VALUE_MAX*.

See also

`Specification.`

5.34.2.6 static int sem_timedwait (cobalt_sem_t * *sem*, const struct timespec * *abs_timeout*) `[static]`

Attempt, during a bounded time, to decrement a semaphore.

This service is equivalent to sem_wait(), except that the caller is only blocked until the timeout *abs_timeout* expires.

Parameters

sem the semaphore to be decremented;

abs_timeout the timeout, expressed as an absolute value of the relevant clock for the semaphore, either CLOCK_MONOTONIC if SEM_RAWCLOCK was mentioned via sem_init_np(), or CLOCK_REALTIME otherwise.

Return values

0 on success;

-1 with *errno* set if:
- EPERM, the caller context is invalid;
- EINVAL, the semaphore is invalid or uninitialized;
- EINVAL, the specified timeout is invalid;
- EPERM, the semaphore *sm* is not process-shared and does not belong to the current process;
- EINTR, the caller was interrupted by a signal while blocked in this service;
- ETIMEDOUT, the semaphore could not be decremented and the specified timeout expired.

Valid contexts:

- Xenomai kernel-space thread,
- Xenomai user-space thread (switches to primary mode).

See also

`Specification.`

Referenced by rtdm_sem_timeddown().

5.34.2.7 static int sem_trywait (cobalt_sem_t * *sem*) `[static]`

Attempt to decrement a semaphore.

This service is equivalent to sem_wait(), except that it returns immediately if the semaphore *sm* is currently depleted, and that it is not a cancellation point.

Parameters

sem the semaphore to be decremented.

Return values

0 on success;

-1 with *errno* set if:
- EINVAL, the specified semaphore is invalid or uninitialized;
- EPERM, the semaphore *sm* is not process-shared and does not belong to the current process;
- EAGAIN, the specified semaphore is currently fully depleted.

*

See also

`Specification.`

5.34.2.8 int sem_unlink (const char * *name*)

Unlink a named semaphore.

This service unlinks the semaphore named *name*. This semaphore is not destroyed until all references obtained with sem_open() are closed by calling sem_close(). However, the unlinked semaphore may no longer be reached with the sem_open() service.

When a semaphore is destroyed, the memory it used is returned to the system heap, so that further references to this semaphore are not guaranteed to fail, as is the case for unnamed semaphores.

Parameters

name the name of the semaphore to be unlinked.

Return values

0 on success;

-1 with *errno* set if:

- ENAMETOOLONG, the length of the *name* argument exceeds 64 characters;
- ENOENT, the named semaphore does not exist.

See also

`Specification.`

5.34.2.9 static int sem_wait (cobalt_sem_t * *sem*) [static]

Decrement a semaphore.

This service decrements the semaphore *sm* if it is currently if its value is greater than 0. If the semaphore's value is currently zero, the calling thread is suspended until the semaphore is posted, or a signal is delivered to the calling thread.

This service is a cancellation point for Xenomai POSIX skin threads (created with the pthread_-create() service). When such a thread is cancelled while blocked in a call to this service, the semaphore state is left unchanged before the cancellation cleanup handlers are called.

Parameters

sem the semaphore to be decremented.

Return values

0 on success;

-1 with *errno* set if:

- EPERM, the caller context is invalid;
- EINVAL, the semaphore is invalid or uninitialized;
- EPERM, the semaphore *sm* is not process-shared and does not belong to the current process;
- EINTR, the caller was interrupted by a signal while blocked in this service.

Valid contexts:

- Xenomai kernel-space thread,

- Xenomai user-space thread (switches to primary mode).

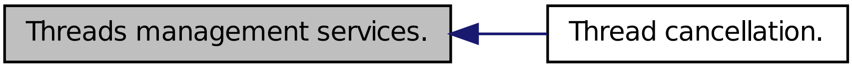
See also

Specification.

5.35 Threads management services.

Threads management services.

Collaboration diagram for Threads management services.:



Modules

- Thread cancellation.

  Thread cancellation.

Functions

- static int pthread_getschedparam (pthread_t tid, int *pol, struct sched_param *par)

  Get the scheduling policy and parameters of the specified thread.

- static int pthread_getschedparam_ex (pthread_t tid, int *pol, struct sched_param_ex *par)

  Get the extended scheduling policy and parameters of the specified thread.

- static int pthread_create (pthread_t *tid, const pthread_attr_t *attr)

  Create a thread.

- static int pthread_make_periodic_np (pthread_t thread, clockid_t clock_id, struct timespec *starttp, struct timespec *periodtp)

  Make a thread periodic.

- static int pthread_set_mode_np (int clrmask, int setmask, int *mode_r)

  Set the mode of the current thread.

- static int pthread_set_name_np (pthread_t thread, const char *name)

  Set a thread name.

- static int pthread_setschedparam (pthread_t tid, int pol, const struct sched_param *par)

  Set the scheduling policy and parameters of the specified thread.

- static int pthread_setschedparam_ex (pthread_t tid, int pol, const struct sched_param_ex *par)

  Set the extended scheduling policy and parameters of the specified thread.

5.35.1 Detailed Description

Threads management services.

See also

`Specification.`

5.35.2 Function Documentation

5.35.2.1 static int pthread_create (pthread_t * *tid*, const pthread_attr_t * *attr*) `[inline, static]`

Create a thread.

This service creates a Cobalt thread control block. The created thread may use Cobalt API services.

The new thread control block can be mapped over a regular Linux thread, forming a Xenomai shadow.

The new thread signal mask is inherited from the current thread, if it was also created with pthread_create(), otherwise the new thread signal mask is empty.

Other attributes of the new thread depend on the *attr* argument. If *attr* is null, default values for these attributes are used.

Returning from the *start* routine has the same effect as calling pthread_exit() with the return value.

Parameters

tid address where the identifier of the new thread will be stored on success;

attr thread attributes;

Returns

0 on success;
an error number if:

- EINVAL, *attr* is invalid;
- EAGAIN, insufficient memory exists in the system heap to create a new thread, increase CONFIG_XENO_OPT_SYS_HEAPSZ;
- EINVAL, thread attribute *inheritsched* is set to PTHREAD_INHERIT_SCHED and the calling thread does not belong to the POSIX skin;

See also

`Specification.`

Note

When creating or shadowing a Xenomai thread for the first time in user-space, Xenomai installs a handler for the SIGWINCH signal. If you had installed a handler before that, it will be automatically called by Xenomai for SIGWINCH signals that it has not sent.

If, however, you install a signal handler for SIGWINCH after creating or shadowing the first Xenomai thread, you have to explicitly call the function xeno_sigwinch_handler at the beginning of your signal handler, using its return to know if the signal was in fact an internal signal of Xenomai (in which case it returns 1), or if you should handle the signal (in which case it returns 0). xeno_sigwinch_handler prototype is:

int xeno_sigwinch_handler(int sig, siginfo_t *si, void *ctxt);

Which means that you should register your handler with sigaction, using the SA_SIGINFO flag, and pass all the arguments you received to xeno_sigwinch_handler.

References pthread_getschedparam_ex(), xnpod_init_thread(), xnpod_set_thread_tslice(), and xnsynch_init().

5.35.2.2 static int pthread_getschedparam (pthread_t *tid*, int * *pol*, struct sched_param * *par*) `[inline, static]`

Get the scheduling policy and parameters of the specified thread.

This service returns, at the addresses *pol* and *par*, the current scheduling policy and scheduling parameters (i.e. priority) of the Xenomai POSIX skin thread *tid*. If this service is called from user-space and *tid* is not the identifier of a Xenomai POSIX skin thread, this service fallback to Linux regular pthread_getschedparam service.

Parameters

tid target thread;

pol address where the scheduling policy of *tid* is stored on success;

par address where the scheduling parameters of *tid* is stored on success.

Returns

0 on success;
an error number if:

- ESRCH, *tid* is invalid.

See also

`Specification.`

5.35.2.3 static int pthread_getschedparam_ex (pthread_t *tid*, int * *pol*, struct sched_param_ex * *par*) `[inline, static]`

Get the extended scheduling policy and parameters of the specified thread.

This service is an extended version of pthread_getschedparam(), that also supports Xenomai-specific or additional POSIX scheduling policies, which are not available with the host Linux environment.

Typically, SCHED_SPORADIC parameters can be retrieved from this call.

Parameters

tid target thread;

pol address where the scheduling policy of *tid* is stored on success;

par address where the scheduling parameters of *tid* is stored on success.

Returns

0 on success;
an error number if:

- ESRCH, *tid* is invalid.

See also

`Specification.`

References XNRRB.

Referenced by pthread_create().

5.35.2.4 static int pthread_make_periodic_np (pthread_t *thread*, clockid_t *clock_id*, struct timespec * *starttp*, struct timespec * *periodtp*) `[inline, static]`

Make a thread periodic.

This service make the POSIX skin thread *thread* periodic.

This service is a non-portable extension of the POSIX interface.

Parameters

thread thread identifier. This thread is immediately delayed until the first periodic release point is reached.

clock_id clock identifier, either CLOCK_REALTIME, CLOCK_MONOTONIC or CLOCK_-MONOTONIC_RAW.

starttp start time, expressed as an absolute value of the clock *clock_id*. The affected thread will be delayed until this point is reached.

periodtp period, expressed as a time interval.

Returns

0 on success;
an error number if:

- ESRCH, *thread* is invalid;
- ETIMEDOUT, the start time has already passed.
- ENOTSUP, the specified clock is unsupported;

Rescheduling: always, until the *starttp* start time has been reached.

References xnpod_set_thread_periodic().

5.35.2.5 static int pthread_set_mode_np (int *clrmask*, int *setmask*, int * *mode_r*) [inline, static]

Set the mode of the current thread.

This service sets the mode of the calling thread. *clrmask* and *setmask* are two bit masks which are respectively cleared and set in the calling thread status. They are a bitwise OR of the following values:

- PTHREAD_LOCK_SCHED, when set, locks the scheduler, which prevents the current thread from being switched out until the scheduler is unlocked;
- PTHREAD_WARNSW, when set, causes the signal SIGXCPU to be sent to the current thread, whenever it involontary switches to secondary mode;
- PTHREAD_CONFORMING can be passed in *setmask* to switch the current user-space task to its preferred runtime mode. The only meaningful use of this switch is to force a real-time shadow back to primary mode. Any other use either cause to a nop, or an error.

PTHREAD_LOCK_SCHED is valid for any Xenomai thread, the other bits are only valid for Xenomai user-space threads.

This service is a non-portable extension of the POSIX interface.

Parameters

clrmask set of bits to be cleared;

setmask set of bits to be set.

mode_r If non-NULL, *mode_r* must be a pointer to a memory location which will be written upon success with the previous set of active mode bits. If NULL, the previous set of active mode bits will not be returned.

Returns

0 on success;
an error number if:

- EINVAL, some bit in *clrmask* or *setmask* is invalid.

References XNLOCK, xnpod_schedule(), xnpod_set_thread_mode(), and XNSHADOW.

5.35.2.6 static int pthread_set_name_np (pthread_t *thread*, const char * *name*) [inline, static]

Set a thread name.

This service set to *name*, the name of *thread*. This name is used for displaying information in /proc/xenomai/sched.

This service is a non-portable extension of the POSIX interface.

Parameters

thread target thread;

name name of the thread.

Returns

0 on success;
an error number if:

- ESRCH, *thread* is invalid.

5.35.2.7 static int pthread_setschedparam (pthread_t *tid*, int *pol*, const struct sched_param * *par*) [inline, static]

Set the scheduling policy and parameters of the specified thread.

This service set the scheduling policy of the Xenomai POSIX skin thread *tid* to the value *pol*, and its scheduling parameters (i.e. its priority) to the value pointed to by *par*.

When used in user-space, passing the current thread ID as *tid* argument, this service turns the current thread into a Xenomai POSIX skin thread. If *tid* is neither the identifier of the current thread nor the identifier of a Xenomai POSIX skin thread this service falls back to the regular pthread_setschedparam() service, hereby causing the current thread to switch to secondary mode if it is Xenomai thread.

Parameters

tid target thread;

pol scheduling policy, one of SCHED_FIFO, SCHED_COBALT, SCHED_RR or SCHED_-OTHER;

par scheduling parameters address.

Returns

0 on success;
an error number if:

- ESRCH, *tid* is invalid;
- EINVAL, *pol* or *par->sched_priority* is invalid;
- EAGAIN, in user-space, insufficient memory exists in the system heap, increase CONFIG_XENO_OPT_SYS_HEAPSZ;
- EFAULT, in user-space, *par* is an invalid address;
- EPERM, in user-space, the calling process does not have superuser permissions.

See also

`Specification.`

Note

When creating or shadowing a Xenomai thread for the first time in user-space, Xenomai installs a handler for the SIGWINCH signal. If you had installed a handler before that, it will be automatically called by Xenomai for SIGWINCH signals that it has not sent.

If, however, you install a signal handler for SIGWINCH after creating or shadowing the first Xenomai thread, you have to explicitly call the function xeno_sigwinch_handler at the beginning of your signal handler, using its return to know if the signal was in fact an internal signal of

Xenomai (in which case it returns 1), or if you should handle the signal (in which case it returns 0). xeno_sigwinch_handler prototype is:

int xeno_sigwinch_handler(int sig, siginfo_t *si, void *ctxt);

Which means that you should register your handler with sigaction, using the SA_SIGINFO flag, and pass all the arguments you received to xeno_sigwinch_handler.

References xnpod_schedule(), xnpod_set_thread_schedparam(), and xnpod_set_thread_tslice().

Referenced by pthread_setschedparam_ex().

5.35.2.8 static int pthread_setschedparam_ex (pthread_t *tid*, int *pol*, const struct sched_param_ex * *par*) `[inline, static]`

Set the extended scheduling policy and parameters of the specified thread.

This service is an extended version of pthread_setschedparam(), that supports Xenomai-specific or additional scheduling policies, which are not available with the host Linux environment.

Typically, a Xenomai thread policy can be set to SCHED_SPORADIC using this call.

Parameters

tid target thread;

pol address where the scheduling policy of *tid* is stored on success;

par address where the scheduling parameters of *tid* is stored on success.

Returns

0 on success;
an error number if:

- ESRCH, *tid* is invalid.
- EINVAL, *par* contains invalid parameters.
- ENOMEM, lack of memory to perform the operation.

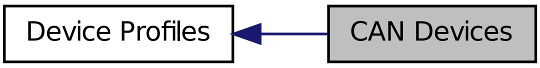
See also

`Specification.`

References pthread_setschedparam(), xnpod_schedule(), xnpod_set_thread_schedparam(), and xnpod_set_thread_tslice().

5.36 CAN Devices

Collaboration diagram for CAN Devices:



Data Structures

- struct can_bittime_std
  Standard bit-time parameters according to Bosch.
- struct can_bittime_btr
  Hardware-specific BTR bit-times.
- struct can_bittime
  Custom CAN bit-time definition.
- struct can_filter
  Filter for reception of CAN messages.
- struct sockaddr_can
  Socket address structure for the CAN address family.
- struct can_frame
  Raw CAN frame.

Files

- file rtcan.h
  Real-Time Driver Model for RT-Socket-CAN, CAN device profile header.

Defines

- #define AF_CAN 29
  CAN address family.
- #define PF_CAN AF_CAN
  CAN protocol family.

- #define SOL_CAN_RAW 103

  CAN socket levels.

Typedefs

- typedef uint32_t can_id_t

  Type of CAN id (see CAN_xxx_MASK and CAN_xxx_FLAG).

- typedef can_id_t can_err_mask_t

  Type of CAN error mask.

- typedef uint32_t can_baudrate_t

  Baudrate definition in bits per second.

- typedef enum CAN_BITTIME_TYPE can_bittime_type_t

  See CAN_BITTIME_TYPE.

- typedef enum CAN_MODE can_mode_t

  See CAN_MODE.

- typedef int can_ctrlmode_t

  See CAN_CTRLMODE.

- typedef enum CAN_STATE can_state_t

  See CAN_STATE.

- typedef struct can_filter can_filter_t

  Filter for reception of CAN messages.

- typedef struct can_frame can_frame_t

  Raw CAN frame.

Enumerations

- enum CAN_BITTIME_TYPE { CAN_BITTIME_STD, CAN_BITTIME_BTR }

  Supported CAN bit-time types.

CAN operation modes

Modes into which CAN controllers can be set

- enum CAN_MODE { CAN_MODE_STOP = 0, CAN_MODE_START, CAN_MODE_SLEEP }

CAN controller states

States a CAN controller can be in.

- enum CAN_STATE {
  CAN_STATE_ACTIVE = 0, CAN_STATE_BUS_WARNING, CAN_STATE_BUS_PASSIVE, CAN_STATE_BUS_OFF,
  CAN_STATE_SCANNING_BAUDRATE, CAN_STATE_STOPPED, CAN_STATE_-SLEEPING }

CAN ID masks

Bit masks for masking CAN IDs

- #define CAN_EFF_MASK 0x1FFFFFFF
  Bit mask for extended CAN IDs.
- #define CAN_SFF_MASK 0x000007FF
  Bit mask for standard CAN IDs.

CAN ID flags

Flags within a CAN ID indicating special CAN frame attributes

- #define CAN_EFF_FLAG 0x80000000
  Extended frame.
- #define CAN_RTR_FLAG 0x40000000
  Remote transmission frame.
- #define CAN_ERR_FLAG 0x20000000
  Error frame (see Errors), not valid in struct can_filter.
- #define CAN_INV_FILTER CAN_ERR_FLAG
  Invert CAN filter definition, only valid in struct can_filter.

Particular CAN protocols

Possible protocols for the PF_CAN protocol family

Currently only the RAW protocol is supported.

- #define CAN_RAW 1
  Raw protocol of `PF_CAN`, applicable to socket type `SOCK_RAW`.

CAN controller modes

Special CAN controllers modes, which can be or'ed together.

Note

These modes are hardware-dependent. Please consult the hardware manual of the CAN controller for more detailed information.

- #define CAN_CTRLMODE_LISTENONLY 0x1
- #define CAN_CTRLMODE_LOOPBACK 0x2

Timestamp switches

Arguments to pass to RTCAN_RTIOC_TAKE_TIMESTAMP

- #define RTCAN_TAKE_NO_TIMESTAMPS 0

  Switch off taking timestamps.

- #define RTCAN_TAKE_TIMESTAMPS 1

  Do take timestamps.

RAW socket options

Setting and getting CAN RAW socket options.

- #define CAN_RAW_FILTER 0x1

  CAN filter definition.

- #define CAN_RAW_ERR_FILTER 0x2

  CAN error mask.

- #define CAN_RAW_LOOPBACK 0x3

  CAN TX loopback.

- #define CAN_RAW_RECV_OWN_MSGS 0x4

  CAN receive own messages.

IOCTLs

CAN device IOCTLs

- #define SIOCGIFINDEX defined_by_kernel_header_file

  Get CAN interface index by name.

- #define SIOCSCANBAUDRATE _IOW(RTIOC_TYPE_CAN, 0x01, struct ifreq)

  Set baud rate.

- #define SIOCGCANBAUDRATE _IOWR(RTIOC_TYPE_CAN, 0x02, struct ifreq)

  Get baud rate.

- #define SIOCSCANCUSTOMBITTIME _IOW(RTIOC_TYPE_CAN, 0x03, struct ifreq)

  Set custom bit time parameter.

- #define SIOCGCANCUSTOMBITTIME _IOWR(RTIOC_TYPE_CAN, 0x04, struct ifreq)

  Get custum bit-time parameters.

- #define SIOCSCANMODE _IOW(RTIOC_TYPE_CAN, 0x05, struct ifreq)

  Set operation mode of CAN controller.

- #define SIOCGCANSTATE _IOWR(RTIOC_TYPE_CAN, 0x06, struct ifreq)

  Get current state of CAN controller.

- #define SIOCSCANCTRLMODE _IOW(RTIOC_TYPE_CAN, 0x07, struct ifreq)

  Set special controller modes.

- #define SIOCGCANCTRLMODE _IOWR(RTIOC_TYPE_CAN, 0x08, struct ifreq)

  Get special controller modes.

- #define RTCAN_RTIOC_TAKE_TIMESTAMP _IOW(RTIOC_TYPE_CAN, 0x09, int)

  Enable or disable storing a high precision timestamp upon reception of a CAN frame.

- #define RTCAN_RTIOC_RCV_TIMEOUT _IOW(RTIOC_TYPE_CAN, 0x0A, nanosecs_rel_-t)

  Specify a reception timeout for a socket.

- #define RTCAN_RTIOC_SND_TIMEOUT _IOW(RTIOC_TYPE_CAN, 0x0B, nanosecs_rel_-t)

  Specify a transmission timeout for a socket.

Error mask

Error class (mask) in `can_id` field of struct can_frame to be used with CAN_RAW_ERR_FILTER.

Note: Error reporting is hardware dependent and most CAN controllers report less detailed error conditions than the SJA1000.

Note: In case of a bus-off error condition (CAN_ERR_BUSOFF), the CAN controller is **not** restarted automatically. It is the application's responsibility to react appropriately, e.g. calling CAN_-MODE_START.

Note: Bus error interrupts (CAN_ERR_BUSERROR) are enabled when an application is calling a Recv function on a socket listening on bus errors (using CAN_RAW_ERR_FILTER). After one bus error has occured, the interrupt will be disabled to allow the application time for error processing and to efficiently avoid bus error interrupt flooding.

- #define CAN_ERR_TX_TIMEOUT 0x00000001U

  TX timeout (netdevice driver).

- #define CAN_ERR_LOSTARB 0x00000002U
  Lost arbitration (see data[0]).
- #define CAN_ERR_CRTL 0x00000004U
  Controller problems (see data[1]).
- #define CAN_ERR_PROT 0x00000008U
  Protocol violations (see data[2], data[3]).
- #define CAN_ERR_TRX 0x00000010U
  Transceiver status (see data[4]).
- #define CAN_ERR_ACK 0x00000020U
  Received no ACK on transmission.
- #define CAN_ERR_BUSOFF 0x00000040U
  Bus off.
- #define CAN_ERR_BUSERROR 0x00000080U
  Bus error (may flood!).
- #define CAN_ERR_RESTARTED 0x00000100U
  Controller restarted.
- #define CAN_ERR_MASK 0x1FFFFFFFU
  Omit EFF, RTR, ERR flags.

Arbitration lost error

Error in the data[0] field of struct can_frame.

- #define CAN_ERR_LOSTARB_UNSPEC 0x00
  unspecified

Controller problems

Error in the data[1] field of struct can_frame.

- #define CAN_ERR_CRTL_UNSPEC 0x00
  unspecified
- #define CAN_ERR_CRTL_RX_OVERFLOW 0x01
  RX buffer overflow.
- #define CAN_ERR_CRTL_TX_OVERFLOW 0x02
  TX buffer overflow.

- #define CAN_ERR_CRTL_RX_WARNING 0x04
  reached warning level for RX errors
- #define CAN_ERR_CRTL_TX_WARNING 0x08
  reached warning level for TX errors
- #define CAN_ERR_CRTL_RX_PASSIVE 0x10
  reached passive level for RX errors
- #define CAN_ERR_CRTL_TX_PASSIVE 0x20
  reached passive level for TX errors

Protocol error type

Error in the data[2] field of struct can_frame.

- #define CAN_ERR_PROT_UNSPEC 0x00
  unspecified
- #define CAN_ERR_PROT_BIT 0x01
  single bit error
- #define CAN_ERR_PROT_FORM 0x02
  frame format error
- #define CAN_ERR_PROT_STUFF 0x04
  bit stuffing error
- #define CAN_ERR_PROT_BIT0 0x08
  unable to send dominant bit
- #define CAN_ERR_PROT_BIT1 0x10
  unable to send recessive bit
- #define CAN_ERR_PROT_OVERLOAD 0x20
  bus overload
- #define CAN_ERR_PROT_ACTIVE 0x40
  active error announcement
- #define CAN_ERR_PROT_TX 0x80
  error occured on transmission

Protocol error location

Error in the data[4] field of struct can_frame.

- #define CAN_ERR_PROT_LOC_UNSPEC 0x00
  unspecified
- #define CAN_ERR_PROT_LOC_SOF 0x03
  start of frame
- #define CAN_ERR_PROT_LOC_ID28_21 0x02
  ID bits 28 - 21 (SFF: 10 - 3).
- #define CAN_ERR_PROT_LOC_ID20_18 0x06
  ID bits 20 - 18 (SFF: 2 - 0).
- #define CAN_ERR_PROT_LOC_SRTR 0x04
  substitute RTR (SFF: RTR)
- #define CAN_ERR_PROT_LOC_IDE 0x05
  identifier extension
- #define CAN_ERR_PROT_LOC_ID17_13 0x07
  ID bits 17-13.
- #define CAN_ERR_PROT_LOC_ID12_05 0x0F
  ID bits 12-5.
- #define CAN_ERR_PROT_LOC_ID04_00 0x0E
  ID bits 4-0.
- #define CAN_ERR_PROT_LOC_RTR 0x0C
  RTR.
- #define CAN_ERR_PROT_LOC_RES1 0x0D
  reserved bit 1
- #define CAN_ERR_PROT_LOC_RES0 0x09
  reserved bit 0
- #define CAN_ERR_PROT_LOC_DLC 0x0B
  data length code
- #define CAN_ERR_PROT_LOC_DATA 0x0A
  data section
- #define CAN_ERR_PROT_LOC_CRC_SEQ 0x08
  CRC sequence.
- #define CAN_ERR_PROT_LOC_CRC_DEL 0x18

CRC delimiter.

- #define CAN_ERR_PROT_LOC_ACK 0x19

  ACK slot.

- #define CAN_ERR_PROT_LOC_ACK_DEL 0x1B

  ACK delimiter.

- #define CAN_ERR_PROT_LOC_EOF 0x1A

  end of frame

- #define CAN_ERR_PROT_LOC_INTERM 0x12

  intermission

- #define CAN_ERR_TRX_UNSPEC 0x00

  0000 0000

- #define CAN_ERR_TRX_CANH_NO_WIRE 0x04

  0000 0100

- #define CAN_ERR_TRX_CANH_SHORT_TO_BAT 0x05

  0000 0101

- #define CAN_ERR_TRX_CANH_SHORT_TO_VCC 0x06

  0000 0110

- #define CAN_ERR_TRX_CANH_SHORT_TO_GND 0x07

  0000 0111

- #define CAN_ERR_TRX_CANL_NO_WIRE 0x40

  0100 0000

- #define CAN_ERR_TRX_CANL_SHORT_TO_BAT 0x50

  0101 0000

- #define CAN_ERR_TRX_CANL_SHORT_TO_VCC 0x60

  0110 0000

- #define CAN_ERR_TRX_CANL_SHORT_TO_GND 0x70

  0111 0000

- #define CAN_ERR_TRX_CANL_SHORT_TO_CANH 0x80

  1000 0000

5.36.1 Detailed Description

This is the common interface a RTDM-compliant CAN device has to provide. Feel free to report bugs and comments on this profile to the "Socketcan" mailing list (Socketcan-core@lists.berlios.de) or directly to the authors (wg@grandegger.com or Sebastian.Smolorz@stud.uni-hannover.de).

Profile Revision: 2

Device Characteristics

Device Flags: `RTDM_PROTOCOL_DEVICE`
Protocol Family: `PF_CAN`
Socket Type: `SOCK_RAW`
Device Class: `RTDM_CLASS_CAN`

Supported Operations

Socket
Environments: non-RT (RT optional, deprecated)
Specific return values:

- -EPROTONOSUPPORT (Protocol is not supported by the driver. See CAN protocols for possible protocols.)

Close
Blocking calls to any of the Send or Receive functions will be unblocked when the socket is closed and return with an error.
Environments: non-RT (RT optional, deprecated)
Specific return values: none
IOCTL
Mandatory Environments: see below
Specific return values: see below
Bind
Binds a socket to one or all CAN devices (see struct sockaddr_can). If a filter list has been defined with setsockopt (see Sockopts), it will be used upon reception of CAN frames to decide whether the bound socket will receive a frame. If no filter has been defined, the socket will receive **all** CAN frames on the specified interface(s).
Binding to special interface index `0` will make the socket receive CAN frames from all CAN interfaces.
Binding to an interface index is also relevant for the Send functions because they will transmit a message over the interface the socket is bound to when no socket address is given to them.
Environments: non-RT (RT optional)
Specific return values:

- -EFAULT (It was not possible to access user space memory area at the specified address.)
- -ENOMEM (Not enough memory to fulfill the operation)
- -EINVAL (Invalid address family, or invalid length of address structure)
- -ENODEV (Invalid CAN interface index)
- -ENOSPC (No enough space for filter list)
- -EBADF (Socket is about to be closed)
- -EAGAIN (Too many receivers. Old binding (if any) is still active. Close some sockets and try again.)

Setsockopt, Getsockopt

These functions allow to set and get various socket options. Currently, only CAN raw sockets are supported.
Supported Levels and Options:

- Level **SOL_CAN_RAW** : CAN RAW protocol (see CAN_RAW)
  - Option CAN_RAW_FILTER : CAN filter list
  - Option CAN_RAW_ERR_FILTER : CAN error mask
  - Option CAN_RAW_LOOPBACK : CAN TX loopback to local sockets

Environments: non-RT (RT optional)
Specific return values: see links to options above.
Recv, Recvfrom, Recvmsg
These functions receive CAN messages from a socket. Only one message per call can be received, so only one buffer with the correct length must be passed. For `SOCK_RAW`, this is the size of struct can_frame.
Unlike a call to one of the Send functions, a Recv function will not return with an error if an interface is down (due to bus-off or setting of stop mode) or in sleep mode. Moreover, in such a case there may still be some CAN messages in the socket buffer which could be read out successfully.
It is possible to receive a high precision timestamp with every CAN message. The condition is a former instruction to the socket via RTCAN_RTIOC_TAKE_TIMESTAMP. The timestamp will be copied to the `msg_control` buffer of `struct msghdr` if it points to a valid memory location with size of nanosecs_abs_t. If this is a NULL pointer the timestamp will be discarded silently.
Note: A `msg_controllen` of `0` upon completion of the function call indicates that no timestamp is available for that message.
Supported Flags [in]:

- MSG_DONTWAIT (By setting this flag the operation will only succeed if it would not block, i.e. if there is a message in the socket buffer. This flag takes precedence over a timeout specified by RTCAN_RTIOC_RCV_TIMEOUT.)
- MSG_PEEK (Receive a message but leave it in the socket buffer. The next receive operation will get that message again.)

Supported Flags [out]: none
Environments: RT (non-RT optional)
Specific return values:

- Non-negative value (Indicating the successful reception of a CAN message. For `SOCK_RAW`, this is the size of struct can_frame regardless of the actual size of the payload.)
- -EFAULT (It was not possible to access user space memory area at one of the specified addresses.)
- -EINVAL (Unsupported flag detected, or invalid length of socket address buffer, or invalid length of message control buffer)
- -EMSGSIZE (Zero or more than one iovec buffer passed, or buffer too small)
- -EAGAIN (No data available in non-blocking mode)
- -EBADF (Socket was closed.)
- -EINTR (Operation was interrupted explicitly or by signal.)
- -ETIMEDOUT (Timeout)

Send, Sendto, Sendmsg
These functions send out CAN messages. Only one message per call can be transmitted, so only one buffer with the correct length must be passed. For `SOCK_RAW`, this is the size of struct can_frame.

The following only applies to `SOCK_RAW`: If a socket address of struct sockaddr_can is given, only `can_ifindex` is used. It is also possible to omit the socket address. Then the interface the socket is bound to will be used for sending messages.
If an interface goes down (due to bus-off or setting of stop mode) all senders that were blocked on this interface will be woken up.
Supported Flags:

- MSG_DONTWAIT (By setting this flag the transmit operation will only succeed if it would not block. This flag takes precedence over a timeout specified by RTCAN_-RTIOC_SND_TIMEOUT.)

Environments: RT (non-RT optional)
Specific return values:

- Non-negative value equal to given buffer size (Indicating the successful completion of the function call. See also note.)
- -EOPNOTSUPP (MSG_OOB flag is not supported.)
- -EINVAL (Unsupported flag detected *or:* Invalid length of socket address *or:* Invalid address family *or:* Data length code of CAN frame not between 0 and 15 *or:* CAN standard frame has got an ID not between 0 and 2031)
- -EMSGSIZE (Zero or more than one buffer passed or invalid size of buffer)
- -EFAULT (It was not possible to access user space memory area at one of the specified addresses.)
- -ENXIO (Invalid CAN interface index - `0` is not allowed here - or socket not bound or rather bound to all interfaces.)
- -ENETDOWN (Controller is bus-off or in stopped state.)
- -ECOMM (Controller is sleeping)
- -EAGAIN (Cannot transmit without blocking but a non-blocking call was requested.)
- -EINTR (Operation was interrupted explicitly or by signal)
- -EBADF (Socket was closed.)
- -ETIMEDOUT (Timeout)

Note: A successful completion of the function call does not implicate a successful transmission of the message.

5.36.2 Define Documentation

5.36.2.1 #define CAN_CTRLMODE_LISTENONLY 0x1

Listen-Only mode

In this mode the CAN controller would give no acknowledge to the CAN-bus, even if a message is received successfully and messages would not be transmitted. This mode might be useful for bus-monitoring, hot-plugging or throughput analysis.

5.36.2.2 #define CAN_CTRLMODE_LOOPBACK 0x2

Loopback mode

In this mode the CAN controller does an internal loop-back, a message is transmitted and simultaneously received. That mode can be used for self test operation.

5.36.2.3 #define CAN_ERR_LOSTARB_UNSPEC 0x00

unspecified

else bit number in bitstream

5.36.2.4 #define CAN_RAW_ERR_FILTER 0x2

CAN error mask.

A CAN error mask (see Errors) can be set with `setsockopt`. This mask is then used to decide if error frames are delivered to this socket in case of error condidtions. The error frames are marked with the CAN_ERR_FLAG of CAN_xxx_FLAG and must be handled by the application properly. A detailed description of the errors can be found in the `can_id` and the `data` fields of struct can_frame (see Errors for futher details).

Parameters

`[in]` *level* **SOL_CAN_RAW**

`[in]` *optname* **CAN_RAW_ERR_FILTER**

`[in]` *optval* Pointer to error mask of type can_err_mask_t.

`[in]` *optlen* Size of error mask: sizeof(can_err_mask_t).

Environments: non-RT (RT optional)

Specific return values:

- -EFAULT (It was not possible to access user space memory area at the specified address.)
- -EINVAL (Invalid length "optlen")

5.36.2.5 #define CAN_RAW_FILTER 0x1

CAN filter definition.

A CAN raw filter list with elements of struct can_filter can be installed with `setsockopt`. This list is used upon reception of CAN frames to decide whether the bound socket will receive a frame. An empty filter list can also be defined using optlen = 0, which is recommanded for write-only sockets.

If the socket was already bound with Bind, the old filter list gets replaced with the new one. Be aware that already received, but not read out CAN frames may stay in the socket buffer.

Parameters

`[in]` *level* **SOL_CAN_RAW**

`[in]` *optname* **CAN_RAW_FILTER**

`[in]` *optval* Pointer to array of struct can_filter.

`[in]` *optlen* Size of filter list: count * sizeof(struct can_filter).

Environments: non-RT (RT optional)

Specific return values:

- -EFAULT (It was not possible to access user space memory area at the specified address.)

- -ENOMEM (Not enough memory to fulfill the operation)
- -EINVAL (Invalid length "optlen")
- -ENOSPC (No space to store filter list, check RT-Socket-CAN kernel parameters)

5.36.2.6 #define CAN_RAW_LOOPBACK 0x3

CAN TX loopback.

The TX loopback to other local sockets can be selected with this `setsockopt`.

Note

The TX loopback feature must be enabled in the kernel and then the loopback to other local TX sockets is enabled by default.

Parameters

`[in]` *level* **SOL_CAN_RAW**

`[in]` *optname* **CAN_RAW_LOOPBACK**

`[in]` *optval* Pointer to integer value.

`[in]` *optlen* Size of int: sizeof(int).

Environments: non-RT (RT optional)

Specific return values:

- -EFAULT (It was not possible to access user space memory area at the specified address.)
- -EINVAL (Invalid length "optlen")
- -EOPNOTSUPP (not supported, check RT-Socket-CAN kernel parameters).

5.36.2.7 #define CAN_RAW_RECV_OWN_MSGS 0x4

CAN receive own messages.

Not supported by RT-Socket-CAN, but defined for compatibility with Socket-CAN.

5.36.2.8 #define RTCAN_RTIOC_RCV_TIMEOUT _IOW(RTIOC_TYPE_CAN, 0x0A, nanosecs_rel_t)

Specify a reception timeout for a socket.

Defines a timeout for all receive operations via a socket which will take effect when one of the receive functions is called without the `MSG_DONTWAIT` flag set.

The default value for a newly created socket is an infinite timeout.

Note

The setting of the timeout value is not done atomically to avoid locks. Please set the value before receiving messages from the socket.

Parameters

[in] *arg* Pointer to nanosecs_rel_t variable. The value is interpreted as relative timeout in nanoseconds in case of a positive value. See Timeouts for special timeouts.

Returns

0 on success, otherwise:

- -EFAULT: It was not possible to access user space memory area at the specified address.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.36.2.9 #define RTCAN_RTIOC_SND_TIMEOUT _IOW(RTIOC_TYPE_CAN, 0x0B, nanosecs_rel_t)

Specify a transmission timeout for a socket.

Defines a timeout for all send operations via a socket which will take effect when one of the send functions is called without the `MSG_DONTWAIT` flag set.

The default value for a newly created socket is an infinite timeout.

Note

The setting of the timeout value is not done atomically to avoid locks. Please set the value before sending messages to the socket.

Parameters

[in] *arg* Pointer to nanosecs_rel_t variable. The value is interpreted as relative timeout in nanoseconds in case of a positive value. See Timeouts for special timeouts.

Returns

0 on success, otherwise:

- -EFAULT: It was not possible to access user space memory area at the specified address.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.36.2.10 #define RTCAN_RTIOC_TAKE_TIMESTAMP _IOW(RTIOC_TYPE_CAN, 0x09, int)

Enable or disable storing a high precision timestamp upon reception of a CAN frame.

A newly created socket takes no timestamps by default.

Parameters

`[in]` *arg* int variable, see Timestamp switches

Returns

0 on success.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Note

Activating taking timestamps only has an effect on newly received CAN messages from the bus. Frames that already are in the socket buffer do not have timestamps if it was deactivated before. See Receive for more details.

Rescheduling: never.

5.36.2.11 #define SIOCGCANBAUDRATE _IOWR(RTIOC_TYPE_CAN, 0x02, struct ifreq)

Get baud rate.

Parameters

`[in,out]` *arg* Pointer to interface request structure buffer (`struct ifreq` from linux/if.h). `ifr_name` must hold a valid CAN interface name, `ifr_ifru` will be filled with an instance of can_baudrate_t.

Returns

0 on success, otherwise:

- -EFAULT: It was not possible to access user space memory area at the specified address.
- -ENODEV: No device with specified name exists.
- -EINVAL: No baud rate was set yet.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code

- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.36.2.12 #define SIOCGCANCTRLMODE _IOWR(RTIOC_TYPE_CAN, 0x08, struct ifreq)

Get special controller modes.

Parameters

`[in]` *arg* Pointer to interface request structure buffer (`struct ifreq` from linux/if.h). `ifr_name` must hold a valid CAN interface name, `ifr_ifru` must be filled with an instance of can_ctrlmode_t.

Returns

0 on success, otherwise:

- -EFAULT: It was not possible to access user space memory area at the specified address.
- -ENODEV: No device with specified name exists.
- -EINVAL: No baud rate was set yet.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: possible.

5.36.2.13 #define SIOCGCANCUSTOMBITTIME _IOWR(RTIOC_TYPE_CAN, 0x04, struct ifreq)

Get custum bit-time parameters.

Parameters

`[in,out]` *arg* Pointer to interface request structure buffer (`struct ifreq` from linux/if.h). `ifr_name` must hold a valid CAN interface name, `ifr_ifru` will be filled with an instance of struct can_bittime.

Returns

0 on success, otherwise:

- -EFAULT: It was not possible to access user space memory area at the specified address.
- -ENODEV: No device with specified name exists.
- -EINVAL: No baud rate was set yet.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.36.2.14 #define SIOCGCANSTATE _IOWR(RTIOC_TYPE_CAN, 0x06, struct ifreq)

Get current state of CAN controller.

States are divided into main states and additional error indicators. A CAN controller is always in exactly one main state. CAN bus errors are registered by the CAN hardware and collected by the driver. There is one error indicator (bit) per error type. If this IOCTL is triggered the error types which occured since the last call of this IOCTL are reported and thereafter the error indicators are cleared. See also CAN controller states.

Parameters

`[in,out]` *arg* Pointer to interface request structure buffer (`struct ifreq` from linux/if.h). `ifr_name` must hold a valid CAN interface name, `ifr_ifru` will be filled with an instance of can_mode_t.

Returns

0 on success, otherwise:

- -EFAULT: It was not possible to access user space memory area at the specified address.
- -ENODEV: No device with specified name exists.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: possible.

5.36.2.15 #define SIOCGIFINDEX defined_by_kernel_header_file

Get CAN interface index by name.

Parameters

`[in,out]` *arg* Pointer to interface request structure buffer (`struct ifreq` from linux/if.h). If `ifr_name` holds a valid CAN interface name `ifr_ifindex` will be filled with the corresponding interface index.

Returns

0 on success, otherwise:

- -EFAULT: It was not possible to access user space memory area at the specified address.
- -ENODEV: No device with specified name exists.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.36.2.16 #define SIOCSCANBAUDRATE _IOW(RTIOC_TYPE_CAN, 0x01, struct ifreq)

Set baud rate.

The baudrate must be specified in bits per second. The driver will try to calculate resonable CAN bit-timing parameters. You can use SIOCSCANCUSTOMBITTIME to set custom bit-timing.

Parameters

`[in]` *arg* Pointer to interface request structure buffer (`struct ifreq` from linux/if.h). `ifr_name` must hold a valid CAN interface name, `ifr_ifru` must be filled with an instance of can_baudrate_t.

Returns

0 on success, otherwise:

- -EFAULT: It was not possible to access user space memory area at the specified address.
- -ENODEV: No device with specified name exists.
- -EINVAL: No valid baud rate, see can_baudrate_t.
- -EDOM : Baud rate not possible.
- -EAGAIN: Request could not be successully fulfilled. Try again.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Note

Setting the baud rate is a configuration task. It should be done deliberately or otherwise CAN messages will likely be lost.

Rescheduling: possible.

5.36.2.17 #define SIOCSCANCTRLMODE _IOW(RTIOC_TYPE_CAN, 0x07, struct ifreq)

Set special controller modes.

Various special controller modes could be or'ed together (see CAN_CTRLMODE for further information).

Parameters

`[in]` *arg* Pointer to interface request structure buffer (`struct ifreq` from linux/if.h). `ifr_name` must hold a valid CAN interface name, `ifr_ifru` must be filled with an instance of can_ctrlmode_t.

Returns

0 on success, otherwise:

- -EFAULT: It was not possible to access user space memory area at the specified address.
- -ENODEV: No device with specified name exists.
- -EINVAL: No valid baud rate, see can_baudrate_t.
- -EAGAIN: Request could not be successully fulfilled. Try again.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Note

Setting special controller modes is a configuration task. It should be done deliberately or otherwise CAN messages will likely be lost.

Rescheduling: possible.

5.36.2.18 #define SIOCSCANCUSTOMBITTIME _IOW(RTIOC_TYPE_CAN, 0x03, struct ifreq)

Set custom bit time parameter.

Custem-bit time could be defined in various formats (see struct can_bittime).

Parameters

`[in]` *arg* Pointer to interface request structure buffer (`struct ifreq` from linux/if.h). `ifr_name` must hold a valid CAN interface name, `ifr_ifru` must be filled with an instance of struct can_bittime.

Returns

0 on success, otherwise:

- -EFAULT: It was not possible to access user space memory area at the specified address.

- -ENODEV: No device with specified name exists.
- -EINVAL: No valid baud rate, see can_baudrate_t.
- -EAGAIN: Request could not be successfully fulfilled. Try again.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Note

Setting the bit-time is a configuration task. It should be done deliberately or otherwise CAN messages will likely be lost.

Rescheduling: possible.

5.36.2.19 #define SIOCSCANMODE _IOW(RTIOC_TYPE_CAN, 0x05, struct ifreq)

Set operation mode of CAN controller.

See CAN controller modes for available modes.

Parameters

`[in]` *arg* Pointer to interface request structure buffer (`struct ifreq` from linux/if.h). `ifr_name` must hold a valid CAN interface name, `ifr_ifru` must be filled with an instance of can_mode_t.

Returns

0 on success, otherwise:

- -EFAULT: It was not possible to access user space memory area at the specified address.
- -ENODEV: No device with specified name exists.
- -EAGAIN: (CAN_MODE_START, CAN_MODE_STOP) Could not successfully set mode, hardware is busy. Try again.
- -EINVAL: (CAN_MODE_START) Cannot start controller, set baud rate first.
- -ENETDOWN: (CAN_MODE_SLEEP) Cannot go into sleep mode because controller is stopped or bus off.
- -EOPNOTSUPP: unknown mode

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Note

Setting a CAN controller into normal operation after a bus-off can take some time (128 occurrences of 11 consecutive recessive bits). In such a case, although this IOCTL will return immediately with success and SIOCGCANSTATE will report CAN_STATE_ACTIVE, bus-off recovery may still be in progress.

If a controller is bus-off, setting it into stop mode will return no error but the controller remains bus-off.

Rescheduling: possible.

5.36.2.20 #define SOL_CAN_RAW 103

CAN socket levels.

Used for Sockopts for the particular protocols.

5.36.3 Typedef Documentation

5.36.3.1 typedef struct can_filter can_filter_t

Filter for reception of CAN messages.

This filter works as follows: A received CAN ID is AND'ed bitwise with `can_mask` and then compared to `can_id`. This also includes the CAN_EFF_FLAG and CAN_RTR_FLAG of CAN_-xxx_FLAG. If this comparison is true, the message will be received by the socket. The logic can be inverted with the `can_id` flag CAN_INV_FILTER :

```
if (can_id & CAN_INV_FILTER) {
   if ((received_can_id & can_mask) != (can_id & ~CAN_INV_FILTER))
      accept-message;
} else {
   if ((received_can_id & can_mask) == can_id)
      accept-message;
}
```

Multiple filters can be arranged in a filter list and set with Sockopts. If one of these filters matches a CAN ID upon reception of a CAN frame, this frame is accepted.

5.36.3.2 typedef struct can_frame can_frame_t

Raw CAN frame.

Central structure for receiving and sending CAN frames.

5.36.4 Enumeration Type Documentation

5.36.4.1 enum CAN_BITTIME_TYPE

Supported CAN bit-time types.

Enumerator:

CAN_BITTIME_STD Standard bit-time definition according to Bosch.

CAN_BITTIME_BTR Hardware-specific BTR bit-time definition.

5.36.4.2 enum CAN_MODE

Enumerator:

CAN_MODE_STOP Set controller in Stop mode (no reception / transmission possible)

CAN_MODE_START Set controller into normal operation.
Coming from stopped mode or bus off, the controller begins with no errors in CAN_-STATE_ACTIVE.

CAN_MODE_SLEEP Set controller into Sleep mode.
This is only possible if the controller is not stopped or bus-off.
Notice that sleep mode will only be entered when there is no bus activity. If the controller detects bus activity while "sleeping" it will go into operating mode again.
To actively leave sleep mode again trigger `CAN_MODE_START`.

5.36.4.3 enum CAN_STATE

Enumerator:

CAN_STATE_ACTIVE CAN controller is error active.

CAN_STATE_BUS_WARNING CAN controller is error active, warning level is reached.

CAN_STATE_BUS_PASSIVE CAN controller is error passive.

CAN_STATE_BUS_OFF CAN controller went into Bus Off.

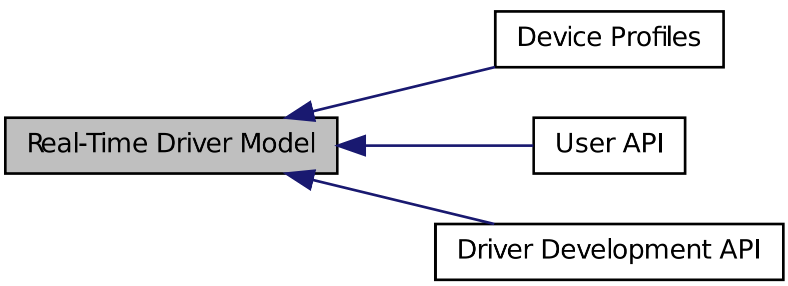
CAN_STATE_SCANNING_BAUDRATE CAN controller is scanning to get the baudrate.

CAN_STATE_STOPPED CAN controller is in stopped mode.

CAN_STATE_SLEEPING CAN controller is in Sleep mode.

5.37 Real-Time Driver Model

Collaboration diagram for Real-Time Driver Model:



Modules

- Driver Development API
- Device Profiles
- User API

Typedefs

- typedef uint64_t nanosecs_abs_t

    RTDM type for representing absolute dates.

- typedef int64_t nanosecs_rel_t

    RTDM type for representing relative intervals.

API Versioning

- #define RTDM_API_VER 8

    Common user and driver API version.

- #define RTDM_API_MIN_COMPAT_VER 6

    Minimum API revision compatible with the current release.

RTDM_TIMEOUT_xxx

Special timeout values

- #define RTDM_TIMEOUT_INFINITE 0

    Block forever.

- #define RTDM_TIMEOUT_NONE (-1)

    Any negative timeout means non-blocking.

5.37.1 Detailed Description

The Real-Time Driver Model (RTDM) provides a unified interface to both users and developers of real-time device drivers. Specifically, it addresses the constraints of mixed RT/non-RT systems like Xenomai. RTDM conforms to POSIX semantics (IEEE Std 1003.1) where available and applicable.

API Revision: 8

5.37.2 Define Documentation

5.37.2.1 #define RTDM_TIMEOUT_INFINITE 0

Block forever.

5.37.2.2 #define RTDM_TIMEOUT_NONE (-1)

Any negative timeout means non-blocking.

5.37.3 Typedef Documentation

5.37.3.1 typedef uint64_t nanosecs_abs_t

RTDM type for representing absolute dates.

Its base type is a 64 bit unsigned integer. The unit is 1 nanosecond.

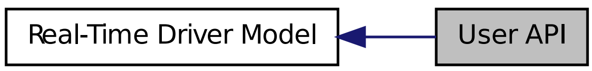
5.37.3.2 typedef int64_t nanosecs_rel_t

RTDM type for representing relative intervals.

Its base type is a 64 bit signed integer. The unit is 1 nanosecond. Relative intervals can also encode the special timeouts "infinite" and "non-blocking", see RTDM_TIMEOUT_xxx.

5.38 User API

Collaboration diagram for User API:



Files

- file rtdm.h

  Real-Time Driver Model for Xenomai, user API header.

Functions

- int rt_dev_open (const char *path, int oflag,...)

  Open a device.

- int rt_dev_socket (int protocol_family, int socket_type, int protocol)

  Create a socket.

- int rt_dev_close (int fd)

Close a device or socket.

- int rt_dev_ioctl (int fd, int request,...)

    Issue an IOCTL.

- ssize_t rt_dev_read (int fd, void *buf, size_t nbyte)

    Read from device.

- ssize_t rt_dev_write (int fd, const void *buf, size_t nbyte)

    Write to device.

- ssize_t rt_dev_recvmsg (int fd, struct msghdr *msg, int flags)

    Receive message from socket.

- ssize_t rt_dev_recvfrom (int fd, void *buf, size_t len, int flags, struct sockaddr *from, socklen_t *fromlen)

    Receive message from socket.

- ssize_t rt_dev_recv (int fd, void *buf, size_t len, int flags)

    Receive message from socket.

- ssize_t rt_dev_sendmsg (int fd, const struct msghdr *msg, int flags)

    Transmit message to socket.

- ssize_t rt_dev_sendto (int fd, const void *buf, size_t len, int flags, const struct sockaddr *to, socklen_t tolen)

    Transmit message to socket.

- ssize_t rt_dev_send (int fd, const void *buf, size_t len, int flags)

    Transmit message to socket.

- int rt_dev_bind (int fd, const struct sockaddr *my_addr, socklen_t addrlen)

    Bind to local address.

- int rt_dev_connect (int fd, const struct sockaddr *serv_addr, socklen_t addrlen)

    Connect to remote address.

- int rt_dev_listen (int fd, int backlog)

    Listen for incomming connection requests.

- int rt_dev_accept (int fd, struct sockaddr *addr, socklen_t *addrlen)

    Accept a connection requests.

- int rt_dev_shutdown (int fd, int how)

    Shut down parts of a connection.

- int rt_dev_getsockopt (int fd, int level, int optname, void *optval, socklen_t *optlen)

    Get socket option.

- int rt_dev_setsockopt (int fd, int level, int optname, const void *optval, socklen_t optlen)

Set socket option.

- int rt_dev_getsockname (int fd, struct sockaddr *name, socklen_t *namelen)
  Get local socket address.
- int rt_dev_getpeername (int fd, struct sockaddr *name, socklen_t *namelen)
  Get socket destination address.

5.38.1 Detailed Description

This is the upper interface of RTDM provided to application programs both in kernel and user space. Note that certain functions may not be implemented by every device. Refer to the Device Profiles for precise information.

5.38.2 Function Documentation

5.38.2.1 int rt_dev_accept (int *fd*, struct sockaddr * *addr*, socklen_t * *addrlen*)

Accept a connection requests.

Parameters

`[in]` ***fd*** File descriptor as returned by rt_dev_socket()

`[out]` ***addr*** Buffer for remote address

`[in,out]` ***addrlen*** Address buffer size

Returns

0 on success, otherwise negative error code

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

See also

`accept()` in IEEE Std 1003.1, `http://www.opengroup.org/onlinepubs/009695399`

5.38.2.2 int rt_dev_bind (int *fd*, const struct sockaddr * *my_addr*, socklen_t *addrlen*)

Bind to local address.

Parameters

`[in]` ***fd*** File descriptor as returned by rt_dev_socket()

`[in]` ***my_addr*** Address buffer

`[in]` ***addrlen*** Address buffer size

Returns

0 on success, otherwise negative error code

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

See also

`bind()` in IEEE Std 1003.1, http://www.opengroup.org/onlinepubs/009695399

5.38.2.3 int rt_dev_close (int *fd*)

Close a device or socket.

Parameters

`[in]` *fd* File descriptor as returned by rt_dev_open() or rt_dev_socket()

Returns

0 on success, otherwise a negative error code.

Note

If the matching rt_dev_open() or rt_dev_socket() call took place in non-real-time context, rt_dev_close() must be issued within non-real-time as well. Otherwise, the call will fail.

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

See also

`close()` in IEEE Std 1003.1, http://www.opengroup.org/onlinepubs/009695399

5.38.2.4 int rt_dev_connect (int *fd*, const struct sockaddr * *serv_addr*, socklen_t *addrlen*)

Connect to remote address.

Parameters

`[in]` *fd* File descriptor as returned by rt_dev_socket()

`[in]` *serv_addr* Address buffer

`[in]` *addrlen* Address buffer size

Returns

0 on success, otherwise negative error code

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

See also

connect() in IEEE Std 1003.1, http://www.opengroup.org/onlinepubs/009695399

5.38.2.5 int rt_dev_getpeername (int *fd*, struct sockaddr * *name*, socklen_t * *namelen*)

Get socket destination address.

Parameters

[in] *fd* File descriptor as returned by rt_dev_socket()
[out] *name* Address buffer
[in,out] *namelen* Address buffer size

Returns

0 on success, otherwise negative error code

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

See also

getpeername() in IEEE Std 1003.1, http://www.opengroup.org/onlinepubs/009695399

5.38.2.6 int rt_dev_getsockname (int *fd*, struct sockaddr * *name*, socklen_t * *namelen*)

Get local socket address.

Parameters

[in] *fd* File descriptor as returned by rt_dev_socket()
[out] *name* Address buffer
[in,out] *namelen* Address buffer size

Returns

0 on success, otherwise negative error code

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

See also

getsockname() in IEEE Std 1003.1, http://www.opengroup.org/onlinepubs/009695399

5.38.2.7 int rt_dev_getsockopt (int *fd*, int *level*, int *optname*, void * *optval*, socklen_t * *optlen*)

Get socket option.

Parameters

`[in]` ***fd*** File descriptor as returned by rt_dev_socket()

`[in]` ***level*** Addressed stack level

`[in]` ***optname*** Option name ID

`[out]` ***optval*** Value buffer

`[in,out]` ***optlen*** Value buffer size

Returns

0 on success, otherwise negative error code

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

See also

`getsockopt()` in IEEE Std 1003.1, `http://www.opengroup.org/onlinepubs/009695399`

5.38.2.8 int rt_dev_ioctl (int *fd*, int *request*, ...)

Issue an IOCTL.

Parameters

`[in]` ***fd*** File descriptor as returned by rt_dev_open() or rt_dev_socket()

`[in]` ***request*** IOCTL code

... Optional third argument, depending on IOCTL function (`void *` or `unsigned long`)

Returns

Positiv value on success, otherwise negative error code

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

See also

`ioctl()` in IEEE Std 1003.1, `http://www.opengroup.org/onlinepubs/009695399`

5.38.2.9 int rt_dev_listen (int *fd*, int *backlog*)

Listen for incomming connection requests.

Parameters

`[in]` ***fd*** File descriptor as returned by rt_dev_socket()

`[in]` ***backlog*** Maximum queue length

Returns

0 on success, otherwise negative error code

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

See also

`lsiten()` in IEEE Std 1003.1, `http://www.opengroup.org/onlinepubs/009695399`

5.38.2.10 int rt_dev_open (const char * *path*, int *oflag*, ...)

Open a device.

Parameters

`[in]` ***path*** Device name

`[in]` ***oflag*** Open flags

... Further parameters will be ignored.

Returns

Positive file descriptor value on success, otherwise a negative error code.

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

See also

`open()` in IEEE Std 1003.1, `http://www.opengroup.org/onlinepubs/009695399`

5.38.2.11 ssize_t rt_dev_read (int *fd*, void * *buf*, size_t *nbyte*)

Read from device.

Parameters

`[in]` ***fd*** File descriptor as returned by rt_dev_open()

[out] *buf* Input buffer

[in] *nbyte* Number of bytes to read

Returns

Number of bytes read, otherwise negative error code

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

See also

`read()` in IEEE Std 1003.1, http://www.opengroup.org/onlinepubs/009695399

5.38.2.12 ssize_t rt_dev_recv (int *fd*, void ∗ *buf*, size_t *len*, int *flags*)

Receive message from socket.

Parameters

[in] *fd* File descriptor as returned by rt_dev_socket()

[out] *buf* Message buffer

[in] *len* Message buffer size

[in] *flags* Message flags

Returns

Number of bytes received, otherwise negative error code

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

See also

`recv()` in IEEE Std 1003.1, http://www.opengroup.org/onlinepubs/009695399

5.38.2.13 ssize_t rt_dev_recvfrom (int *fd*, void ∗ *buf*, size_t *len*, int *flags*, struct sockaddr ∗ *from*, socklen_t ∗ *fromlen*)

Receive message from socket.

Parameters

[in] *fd* File descriptor as returned by rt_dev_socket()

[out] *buf* Message buffer

[in] *len* Message buffer size

`[in]` *flags* Message flags
`[out]` *from* Buffer for message sender address
`[in,out]` *fromlen* Address buffer size

Returns

Number of bytes received, otherwise negative error code

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

See also

`recvfrom()` in IEEE Std 1003.1, http://www.opengroup.org/onlinepubs/009695399

5.38.2.14 ssize_t rt_dev_recvmsg (int *fd*, struct msghdr * *msg*, int *flags*)

Receive message from socket.

Parameters

`[in]` *fd* File descriptor as returned by rt_dev_socket()
`[in,out]` *msg* Message descriptor
`[in]` *flags* Message flags

Returns

Number of bytes received, otherwise negative error code

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

See also

`recvmsg()` in IEEE Std 1003.1, http://www.opengroup.org/onlinepubs/009695399

5.38.2.15 ssize_t rt_dev_send (int *fd*, const void * *buf*, size_t *len*, int *flags*)

Transmit message to socket.

Parameters

`[in]` *fd* File descriptor as returned by rt_dev_socket()
`[in]` *buf* Message buffer
`[in]` *len* Message buffer size
`[in]` *flags* Message flags

Returns

Number of bytes sent, otherwise negative error code

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

See also

`send()` in IEEE Std 1003.1, `http://www.opengroup.org/onlinepubs/009695399`

5.38.2.16 ssize_t rt_dev_sendmsg (int *fd*, const struct msghdr * *msg*, int *flags*)

Transmit message to socket.

Parameters

`[in]` *fd* File descriptor as returned by rt_dev_socket()

`[in]` *msg* Message descriptor

`[in]` *flags* Message flags

Returns

Number of bytes sent, otherwise negative error code

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

See also

`sendmsg()` in IEEE Std 1003.1, `http://www.opengroup.org/onlinepubs/009695399`

5.38.2.17 ssize_t rt_dev_sendto (int *fd*, const void * *buf*, size_t *len*, int *flags*, const struct sockaddr * *to*, socklen_t *tolen*)

Transmit message to socket.

Parameters

`[in]` *fd* File descriptor as returned by rt_dev_socket()

`[in]` *buf* Message buffer

`[in]` *len* Message buffer size

`[in]` *flags* Message flags

`[in]` *to* Buffer for message destination address

`[in]` *tolen* Address buffer size

Returns

Number of bytes sent, otherwise negative error code

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

See also

`sendto()` in IEEE Std 1003.1, http://www.opengroup.org/onlinepubs/009695399

5.38.2.18 int rt_dev_setsockopt (int *fd*, int *level*, int *optname*, const void * *optval*, socklen_t *optlen*)

Set socket option.

Parameters

`[in]` ***fd*** File descriptor as returned by rt_dev_socket()

`[in]` ***level*** Addressed stack level

`[in]` ***optname*** Option name ID

`[in]` ***optval*** Value buffer

`[in]` ***optlen*** Value buffer size

Returns

0 on success, otherwise negative error code

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

See also

`setsockopt()` in IEEE Std 1003.1, http://www.opengroup.org/onlinepubs/009695399

5.38.2.19 int rt_dev_shutdown (int *fd*, int *how*)

Shut down parts of a connection.

Parameters

`[in]` ***fd*** File descriptor as returned by rt_dev_socket()

`[in]` ***how*** Specifies the part to be shut down (`SHUT_xxx`)

Returns

0 on success, otherwise negative error code

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

See also

`shutdown()` in IEEE Std 1003.1, http://www.opengroup.org/onlinepubs/009695399

5.38.2.20 int rt_dev_socket (int *protocol_family*, int *socket_type*, int *protocol*)

Create a socket.

Parameters

`[in]` ***protocol_family*** Protocol family (`PF_xxx`)
`[in]` ***socket_type*** Socket type (`SOCK_xxx`)
`[in]` ***protocol*** Protocol ID, 0 for default

Returns

Positive file descriptor value on success, otherwise a negative error code.

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

See also

`socket()` in IEEE Std 1003.1, http://www.opengroup.org/onlinepubs/009695399

5.38.2.21 ssize_t rt_dev_write (int *fd*, const void * *buf*, size_t *nbyte*)

Write to device.

Parameters

`[in]` ***fd*** File descriptor as returned by rt_dev_open()
`[in]` ***buf*** Output buffer
`[in]` ***nbyte*** Number of bytes to write

Returns

Number of bytes written, otherwise negative error code

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

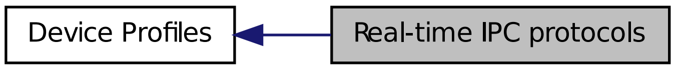
See also

`write()` in IEEE Std 1003.1, http://www.opengroup.org/onlinepubs/009695399

5.39 Real-time IPC protocols

Profile Revision: 1

Collaboration diagram for Real-time IPC protocols:



Data Structures

- struct rtipc_port_label

  Port label information structure.

- struct sockaddr_ipc

  Socket address structure for the RTIPC address family.

Files

- file rtipc.h

  This file is part of the Xenomai project.

Typedefs

- typedef int16_t rtipc_port_t

  Port number type for the RTIPC address family.

RTIPC protocol list

protocols for the PF_RTIPC protocol family

- enum { IPCPROTO_IPC = 0, IPCPROTO_XDDP = 1, IPCPROTO_IDDP = 2, IPCPROTO_-BUFP = 3 }

Supported operations

Standard socket operations supported by the RTIPC protocols.

- int socket__AF_RTIPC (int domain=AF_RTIPC, int type=SOCK_DGRAM, int protocol)
  Create an endpoint for communication in the AF_RTIPC domain.
- int close__AF_RTIPC (int sockfd)
  Close a RTIPC socket descriptor.
- int bind__AF_RTIPC (int sockfd, const struct sockaddr_ipc *addr, socklen_t addrlen)
  Bind a RTIPC socket to a port.
- int connect__AF_RTIPC (int sockfd, const struct sockaddr_ipc *addr, socklen_t addrlen)
  Initiate a connection on a RTIPC socket.
- int setsockopt__AF_RTIPC (int sockfd, int level, int optname, const void *optval, socklen_t optlen)
  Set options on RTIPC sockets.
- int getsockopt__AF_RTIPC (int sockfd, int level, int optname, void *optval, socklen_t *optlen)
  Get options on RTIPC sockets.
- ssize_t sendmsg__AF_RTIPC (int sockfd, const struct msghdr *msg, int flags)
  Send a message on a RTIPC socket.
- ssize_t recvmsg__AF_RTIPC (int sockfd, struct msghdr *msg, int flags)
  Receive a message from a RTIPC socket.
- int getsockname__AF_RTIPC (int sockfd, struct sockaddr_ipc *addr, socklen_t *addrlen)
  Get socket name.
- int getpeername__AF_RTIPC (int sockfd, struct sockaddr_ipc *addr, socklen_t *addrlen)
  Get socket peer.

XDDP socket options

Setting and getting XDDP socket options.

- #define XDDP_LABEL 1
  XDDP label assignment.
- #define XDDP_POOLSZ 2
  XDDP local pool size configuration.
- #define XDDP_BUFSZ 3
  XDDP streaming buffer size configuration.
- #define XDDP_MONITOR 4
  XDDP monitoring callback.

XDDP events

Specific events occurring on XDDP channels, which can be monitored via the XDDP_MONITOR socket option.

- #define XDDP_EVTIN 1
  Monitor writes to the non real-time endpoint.
- #define XDDP_EVTOUT 2
  Monitor reads from the non real-time endpoint.
- #define XDDP_EVTDOWN 3
  Monitor close from the non real-time endpoint.
- #define XDDP_EVTNOBUF 4
  Monitor memory shortage for non real-time datagrams.

IDDP socket options

Setting and getting IDDP socket options.

- #define IDDP_LABEL 1
  IDDP label assignment.
- #define IDDP_POOLSZ 2
  IDDP local pool size configuration.

BUFP socket options

Setting and getting BUFP socket options.

- #define BUFP_LABEL 1
  BUFP label assignment.
- #define BUFP_BUFSZ 2
  BUFP buffer size configuration.

Socket level options

Setting and getting supported standard socket level options.

- #define SO_SNDTIMEO defined_by_kernel_header_file
  IPCPROTO_IDDP and IPCPROTO_BUFP protocols support the standard SO_SNDTIMEO socket option, from the `SOL_SOCKET` level.
- #define SO_RCVTIMEO defined_by_kernel_header_file
  All RTIPC protocols support the standard SO_RCVTIMEO socket option, from the `SOL_SOCKET` level.

5.39.1 Detailed Description

Profile Revision: 1

Device Characteristics

Device Flags: `RTDM_PROTOCOL_DEVICE`
Protocol Family: `PF_RTIPC`
Socket Type: `SOCK_DGRAM`
Device Class: `RTDM_CLASS_RTIPC`

5.39.2 Define Documentation

5.39.2.1 #define BUFP_BUFSZ 2

BUFP buffer size configuration.

All messages written to a BUFP socket are buffered in a single per-socket memory area. Configuring the size of such buffer prior to binding the socket to a destination port is mandatory.

It is not allowed to configure a buffer size after the socket was bound. However, multiple configuration calls are allowed prior to the binding; the last value set will be used.

Note

: the buffer memory is obtained from the host allocator by the bind call.

Parameters

`[in]` *level* SOL_BUFP

`[in]` *optname* **BUFP_BUFSZ**

`[in]` *optval* Pointer to a variable of type size_t, containing the required size of the buffer to reserve at binding time

`[in]` *optlen* sizeof(size_t)

Returns

0 is returned upon success. Otherwise:

- -EFAULT (Invalid data address given)
- -EALREADY (socket already bound)
- -EINVAL (*optlen* is invalid or **optval* is zero)

Calling context:

RT/non-RT

5.39.2.2 #define BUFP_LABEL 1

BUFP label assignment.

ASCII label strings can be attached to BUFP ports, in order to connect sockets to them in a more descriptive way than using plain numeric port values.

When available, this label will be registered when binding, in addition to the port number (see BUFP port binding).

It is not allowed to assign a label after the socket was bound. However, multiple assignment calls are allowed prior to the binding; the last label set will be used.

Parameters

`[in]` ***level*** SOL_BUFP

`[in]` ***optname*** **BUFP_LABEL**

`[in]` ***optval*** Pointer to struct rtipc_port_label

`[in]` ***optlen*** sizeof(struct rtipc_port_label)

Returns

0 is returned upon success. Otherwise:

- -EFAULT (Invalid data address given)
- -EALREADY (socket already bound)
- -EINVAL (*optlen* is invalid)

Calling context:

RT/non-RT

5.39.2.3 #define IDDP_LABEL 1

IDDP label assignment.

ASCII label strings can be attached to IDDP ports, in order to connect sockets to them in a more descriptive way than using plain numeric port values.

When available, this label will be registered when binding, in addition to the port number (see IDDP port binding).

It is not allowed to assign a label after the socket was bound. However, multiple assignment calls are allowed prior to the binding; the last label set will be used.

Parameters

`[in]` ***level*** SOL_IDDP

`[in]` ***optname*** **IDDP_LABEL**

`[in]` ***optval*** Pointer to struct rtipc_port_label

`[in]` ***optlen*** sizeof(struct rtipc_port_label)

Returns

0 is returned upon success. Otherwise:

- -EFAULT (Invalid data address given)

- -EALREADY (socket already bound)
- -EINVAL (*optlen* is invalid)

Calling context:

RT/non-RT

5.39.2.4 #define IDDP_POOLSZ 2

IDDP local pool size configuration.

By default, the memory needed to convey the data is pulled from Xenomai's system pool. Setting a local pool size overrides this default for the socket.

If a non-zero size was configured, a local pool is allocated at binding time. This pool will provide storage for pending datagrams.

It is not allowed to configure a local pool size after the socket was bound. However, multiple configuration calls are allowed prior to the binding; the last value set will be used.

Note

: the pool memory is obtained from the host allocator by the bind call.

Parameters

`[in]` ***level*** SOL_IDDP

`[in]` ***optname*** **IDDP_POOLSZ**

`[in]` ***optval*** Pointer to a variable of type size_t, containing the required size of the local pool to reserve at binding time

`[in]` ***optlen*** sizeof(size_t)

Returns

0 is returned upon success. Otherwise:

- -EFAULT (Invalid data address given)
- -EALREADY (socket already bound)
- -EINVAL (*optlen* is invalid or **optval* is zero)

Calling context:

RT/non-RT

5.39.2.5 #define SO_RCVTIMEO defined_by_kernel_header_file

All RTIPC protocols support the standard SO_RCVTIMEO socket option, from the `SOL_SOCKET` level.

See also

`setsockopt()`, `getsockopt()` in IEEE Std 1003.1, `http://www.opengroup.org/onlinepubs/009695399/`

5.39.2.6 #define SO_SNDTIMEO defined_by_kernel_header_file

IPCPROTO_IDDP and IPCPROTO_BUFP protocols support the standard SO_SNDTIMEO socket option, from the `SOL_SOCKET` level.

See also

`setsockopt()`, `getsockopt()` in IEEE Std 1003.1, `http://www.opengroup.org/onlinepubs/009695399/`

5.39.2.7 #define XDDP_BUFSZ 3

XDDP streaming buffer size configuration.

In addition to sending datagrams, real-time threads may stream data in a byte-oriented mode through the port as well. This increases the bandwidth and reduces the overhead, when the overall data to send to the Linux domain is collected by bits, and keeping the message boundaries is not required.

This feature is enabled when a non-zero buffer size is set for the socket. In that case, the real-time data accumulates into the streaming buffer when MSG_MORE is passed to any of the send functions, until:

- the receiver from the Linux domain wakes up and consumes it,
- a different source port attempts to send data to the same destination port,
- MSG_MORE is absent from the send flags,
- the buffer is full,

whichever comes first.

Setting **optval* to zero disables the streaming buffer, in which case all sendings are conveyed in separate datagrams, regardless of MSG_MORE.

Note

only a single streaming buffer exists per socket. When this buffer is full, the real-time data stops accumulating and sending operations resume in mere datagram mode. Accumulation may happen again after some or all data in the streaming buffer is consumed from the Linux domain endpoint.

The streaming buffer size may be adjusted multiple times during the socket lifetime; the latest configuration change will take effect when the accumulation resumes after the previous buffer was flushed.

Parameters

`[in]` ***level*** SOL_XDDP

`[in]` ***optname*** **XDDP_BUFSZ**

`[in]` ***optval*** Pointer to a variable of type size_t, containing the required size of the streaming buffer

`[in]` ***optlen*** sizeof(size_t)

Returns

0 is returned upon success. Otherwise:

- -EFAULT (Invalid data address given)
- -ENOMEM (Not enough memory)
- -EINVAL (*optlen* is invalid)

Calling context:

RT/non-RT

5.39.2.8 #define XDDP_EVTDOWN 3

Monitor close from the non real-time endpoint.

XDDP_EVTDOWN is sent when the non real-time endpoint is closed. The argument is always 0.

5.39.2.9 #define XDDP_EVTIN 1

Monitor writes to the non real-time endpoint.

XDDP_EVTIN is sent when data is written to the non real-time endpoint the socket is bound to (i.e. via /dev/rtp*N*), which means that some input is pending for the real-time endpoint. The argument is the size of the incoming message.

5.39.2.10 #define XDDP_EVTNOBUF 4

Monitor memory shortage for non real-time datagrams.

XDDP_EVTNOBUF is sent when no memory is available from the pool to hold the message currently sent from the non real-time endpoint. The argument is the size of the failed allocation. Upon return from the callback, the caller will block and retry until enough space is available from the pool; during that process, the callback might be invoked multiple times, each time a new attempt to get the required memory fails.

5.39.2.11 #define XDDP_EVTOUT 2

Monitor reads from the non real-time endpoint.

XDDP_EVTOUT is sent when the non real-time endpoint successfully reads a complete message (i.e. via /dev/rtp*N*). The argument is the size of the outgoing message.

5.39.2.12 #define XDDP_LABEL 1

XDDP label assignment.

ASCII label strings can be attached to XDDP ports, so that opening the non-RT endpoint can be done by specifying this symbolic device name rather than referring to a raw pseudo-device entry (i.e. /dev/rtp*N*).

When available, this label will be registered when binding, in addition to the port number (see XDDP port binding).

It is not allowed to assign a label after the socket was bound. However, multiple assignment calls are allowed prior to the binding; the last label set will be used.

Parameters

[in] ***level*** SOL_XDDP

[in] ***optname*** **XDDP_LABEL**

[in] ***optval*** Pointer to struct rtipc_port_label

[in] ***optlen*** sizeof(struct rtipc_port_label)

Returns

0 is returned upon success. Otherwise:

- -EFAULT (Invalid data address given)
- -EALREADY (socket already bound)
- -EINVAL (*optlen* invalid)

Calling context:

RT/non-RT

5.39.2.13 #define XDDP_MONITOR 4

XDDP monitoring callback.

Other RTDM drivers may install a user-defined callback via the rtdm_setsockopt call from the inter-driver API, in order to collect particular events occurring on the channel.

This notification mechanism is particularly useful to monitor a channel asynchronously while performing other tasks.

The user-provided routine will be passed the RTDM file descriptor of the socket receiving the event, the event code, and an optional argument. Four events are currently defined, see XDDP_-EVENTS.

The XDDP_EVTIN and XDDP_EVTOUT events are fired on behalf of a fully atomic context; therefore, care must be taken to keep their overhead low. In those cases, the Xenomai services that may be called from the callback are restricted to the set allowed to a real-time interrupt handler.

Parameters

[in] ***level*** SOL_XDDP

[in] ***optname*** **XDDP_MONITOR**

[in] ***optval*** Pointer to a pointer to function of type int (*)(int fd, int event, long arg), containing the address of the user-defined callback.Passing a NULL callback pointer in *optval* disables monitoring.

[in] ***optlen*** sizeof(int (*)(int fd, int event, long arg))

Returns

0 is returned upon success. Otherwise:

- -EFAULT (Invalid data address given)
- -EPERM (Operation not allowed from user-space)
- -EINVAL (*optlen* is invalid)

Calling context:

RT/non-RT, kernel space only

5.39.2.14 #define XDDP_POOLSZ 2

XDDP local pool size configuration.

By default, the memory needed to convey the data is pulled from Xenomai's system pool. Setting a local pool size overrides this default for the socket.

If a non-zero size was configured, a local pool is allocated at binding time. This pool will provide storage for pending datagrams.

It is not allowed to configure a local pool size after the socket was bound. However, multiple configuration calls are allowed prior to the binding; the last value set will be used.

Note

: the pool memory is obtained from the host allocator by the bind call.

Parameters

`[in]` ***level*** SOL_XDDP

`[in]` ***optname*** **XDDP_POOLSZ**

`[in]` ***optval*** Pointer to a variable of type size_t, containing the required size of the local pool to reserve at binding time

`[in]` ***optlen*** sizeof(size_t)

Returns

0 is returned upon success. Otherwise:

- -EFAULT (Invalid data address given)
- -EALREADY (socket already bound)
- -EINVAL (*optlen* invalid or **optval* is zero)

Calling context:

RT/non-RT

5.39.3 Enumeration Type Documentation

5.39.3.1 anonymous enum

Enumerator:

IPCPROTO_IPC Default protocol (IDDP).

IPCPROTO_XDDP Cross-domain datagram protocol (RT <-> non-RT). Real-time Xenomai threads and regular Linux threads may want to exchange data in a way that does not require the former to leave the real-time domain (i.e. secondary mode). The RTDM-based XDDP protocol is available for this purpose.

On the Linux domain side, pseudo-device files named /dev/rtp<*minor*> give regular POSIX threads access to non real-time communication endpoints, via the standard character-based I/O interface. On the Xenomai domain side, sockets may be bound to XDDP ports, which act as proxies to send and receive data to/from the associated pseudo-device files. Ports and pseudo-device minor numbers are paired, meaning that e.g. port 7 will proxy the traffic to/from /dev/rtp7.

All data sent through a bound/connected XDDP socket via `sendto(2)` or `write(2)` will be passed to the peer endpoint in the Linux domain, and made available for reading via the standard `read(2)` system call. Conversely, all data sent using `write(2)` through the non real-time endpoint will be conveyed to the real-time socket endpoint, and made available to the `recvfrom(2)` or `read(2)` system calls.

IPCPROTO_IDDP Intra-domain datagram protocol (RT <-> RT). The RTDM-based IDDP protocol enables real-time threads to exchange datagrams within the Xenomai domain, via socket endpoints.

IPCPROTO_BUFP Buffer protocol (RT <-> RT, byte-oriented). The RTDM-based BUFP protocol implements a lightweight, byte-oriented, one-way Producer-Consumer data path. All messages written are buffered into a single memory area in strict FIFO order, until read by the consumer.

This protocol always prevents short writes, and only allows short reads when a potential deadlock situation arises (i.e. readers and writers waiting for each other indefinitely).

5.39.4 Function Documentation

5.39.4.1 int bind__AF_RTIPC (int *sockfd*, const struct sockaddr_ipc * *addr*, socklen_t *addrlen*)

Bind a RTIPC socket to a port.

Bind the socket to a destination port.

Parameters

`[in]` ***sockfd*** The RTDM file descriptor obtained from the socket creation call.

`[in]` ***addr*** The address to bind the socket to (see struct sockaddr_ipc). The meaning of such address depends on the RTIPC protocol in use for the socket:

- IPCPROTO_XDDP

This action creates an endpoint for channelling traffic between the Xenomai and Linux domains.

sipc_family must be AF_RTIPC, *sipc_port* is either -1, or a valid free port number between 0 and CONFIG_XENO_OPT_PIPE_NRDEV-1.

If *sipc_port* is -1, a free port will be assigned automatically.

Upon success, the pseudo-device /dev/rtp*N* will be reserved for this communication channel, where *N* is the assigned port number. The non real-time side shall open this device to exchange data over the bound socket.

If a label was assigned (see XDDP_LABEL) prior to binding the socket to a port, a registry link referring to the created pseudo-device will be automatically set up as `/proc/xenomai/registry/rtipc/xddp/label`, where *label* is the label string passed to setsockopt() for the XDDP_LABEL option.

- IPCPROTO_IDDP

This action creates an endpoint for exchanging datagrams within the Xenomai domain.

sipc_family must be AF_RTIPC, *sipc_port* is either -1, or a valid free port number between 0 and CONFIG_XENO_OPT_IDDP_NRPORT-1.

If *sipc_port* is -1, a free port will be assigned automatically. The real-time peer shall connect to the same port for exchanging data over the bound socket.

If a label was assigned (see IDDP_LABEL) prior to binding the socket to a port, a registry link referring to the assigned port number will be automatically set up as `/proc/xenomai/registry/rtipc/iddp/label`, where *label* is the label string passed to setsockopt() for the IDDP_LABEL option.

- IPCPROTO_BUFP

This action creates an endpoint for a one-way byte stream within the Xenomai domain.

sipc_family must be AF_RTIPC, *sipc_port* is either -1, or a valid free port number between 0 and CONFIG_XENO_OPT_BUFP_NRPORT-1.

If *sipc_port* is -1, an available port will be assigned automatically. The real-time peer shall connect to the same port for exchanging data over the bound socket.

If a label was assigned (see BUFP_LABEL) prior to binding the socket to a port, a registry link referring to the assigned port number will be automatically set up as `/proc/xenomai/registry/rtipc/bufp/label`, where *label* is the label string passed to setsockopt() for the *BUFP_LABEL* option.

Parameters

`[in]` ***addrlen*** The size in bytes of the structure pointed to by *addr*.

Returns

In addition to the standard error codes for `bind(2)`, the following specific error code may be returned:

- -EFAULT (Invalid data address given)
- -ENOMEM (Not enough memory)
- -EINVAL (Invalid parameter)
- -EADDRINUSE (Socket already bound to a port, or no port available)

Calling context:

non-RT

5.39.4.2 int close__AF_RTIPC (int *sockfd*)

Close a RTIPC socket descriptor.

Blocking calls to any of the sendmsg or recvmsg functions will be unblocked when the socket is closed and return with an error.

Returns

In addition to the standard error codes for `close(2)`, the following specific error code may be returned: none

Calling context:

non-RT

5.39.4.3 int connect__AF_RTIPC (int *sockfd*, const struct sockaddr_ipc * *addr*, socklen_t *addrlen*)

Initiate a connection on a RTIPC socket.

Parameters

`[in]` ***sockfd*** The RTDM file descriptor obtained from the socket creation call.

`[in]` ***addr*** The address to connect the socket to (see struct sockaddr_ipc).

- If sipc_port is a valid port for the protocol, it is used verbatim and the connection succeeds immediately, regardless of whether the destination is bound at the time of the call.

- If sipc_port is -1 and a label was assigned to the socket, connect() blocks for the requested amount of time (see SO_RCVTIMEO) until a socket is bound to the same label via `bind(2)` (see XDDP_LABEL, IDDP_LABEL, BUFP_LABEL), in which case a connection is established between both endpoints.

- If sipc_port is -1 and no label was assigned to the socket, the default destination address is cleared, meaning that any subsequent write to the socket will return -EDESTADDRREQ, until a valid destination address is set via `connect(2)` or `bind(2)`.

Parameters

`[in]` ***addrlen*** The size in bytes of the structure pointed to by *addr*.

Returns

In addition to the standard error codes for `connect(2)`, the following specific error code may be returned: none.

Calling context:

RT/non-RT

5.39.4.4 int getpeername__AF_RTIPC (int *sockfd*, struct sockaddr_ipc * *addr*, socklen_t * *addrlen*)

Get socket peer.

The name of the remote endpoint for the socket is copied back (see struct sockaddr_ipc). This is the default destination address for messages sent on the socket. It can be set either explicitly via `connect(2)`, or implicitly via `bind(2)` if no `connect(2)` was called prior to binding the socket to a port, in which case both the local and remote names are equal.

Returns

In addition to the standard error codes for `getpeername(2)`, the following specific error code may be returned: none.

Calling context:

RT/non-RT

5.39.4.5 int getsockname__AF_RTIPC (int *sockfd*, struct sockaddr_ipc * *addr*, socklen_t * *addrlen*)

Get socket name.

The name of the local endpoint for the socket is copied back (see struct sockaddr_ipc).

Returns

In addition to the standard error codes for `getsockname(2)`, the following specific error code may be returned: none.

Calling context:

RT/non-RT

5.39.4.6 int getsockopt__AF_RTIPC (int *sockfd*, int *level*, int *optname*, void * *optval*, socklen_t * *optlen*)

Get options on RTIPC sockets.

These functions allow to get various socket options. Supported Levels and Options:

- Level SOL_SOCKET
- Level SOL_XDDP
- Level SOL_IDDP
- Level SOL_BUFP

Returns

In addition to the standard error codes for `getsockopt(2)`, the following specific error code may be returned: follow the option links above.

Calling context:

RT/non-RT

5.39.4.7 ssize_t recvmsg__AF_RTIPC (int *sockfd*, struct msghdr * *msg*, int *flags*)

Receive a message from a RTIPC socket.

Parameters

`[in]` ***sockfd*** The RTDM file descriptor obtained from the socket creation call.

`[out]` ***msg*** The address the message header will be copied at.

`[in]` ***flags*** Operation flags:

- MSG_DONTWAIT Non-blocking I/O operation. The caller will not be blocked whenever no message is immediately available for receipt at the time of the call, but will rather return with -EWOULDBLOCK.

Note

IPCPROTO_BUFP does not allow for short reads and always returns the requested amount of bytes, except in one situation: whenever some writer is waiting for sending data upon a buffer full condition, while the caller would have to wait for receiving a complete message. This is usually the sign of a pathological use of the BUFP socket, like defining an incorrect buffer size via BUFP_BUFSZ. In that case, a short read is allowed to prevent a deadlock.

Returns

In addition to the standard error codes for `recvmsg(2)`, the following specific error code may be returned: none.

Calling context:

RT

5.39.4.8 ssize_t sendmsg__AF_RTIPC (int *sockfd*, const struct msghdr * *msg*, int *flags*)

Send a message on a RTIPC socket.

Parameters

`[in]` ***sockfd*** The RTDM file descriptor obtained from the socket creation call.

`[in]` ***msg*** The address of the message header conveying the datagram.

`[in]` ***flags*** Operation flags:

- MSG_OOB Send out-of-band message. For all RTIPC protocols except IPCPROTO_BUFP, sending out-of-band data actually means pushing them to the head of the receiving queue, so that the reader will always receive them before normal messages. IPCPROTO_BUFP does not support out-of-band sending.

- MSG_DONTWAIT Non-blocking I/O operation. The caller will not be blocked whenever the message cannot be sent immediately at the time of the call (e.g. memory shortage), but will rather return with -EWOULDBLOCK. Unlike other RTIPC protocols, IPCPROTO_XDDP accepts but never considers MSG_DONTWAIT since writing to a real-time XDDP endpoint is inherently a non-blocking operation.

- MSG_MORE Accumulate data before sending. This flag is accepted by the IPCPROTO_-XDDP protocol only, and tells the send service to accumulate the outgoing data into an internal streaming buffer, instead of issuing a datagram immediately for it. See XDDP_-BUFSZ for more.

Note

No RTIPC protocol allows for short writes, and only complete messages are sent to the peer.

Returns

In addition to the standard error codes for `sendmsg(2)`, the following specific error code may be returned: none.

Calling context:

RT

5.39.4.9 int setsockopt__AF_RTIPC (int *sockfd,* int *level,* int *optname,* const void * *optval,* socklen_t *optlen*)

Set options on RTIPC sockets.

These functions allow to set various socket options. Supported Levels and Options:

- Level SOL_SOCKET
- Level SOL_XDDP
- Level SOL_IDDP
- Level SOL_BUFP

Returns

In addition to the standard error codes for `setsockopt(2)`, the following specific error code may be returned: follow the option links above.

Calling context:

non-RT

5.39.4.10 int socket__AF_RTIPC (int *domain* = *AF_RTIPC,* int *type* = *SOCK_DGRAM,* int *protocol*)

Create an endpoint for communication in the AF_RTIPC domain.

Parameters

`[in]` *domain* The communication domain. Must be AF_RTIPC.

`[in]` *type* The socket type. Must be SOCK_DGRAM.

`[in]` *protocol* Any of IPCPROTO_XDDP, IPCPROTO_IDDP, or IPCPROTO_BUFP. IPCPROTO_IPC is also valid, and refers to the default RTIPC protocol, namely IPCPROTO_IDDP.

Returns

In addition to the standard error codes for `socket(2)`, the following specific error code may be returned:

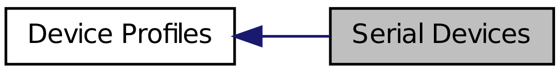
- -ENOPROTOOPT (Protocol is known, but not compiled in the RTIPC driver). See RTIPC protocols for available protocols.

Calling context:

non-RT

5.40 Serial Devices

Collaboration diagram for Serial Devices:



Data Structures

- struct rtser_config

  Serial device configuration.

- struct rtser_status

  Serial device status.

- struct rtser_event

  Additional information about serial device events.

Files

- file rtserial.h

  Real-Time Driver Model for Xenomai, serial device profile header.

Defines

- #define RTSER_RTIOC_BREAK_CTL _IOR(RTIOC_TYPE_SERIAL, 0x06, int)

  Set or clear break on UART output line.

RTSER_BREAK_xxx

Break control

- typedef struct rtser_config rtser_config_t
    Serial device configuration.
- typedef struct rtser_status rtser_status_t
    Serial device status.
- typedef struct rtser_event rtser_event_t
    Additional information about serial device events.
- #define RTSER_BREAK_CLR 0x00
    Serial device configuration.
- #define RTSER_BREAK_SET 0x01
    Serial device configuration.
- #define RTIOC_TYPE_SERIAL RTDM_CLASS_SERIAL
    Serial device configuration.

RTSER_DEF_BAUD

Default baud rate

- #define **RTSER_DEF_BAUD** 9600

RTSER_xxx_PARITY

Number of parity bits

- #define **RTSER_NO_PARITY** 0x00
- #define **RTSER_ODD_PARITY** 0x01
- #define **RTSER_EVEN_PARITY** 0x03
- #define **RTSER_DEF_PARITY** RTSER_NO_PARITY

RTSER_xxx_BITS

Number of data bits

- #define **RTSER_5_BITS** 0x00
- #define **RTSER_6_BITS** 0x01
- #define **RTSER_7_BITS** 0x02
- #define **RTSER_8_BITS** 0x03
- #define **RTSER_DEF_BITS** RTSER_8_BITS

RTSER_xxx_STOPB

Number of stop bits

- #define RTSER_1_STOPB 0x00
  valid only in combination with 5 data bits
- #define RTSER_1_5_STOPB 0x01
  valid only in combination with 5 data bits
- #define RTSER_2_STOPB 0x01
  valid only in combination with 5 data bits
- #define RTSER_DEF_STOPB RTSER_1_STOPB
  valid only in combination with 5 data bits

RTSER_xxx_HAND

Handshake mechanisms

- #define **RTSER_NO_HAND** 0x00
- #define **RTSER_RTSCTS_HAND** 0x01
- #define **RTSER_DEF_HAND** RTSER_NO_HAND

RTSER_FIFO_xxx

Reception FIFO interrupt threshold

- #define **RTSER_FIFO_DEPTH_1** 0x00
- #define **RTSER_FIFO_DEPTH_4** 0x40
- #define **RTSER_FIFO_DEPTH_8** 0x80
- #define **RTSER_FIFO_DEPTH_14** 0xC0
- #define **RTSER_DEF_FIFO_DEPTH** RTSER_FIFO_DEPTH_1

RTSER_TIMEOUT_xxx

Special timeout values, see also RTDM_TIMEOUT_xxx

- #define **RTSER_TIMEOUT_INFINITE** RTDM_TIMEOUT_INFINITE
- #define **RTSER_TIMEOUT_NONE** RTDM_TIMEOUT_NONE
- #define **RTSER_DEF_TIMEOUT** RTDM_TIMEOUT_INFINITE

RTSER_xxx_TIMESTAMP_HISTORY

Timestamp history control

- #define **RTSER_RX_TIMESTAMP_HISTORY** 0x01
- #define **RTSER_DEF_TIMESTAMP_HISTORY** 0x00

RTSER_EVENT_xxx

Events bits

- #define **RTSER_EVENT_RXPEND** 0x01
- #define **RTSER_EVENT_ERRPEND** 0x02
- #define **RTSER_EVENT_MODEMHI** 0x04
- #define **RTSER_EVENT_MODEMLO** 0x08
- #define **RTSER_DEF_EVENT_MASK** 0x00

RTSER_SET_xxx

Configuration mask bits

- #define **RTSER_SET_BAUD** 0x0001
- #define **RTSER_SET_PARITY** 0x0002
- #define **RTSER_SET_DATA_BITS** 0x0004
- #define **RTSER_SET_STOP_BITS** 0x0008
- #define **RTSER_SET_HANDSHAKE** 0x0010
- #define **RTSER_SET_FIFO_DEPTH** 0x0020
- #define **RTSER_SET_TIMEOUT_RX** 0x0100
- #define **RTSER_SET_TIMEOUT_TX** 0x0200
- #define **RTSER_SET_TIMEOUT_EVENT** 0x0400
- #define **RTSER_SET_TIMESTAMP_HISTORY** 0x0800
- #define **RTSER_SET_EVENT_MASK** 0x1000

RTSER_LSR_xxx

Line status bits

- #define **RTSER_LSR_DATA** 0x01
- #define **RTSER_LSR_OVERRUN_ERR** 0x02
- #define **RTSER_LSR_PARITY_ERR** 0x04
- #define **RTSER_LSR_FRAMING_ERR** 0x08
- #define **RTSER_LSR_BREAK_IND** 0x10
- #define **RTSER_LSR_THR_EMTPY** 0x20
- #define **RTSER_LSR_TRANSM_EMPTY** 0x40
- #define **RTSER_LSR_FIFO_ERR** 0x80
- #define **RTSER_SOFT_OVERRUN_ERR** 0x0100

RTSER_MSR_xxx

Modem status bits

- #define **RTSER_MSR_DCTS** 0x01
- #define **RTSER_MSR_DDSR** 0x02
- #define **RTSER_MSR_TERI** 0x04
- #define **RTSER_MSR_DDCD** 0x08

- #define **RTSER_MSR_CTS** 0x10
- #define **RTSER_MSR_DSR** 0x20
- #define **RTSER_MSR_RI** 0x40
- #define **RTSER_MSR_DCD** 0x80

RTSER_MCR_xxx

Modem control bits

- #define **RTSER_MCR_DTR** 0x01
- #define **RTSER_MCR_RTS** 0x02
- #define **RTSER_MCR_OUT1** 0x04
- #define **RTSER_MCR_OUT2** 0x08
- #define **RTSER_MCR_LOOP** 0x10

Sub-Classes of RTDM_CLASS_SERIAL

- #define **RTDM_SUBCLASS_16550A** 0

IOCTLs

Serial device IOCTLs

- #define RTSER_RTIOC_GET_CONFIG _IOR(RTIOC_TYPE_SERIAL, 0x00, struct rtser_-config)

    Get serial device configuration.

- #define RTSER_RTIOC_SET_CONFIG _IOW(RTIOC_TYPE_SERIAL, 0x01, struct rtser_-config)

    Set serial device configuration.

- #define RTSER_RTIOC_GET_STATUS _IOR(RTIOC_TYPE_SERIAL, 0x02, struct rtser_-status)

    Get serial device status.

- #define RTSER_RTIOC_GET_CONTROL _IOR(RTIOC_TYPE_SERIAL, 0x03, int)

    Get serial device's modem contol register.

- #define RTSER_RTIOC_SET_CONTROL _IOW(RTIOC_TYPE_SERIAL, 0x04, int)

    Set serial device's modem contol register.

- #define RTSER_RTIOC_WAIT_EVENT _IOR(RTIOC_TYPE_SERIAL, 0x05, struct rtser_-event)

    Wait on serial device events according to previously set mask.

5.40.1 Detailed Description

This is the common interface a RTDM-compliant serial device has to provide. Feel free to comment on this profile via the Xenomai mailing list (Xenomai-core@gna.org) or directly to the author (jan.kiszka@web.de).

Profile Revision: 3

Device Characteristics

Device Flags: `RTDM_NAMED_DEVICE`, `RTDM_EXCLUSIVE`
Device Name: `"rtser<N>"`, N >= 0
Device Class: `RTDM_CLASS_SERIAL`

Supported Operations

Open
Environments: non-RT (RT optional, deprecated)
Specific return values: none
Close
Environments: non-RT (RT optional, deprecated)
Specific return values: none
IOCTL
Mandatory Environments: see below
Specific return values: see below
Read
Environments: RT (non-RT optional)
Specific return values:

- -ETIMEDOUT
- -EINTR (interrupted explicitly or by signal)
- -EAGAIN (no data available in non-blocking mode)
- -EBADF (device has been closed while reading)
- -EIO (hardware error or broken bit stream)

Write
Environments: RT (non-RT optional)
Specific return values:

- -ETIMEDOUT
- -EINTR (interrupted explicitly or by signal)
- -EAGAIN (no data written in non-blocking mode)
- -EBADF (device has been closed while writing)

5.40.2 Define Documentation

5.40.2.1 #define RTSER_RTIOC_BREAK_CTL _IOR(RTIOC_TYPE_SERIAL, 0x06, int)

Set or clear break on UART output line.

Parameters

`[in]` *arg* `RTSER_BREAK_SET` or `RTSER_BREAK_CLR` (int)

Returns

0 on success, otherwise negative error code

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Note

A set break condition may also be cleared on UART line reconfiguration.

Rescheduling: never.

5.40.2.2 #define RTSER_RTIOC_GET_CONFIG _IOR(RTIOC_TYPE_SERIAL, 0x00, struct rtser_config)

Get serial device configuration.

Parameters

`[out]` *arg* Pointer to configuration buffer (struct rtser_config)

Returns

0 on success, otherwise negative error code

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.40.2.3 #define RTSER_RTIOC_GET_CONTROL _IOR(RTIOC_TYPE_SERIAL, 0x03, int)

Get serial device's modem contol register.

Parameters

`[out]` *arg* Pointer to variable receiving the content (int, see RTSER_MCR_xxx)

Returns

0 on success, otherwise negative error code

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.40.2.4 #define RTSER_RTIOC_GET_STATUS _IOR(RTIOC_TYPE_SERIAL, 0x02, struct rtser_status)

Get serial device status.

Parameters

`[out]` *arg* Pointer to status buffer (struct rtser_status)

Returns

0 on success, otherwise negative error code

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Note

The error states `RTSER_LSR_OVERRUN_ERR`, `RTSER_LSR_PARITY_ERR`, `RTSER_LSR_FRAMING_ERR`, and `RTSER_SOFT_OVERRUN_ERR` that may have occured during previous read accesses to the device will be saved for being reported via this IOCTL. Upon return from `RTSER_RTIOC_-GET_STATUS`, the saved state will be cleared.

Rescheduling: never.

5.40.2.5 #define RTSER_RTIOC_SET_CONFIG _IOW(RTIOC_TYPE_SERIAL, 0x01, struct rtser_config)

Set serial device configuration.

Parameters

`[in]` *arg* Pointer to configuration buffer (struct rtser_config)

Returns

0 on success, otherwise:

- -EPERM is returned if the caller's context is invalid, see note below.
- -ENOMEM is returned if a new history buffer for timestamps cannot be allocated.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Note

If rtser_config contains a valid timestamp_history and the addressed device has been opened in non-real-time context, this IOCTL must be issued in non-real-time context as well. Otherwise, this command will fail.

Rescheduling: never.

5.40.2.6 #define RTSER_RTIOC_SET_CONTROL _IOW(RTIOC_TYPE_SERIAL, 0x04, int)

Set serial device's modem contol register.

Parameters

`[in]` *arg* New control register content (int, see RTSER_MCR_xxx)

Returns

0 on success, otherwise negative error code

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.40.2.7 #define RTSER_RTIOC_WAIT_EVENT _IOR(RTIOC_TYPE_SERIAL, 0x05, struct rtser_event)

Wait on serial device events according to previously set mask.

Parameters

`[out]` *arg* Pointer to event information buffer (struct rtser_event)

Returns

0 on success, otherwise:

- -EBUSY is returned if another task is already waiting on events of this device.
- -EBADF is returned if the file descriptor is invalid or the device has just been closed.

Environments:

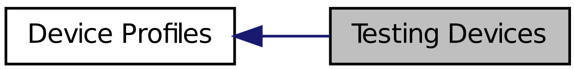
This service can be called from:

- Kernel-based task
- User-space task (RT)

Rescheduling: possible.

5.41 Testing Devices

Collaboration diagram for Testing Devices:



Files

- file rttesting.h
  Real-Time Driver Model for Xenomai, testing device profile header.

Sub-Classes of RTDM_CLASS_TESTING

- #define RTDM_SUBCLASS_TIMERBENCH 0
  subclass name: "timerbench"
- #define RTDM_SUBCLASS_IRQBENCH 1
  subclass name: "irqbench"
- #define RTDM_SUBCLASS_SWITCHTEST 2
  subclass name: "switchtest"
- #define RTDM_SUBCLASS_RTDMTEST 3
  subclase name: "rtdm"

IOCTLs

Testing device IOCTLs

- #define **RTTST_RTIOC_INTERM_BENCH_RES** _IOWR(RTIOC_TYPE_TESTING, 0x00, struct rttst_interm_bench_res)
- #define **RTTST_RTIOC_TMBENCH_START** _IOW(RTIOC_TYPE_TESTING, 0x10, struct rttst_tmbench_config)
- #define **RTTST_RTIOC_TMBENCH_STOP** _IOWR(RTIOC_TYPE_TESTING, 0x11, struct rttst_overall_bench_res)
- #define **RTTST_RTIOC_IRQBENCH_START** _IOW(RTIOC_TYPE_TESTING, 0x20, struct rttst_irqbench_config)
- #define **RTTST_RTIOC_IRQBENCH_STOP** _IO(RTIOC_TYPE_TESTING, 0x21)
- #define **RTTST_RTIOC_IRQBENCH_GET_STATS** _IOR(RTIOC_TYPE_TESTING, 0x22, struct rttst_irqbench_stats)
- #define **RTTST_RTIOC_IRQBENCH_WAIT_IRQ** _IO(RTIOC_TYPE_TESTING, 0x23)
- #define **RTTST_RTIOC_IRQBENCH_REPLY_IRQ** _IO(RTIOC_TYPE_TESTING, 0x24)
- #define **RTTST_RTIOC_SWTEST_SET_TASKS_COUNT** _IOW(RTIOC_TYPE_TESTING, 0x30, unsigned long)
- #define **RTTST_RTIOC_SWTEST_SET_CPU** _IOW(RTIOC_TYPE_TESTING, 0x31, unsigned long)
- #define **RTTST_RTIOC_SWTEST_REGISTER_UTASK** _IOW(RTIOC_TYPE_TESTING, 0x32, struct rttst_swtest_task)
- #define **RTTST_RTIOC_SWTEST_CREATE_KTASK** _IOWR(RTIOC_TYPE_TESTING, 0x33, struct rttst_swtest_task)
- #define **RTTST_RTIOC_SWTEST_PEND** _IOR(RTIOC_TYPE_TESTING, 0x34, struct rttst_swtest_task)
- #define **RTTST_RTIOC_SWTEST_SWITCH_TO** _IOR(RTIOC_TYPE_TESTING, 0x35, struct rttst_swtest_dir)
- #define **RTTST_RTIOC_SWTEST_GET_SWITCHES_COUNT** _IOR(RTIOC_TYPE_TESTING, 0x36, unsigned long)
- #define **RTTST_RTIOC_SWTEST_GET_LAST_ERROR** _IOR(RTIOC_TYPE_TESTING, 0x37, struct rttst_swtest_error)
- #define **RTTST_RTIOC_SWTEST_SET_PAUSE** _IOW(RTIOC_TYPE_TESTING, 0x38, unsigned long)
- #define **RTTST_RTIOC_RTDM_DEFER_CLOSE** _IOW(RTIOC_TYPE_TESTING, 0x40, unsigned long)

5.41.1 Detailed Description

This group of devices is intended to provide in-kernel testing results. Feel free to comment on this profile via the Xenomai mailing list (`xenomai-core@gna.org`) or directly to the author (`jan.kiszka@web.de`).

Profile Revision: 2

Device Characteristics

Device Flags: `RTDM_NAMED_DEVICE`
Device Name: `"rttest[-<subclass>]<N>"`, N >= 0, optional subclass name to simplify device discovery
Device Class: `RTDM_CLASS_TESTING`

Supported Operations

Open
Environments: non-RT (RT optional, deprecated)
Specific return values: none
Close
Environments: non-RT (RT optional, deprecated)
Specific return values: none
IOCTL
Mandatory Environments: see TSTIOCTLs below
Specific return values: see TSTIOCTLs below

5.42 Sched

Data Structures

- struct xnsched

  Scheduling information structure.

Files

- file sched.h

  Scheduler interface header.

- file sched-idle.c

  Idle scheduling class implementation (i.e. Linux placeholder).

- file sched-rt.c

  Common real-time scheduling class implementation (FIFO + RR).

- file sched-sporadic.c

  POSIX SCHED_SPORADIC scheduling class.

- file sched-tp.c

  Temporal partitioning (typical of IMA systems).

- file sched.c

Typedefs

- typedef struct xnsched xnsched_t

  Scheduling information structure.

Functions

- static void xnsched_rotate (struct xnsched *sched, struct xnsched_class *sched_class, const union xnsched_policy_param *sched_param)

Rotate a scheduler runqueue.

5.42.1 Function Documentation

5.42.1.1 void xnsched_rotate (struct xnsched * *sched,* struct xnsched_class * *sched_class,* const union xnsched_policy_param * *sched_param*) [inline, static]

Rotate a scheduler runqueue.

The specified scheduling class is requested to rotate its runqueue for the given scheduler. Rotation is performed according to the scheduling parameter specified by *sched_param*.

Note

The nucleus supports round-robin scheduling for the members of the RT class.

Parameters

sched The per-CPU scheduler hosting the target scheduling class.

sched_class The scheduling class which should rotate its runqueue.

sched_param The scheduling parameter providing rotation information to the specified scheduling class.

Environments:

This service should be called from:

- Kernel-based task
- Interrupt service routine
- User-space task (primary mode only)

Rescheduling: never.

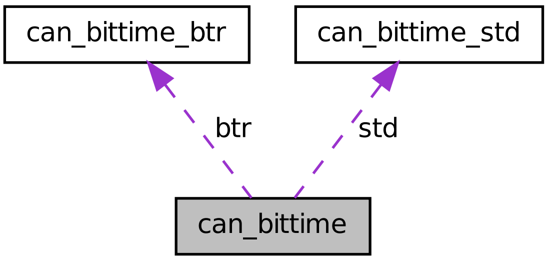
Chapter 6

Data Structure Documentation

6.1 can_bittime Struct Reference

Custom CAN bit-time definition.

Collaboration diagram for can_bittime:



Data Fields

- can_bittime_type_t type

    Type of bit-time definition.

- struct can_bittime_std std

    Standard bit-time.

- struct can_bittime_btr btr

    Hardware-spcific BTR bit-time.

6.1.1 Detailed Description

Custom CAN bit-time definition.

The documentation for this struct was generated from the following file:

- include/rtdm/rtcan.h

6.2 can_bittime_btr Struct Reference

Hardware-specific BTR bit-times.

Data Fields

- uint8_t btr0
  Bus timing register 0.
- uint8_t btr1
  Bus timing register 1.

6.2.1 Detailed Description

Hardware-specific BTR bit-times.

The documentation for this struct was generated from the following file:

- include/rtdm/rtcan.h

6.3 can_bittime_std Struct Reference

Standard bit-time parameters according to Bosch.

Data Fields

- uint32_t brp
  Baud rate prescaler.
- uint8_t prop_seg
  from 1 to 8
- uint8_t phase_seg1
  from 1 to 8
- uint8_t phase_seg2
  from 1 to 8

- uint8_t sjw:7

  from 1 to 4

- uint8_t sam:1

  1 - enable triple sampling

6.3.1 Detailed Description

Standard bit-time parameters according to Bosch.

The documentation for this struct was generated from the following file:

- include/rtdm/rtcan.h

6.4 can_filter Struct Reference

Filter for reception of CAN messages.

Data Fields

- uint32_t can_id

  CAN ID which must match with incoming IDs after passing the mask.

- uint32_t can_mask

  Mask which is applied to incoming IDs.

6.4.1 Detailed Description

Filter for reception of CAN messages. This filter works as follows: A received CAN ID is AND'ed bitwise with `can_mask` and then compared to `can_id`. This also includes the CAN_EFF_FLAG and CAN_RTR_FLAG of CAN_xxx_FLAG. If this comparison is true, the message will be received by the socket. The logic can be inverted with the `can_id` flag CAN_INV_FILTER :

```
if (can_id & CAN_INV_FILTER) {
   if ((received_can_id & can_mask) != (can_id & ~CAN_INV_FILTER))
      accept-message;
} else {
   if ((received_can_id & can_mask) == can_id)
      accept-message;
}
```

Multiple filters can be arranged in a filter list and set with Sockopts. If one of these filters matches a CAN ID upon reception of a CAN frame, this frame is accepted.

6.4.2 Field Documentation

6.4.2.1 uint32_t can_filter::can_id

CAN ID which must match with incoming IDs after passing the mask.

The filter logic can be inverted with the flag CAN_INV_FILTER.

6.4.2.2 uint32_t can_filter::can_mask

Mask which is applied to incoming IDs.

See CAN ID masks if exactly one CAN ID should come through.

The documentation for this struct was generated from the following file:

- include/rtdm/rtcan.h

6.5 can_frame Struct Reference

Raw CAN frame.

Public Member Functions

- uint8_t data[8] __attribute__ ((aligned(8)))
  Payload data bytes.

Data Fields

- can_id_t can_id
  CAN ID of the frame.
- uint8_t can_dlc
  Size of the payload in bytes.

6.5.1 Detailed Description

Raw CAN frame. Central structure for receiving and sending CAN frames.

6.5.2 Field Documentation

6.5.2.1 can_id_t can_frame::can_id

CAN ID of the frame.

See CAN ID flags for special bits.

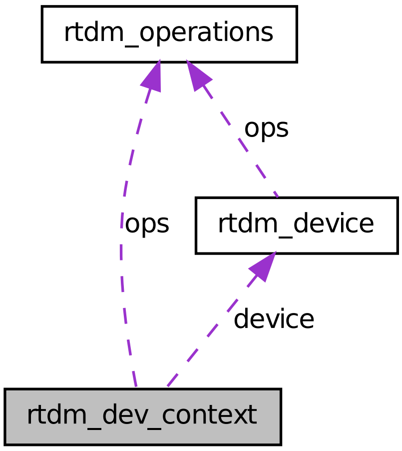
The documentation for this struct was generated from the following file:

- include/rtdm/rtcan.h

6.6 rtdm_dev_context Struct Reference

Device context.

Collaboration diagram for rtdm_dev_context:



Data Fields

- unsigned long context_flags

  Context flags, see Context Flags for details.

- int fd

  Associated file descriptor.

- atomic_t close_lock_count

  Lock counter of context, held while structure is referenced by an operation handler.

- struct rtdm_operations * ops

  Set of active device operation handlers.

- struct rtdm_device * device

  Reference to owning device.

- struct rtdm_devctx_reserved reserved

  Data stored by RTDM inside a device context (internal use only).

- char dev_private [0]

  Begin of driver defined context data structure.

6.6.1 Detailed Description

Device context. A device context structure is associated with every open device instance. RTDM takes care of its creation and destruction and passes it to the operation handlers when being invoked.

Drivers can attach arbitrary data immediately after the official structure. The size of this data is provided via rtdm_device.context_size during device registration.

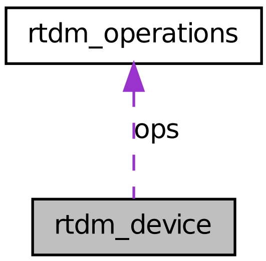
The documentation for this struct was generated from the following file:

- include/rtdm/rtdm_driver.h

6.7 rtdm_device Struct Reference

RTDM device.

Collaboration diagram for rtdm_device:



Data Fields

- int struct_version

  Revision number of this structure, see Driver Versioning defines.

- int device_flags

  Device flags, see Device Flags for details.

- size_t context_size

  Size of driver defined appendix to struct rtdm_dev_context.

- char device_name [RTDM_MAX_DEVNAME_LEN+1]

  Named device identification (orthogonal to Linux device name space).

- int protocol_family

  Protocol device identification: protocol family (PF_xxx).

- int socket_type

  Protocol device identification: socket type (SOCK_xxx).

- rtdm_open_handler_t open_rt

  Named device instance creation for real-time contexts, optional (but deprecated) if open_nrt is non-NULL, ignored for protocol devices.

- rtdm_open_handler_t open_nrt

  Named device instance creation for non-real-time contexts, optional if open_rt is non-NULL, ignored for protocol devices.

- rtdm_socket_handler_t socket_rt

  Protocol socket creation for real-time contexts, optional (but deprecated) if socket_nrt is non-NULL, ignored for named devices.

- rtdm_socket_handler_t socket_nrt

  Protocol socket creation for non-real-time contexts, optional if socket_rt is non-NULL, ignored for named devices.

- struct rtdm_operations ops

  Default operations on newly opened device instance.

- int device_class

  Device class ID, see RTDM_CLASS_xxx.

- int device_sub_class

  Device sub-class, see RTDM_SUBCLASS_xxx definition in the Device Profiles.

- int profile_version

  Supported device profile version.

- const char * driver_name

  Informational driver name (reported via /proc).

- int driver_version

  Driver version, see Driver Versioning defines.

- const char * peripheral_name

  Informational peripheral name the device is attached to (reported via /proc).

- const char * provider_name

  Informational driver provider name (reported via /proc).

- const char * proc_name

Name of /proc entry for the device, must not be NULL.

- int device_id

  Driver definable device ID.

- void * device_data

  Driver definable device data.

- struct rtdm_dev_reserved reserved

  Data stored by RTDM inside a registered device (internal use only).

6.7.1 Detailed Description

RTDM device. This structure specifies a RTDM device. As some fields, especially the reserved area, will be modified by RTDM during runtime, the structure must not reside in write-protected memory.

6.7.2 Field Documentation

6.7.2.1 rtdm_open_handler_t rtdm_device::open_rt

Named device instance creation for real-time contexts, optional (but deprecated) if open_nrt is non-NULL, ignored for protocol devices.

Deprecated

Only use non-real-time open handler in new drivers.

Referenced by rtdm_dev_register().

6.7.2.2 rtdm_socket_handler_t rtdm_device::socket_rt

Protocol socket creation for real-time contexts, optional (but deprecated) if socket_nrt is non-NULL, ignored for named devices.

Deprecated

Only use non-real-time socket creation handler in new drivers.

Referenced by rtdm_dev_register().

The documentation for this struct was generated from the following file:

- include/rtdm/rtdm_driver.h

6.8 rtdm_device_info Struct Reference

Device information.

Data Fields

- int device_flags

  Device flags, see Device Flags for details.

- int device_class

  Device class ID, see RTDM_CLASS_xxx.

- int device_sub_class

  Device sub-class, either RTDM_SUBCLASS_GENERIC or a RTDM_SUBCLASS_xxx definition of the related Device Profile.

- int profile_version

  Supported device profile version.

6.8.1 Detailed Description

Device information.

The documentation for this struct was generated from the following file:

- include/rtdm/rtdm.h

6.9 rtdm_operations Struct Reference

Device operations.

Data Fields

Common Operations

- rtdm_close_handler_t close_rt

  Close handler for real-time contexts (optional, deprecated).

- rtdm_close_handler_t close_nrt

  Close handler for non-real-time contexts (required).

- rtdm_ioctl_handler_t ioctl_rt

  IOCTL from real-time context (optional).

- rtdm_ioctl_handler_t ioctl_nrt

  IOCTL from non-real-time context (optional).

- rtdm_select_bind_handler_t select_bind

  Select binding handler for any context (optional).

Stream-Oriented Device Operations

- rtdm_read_handler_t read_rt
  Read handler for real-time context (optional).

- rtdm_read_handler_t read_nrt
  Read handler for non-real-time context (optional).

- rtdm_write_handler_t write_rt
  Write handler for real-time context (optional).

- rtdm_write_handler_t write_nrt
  Write handler for non-real-time context (optional).

Message-Oriented Device Operations

- rtdm_recvmsg_handler_t recvmsg_rt
  Receive message handler for real-time context (optional).

- rtdm_recvmsg_handler_t recvmsg_nrt
  Receive message handler for non-real-time context (optional).

- rtdm_sendmsg_handler_t sendmsg_rt
  Transmit message handler for real-time context (optional).

- rtdm_sendmsg_handler_t sendmsg_nrt
  Transmit message handler for non-real-time context (optional).

6.9.1 Detailed Description

Device operations.

6.9.2 Field Documentation

6.9.2.1 rtdm_close_handler_t rtdm_operations::close_rt

Close handler for real-time contexts (optional, deprecated).

Deprecated

Only use non-real-time close handler in new drivers.

Referenced by rtdm_dev_register().

The documentation for this struct was generated from the following file:

- include/rtdm/rtdm_driver.h

6.10 rtipc_port_label Struct Reference

Port label information structure.

Data Fields

- char label [XNOBJECT_NAME_LEN]

  Port label string, null-terminated.

6.10.1 Detailed Description

Port label information structure.

6.10.2 Field Documentation

6.10.2.1 char rtipc_port_label::label[XNOBJECT_NAME_LEN]

Port label string, null-terminated.

The documentation for this struct was generated from the following file:

- include/rtdm/rtipc.h

6.11 rtser_config Struct Reference

Serial device configuration.

Data Fields

- int config_mask

  mask specifying valid fields, see RTSER_SET_xxx

- int baud_rate

  baud rate, default RTSER_DEF_BAUD

- int parity

  number of parity bits, see RTSER_xxx_PARITY

- int data_bits

  number of data bits, see RTSER_xxx_BITS

- int stop_bits

  number of stop bits, see RTSER_xxx_STOPB

- int handshake

  handshake mechanisms, see RTSER_xxx_HAND

- int fifo_depth

  reception FIFO interrupt threshold, see RTSER_FIFO_xxx

- nanosecs_rel_t rx_timeout

reception timeout, see RTSER_TIMEOUT_xxx for special values

- nanosecs_rel_t tx_timeout

  transmission timeout, see RTSER_TIMEOUT_xxx for special values

- nanosecs_rel_t event_timeout

  event timeout, see RTSER_TIMEOUT_xxx for special values

- int timestamp_history

  enable timestamp history, see RTSER_xxx_TIMESTAMP_HISTORY

- int event_mask

  event mask to be used with RTSER_RTIOC_WAIT_EVENT, see RTSER_EVENT_xxx

6.11.1 Detailed Description

Serial device configuration.

The documentation for this struct was generated from the following file:

- include/rtdm/rtserial.h

6.12 rtser_event Struct Reference

Additional information about serial device events.

Data Fields

- int events

  signalled events, see RTSER_EVENT_xxx

- int rx_pending

  number of pending input characters

- nanosecs_abs_t last_timestamp

  last interrupt timestamp

- nanosecs_abs_t rxpend_timestamp

  reception timestamp of oldest character in input queue

6.12.1 Detailed Description

Additional information about serial device events.

The documentation for this struct was generated from the following file:

- include/rtdm/rtserial.h

6.13 rtser_status Struct Reference

Serial device status.

Data Fields

- int line_status
  line status register, see RTSER_LSR_xxx
- int modem_status
  modem status register, see RTSER_MSR_xxx

6.13.1 Detailed Description

Serial device status.

The documentation for this struct was generated from the following file:

- include/rtdm/rtserial.h

6.14 sockaddr_can Struct Reference

Socket address structure for the CAN address family.

Data Fields

- sa_family_t can_family
  CAN address family, must be `AF_CAN`.
- int can_ifindex
  Interface index of CAN controller.

6.14.1 Detailed Description

Socket address structure for the CAN address family.

6.14.2 Field Documentation

6.14.2.1 int sockaddr_can::can_ifindex

Interface index of CAN controller.

See SIOCGIFINDEX.

The documentation for this struct was generated from the following file:

- include/rtdm/rtcan.h

6.15 sockaddr_ipc Struct Reference

Socket address structure for the RTIPC address family.

Data Fields

- sa_family_t sipc_family

  RTIPC address family, must be `AF_RTIPC`.

- rtipc_port_t sipc_port

  Port number.

6.15.1 Detailed Description

Socket address structure for the RTIPC address family.

6.15.2 Field Documentation

6.15.2.1 rtipc_port_t sockaddr_ipc::sipc_port

Port number.

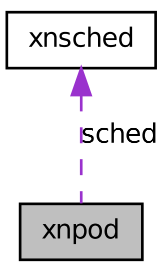
The documentation for this struct was generated from the following file:

- include/rtdm/rtipc.h

6.16 xnpod Struct Reference

Real-time pod descriptor.

Collaboration diagram for xnpod:



Data Fields

- xnflags_t status
- xnsched_t sched [XNARCH_NR_CPUS]
- xnqueue_t threadq
- xnqueue_t tstartq
- xnqueue_t tswitchq
- xnqueue_t tdeleteq
- atomic_counter_t timerlck
- int refcnt

6.16.1 Detailed Description

Real-time pod descriptor. The source of all Xenomai magic.

6.16.2 Field Documentation

6.16.2.1 int xnpod::refcnt

Reference count.

Referenced by xnpod_init().

6.16.2.2 xnsched_t xnpod::sched[XNARCH_NR_CPUS]

Per-cpu scheduler slots.

Referenced by xnpod_init().

6.16.2.3 xnflags_t xnpod::status

Status bitmask.

Referenced by xnpod_init().

6.16.2.4 xnqueue_t xnpod::tdeleteq

Thread delete hook queue.

Referenced by xnpod_init().

6.16.2.5 xnqueue_t xnpod::threadq

All existing threads.

Referenced by xnpod_init().

6.16.2.6 atomic_counter_t xnpod::timerlck

Timer lock depth.

Referenced by xnpod_init().

6.16.2.7 xnqueue_t xnpod::tstartq

Thread start hook queue.

Referenced by xnpod_init().

6.16.2.8 xnqueue_t xnpod::tswitchq

Thread switch hook queue.

Referenced by xnpod_init().

The documentation for this struct was generated from the following file:

- include/cobalt/nucleus/pod.h

6.17 xnsched Struct Reference

Scheduling information structure.

Data Fields

- xnflags_t status
- xnflags_t lflags
- struct xnthread * curr
- struct xnsched_rt rt
- volatile unsigned inesting
- struct xntimer htimer
- struct xnthread rootcb

6.17.1 Detailed Description

Scheduling information structure.

6.17.2 Field Documentation

6.17.2.1 struct xnthread* xnsched::curr

Current thread.

Referenced by xnpod_delete_thread(), and xnpod_suspend_thread().

6.17.2.2 struct xntimer xnsched::htimer

Host timer.

Referenced by xnpod_enable_timesource(), and xntimer_tick().

6.17.2.3 volatile unsigned xnsched::inesting

Interrupt nesting level.

6.17.2.4 xnflags_t xnsched::lflags

Scheduler specific local flags bitmask.

Referenced by xnpod_schedule(), and xntimer_tick().

6.17.2.5 struct xnthread xnsched::rootcb

Root thread control block.

Referenced by xnpod_init().

6.17.2.6 struct xnsched_rt xnsched::rt

Context of built-in real-time class.

6.17.2.7 xnflags_t xnsched::status

Scheduler specific status bitmask.

Referenced by xnpod_delete_thread(), xnpod_schedule(), xntimer_freeze(), and xntimer_tick().

The documentation for this struct was generated from the following file:

- include/cobalt/nucleus/sched.h

6.18 xnthread_info Struct Reference

Structure containing thread information.

Data Fields

- unsigned long state

  Thread state,.

- int bprio

  Base priority.

- int cprio

Current priority.

- int cpu

  CPU the thread currently runs on.

- unsigned long affinity

  Thread's CPU affinity.

- unsigned long long relpoint

  Time of next release.

- unsigned long long exectime

  Execution time in primary mode in nanoseconds.

- unsigned long modeswitches

  Number of primary->secondary mode switches.

- unsigned long ctxswitches

  Number of context switches.

- unsigned long pagefaults

  Number of triggered page faults.

- unsigned long syscalls

  Number of Xenomai syscalls.

- char name [XNOBJECT_NAME_LEN]

  Symbolic name assigned at creation.

6.18.1 Detailed Description

Structure containing thread information.

6.18.2 Field Documentation

6.18.2.1 unsigned long xnthread_info::affinity

Thread's CPU affinity.

6.18.2.2 int xnthread_info::bprio

Base priority.

6.18.2.3 int xnthread_info::cprio

Current priority.

May change through Priority Inheritance.

6.18.2.4 int xnthread_info::cpu

CPU the thread currently runs on.

6.18.2.5 unsigned long xnthread_info::ctxswitches

Number of context switches.

6.18.2.6 unsigned long long xnthread_info::exectime

Execution time in primary mode in nanoseconds.

6.18.2.7 unsigned long xnthread_info::modeswitches

Number of primary->secondary mode switches.

6.18.2.8 char xnthread_info::name[XNOBJECT_NAME_LEN]

Symbolic name assigned at creation.

6.18.2.9 unsigned long xnthread_info::pagefaults

Number of triggered page faults.

6.18.2.10 unsigned long long xnthread_info::relpoint

Time of next release.

6.18.2.11 unsigned long xnthread_info::state

Thread state,.

See also

Thread state flags.

6.18.2.12 unsigned long xnthread_info::syscalls

Number of Xenomai syscalls.

The documentation for this struct was generated from the following file:

- include/cobalt/nucleus/thread.h

6.19 xnvfile_lock_ops Struct Reference

Vfile locking operations.

Data Fields

- int(* get)(struct xnvfile *vfile)
- void(* put)(struct xnvfile *vfile)

6.19.1 Detailed Description

Vfile locking operations.

This structure describes the operations to be provided for implementing locking support on vfiles. They apply to both snapshot-driven and regular vfiles.

6.19.2 Field Documentation

6.19.2.1 int(* xnvfile_lock_ops::get)(struct xnvfile *vfile)

This handler should grab the desired lock.

Parameters

vfile A pointer to the virtual file which needs locking.

Returns

zero should be returned if the call succeeds. Otherwise, a negative error code can be returned; upon error, the current vfile operation is aborted, and the user-space caller is passed back the error value.

6.19.2.2 void(* xnvfile_lock_ops::put)(struct xnvfile *vfile)

This handler should release the lock previously grabbed by the get() handler.

Parameters

vfile A pointer to the virtual file which currently holds the lock to release.

The documentation for this struct was generated from the following file:

- include/cobalt/nucleus/vfile.h

6.20 xnvfile_regular_iterator Struct Reference

Regular vfile iterator.

Data Fields

- loff_t pos

  Current record position while iterating.

- struct seq_file * seq

  Backlink to the host sequential file supporting the vfile.

- struct xnvfile_regular * vfile

  Backlink to the vfile being read.

- char private [0]

  Start of private area.

6.20.1 Detailed Description

Regular vfile iterator.

This structure defines an iterator over a regular vfile.

6.20.2 Field Documentation

6.20.2.1 loff_t xnvfile_regular_iterator::pos

Current record position while iterating.

6.20.2.2 char xnvfile_regular_iterator::private[0]

Start of private area.

Use xnvfile_iterator_priv() to address it.

6.20.2.3 struct seq_file* xnvfile_regular_iterator::seq

Backlink to the host sequential file supporting the vfile.

6.20.2.4 struct xnvfile_regular* xnvfile_regular_iterator::vfile

Backlink to the vfile being read.

The documentation for this struct was generated from the following file:

- include/cobalt/nucleus/vfile.h

6.21 xnvfile_regular_ops Struct Reference

Regular vfile operation descriptor.

Data Fields

- int(* rewind)(struct xnvfile_regular_iterator *it)
- void *(* begin)(struct xnvfile_regular_iterator *it)

- void *(* next)(struct xnvfile_regular_iterator *it)
- void(* end)(struct xnvfile_regular_iterator *it)
- int(* show)(struct xnvfile_regular_iterator *it, void *data)
- ssize_t(* store)(struct xnvfile_input *input)

6.21.1 Detailed Description

Regular vfile operation descriptor.

This structure describes the operations available with a regular vfile. It defines handlers for sending back formatted kernel data upon a user-space read request, and for obtaining user data upon a user-space write request.

6.21.2 Field Documentation

6.21.2.1 void*(* xnvfile_regular_ops::begin)(struct xnvfile_regular_iterator *it)

This handler should prepare for iterating over the records upon a read request, starting from the specified position.

Parameters

it A pointer to the current vfile iterator. On entry, it->pos is set to the (0-based) position of the first record to output. This handler may be called multiple times with different position requests.

Returns

A pointer to the first record to format and output, to be passed to the show() handler as its *data* parameter, if the call succeeds. Otherwise:

- NULL in case no record is available, in which case the read operation will terminate immediately with no output.
- VFILE_SEQ_START, a special value indicating that the show() handler should receive a NULL data pointer first, in order to output a header.
- ERR_PTR(errno), where errno is a negative error code; upon error, the current operation will be aborted immediately.

Note

This handler is optional; if none is given in the operation descriptor (i.e. NULL value), the show() handler() will be called only once for a read operation, with a NULL *data* parameter. This particular setting is convenient for simple regular vfiles having a single, fixed record to output.

6.21.2.2 void(* xnvfile_regular_ops::end)(struct xnvfile_regular_iterator *it)

This handler is called after all records have been output.

Parameters

it A pointer to the current vfile iterator.

Note

This handler is optional and the pointer may be NULL.

6.21.2.3 void*(* xnvfile_regular_ops::next)(struct xnvfile_regular_iterator *it)

This handler should return the address of the next record to format and output by the show() handler".

Parameters

it A pointer to the current vfile iterator. On entry, it->pos is set to the (0-based) position of the next record to output.

Returns

A pointer to the next record to format and output, to be passed to the show() handler as its *data* parameter, if the call succeeds. Otherwise:

- NULL in case no record is available, in which case the read operation will terminate immediately with no output.
- ERR_PTR(errno), where errno is a negative error code; upon error, the current operation will be aborted immediately.

Note

This handler is optional; if none is given in the operation descriptor (i.e. NULL value), the read operation will stop after the first invocation of the show() handler.

6.21.2.4 int(* xnvfile_regular_ops::rewind)(struct xnvfile_regular_iterator *it)

This handler is called only once, when the virtual file is opened, before the begin() handler is invoked.

Parameters

it A pointer to the vfile iterator which will be used to read the file contents.

Returns

Zero should be returned upon success. Otherwise, a negative error code aborts the operation, and is passed back to the reader.

Note

This handler is optional. It should not be used to allocate resources but rather to perform consistency checks, since no closure call is issued in case the open sequence eventually fails.

6.21.2.5 int(* xnvfile_regular_ops::show)(struct xnvfile_regular_iterator *it, void *data)

This handler should format and output a record.

xnvfile_printf(), xnvfile_write(), xnvfile_puts() and xnvfile_putc() are available to format and/or emit the output. All routines take the iterator argument *it* as their first parameter.

Parameters

it A pointer to the current vfile iterator.

data A pointer to the record to format then output. The first call to the handler may receive a NULL *data* pointer, depending on the presence and/or return of a hander; the show handler should test this special value to output any header that fits, prior to receiving more calls with actual records.

Returns

zero if the call succeeds, also indicating that the handler should be called for the next record if any. Otherwise:

- A negative error code. This will abort the output phase, and return this status to the reader.

- VFILE_SEQ_SKIP, a special value indicating that the current record should be skipped and will not be output.

6.21.2.6 ssize_t(* xnvfile_regular_ops::store)(struct xnvfile_input *input)

This handler receives data written to the vfile, likely for updating some kernel setting, or triggering any other action which fits. This is the only handler which deals with the write-side of a vfile. It is called when writing to the /proc entry of the vfile from a user-space process.

The input data is described by a descriptor passed to the handler, which may be subsequently passed to parsing helper routines. For instance, xnvfile_get_string() will accept the input descriptor for returning the written data as a null-terminated character string. On the other hand, xnvfile_get_integer() will attempt to return a long integer from the input data.

Parameters

input A pointer to an input descriptor. It refers to an opaque data from the handler's standpoint.

Returns

the number of bytes read from the input descriptor if the call succeeds. Otherwise, a negative error code. Return values from parsing helper routines are commonly passed back to the caller by the store() handler.

Note

This handler is optional, and may be omitted for read-only vfiles.

The documentation for this struct was generated from the following file:

- include/cobalt/nucleus/vfile.h

6.22 xnvfile_rev_tag Struct Reference

Snapshot revision tag.

Data Fields

- int rev

  Current revision number.

6.22.1 Detailed Description

Snapshot revision tag.

This structure defines a revision tag to be used with snapshot-driven vfiles.

6.22.2 Field Documentation

6.22.2.1 int xnvfile_rev_tag::rev

Current revision number.

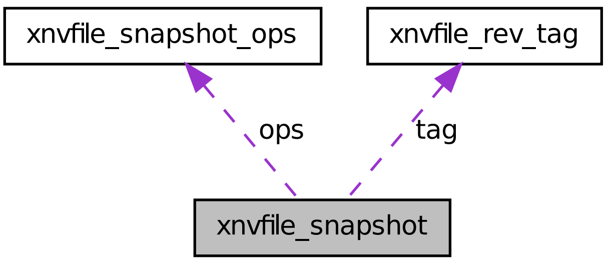
The documentation for this struct was generated from the following file:

- include/cobalt/nucleus/vfile.h

6.23 xnvfile_snapshot Struct Reference

Snapshot vfile descriptor.

Collaboration diagram for xnvfile_snapshot:



6.23.1 Detailed Description

Snapshot vfile descriptor.

This structure describes a snapshot-driven vfile. Reading from such a vfile involves a preliminary data collection phase under lock protection, and a subsequent formatting and output phase of the collected data records. Locking is done in a way that does not increase worst-case latency, regardless of the number of records to be collected for output.

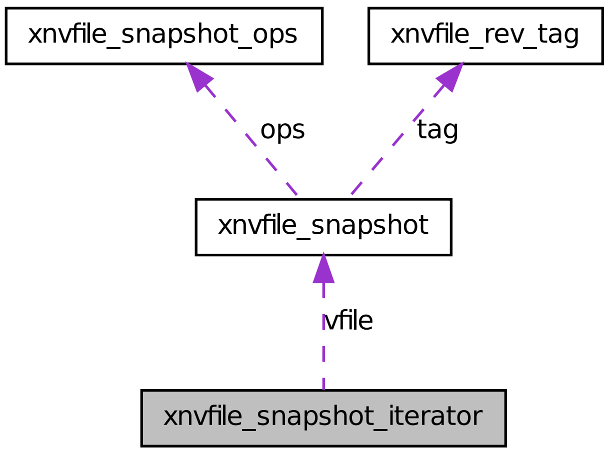
The documentation for this struct was generated from the following file:

- include/cobalt/nucleus/vfile.h

6.24 xnvfile_snapshot_iterator Struct Reference

Snapshot-driven vfile iterator.

Collaboration diagram for xnvfile_snapshot_iterator:



Data Fields

- int nrdata

  Number of collected records.

- caddr_t databuf

  Address of record buffer.

- struct seq_file * seq

  Backlink to the host sequential file supporting the vfile.

- struct xnvfile_snapshot * vfile

    Backlink to the vfile being read.

- void(* endfn)(struct xnvfile_snapshot_iterator *it, void *buf)

    Buffer release handler.

- char private [0]

    Start of private area.

6.24.1 Detailed Description

Snapshot-driven vfile iterator.

This structure defines an iterator over a snapshot-driven vfile.

6.24.2 Field Documentation

6.24.2.1 caddr_t xnvfile_snapshot_iterator::databuf

Address of record buffer.

6.24.2.2 void(* xnvfile_snapshot_iterator::endfn)(struct xnvfile_snapshot_iterator *it, void *buf)

Buffer release handler.

6.24.2.3 int xnvfile_snapshot_iterator::nrdata

Number of collected records.

6.24.2.4 char xnvfile_snapshot_iterator::private[0]

Start of private area.

Use xnvfile_iterator_priv() to address it.

6.24.2.5 struct seq_file* xnvfile_snapshot_iterator::seq

Backlink to the host sequential file supporting the vfile.

6.24.2.6 struct xnvfile_snapshot* xnvfile_snapshot_iterator::vfile

Backlink to the vfile being read.

The documentation for this struct was generated from the following file:

- include/cobalt/nucleus/vfile.h

6.25 xnvfile_snapshot_ops Struct Reference

Snapshot vfile operation descriptor.

Data Fields

- int(* rewind)(struct xnvfile_snapshot_iterator *it)
- void *(* begin)(struct xnvfile_snapshot_iterator *it)
- void(* end)(struct xnvfile_snapshot_iterator *it, void *buf)
- int(* next)(struct xnvfile_snapshot_iterator *it, void *data)
- int(* show)(struct xnvfile_snapshot_iterator *it, void *data)
- ssize_t(* store)(struct xnvfile_input *input)

6.25.1 Detailed Description

Snapshot vfile operation descriptor.

This structure describes the operations available with a snapshot-driven vfile. It defines handlers for returning a printable snapshot of some Xenomai object contents upon a user-space read request, and for updating this object upon a user-space write request.

6.25.2 Field Documentation

6.25.2.1 void*(* xnvfile_snapshot_ops::begin)(struct xnvfile_snapshot_iterator *it)

This handler should allocate the snapshot buffer to hold records during the data collection phase. When specified, all records collected via the next() handler" will be written to a cell from the memory area returned by begin().

Parameters

it A pointer to the current snapshot iterator.

Returns

A pointer to the record buffer, if the call succeeds. Otherwise:

- NULL in case of allocation error. This will abort the data collection, and return -ENOMEM to the reader.

- VFILE_SEQ_EMPTY, a special value indicating that no record will be output. In such a case, the next() handler will not be called, and the data collection will stop immediately. However, the show() handler will still be called once, with a NULL data pointer (i.e. header display request).

Note

This handler is optional; if none is given, an internal allocation depending on the value returned by the rewind() handler can be obtained.

6.25.2.2 void(* xnvfile_snapshot_ops::end)(struct xnvfile_snapshot_iterator *it, void *buf)

This handler releases the memory buffer previously obtained from begin(). It is usually called after the snapshot data has been output by show(), but it may also be called before rewinding the vfile after a revision change, to release the dropped buffer.

Parameters

it A pointer to the current snapshot iterator.

buf A pointer to the buffer to release.

Note

This routine is optional and the pointer may be NULL. It is not needed upon internal buffer allocation; see the description of the rewind() handler".

6.25.2.3 int(* xnvfile_snapshot_ops::next)(struct xnvfile_snapshot_iterator *it, void *data)

This handler fetches the next record, as part of the snapshot data to be sent back to the reader via the show().

Parameters

it A pointer to the current snapshot iterator.

data A pointer to the record to fill in.

Returns

a strictly positive value, if the call succeeds and leaves a valid record into *data*, which should be passed to the show() handler() during the formatting and output phase. Otherwise:

- A negative error code. This will abort the data collection, and return this status to the reader.
- VFILE_SEQ_SKIP, a special value indicating that the current record should be skipped. In such a case, the *data* pointer is not advanced to the next position before the next() handler is called anew.

Note

This handler is called with the vfile lock held. Before each invocation of this handler, the vfile core checks whether the revision tag has been touched, in which case the data collection is restarted from scratch. A data collection phase succeeds whenever all records can be fetched via the next() handler, while the revision tag remains unchanged, which indicates that a consistent snapshot of the object state was taken.

6.25.2.4 int(* xnvfile_snapshot_ops::rewind)(struct xnvfile_snapshot_iterator *it)

This handler (re-)initializes the data collection, moving the seek pointer at the first record. When the file revision tag is touched while collecting data, the current reading is aborted, all collected data dropped, and the vfile is eventually rewound.

Parameters

it A pointer to the current snapshot iterator. Two useful information can be retrieved from this iterator in this context:

- it->vfile is a pointer to the descriptor of the virtual file being rewound.
- xnvfile_iterator_priv(it) returns a pointer to the private data area, available from the descriptor, which size is vfile->privsz. If the latter size is zero, the returned pointer is meaningless and should not be used.

Returns

A negative error code aborts the data collection, and is passed back to the reader. Otherwise:

- a strictly positive value is interpreted as the total number of records which will be returned by the next() handler during the data collection phase. If no begin() handler is provided in the operation descriptor, this value is used to allocate the snapshot buffer internally. The size of this buffer would then be vfile->datasz * value.
- zero leaves the allocation to the begin() handler if present, or indicates that no record is to be output in case such handler is not given.

Note

This handler is optional; a NULL value indicates that nothing needs to be done for rewinding the vfile. It is called with the vfile lock held.

6.25.2.5 int(* xnvfile_snapshot_ops::show)(struct xnvfile_snapshot_iterator *it, void *data)

This handler should format and output a record from the collected data.

xnvfile_printf(), xnvfile_write(), xnvfile_puts() and xnvfile_putc() are available to format and/or emit the output. All routines take the iterator argument *it* as their first parameter.

Parameters

it A pointer to the current snapshot iterator.

data A pointer to the record to format then output. The first call to the handler is always passed a NULL *data* pointer; the show handler should test this special value to output any header that fits, prior to receiving more calls with actual records.

Returns

zero if the call succeeds, also indicating that the handler should be called for the next record if any. Otherwise:

- A negative error code. This will abort the output phase, and return this status to the reader.
- VFILE_SEQ_SKIP, a special value indicating that the current record should be skipped and will not be output.

6.25.2.6 ssize_t(* xnvfile_snapshot_ops::store)(struct xnvfile_input *input)

This handler receives data written to the vfile, likely for updating the associated Xenomai object's state, or triggering any other action which fits. This is the only handler which deals with the write-side of a vfile. It is called when writing to the /proc entry of the vfile from a user-space process.

The input data is described by a descriptor passed to the handler, which may be subsequently passed to parsing helper routines. For instance, xnvfile_get_string() will accept the input descriptor for returning the written data as a null-terminated character string. On the other hand, xnvfile_get_integer() will attempt to return a long integer from the input data.

Parameters

input A pointer to an input descriptor. It refers to an opaque data from the handler's standpoint.

Returns

the number of bytes read from the input descriptor if the call succeeds. Otherwise, a negative error code. Return values from parsing helper routines are commonly passed back to the caller by the store() handler.

Note

This handler is optional, and may be omitted for read-only vfiles.

Referenced by xnvfile_init_snapshot().

The documentation for this struct was generated from the following file:

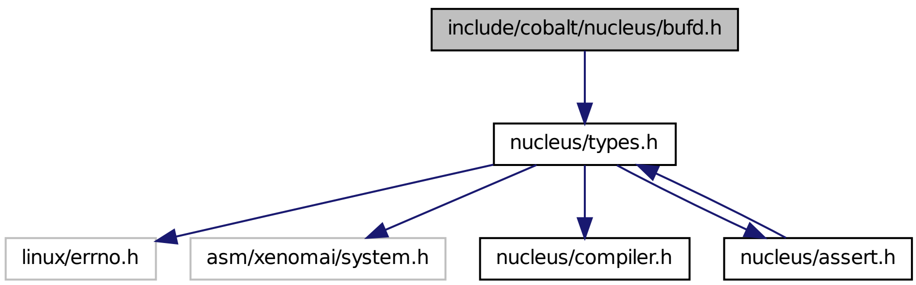
- include/cobalt/nucleus/vfile.h

Chapter 7

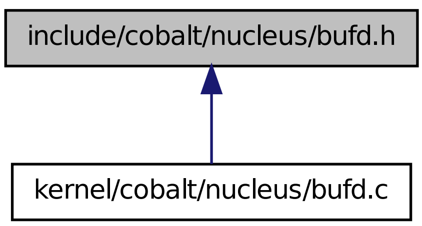
File Documentation

7.1 include/cobalt/nucleus/bufd.h File Reference

Include dependency graph for bufd.h:



This graph shows which files directly or indirectly include this file:



Functions

- static void xnbufd_map_uread (struct xnbufd *bufd, const void __user *ptr, size_t len)
  Initialize a buffer descriptor for reading from user memory.
- static void xnbufd_map_uwrite (struct xnbufd *bufd, void __user *ptr, size_t len)
  Initialize a buffer descriptor for writing to user memory.
- ssize_t xnbufd_unmap_uread (struct xnbufd *bufd)
  Finalize a buffer descriptor obtained from xnbufd_map_uread().
- ssize_t xnbufd_unmap_uwrite (struct xnbufd *bufd)
  Finalize a buffer descriptor obtained from xnbufd_map_uwrite().
- static void xnbufd_map_kread (struct xnbufd *bufd, const void *ptr, size_t len)
  Initialize a buffer descriptor for reading from kernel memory.
- static void xnbufd_map_kwrite (struct xnbufd *bufd, void *ptr, size_t len)
  Initialize a buffer descriptor for writing to kernel memory.
- ssize_t xnbufd_unmap_kread (struct xnbufd *bufd)
  Finalize a buffer descriptor obtained from xnbufd_map_kread().
- ssize_t xnbufd_unmap_kwrite (struct xnbufd *bufd)
  Finalize a buffer descriptor obtained from xnbufd_map_kwrite().
- ssize_t xnbufd_copy_to_kmem (void *ptr, struct xnbufd *bufd, size_t len)
  Copy memory covered by a buffer descriptor to kernel memory.
- ssize_t xnbufd_copy_from_kmem (struct xnbufd *bufd, void *from, size_t len)
  Copy kernel memory to the area covered by a buffer descriptor.
- void xnbufd_invalidate (struct xnbufd *bufd)
  Invalidate a buffer descriptor.
- static void xnbufd_reset (struct xnbufd *bufd)
  Reset a buffer descriptor.

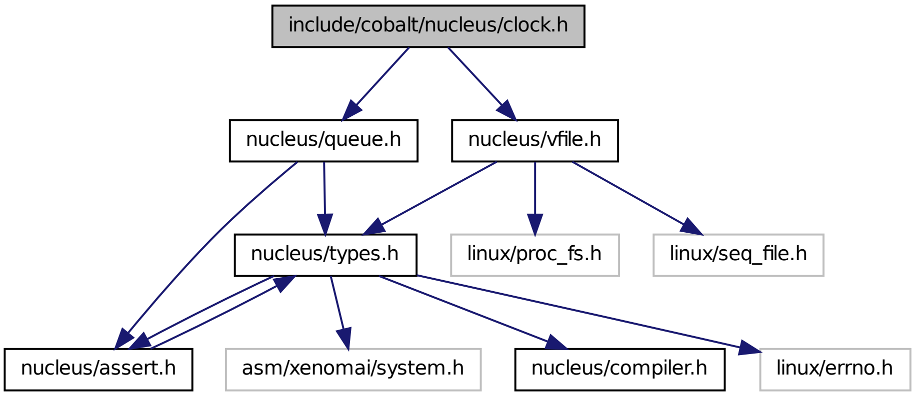
7.1.1 Detailed Description

Note

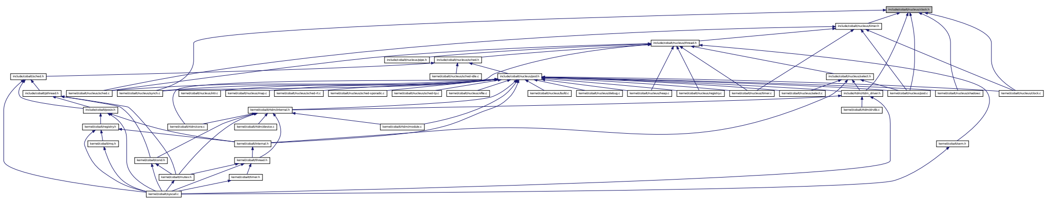
You should have received a copy of the GNU General Public License along with Xenomai; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

7.2 include/cobalt/nucleus/clock.h File Reference

Include dependency graph for clock.h:



This graph shows which files directly or indirectly include this file:



Functions

- void xnclock_adjust (xnsticks_t delta)

    Adjust the clock time for the system.

7.2.1 Detailed Description

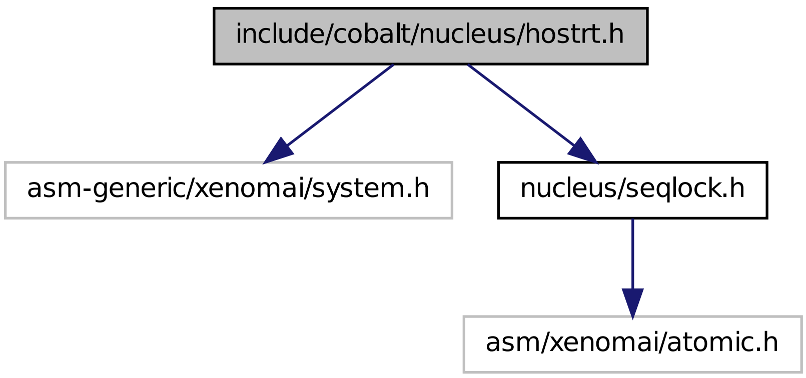
Note

Copyright (C) 2006,2007 Philippe Gerum <rpm@xenomai.org>.

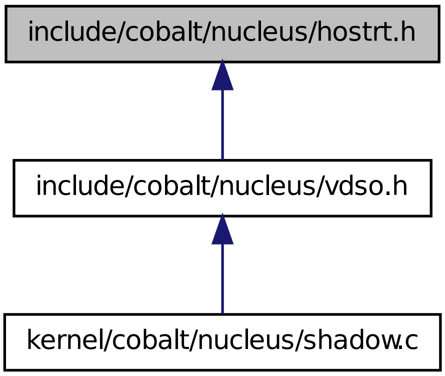
7.3 include/cobalt/nucleus/hostrt.h File Reference

Definitions for global semaphore heap shared objects.

Include dependency graph for hostrt.h:



This graph shows which files directly or indirectly include this file:



7.3.1 Detailed Description

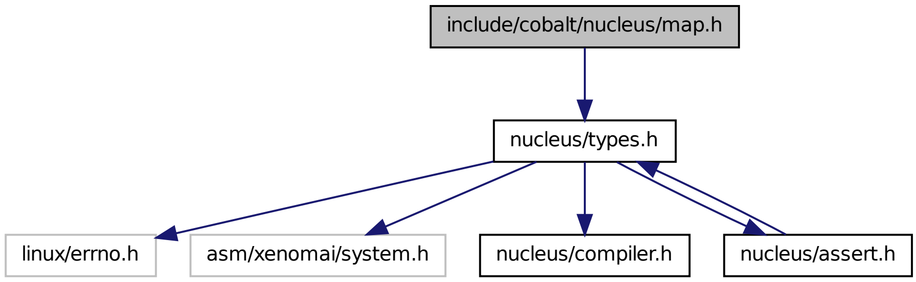
Definitions for global semaphore heap shared objects.

Author

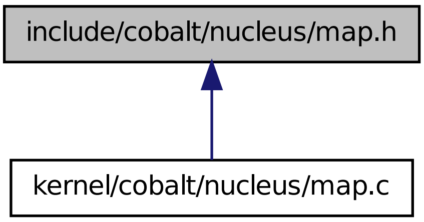
Wolfgang Mauerer

7.4 include/cobalt/nucleus/map.h File Reference

Include dependency graph for map.h:



This graph shows which files directly or indirectly include this file:



Functions

- xnmap_t * xnmap_create (int nkeys, int reserve, int offset)
  Create a map.
- void xnmap_delete (xnmap_t *map)
  Delete a map.
- int xnmap_enter (xnmap_t *map, int key, void *objaddr)
  Index an object into a map.
- int xnmap_remove (xnmap_t *map, int key)
  Remove an object reference from a map.

- static void * xnmap_fetch_nocheck (xnmap_t *map, int key)

  Search an object into a map - unchecked form.

- static void * xnmap_fetch (xnmap_t *map, int key)

  Search an object into a map.

7.4.1 Detailed Description

Note

Copyright (C) 2007 Philippe Gerum <rpm@xenomai.org>.

Xenomai is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

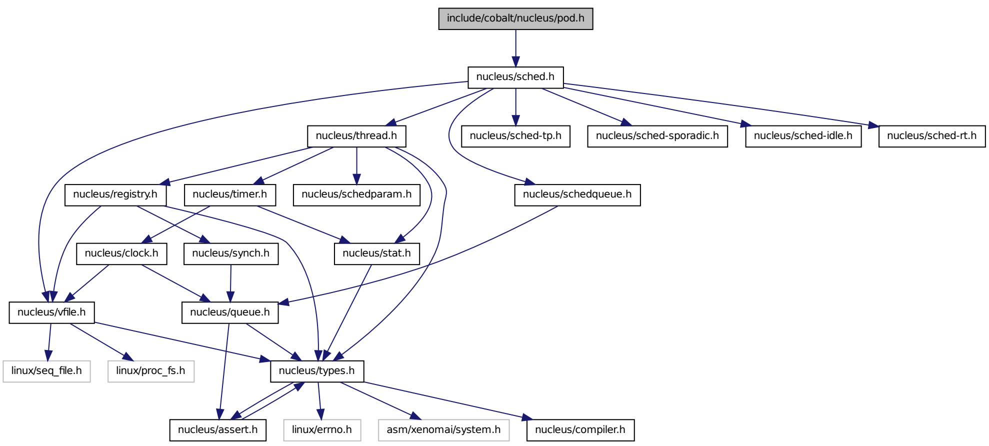
Xenomai is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with Xenomai; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

7.5 include/cobalt/nucleus/pod.h File Reference

Real-time pod interface header.

Include dependency graph for pod.h:



This graph shows which files directly or indirectly include this file:

Data Structures

- struct xnpod

  Real-time pod descriptor.

Functions

- void __xnpod_reset_thread (struct xnthread *thread)

  Reset the thread.

- int xnpod_init (void)

  Initialize the core pod.

- int xnpod_enable_timesource (void)

  Activate the core time source.

- void xnpod_disable_timesource (void)

  Stop the core time source.

- void xnpod_shutdown (int xtype)

  Shutdown the current pod.

- int xnpod_init_thread (struct xnthread *thread, const struct xnthread_init_attr *attr, struct xnsched_class *sched_class, const union xnsched_policy_param *sched_param)

  Initialize a new thread.

- int xnpod_start_thread (xnthread_t *thread, const struct xnthread_start_attr *attr)

  Initial start of a newly created thread.

- void xnpod_stop_thread (xnthread_t *thread)

  Stop a thread.

- void xnpod_delete_thread (xnthread_t *thread)

  Delete a thread.

- void xnpod_abort_thread (xnthread_t *thread)

  Abort a thread.

- xnflags_t xnpod_set_thread_mode (xnthread_t *thread, xnflags_t clrmask, xnflags_t setmask)

  Change a thread's control mode.

- void xnpod_suspend_thread (xnthread_t *thread, xnflags_t mask, xnticks_t timeout, xntmode_t timeout_mode, struct xnsynch *wchan)

  Suspend a thread.

- void xnpod_resume_thread (xnthread_t *thread, xnflags_t mask)

  Resume a thread.

- int xnpod_unblock_thread (xnthread_t *thread)

  Unblock a thread.

- int xnpod_set_thread_schedparam (struct xnthread *thread, struct xnsched_class *sched_-class, const union xnsched_policy_param *sched_param)

  Change the base scheduling parameters of a thread.

- int xnpod_migrate_thread (int cpu)

  Migrate the current thread.

- void xnpod_dispatch_signals (void)

  Deliver pending asynchronous signals to the running thread.

- static void xnpod_schedule (void)

  Rescheduling procedure entry point.

- int xnpod_handle_exception (struct ipipe_trap_data *d)

  Exception handler.

- int xnpod_set_thread_periodic (xnthread_t *thread, xnticks_t idate, xntmode_t timeout_-mode, xnticks_t period)

  Make a thread periodic.

- int xnpod_wait_thread_period (unsigned long *overruns_r)

  Wait for the next periodic release point.

- int xnpod_set_thread_tslice (struct xnthread *thread, xnticks_t quantum)

  Set thread time-slicing information.

- int xnpod_add_hook (int type, void(*routine)(xnthread_t *))

  Install a nucleus hook.

- int xnpod_remove_hook (int type, void(*routine)(xnthread_t *))

  Remove a nucleus hook.

7.5.1 Detailed Description

Real-time pod interface header.

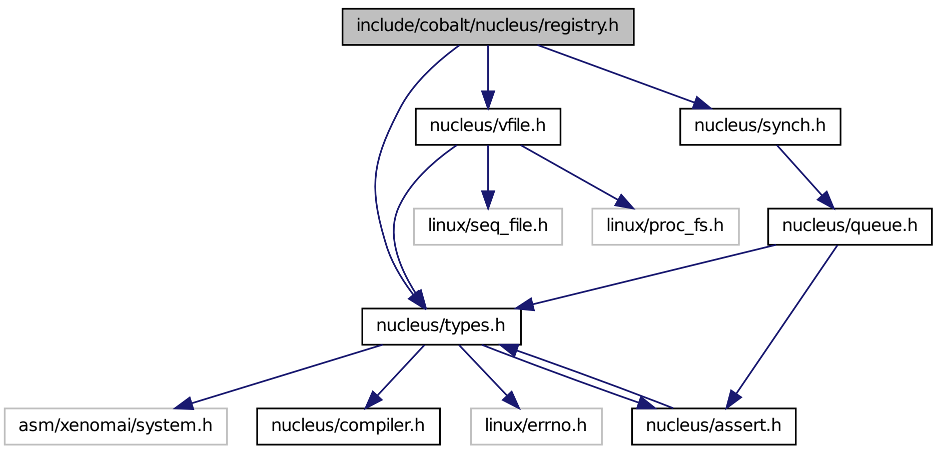
Author

Philippe Gerum

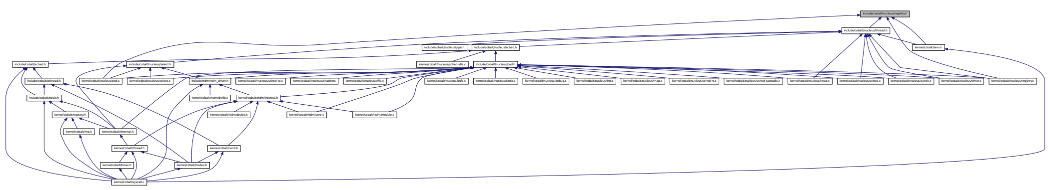
7.6 include/cobalt/nucleus/registry.h File Reference

This file is part of the Xenomai project.

Include dependency graph for registry.h:



This graph shows which files directly or indirectly include this file:



Functions

- int xnregistry_enter (const char *key, void *objaddr, xnhandle_t *phandle, struct xnpnode *pnode)

  Register a real-time object.

- int xnregistry_bind (const char *key, xnticks_t timeout, int timeout_mode, xnhandle_t *phandle)

  Bind to a real-time object.

- int xnregistry_remove (xnhandle_t handle)

  Forcibly unregister a real-time object.

- int xnregistry_remove_safe (xnhandle_t handle, xnticks_t timeout)

  Unregister an idle real-time object.

- void * xnregistry_get (xnhandle_t handle)

  Find and lock a real-time object into the registry.

- void * xnregistry_fetch (xnhandle_t handle)

  Find a real-time object into the registry.

- u_long xnregistry_put (xnhandle_t handle)

  Unlock a real-time object from the registry.

7.6.1 Detailed Description

This file is part of the Xenomai project.

Note

Copyright (C) 2004 Philippe Gerum <rpm@xenomai.org>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

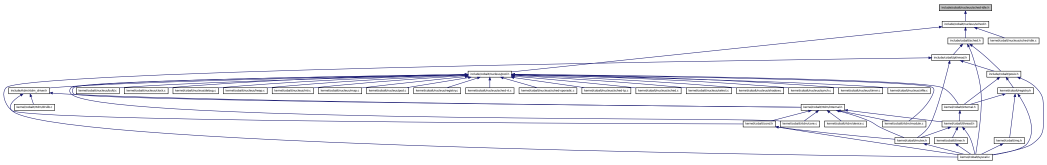
This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

7.7 include/cobalt/nucleus/sched-idle.h File Reference

Definitions for the IDLE scheduling class.

This graph shows which files directly or indirectly include this file:



7.7.1 Detailed Description

Definitions for the IDLE scheduling class.

Author

Philippe Gerum

Copyright (C) 2008 Philippe Gerum <rpm@xenomai.org>.

Xenomai is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

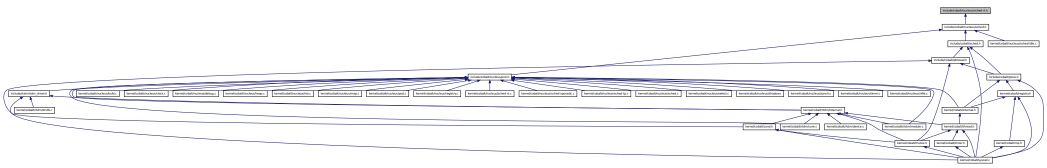
Xenomai is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with Xenomai; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

7.8 include/cobalt/nucleus/sched-rt.h File Reference

Definitions for the RT scheduling class.

This graph shows which files directly or indirectly include this file:



7.8.1 Detailed Description

Definitions for the RT scheduling class.

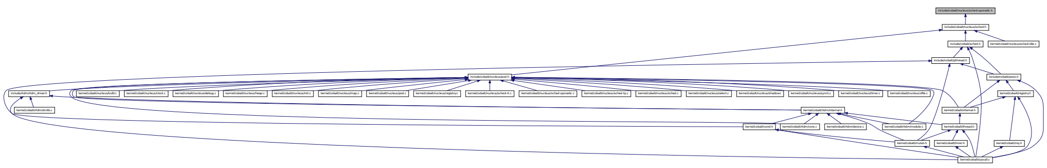
Author

Philippe Gerum

7.9 include/cobalt/nucleus/sched-sporadic.h File Reference

Definitions for the SSP scheduling class.

This graph shows which files directly or indirectly include this file:



7.9.1 Detailed Description

Definitions for the SSP scheduling class.

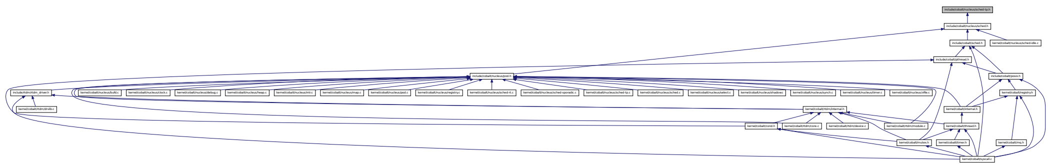
Author

Philippe Gerum

7.10 include/cobalt/nucleus/sched-tp.h File Reference

Definitions for the TP scheduling class.

This graph shows which files directly or indirectly include this file:



7.10.1 Detailed Description

Definitions for the TP scheduling class.

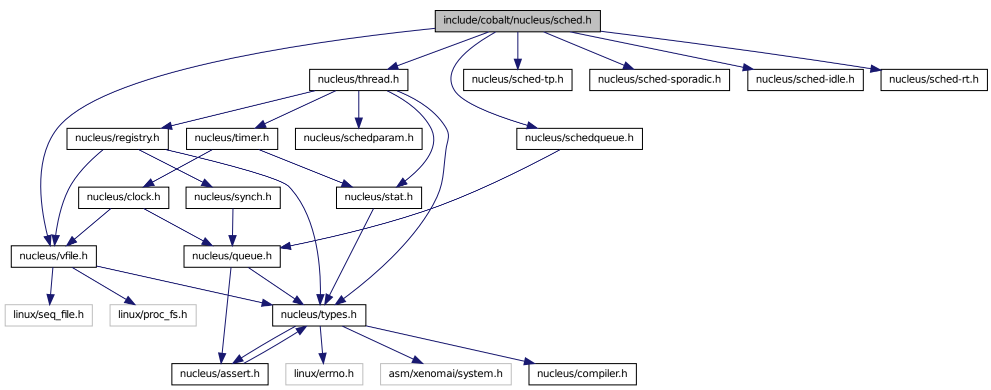
Author

Philippe Gerum

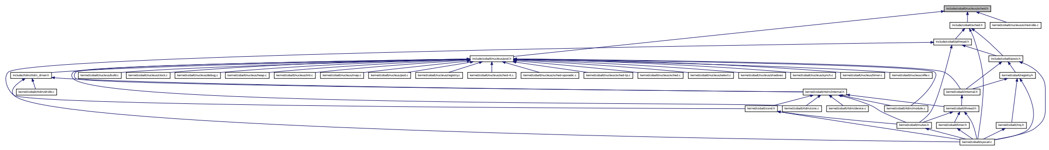
7.11 include/cobalt/nucleus/sched.h File Reference

Scheduler interface header.

Include dependency graph for sched.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct xnsched

  Scheduling information structure.

Typedefs

- typedef struct xnsched xnsched_t

  Scheduling information structure.

Functions

- static void xnsched_rotate (struct xnsched *sched, struct xnsched_class *sched_class, const union xnsched_policy_param *sched_param)

  Rotate a scheduler runqueue.

7.11.1 Detailed Description

Scheduler interface header.

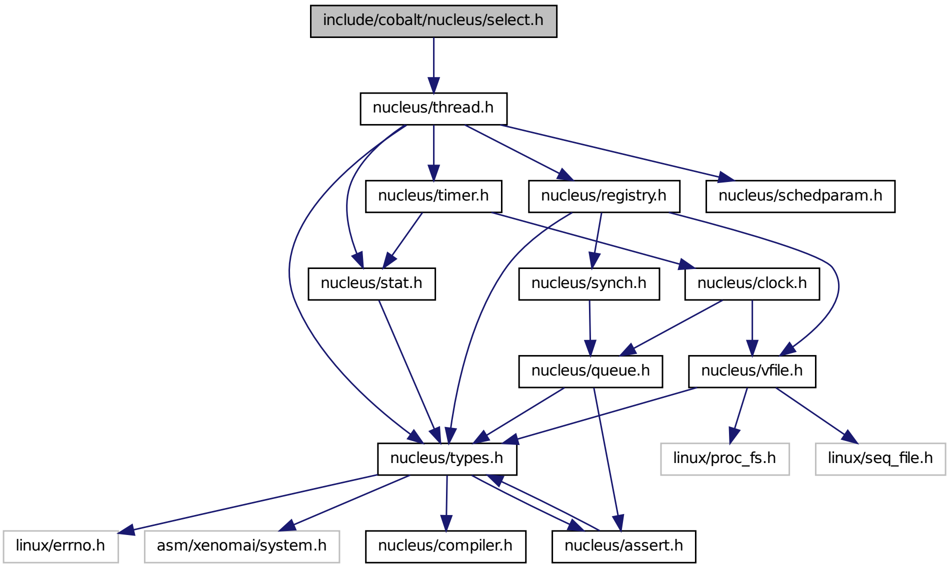
Author

Philippe Gerum

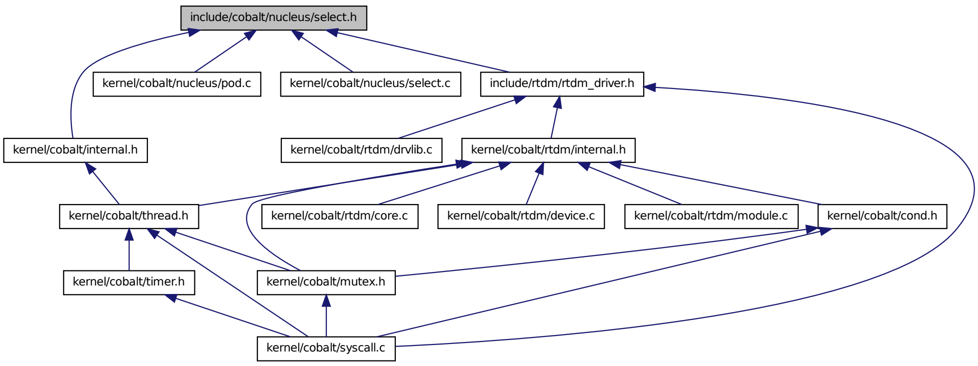
7.12 include/cobalt/nucleus/select.h File Reference

file descriptors events multiplexing header.

Include dependency graph for select.h:



This graph shows which files directly or indirectly include this file:



Functions

- void xnselect_init (struct xnselect *select_block)

  Initialize a struct xnselect structure.

- int xnselect_bind (struct xnselect *select_block, struct xnselect_binding *binding, struct xnselector *selector, unsigned type, unsigned index, unsigned state)

  Bind a file descriptor (represented by its xnselect structure) to a selector block.

- static int xnselect_signal (struct xnselect *select_block, unsigned state)

  Signal a file descriptor state change.

- void xnselect_destroy (struct xnselect *select_block)

  Destroy the xnselect structure associated with a file descriptor.

- int xnselector_init (struct xnselector *selector)

  Initialize a selector structure.

- int xnselect (struct xnselector *selector, fd_set *out_fds[XNSELECT_MAX_TYPES], fd_set *in_fds[XNSELECT_MAX_TYPES], int nfds, xnticks_t timeout, xntmode_t timeout_mode)

  Check the state of a number of file descriptors, wait for a state change if no descriptor is ready.

- void xnselector_destroy (struct xnselector *selector)

  Destroy a selector block.

7.12.1 Detailed Description

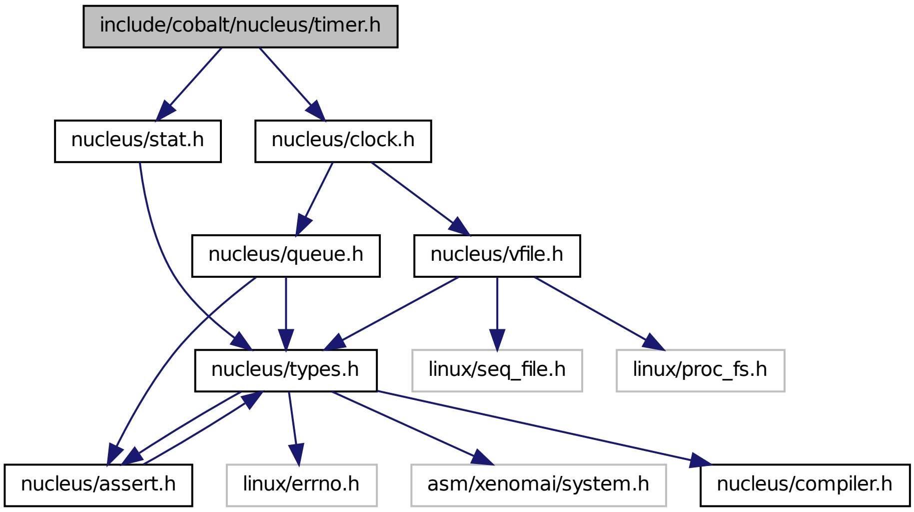
file descriptors events multiplexing header.

Author

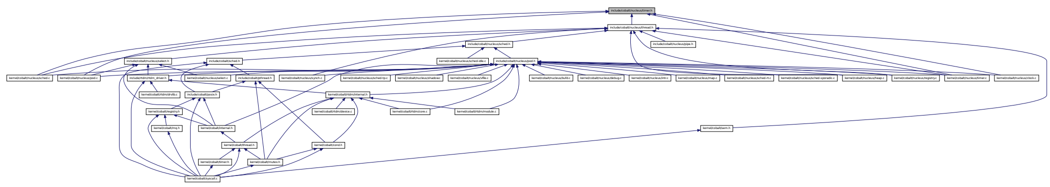
Gilles Chanteperdrix

7.13 include/cobalt/nucleus/timer.h File Reference

Include dependency graph for timer.h:



This graph shows which files directly or indirectly include this file:



Functions

- void xntimer_destroy (xntimer_t *timer)

  Release a timer object.

- int xntimer_start (xntimer_t *timer, xnticks_t value, xnticks_t interval, xntmode_t mode)

  Arm a timer.

- xnticks_t xntimer_get_date (xntimer_t *timer)

  Return the absolute expiration date.

- xnticks_t xntimer_get_timeout (xntimer_t *timer)

  Return the relative expiration date.

- xnticks_t xntimer_get_interval (xntimer_t *timer)

  Return the timer interval value.

- static void xntimer_stop (xntimer_t *timer)

  Disarm a timer.

- unsigned long xntimer_get_overruns (xntimer_t *timer, xnticks_t now)

  Get the count of overruns for the last tick.

- void xntimer_freeze (void)

  Freeze all timers (from every time bases).

- void xntimer_tick (void)

  Process a timer tick.

7.13.1 Detailed Description

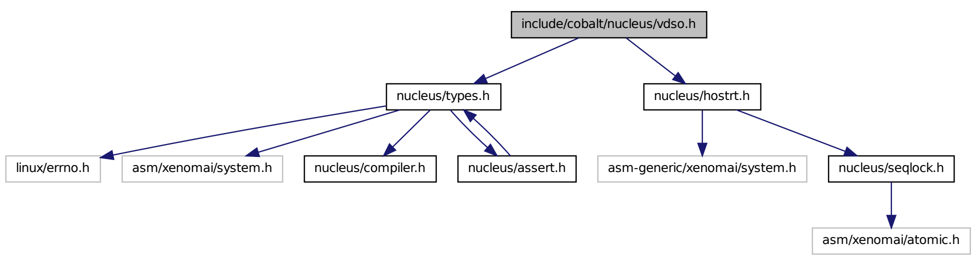
Note

Copyright (C) 2001,2002,2003 Philippe Gerum <rpm@xenomai.org>.

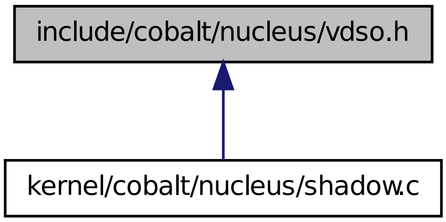
7.14 include/cobalt/nucleus/vdso.h File Reference

Definitions for global semaphore heap shared objects.

Include dependency graph for vdso.h:



This graph shows which files directly or indirectly include this file:



7.14.1 Detailed Description

Definitions for global semaphore heap shared objects.

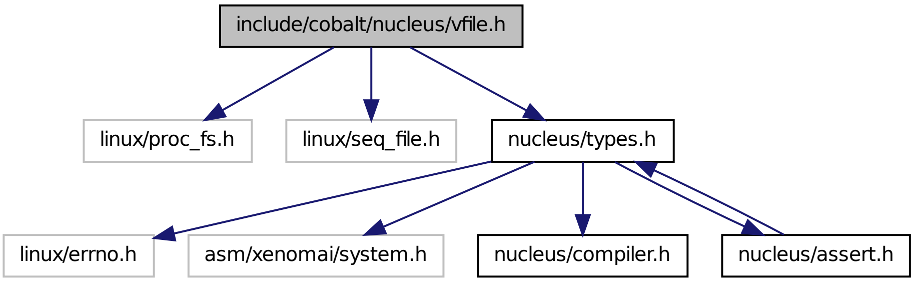
Author

Wolfgang Mauerer

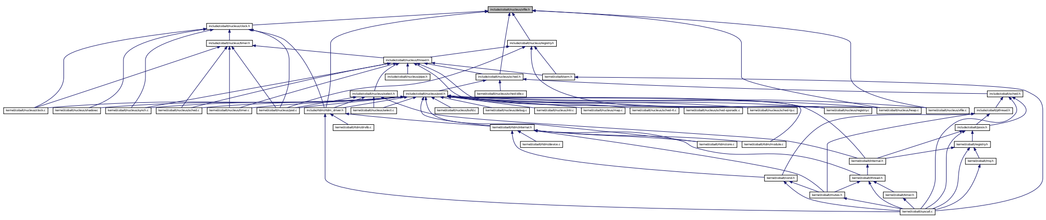
7.15 include/cobalt/nucleus/vfile.h File Reference

This file is part of the Xenomai project.

Include dependency graph for vfile.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct xnvfile_lock_ops

  Vfile locking operations.

- struct xnvfile_regular_ops

  Regular vfile operation descriptor.

- struct xnvfile_regular_iterator

  Regular vfile iterator.

- struct xnvfile_snapshot_ops

  Snapshot vfile operation descriptor.

- struct xnvfile_rev_tag

  Snapshot revision tag.

- struct xnvfile_snapshot

  Snapshot vfile descriptor.

- struct xnvfile_snapshot_iterator

  Snapshot-driven vfile iterator.

Functions

- int xnvfile_init_snapshot (const char *name, struct xnvfile_snapshot *vfile, struct xnvfile_directory *parent)

  Initialize a snapshot-driven vfile.

- int xnvfile_init_regular (const char *name, struct xnvfile_regular *vfile, struct xnvfile_directory *parent)

  Initialize a regular vfile.

- int xnvfile_init_dir (const char *name, struct xnvfile_directory *vdir, struct xnvfile_directory *parent)

  Initialize a virtual directory entry.

- int xnvfile_init_link (const char *from, const char *to, struct xnvfile_link *vlink, struct xnvfile_directory *parent)

  Initialize a virtual link entry.

- void xnvfile_destroy (struct xnvfile *vfile)

  Removes a virtual file entry.

- ssize_t xnvfile_get_blob (struct xnvfile_input *input, void *data, size_t size)

  Read in a data bulk written to the vfile.

- ssize_t xnvfile_get_string (struct xnvfile_input *input, char *s, size_t maxlen)

  Read in a C-string written to the vfile.

- ssize_t xnvfile_get_integer (struct xnvfile_input *input, long *valp)

  Evaluate the string written to the vfile as a long integer.

Variables

- struct xnvfile_directory nkvfroot

  Xenomai vfile root directory.

7.15.1 Detailed Description

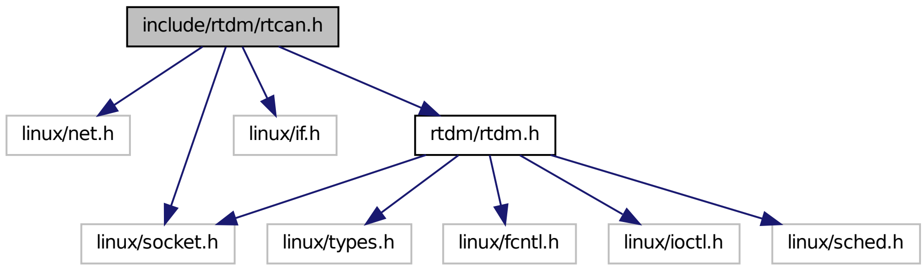
This file is part of the Xenomai project.

Note

7.16 include/rtdm/rtcan.h File Reference

Real-Time Driver Model for RT-Socket-CAN, CAN device profile header.

Include dependency graph for rtcan.h:



Data Structures

- struct can_bittime_std

  Standard bit-time parameters according to Bosch.

- struct can_bittime_btr

  Hardware-specific BTR bit-times.

- struct can_bittime

  Custom CAN bit-time definition.

- struct can_filter

  Filter for reception of CAN messages.

- struct sockaddr_can

  Socket address structure for the CAN address family.

- struct can_frame

  Raw CAN frame.

Defines

- #define AF_CAN 29

CAN address family.

- #define PF_CAN AF_CAN

  CAN protocol family.

- #define SOL_CAN_RAW 103

  CAN socket levels.

CAN ID masks

Bit masks for masking CAN IDs

- #define CAN_EFF_MASK 0x1FFFFFFF

  Bit mask for extended CAN IDs.

- #define CAN_SFF_MASK 0x000007FF

  Bit mask for standard CAN IDs.

CAN ID flags

Flags within a CAN ID indicating special CAN frame attributes

- #define CAN_EFF_FLAG 0x80000000

  Extended frame.

- #define CAN_RTR_FLAG 0x40000000

  Remote transmission frame.

- #define CAN_ERR_FLAG 0x20000000

  Error frame (see Errors), not valid in struct can_filter.

- #define CAN_INV_FILTER CAN_ERR_FLAG

  Invert CAN filter definition, only valid in struct can_filter.

Particular CAN protocols

Possible protocols for the PF_CAN protocol family

Currently only the RAW protocol is supported.

- #define CAN_RAW 1

  Raw protocol of `PF_CAN`, applicable to socket type `SOCK_RAW`.

CAN controller modes

Special CAN controllers modes, which can be or'ed together.

Note

These modes are hardware-dependent. Please consult the hardware manual of the CAN controller for more detailed information.

- #define CAN_CTRLMODE_LISTENONLY 0x1
- #define CAN_CTRLMODE_LOOPBACK 0x2

Timestamp switches

Arguments to pass to RTCAN_RTIOC_TAKE_TIMESTAMP

- #define RTCAN_TAKE_NO_TIMESTAMPS 0
  Switch off taking timestamps.
- #define RTCAN_TAKE_TIMESTAMPS 1
  Do take timestamps.

RAW socket options

Setting and getting CAN RAW socket options.

- #define CAN_RAW_FILTER 0x1
  CAN filter definition.
- #define CAN_RAW_ERR_FILTER 0x2
  CAN error mask.
- #define CAN_RAW_LOOPBACK 0x3
  CAN TX loopback.
- #define CAN_RAW_RECV_OWN_MSGS 0x4
  CAN receive own messages.

IOCTLs

CAN device IOCTLs

- #define SIOCGIFINDEX defined_by_kernel_header_file
  Get CAN interface index by name.
- #define SIOCSCANBAUDRATE _IOW(RTIOC_TYPE_CAN, 0x01, struct ifreq)
  Set baud rate.
- #define SIOCGCANBAUDRATE _IOWR(RTIOC_TYPE_CAN, 0x02, struct ifreq)
  Get baud rate.
- #define SIOCSCANCUSTOMBITTIME _IOW(RTIOC_TYPE_CAN, 0x03, struct ifreq)
  Set custom bit time parameter.
- #define SIOCGCANCUSTOMBITTIME _IOWR(RTIOC_TYPE_CAN, 0x04, struct ifreq)
  Get custum bit-time parameters.
- #define SIOCSCANMODE _IOW(RTIOC_TYPE_CAN, 0x05, struct ifreq)
  Set operation mode of CAN controller.
- #define SIOCGCANSTATE _IOWR(RTIOC_TYPE_CAN, 0x06, struct ifreq)
  Get current state of CAN controller.
- #define SIOCSCANCTRLMODE _IOW(RTIOC_TYPE_CAN, 0x07, struct ifreq)
  Set special controller modes.
- #define SIOCGCANCTRLMODE _IOWR(RTIOC_TYPE_CAN, 0x08, struct ifreq)

Get special controller modes.

- #define RTCAN_RTIOC_TAKE_TIMESTAMP _IOW(RTIOC_TYPE_CAN, 0x09, int)
  Enable or disable storing a high precision timestamp upon reception of a CAN frame.
- #define RTCAN_RTIOC_RCV_TIMEOUT _IOW(RTIOC_TYPE_CAN, 0x0A, nanosecs_-rel_t)
  Specify a reception timeout for a socket.
- #define RTCAN_RTIOC_SND_TIMEOUT _IOW(RTIOC_TYPE_CAN, 0x0B, nanosecs_-rel_t)
  Specify a transmission timeout for a socket.

Error mask

Error class (mask) in `can_id` *field of struct can_frame to be used with CAN_RAW_ERR_FILTER.*

Note: *Error reporting is hardware dependent and most CAN controllers report less detailed error conditions than the SJA1000.*

Note: *In case of a bus-off error condition (CAN_ERR_BUSOFF), the CAN controller is* ***not*** *restarted automatically. It is the application's responsibility to react appropriately, e.g. calling CAN_MODE_-START.*

Note: *Bus error interrupts (CAN_ERR_BUSERROR) are enabled when an application is calling a Recv function on a socket listening on bus errors (using CAN_RAW_ERR_FILTER). After one bus error has occured, the interrupt will be disabled to allow the application time for error processing and to efficiently avoid bus error interrupt flooding.*

- #define CAN_ERR_TX_TIMEOUT 0x00000001U
  TX timeout (netdevice driver).
- #define CAN_ERR_LOSTARB 0x00000002U
  Lost arbitration (see data[0]).
- #define CAN_ERR_CRTL 0x00000004U
  Controller problems (see data[1]).
- #define CAN_ERR_PROT 0x00000008U
  Protocol violations (see data[2], data[3]).
- #define CAN_ERR_TRX 0x00000010U
  Transceiver status (see data[4]).
- #define CAN_ERR_ACK 0x00000020U
  Received no ACK on transmission.
- #define CAN_ERR_BUSOFF 0x00000040U
  Bus off.
- #define CAN_ERR_BUSERROR 0x00000080U
  Bus error (may flood!).
- #define CAN_ERR_RESTARTED 0x00000100U
  Controller restarted.

- #define CAN_ERR_MASK 0x1FFFFFFFU
  Omit EFF, RTR, ERR flags.

Arbitration lost error

Error in the data[0] field of struct can_frame.

- #define CAN_ERR_LOSTARB_UNSPEC 0x00
  unspecified

Controller problems

Error in the data[1] field of struct can_frame.

- #define CAN_ERR_CRTL_UNSPEC 0x00
  unspecified
- #define CAN_ERR_CRTL_RX_OVERFLOW 0x01
  RX buffer overflow.
- #define CAN_ERR_CRTL_TX_OVERFLOW 0x02
  TX buffer overflow.
- #define CAN_ERR_CRTL_RX_WARNING 0x04
  reached warning level for RX errors
- #define CAN_ERR_CRTL_TX_WARNING 0x08
  reached warning level for TX errors
- #define CAN_ERR_CRTL_RX_PASSIVE 0x10
  reached passive level for RX errors
- #define CAN_ERR_CRTL_TX_PASSIVE 0x20
  reached passive level for TX errors

Protocol error type

Error in the data[2] field of struct can_frame.

- #define CAN_ERR_PROT_UNSPEC 0x00
  unspecified
- #define CAN_ERR_PROT_BIT 0x01
  single bit error
- #define CAN_ERR_PROT_FORM 0x02
  frame format error
- #define CAN_ERR_PROT_STUFF 0x04
  bit stuffing error
- #define CAN_ERR_PROT_BIT0 0x08
  unable to send dominant bit

- #define CAN_ERR_PROT_BIT1 0x10
  unable to send recessive bit
- #define CAN_ERR_PROT_OVERLOAD 0x20
  bus overload
- #define CAN_ERR_PROT_ACTIVE 0x40
  active error announcement
- #define CAN_ERR_PROT_TX 0x80
  error occured on transmission

Protocol error location

Error in the data[4] field of struct can_frame.

- #define CAN_ERR_PROT_LOC_UNSPEC 0x00
  unspecified
- #define CAN_ERR_PROT_LOC_SOF 0x03
  start of frame
- #define CAN_ERR_PROT_LOC_ID28_21 0x02
  ID bits 28 - 21 (SFF: 10 - 3).
- #define CAN_ERR_PROT_LOC_ID20_18 0x06
  ID bits 20 - 18 (SFF: 2 - 0).
- #define CAN_ERR_PROT_LOC_SRTR 0x04
  substitute RTR (SFF: RTR)
- #define CAN_ERR_PROT_LOC_IDE 0x05
  identifier extension
- #define CAN_ERR_PROT_LOC_ID17_13 0x07
  ID bits 17-13.
- #define CAN_ERR_PROT_LOC_ID12_05 0x0F
  ID bits 12-5.
- #define CAN_ERR_PROT_LOC_ID04_00 0x0E
  ID bits 4-0.
- #define CAN_ERR_PROT_LOC_RTR 0x0C
  RTR.
- #define CAN_ERR_PROT_LOC_RES1 0x0D
  reserved bit 1
- #define CAN_ERR_PROT_LOC_RES0 0x09
  reserved bit 0

- #define CAN_ERR_PROT_LOC_DLC 0x0B
  data length code
- #define CAN_ERR_PROT_LOC_DATA 0x0A
  data section
- #define CAN_ERR_PROT_LOC_CRC_SEQ 0x08
  CRC sequence.
- #define CAN_ERR_PROT_LOC_CRC_DEL 0x18
  CRC delimiter.
- #define CAN_ERR_PROT_LOC_ACK 0x19
  ACK slot.
- #define CAN_ERR_PROT_LOC_ACK_DEL 0x1B
  ACK delimiter.
- #define CAN_ERR_PROT_LOC_EOF 0x1A
  end of frame
- #define CAN_ERR_PROT_LOC_INTERM 0x12
  intermission
- #define CAN_ERR_TRX_UNSPEC 0x00
  0000 0000
- #define CAN_ERR_TRX_CANH_NO_WIRE 0x04
  0000 0100
- #define CAN_ERR_TRX_CANH_SHORT_TO_BAT 0x05
  0000 0101
- #define CAN_ERR_TRX_CANH_SHORT_TO_VCC 0x06
  0000 0110
- #define CAN_ERR_TRX_CANH_SHORT_TO_GND 0x07
  0000 0111
- #define CAN_ERR_TRX_CANL_NO_WIRE 0x40
  0100 0000
- #define CAN_ERR_TRX_CANL_SHORT_TO_BAT 0x50
  0101 0000
- #define CAN_ERR_TRX_CANL_SHORT_TO_VCC 0x60
  0110 0000
- #define CAN_ERR_TRX_CANL_SHORT_TO_GND 0x70
  0111 0000
- #define CAN_ERR_TRX_CANL_SHORT_TO_CANH 0x80
  1000 0000

Typedefs

- typedef uint32_t can_id_t
  Type of CAN id (see CAN_xxx_MASK and CAN_xxx_FLAG).
- typedef can_id_t can_err_mask_t
  Type of CAN error mask.
- typedef uint32_t can_baudrate_t
  Baudrate definition in bits per second.
- typedef enum CAN_BITTIME_TYPE can_bittime_type_t
  See CAN_BITTIME_TYPE.
- typedef enum CAN_MODE can_mode_t
  See CAN_MODE.
- typedef int can_ctrlmode_t
  See CAN_CTRLMODE.
- typedef enum CAN_STATE can_state_t
  See CAN_STATE.
- typedef struct can_filter can_filter_t
  Filter for reception of CAN messages.
- typedef struct can_frame can_frame_t
  Raw CAN frame.

Enumerations

- enum CAN_BITTIME_TYPE { CAN_BITTIME_STD, CAN_BITTIME_BTR }
  Supported CAN bit-time types.

CAN operation modes

Modes into which CAN controllers can be set

- enum CAN_MODE { CAN_MODE_STOP = 0, CAN_MODE_START, CAN_MODE_-SLEEP }

CAN controller states

States a CAN controller can be in.

- enum CAN_STATE {
  CAN_STATE_ACTIVE = 0, CAN_STATE_BUS_WARNING, CAN_STATE_BUS_-PASSIVE, CAN_STATE_BUS_OFF,
  CAN_STATE_SCANNING_BAUDRATE, CAN_STATE_STOPPED, CAN_STATE_-SLEEPING }

7.16.1 Detailed Description

Real-Time Driver Model for RT-Socket-CAN, CAN device profile header.

Note

Copyright (C) 2006 Wolfgang Grandegger <wg@grandegger.com>
Copyright (C) 2005, 2006 Sebastian Smolorz <Sebastian.Smolorz@stud.uni-hannover.de>

This RTDM CAN device profile header is based on:

include/linux/can.h, include/linux/socket.h, net/can/pf_can.h in linux-can.patch, a CAN socket framework for Linux

Copyright (C) 2004, 2005, Robert Schwebel, Benedikt Spranger, Marc Kleine-Budde, Pengutronix

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU

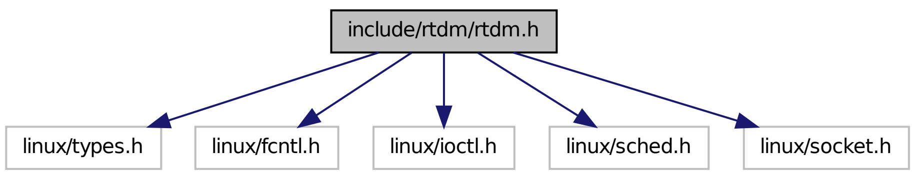
General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

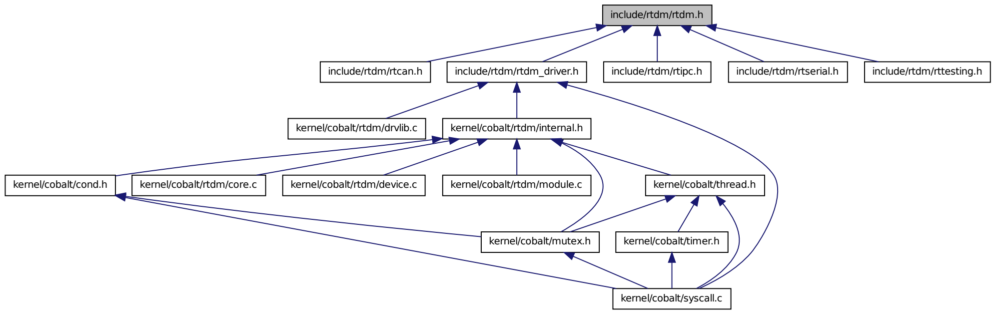
7.17 include/rtdm/rtdm.h File Reference

Real-Time Driver Model for Xenomai, user API header.

Include dependency graph for rtdm.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct rtdm_device_info
  Device information.

Defines

API Versioning

- #define RTDM_API_VER 8
  Common user and driver API version.
- #define RTDM_API_MIN_COMPAT_VER 6
  Minimum API revision compatible with the current release.

RTDM_TIMEOUT_xxx

Special timeout values

- #define RTDM_TIMEOUT_INFINITE 0
  Block forever.
- #define RTDM_TIMEOUT_NONE (-1)
  Any negative timeout means non-blocking.

RTDM_CLASS_xxx

Device classes

- #define **RTDM_CLASS_PARPORT** 1
- #define **RTDM_CLASS_SERIAL** 2
- #define **RTDM_CLASS_CAN** 3
- #define **RTDM_CLASS_NETWORK** 4
- #define **RTDM_CLASS_RTMAC** 5
- #define **RTDM_CLASS_TESTING** 6

- #define **RTDM_CLASS_RTIPC** 7
- #define **RTDM_CLASS_EXPERIMENTAL** 224
- #define **RTDM_CLASS_MAX** 255

Device Naming

Maximum length of device names (excluding the final null character)

- #define **RTDM_MAX_DEVNAME_LEN** 31

RTDM_PURGE_xxx_BUFFER

Flags selecting buffers to be purged

- #define **RTDM_PURGE_RX_BUFFER** 0x0001
- #define **RTDM_PURGE_TX_BUFFER** 0x0002

Common IOCTLs

The following IOCTLs are common to all device profiles.

- #define RTIOC_DEVICE_INFO _IOR(RTIOC_TYPE_COMMON, 0x00, struct rtdm_-device_info)

  Retrieve information about a device or socket.

- #define RTIOC_PURGE _IOW(RTIOC_TYPE_COMMON, 0x10, int)

  Purge internal device or socket buffers.

Typedefs

- typedef uint64_t nanosecs_abs_t

  RTDM type for representing absolute dates.

- typedef int64_t nanosecs_rel_t

  RTDM type for representing relative intervals.

- typedef struct rtdm_device_info rtdm_device_info_t

  Device information.

7.17.1 Detailed Description

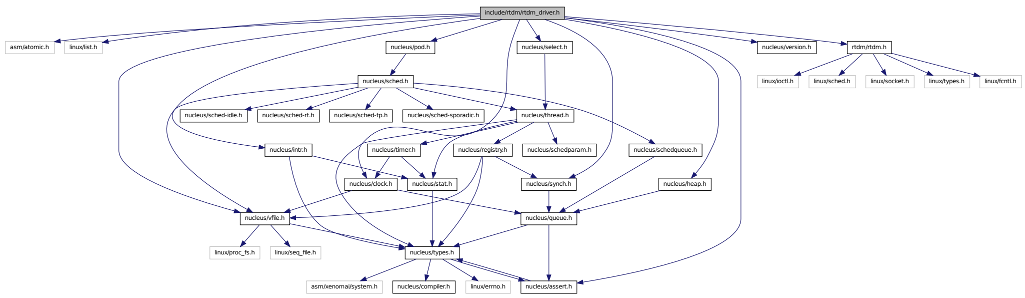
Real-Time Driver Model for Xenomai, user API header.

Note

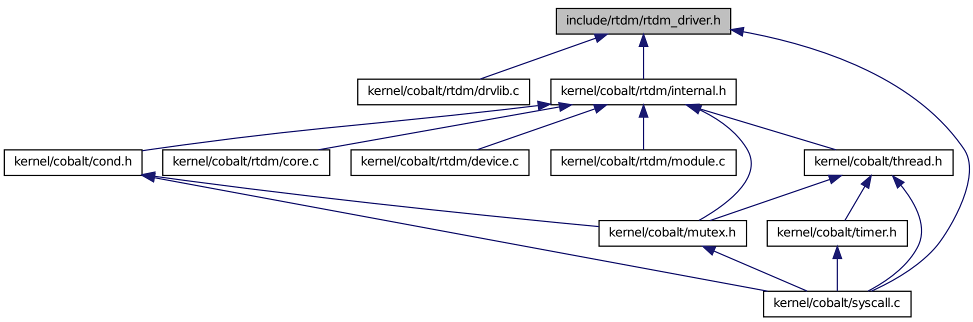
7.18 include/rtdm/rtdm_driver.h File Reference

Real-Time Driver Model for Xenomai, driver API header.

Include dependency graph for rtdm_driver.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct rtdm_operations

  Device operations.

- struct rtdm_dev_context

  Device context.

- struct rtdm_device
  RTDM device.

Defines

- #define rtdm_irq_get_arg(irq_handle, type) ((type *)irq_handle->cookie)
  Retrieve IRQ handler argument.

Device Flags

Static flags describing a RTDM device

- #define RTDM_EXCLUSIVE 0x0001
  If set, only a single instance of the device can be requested by an application.
- #define RTDM_NAMED_DEVICE 0x0010
  If set, the device is addressed via a clear-text name.
- #define RTDM_PROTOCOL_DEVICE 0x0020
  If set, the device is addressed via a combination of protocol ID and socket type.
- #define RTDM_DEVICE_TYPE_MASK 0x00F0
  Mask selecting the device type.

Context Flags

Dynamic flags describing the state of an open RTDM device (bit numbers)

- #define RTDM_CREATED_IN_NRT 0
  Set by RTDM if the device instance was created in non-real-time context.
- #define RTDM_CLOSING 1
  Set by RTDM when the device is being closed.
- #define RTDM_USER_CONTEXT_FLAG 8
  Lowest bit number the driver developer can use freely.

Driver Versioning

Current revisions of RTDM structures, encoding of driver versions. See API Versioning for the interface revision.

- #define RTDM_DEVICE_STRUCT_VER 5
  Version of struct rtdm_device.
- #define RTDM_CONTEXT_STRUCT_VER 3
  Version of struct rtdm_dev_context.
- #define RTDM_SECURE_DEVICE 0x80000000
  Flag indicating a secure variant of RTDM (not supported here).

- #define RTDM_DRIVER_VER(major, minor, patch) (((major & 0xFF) << 16) | ((minor & 0xFF) << 8) | (patch & 0xFF))

  Version code constructor for driver revisions.

- #define RTDM_DRIVER_MAJOR_VER(ver) (((ver) >> 16) & 0xFF)

  Get major version number from driver revision code.

- #define RTDM_DRIVER_MINOR_VER(ver) (((ver) >> 8) & 0xFF)

  Get minor version number from driver revision code.

- #define RTDM_DRIVER_PATCH_VER(ver) ((ver) & 0xFF)

  Get patch version number from driver revision code.

Global Lock across Scheduler Invocation

- #define RTDM_EXECUTE_ATOMICALLY(code_block)

  Execute code block atomically.

RTDM_IRQTYPE_xxx

Interrupt registrations flags

- #define RTDM_IRQTYPE_SHARED XN_ISR_SHARED

  Enable IRQ-sharing with other real-time drivers.

- #define RTDM_IRQTYPE_EDGE XN_ISR_EDGE

  Mark IRQ as edge-triggered, relevant for correct handling of shared edge-triggered IRQs.

RTDM_IRQ_xxx

Return flags of interrupt handlers

- #define RTDM_IRQ_NONE XN_ISR_NONE

  Unhandled interrupt.

- #define RTDM_IRQ_HANDLED XN_ISR_HANDLED

  Denote handled interrupt.

Task Priority Range

Maximum and minimum task priorities

- #define **RTDM_TASK_LOWEST_PRIORITY** XNSCHED_LOW_PRIO
- #define **RTDM_TASK_HIGHEST_PRIORITY** XNSCHED_HIGH_PRIO

Task Priority Modification

Raise or lower task priorities by one level

- #define **RTDM_TASK_RAISE_PRIORITY** (+1)
- #define **RTDM_TASK_LOWER_PRIORITY** (-1)

Typedefs

- typedef int(* rtdm_irq_handler_t)(rtdm_irq_t *irq_handle)
  Interrupt handler.
- typedef void(* rtdm_nrtsig_handler_t)(rtdm_nrtsig_t nrt_sig, void *arg)
  Non-real-time signal handler.
- typedef void(* rtdm_timer_handler_t)(rtdm_timer_t *timer)
  Timer handler.
- typedef void(* rtdm_task_proc_t)(void *arg)
  Real-time task procedure.

Operation Handler Prototypes

- typedef int(* rtdm_open_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info, int oflag)
  Named device open handler.
- typedef int(* rtdm_socket_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info, int protocol)
  Socket creation handler for protocol devices.
- typedef int(* rtdm_close_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info)
  Close handler.
- typedef int(* rtdm_ioctl_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info, unsigned int request, void __user *arg)
  IOCTL handler.
- typedef int(* rtdm_select_bind_handler_t)(struct rtdm_dev_context *context, rtdm_selector_t *selector, enum rtdm_selecttype type, unsigned fd_index)
  Select binding handler.
- typedef ssize_t(* rtdm_read_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info, void *buf, size_t nbyte)
  Read handler.
- typedef ssize_t(* rtdm_write_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info, const void *buf, size_t nbyte)
  Write handler.
- typedef ssize_t(* rtdm_recvmsg_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info, struct msghdr *msg, int flags)
  Receive message handler.
- typedef ssize_t(* rtdm_sendmsg_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info, const struct msghdr *msg, int flags)
  Transmit message handler.

Enumerations

RTDM_SELECTTYPE_xxx

Event types select can bind to

- enum rtdm_selecttype { RTDM_SELECTTYPE_READ = XNSELECT_READ, RTDM_SELECTTYPE_WRITE = XNSELECT_WRITE, RTDM_SELECTTYPE_EXCEPT = XNSELECT_EXCEPT }

RTDM_TIMERMODE_xxx

Timer operation modes

- enum rtdm_timer_mode { RTDM_TIMERMODE_RELATIVE = XN_RELATIVE, RTDM_TIMERMODE_ABSOLUTE = XN_ABSOLUTE, RTDM_TIMERMODE_REALTIME = XN_REALTIME }

Functions

- static void * rtdm_context_to_private (struct rtdm_dev_context *context)

  Locate the driver private area associated to a device context structure.

- static struct rtdm_dev_context * rtdm_private_to_context (void *dev_private)

  Locate a device context structure from its driver private area.

- int rtdm_dev_register (struct rtdm_device *device)

  Register a RTDM device.

- int rtdm_dev_unregister (struct rtdm_device *device, unsigned int poll_delay)

  Unregisters a RTDM device.

- struct rtdm_dev_context * rtdm_context_get (int fd)

  Retrieve and lock a device context.

- int rtdm_select_bind (int fd, rtdm_selector_t *selector, enum rtdm_selecttype type, unsigned fd_index)

  Bind a selector to specified event types of a given file descriptor.

- int rtdm_irq_request (rtdm_irq_t *irq_handle, unsigned int irq_no, rtdm_irq_handler_t handler, unsigned long flags, const char *device_name, void *arg)

  Register an interrupt handler.

- void rtdm_timer_destroy (rtdm_timer_t *timer)

  Destroy a timer.

- int rtdm_timer_start (rtdm_timer_t *timer, nanosecs_abs_t expiry, nanosecs_rel_t interval, enum rtdm_timer_mode mode)

  Start a timer.

- void rtdm_timer_stop (rtdm_timer_t *timer)

  Stop a timer.

- int rtdm_task_init (rtdm_task_t *task, const char *name, rtdm_task_proc_t task_proc, void *arg, int priority, nanosecs_rel_t period)

    Intialise and start a real-time task.

- void rtdm_task_busy_sleep (nanosecs_rel_t delay)

    Busy-wait a specified amount of time.

- void rtdm_toseq_init (rtdm_toseq_t *timeout_seq, nanosecs_rel_t timeout)

    Initialise a timeout sequence.

- void rtdm_event_init (rtdm_event_t *event, unsigned long pending)

    Initialise an event.

- int rtdm_event_select_bind (rtdm_event_t *event, rtdm_selector_t *selector, enum rtdm_selecttype type, unsigned fd_index)

    Bind a selector to an event.

- int rtdm_event_wait (rtdm_event_t *event)

    Wait on event occurrence.

- int rtdm_event_timedwait (rtdm_event_t *event, nanosecs_rel_t timeout, rtdm_toseq_t *timeout_seq)

    Wait on event occurrence with timeout.

- void rtdm_event_signal (rtdm_event_t *event)

    Signal an event occurrence.

- void rtdm_event_clear (rtdm_event_t *event)

    Clear event state.

- void rtdm_sem_init (rtdm_sem_t *sem, unsigned long value)

    Initialise a semaphore.

- int rtdm_sem_select_bind (rtdm_sem_t *sem, rtdm_selector_t *selector, enum rtdm_selecttype type, unsigned fd_index)

    Bind a selector to a semaphore.

- int rtdm_sem_down (rtdm_sem_t *sem)

    Decrement a semaphore.

- int rtdm_sem_timeddown (rtdm_sem_t *sem, nanosecs_rel_t timeout, rtdm_toseq_t *timeout_seq)

    Decrement a semaphore with timeout.

- void rtdm_sem_up (rtdm_sem_t *sem)

    Increment a semaphore.

- void rtdm_mutex_init (rtdm_mutex_t *mutex)

    Initialise a mutex.

- int rtdm_mutex_lock (rtdm_mutex_t *mutex)
  Request a mutex.
- int rtdm_mutex_timedlock (rtdm_mutex_t *mutex, nanosecs_rel_t timeout, rtdm_toseq_t *timeout_seq)
  Request a mutex with timeout.

Spinlock with Preemption Deactivation

- #define RTDM_LOCK_UNLOCKED IPIPE_SPIN_LOCK_UNLOCKED
  Static lock initialisation.
- #define rtdm_lock_init(lock) spin_lock_init(lock)
  Dynamic lock initialisation.
- #define rtdm_lock_get(lock) spin_lock(lock)
  Acquire lock from non-preemptible contexts.
- #define rtdm_lock_put(lock) spin_unlock(lock)
  Release lock without preemption restoration.
- #define rtdm_lock_get_irqsave(lock, context) spin_lock_irqsave(lock, context)
  Acquire lock and disable preemption.
- #define rtdm_lock_put_irqrestore(lock, context) spin_unlock_irqrestore(lock, context)
  Release lock and restore preemption state.
- #define rtdm_lock_irqsave(context) splhigh(context)
  Disable preemption locally.
- #define rtdm_lock_irqrestore(context) splexit(context)
  Restore preemption state.
- typedef ipipe_spinlock_t rtdm_lock_t
  Lock variable.
- typedef unsigned long rtdm_lockctx_t
  Variable to save the context while holding a lock.

7.18.1 Detailed Description

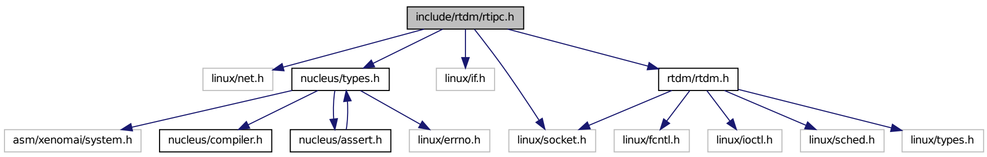
Real-Time Driver Model for Xenomai, driver API header.

Note

7.19 include/rtdm/rtipc.h File Reference

This file is part of the Xenomai project.

Include dependency graph for rtipc.h:



Data Structures

- struct rtipc_port_label

  Port label information structure.

- struct sockaddr_ipc

  Socket address structure for the RTIPC address family.

Defines

XDDP socket options

Setting and getting XDDP socket options.

- #define XDDP_LABEL 1

  XDDP label assignment.

- #define XDDP_POOLSZ 2

  XDDP local pool size configuration.

- #define XDDP_BUFSZ 3

  XDDP streaming buffer size configuration.

- #define XDDP_MONITOR 4

XDDP monitoring callback.

XDDP events

Specific events occurring on XDDP channels, which can be monitored via the XDDP_MONITOR socket option.

- #define XDDP_EVTIN 1
  Monitor writes to the non real-time endpoint.
- #define XDDP_EVTOUT 2
  Monitor reads from the non real-time endpoint.
- #define XDDP_EVTDOWN 3
  Monitor close from the non real-time endpoint.
- #define XDDP_EVTNOBUF 4
  Monitor memory shortage for non real-time datagrams.

IDDP socket options

Setting and getting IDDP socket options.

- #define IDDP_LABEL 1
  IDDP label assignment.
- #define IDDP_POOLSZ 2
  IDDP local pool size configuration.

BUFP socket options

Setting and getting BUFP socket options.

- #define BUFP_LABEL 1
  BUFP label assignment.
- #define BUFP_BUFSZ 2
  BUFP buffer size configuration.

Socket level options

Setting and getting supported standard socket level options.

- #define SO_SNDTIMEO defined_by_kernel_header_file
  IPCPROTO_IDDP and IPCPROTO_BUFP protocols support the standard SO_SNDTIMEO socket option, from the `SOL_SOCKET` *level.*
- #define SO_RCVTIMEO defined_by_kernel_header_file
  All RTIPC protocols support the standard SO_RCVTIMEO socket option, from the `SOL_SOCKET` *level.*

Typedefs

- typedef int16_t rtipc_port_t

  Port number type for the RTIPC address family.

Enumerations

RTIPC protocol list

protocols for the PF_RTIPC protocol family

- enum { IPCPROTO_IPC = 0, IPCPROTO_XDDP = 1, IPCPROTO_IDDP = 2, IPCPROTO_BUFP = 3 }

Functions

Supported operations

Standard socket operations supported by the RTIPC protocols.

- int socket__AF_RTIPC (int domain=AF_RTIPC, int type=SOCK_DGRAM, int protocol)

  Create an endpoint for communication in the AF_RTIPC domain.
- int close__AF_RTIPC (int sockfd)

  Close a RTIPC socket descriptor.
- int bind__AF_RTIPC (int sockfd, const struct sockaddr_ipc *addr, socklen_t addrlen)

  Bind a RTIPC socket to a port.
- int connect__AF_RTIPC (int sockfd, const struct sockaddr_ipc *addr, socklen_t addrlen)

  Initiate a connection on a RTIPC socket.
- int setsockopt__AF_RTIPC (int sockfd, int level, int optname, const void *optval, socklen_t optlen)

  Set options on RTIPC sockets.
- int getsockopt__AF_RTIPC (int sockfd, int level, int optname, void *optval, socklen_t *optlen)

  Get options on RTIPC sockets.
- ssize_t sendmsg__AF_RTIPC (int sockfd, const struct msghdr *msg, int flags)

  Send a message on a RTIPC socket.
- ssize_t recvmsg__AF_RTIPC (int sockfd, struct msghdr *msg, int flags)

  Receive a message from a RTIPC socket.
- int getsockname__AF_RTIPC (int sockfd, struct sockaddr_ipc *addr, socklen_t *addrlen)

  Get socket name.
- int getpeername__AF_RTIPC (int sockfd, struct sockaddr_ipc *addr, socklen_t *addrlen)

  Get socket peer.

7.19.1 Detailed Description

This file is part of the Xenomai project.

Note

Copyright (C) 2009 Philippe Gerum <rpm@xenomai.org>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

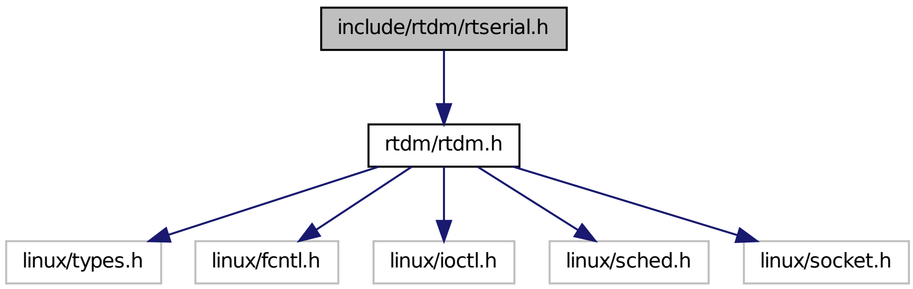
This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

7.20 include/rtdm/rtserial.h File Reference

Real-Time Driver Model for Xenomai, serial device profile header.

Include dependency graph for rtserial.h:



Data Structures

- struct rtser_config

  Serial device configuration.

- struct rtser_status

  Serial device status.

- struct rtser_event

  Additional information about serial device events.

Defines

- #define RTSER_RTIOC_BREAK_CTL _IOR(RTIOC_TYPE_SERIAL, 0x06, int)

  Set or clear break on UART output line.

RTSER_DEF_BAUD

Default baud rate

- #define **RTSER_DEF_BAUD** 9600

RTSER_xxx_PARITY

Number of parity bits

- #define **RTSER_NO_PARITY** 0x00
- #define **RTSER_ODD_PARITY** 0x01
- #define **RTSER_EVEN_PARITY** 0x03
- #define **RTSER_DEF_PARITY** RTSER_NO_PARITY

RTSER_xxx_BITS

Number of data bits

- #define **RTSER_5_BITS** 0x00
- #define **RTSER_6_BITS** 0x01
- #define **RTSER_7_BITS** 0x02
- #define **RTSER_8_BITS** 0x03
- #define **RTSER_DEF_BITS** RTSER_8_BITS

RTSER_xxx_STOPB

Number of stop bits

- #define RTSER_1_STOPB 0x00

  valid only in combination with 5 data bits

- #define RTSER_1_5_STOPB 0x01

  valid only in combination with 5 data bits

- #define RTSER_2_STOPB 0x01

  valid only in combination with 5 data bits

- #define RTSER_DEF_STOPB RTSER_1_STOPB

  valid only in combination with 5 data bits

RTSER_xxx_HAND

Handshake mechanisms

- #define **RTSER_NO_HAND** 0x00
- #define **RTSER_RTSCTS_HAND** 0x01
- #define **RTSER_DEF_HAND** RTSER_NO_HAND

RTSER_FIFO_xxx

Reception FIFO interrupt threshold

- #define **RTSER_FIFO_DEPTH_1** 0x00
- #define **RTSER_FIFO_DEPTH_4** 0x40
- #define **RTSER_FIFO_DEPTH_8** 0x80
- #define **RTSER_FIFO_DEPTH_14** 0xC0
- #define **RTSER_DEF_FIFO_DEPTH** RTSER_FIFO_DEPTH_1

RTSER_TIMEOUT_xxx

Special timeout values, see also RTDM_TIMEOUT_xxx

- #define **RTSER_TIMEOUT_INFINITE** RTDM_TIMEOUT_INFINITE
- #define **RTSER_TIMEOUT_NONE** RTDM_TIMEOUT_NONE
- #define **RTSER_DEF_TIMEOUT** RTDM_TIMEOUT_INFINITE

RTSER_xxx_TIMESTAMP_HISTORY

Timestamp history control

- #define **RTSER_RX_TIMESTAMP_HISTORY** 0x01
- #define **RTSER_DEF_TIMESTAMP_HISTORY** 0x00

RTSER_EVENT_xxx

Events bits

- #define **RTSER_EVENT_RXPEND** 0x01
- #define **RTSER_EVENT_ERRPEND** 0x02
- #define **RTSER_EVENT_MODEMHI** 0x04
- #define **RTSER_EVENT_MODEMLO** 0x08
- #define **RTSER_DEF_EVENT_MASK** 0x00

RTSER_SET_xxx

Configuration mask bits

- #define **RTSER_SET_BAUD** 0x0001
- #define **RTSER_SET_PARITY** 0x0002
- #define **RTSER_SET_DATA_BITS** 0x0004
- #define **RTSER_SET_STOP_BITS** 0x0008
- #define **RTSER_SET_HANDSHAKE** 0x0010
- #define **RTSER_SET_FIFO_DEPTH** 0x0020
- #define **RTSER_SET_TIMEOUT_RX** 0x0100
- #define **RTSER_SET_TIMEOUT_TX** 0x0200
- #define **RTSER_SET_TIMEOUT_EVENT** 0x0400
- #define **RTSER_SET_TIMESTAMP_HISTORY** 0x0800
- #define **RTSER_SET_EVENT_MASK** 0x1000

RTSER_LSR_xxx

Line status bits

- #define **RTSER_LSR_DATA** 0x01
- #define **RTSER_LSR_OVERRUN_ERR** 0x02
- #define **RTSER_LSR_PARITY_ERR** 0x04
- #define **RTSER_LSR_FRAMING_ERR** 0x08
- #define **RTSER_LSR_BREAK_IND** 0x10
- #define **RTSER_LSR_THR_EMTPY** 0x20
- #define **RTSER_LSR_TRANSM_EMPTY** 0x40
- #define **RTSER_LSR_FIFO_ERR** 0x80
- #define **RTSER_SOFT_OVERRUN_ERR** 0x0100

RTSER_MSR_xxx

Modem status bits

- #define **RTSER_MSR_DCTS** 0x01
- #define **RTSER_MSR_DDSR** 0x02
- #define **RTSER_MSR_TERI** 0x04
- #define **RTSER_MSR_DDCD** 0x08
- #define **RTSER_MSR_CTS** 0x10
- #define **RTSER_MSR_DSR** 0x20
- #define **RTSER_MSR_RI** 0x40
- #define **RTSER_MSR_DCD** 0x80

RTSER_MCR_xxx

Modem control bits

- #define **RTSER_MCR_DTR** 0x01
- #define **RTSER_MCR_RTS** 0x02
- #define **RTSER_MCR_OUT1** 0x04
- #define **RTSER_MCR_OUT2** 0x08
- #define **RTSER_MCR_LOOP** 0x10

Sub-Classes of RTDM_CLASS_SERIAL

- #define **RTDM_SUBCLASS_16550A** 0

IOCTLs

Serial device IOCTLs

- #define RTSER_RTIOC_GET_CONFIG _IOR(RTIOC_TYPE_SERIAL, 0x00, struct rtser_-config)

  Get serial device configuration.

- #define RTSER_RTIOC_SET_CONFIG _IOW(RTIOC_TYPE_SERIAL, 0x01, struct rtser_-config)

  Set serial device configuration.

- #define RTSER_RTIOC_GET_STATUS _IOR(RTIOC_TYPE_SERIAL, 0x02, struct rtser_-status)

  Get serial device status.

- #define RTSER_RTIOC_GET_CONTROL _IOR(RTIOC_TYPE_SERIAL, 0x03, int)

  Get serial device's modem contol register.

- #define RTSER_RTIOC_SET_CONTROL _IOW(RTIOC_TYPE_SERIAL, 0x04, int)

  Set serial device's modem contol register.

- #define RTSER_RTIOC_WAIT_EVENT _IOR(RTIOC_TYPE_SERIAL, 0x05, struct rtser_-event)

  Wait on serial device events according to previously set mask.

RTSER_BREAK_xxx

Break control

- #define RTSER_BREAK_CLR 0x00

  Serial device configuration.

- #define RTSER_BREAK_SET 0x01

  Serial device configuration.

- #define RTIOC_TYPE_SERIAL RTDM_CLASS_SERIAL

  Serial device configuration.

- typedef struct rtser_config rtser_config_t

  Serial device configuration.

- typedef struct rtser_status rtser_status_t

  Serial device status.

- typedef struct rtser_event rtser_event_t

  Additional information about serial device events.

7.20.1 Detailed Description

Real-Time Driver Model for Xenomai, serial device profile header.

Note

Copyright (C) 2005-2007 Jan Kiszka <jan.kiszka@web.de>

Xenomai is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

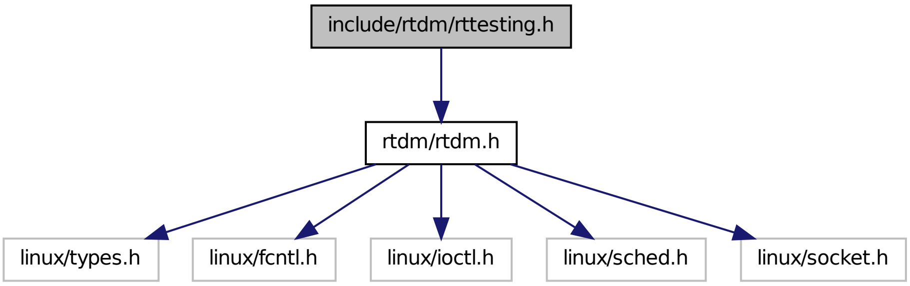
Xenomai is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with Xenomai; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

7.21 include/rtdm/rttesting.h File Reference

Real-Time Driver Model for Xenomai, testing device profile header.

Include dependency graph for rttesting.h:



Defines

Sub-Classes of RTDM_CLASS_TESTING

- #define RTDM_SUBCLASS_TIMERBENCH 0

  subclass name: "timerbench"

- #define RTDM_SUBCLASS_IRQBENCH 1

  subclass name: "irqbench"

- #define RTDM_SUBCLASS_SWITCHTEST 2

  subclass name: "switchtest"

- #define RTDM_SUBCLASS_RTDMTEST 3

  subclase name: "rtdm"

IOCTLs

Testing device IOCTLs

- #define **RTTST_RTIOC_INTERM_BENCH_RES** _IOWR(RTIOC_TYPE_TESTING, 0x00, struct rttst_interm_bench_res)
- #define **RTTST_RTIOC_TMBENCH_START** _IOW(RTIOC_TYPE_TESTING, 0x10, struct rttst_tmbench_config)
- #define **RTTST_RTIOC_TMBENCH_STOP** _IOWR(RTIOC_TYPE_TESTING, 0x11, struct rttst_overall_bench_res)
- #define **RTTST_RTIOC_IRQBENCH_START** _IOW(RTIOC_TYPE_TESTING, 0x20, struct rttst_irqbench_config)
- #define **RTTST_RTIOC_IRQBENCH_STOP** _IO(RTIOC_TYPE_TESTING, 0x21)
- #define **RTTST_RTIOC_IRQBENCH_GET_STATS** _IOR(RTIOC_TYPE_TESTING, 0x22, struct rttst_irqbench_stats)
- #define **RTTST_RTIOC_IRQBENCH_WAIT_IRQ** _IO(RTIOC_TYPE_TESTING, 0x23)
- #define **RTTST_RTIOC_IRQBENCH_REPLY_IRQ** _IO(RTIOC_TYPE_TESTING, 0x24)
- #define **RTTST_RTIOC_SWTEST_SET_TASKS_COUNT** _IOW(RTIOC_TYPE_-TESTING, 0x30, unsigned long)

- #define **RTTST_RTIOC_SWTEST_SET_CPU** _IOW(RTIOC_TYPE_TESTING, 0x31, unsigned long)
- #define **RTTST_RTIOC_SWTEST_REGISTER_UTASK** _IOW(RTIOC_TYPE_TESTING, 0x32, struct rttst_swtest_task)
- #define **RTTST_RTIOC_SWTEST_CREATE_KTASK** _IOWR(RTIOC_TYPE_TESTING, 0x33, struct rttst_swtest_task)
- #define **RTTST_RTIOC_SWTEST_PEND** _IOR(RTIOC_TYPE_TESTING, 0x34, struct rttst_swtest_task)
- #define **RTTST_RTIOC_SWTEST_SWITCH_TO** _IOR(RTIOC_TYPE_TESTING, 0x35, struct rttst_swtest_dir)
- #define **RTTST_RTIOC_SWTEST_GET_SWITCHES_COUNT** _IOR(RTIOC_TYPE_TESTING, 0x36, unsigned long)
- #define **RTTST_RTIOC_SWTEST_GET_LAST_ERROR** _IOR(RTIOC_TYPE_TESTING, 0x37, struct rttst_swtest_error)
- #define **RTTST_RTIOC_SWTEST_SET_PAUSE** _IOW(RTIOC_TYPE_TESTING, 0x38, unsigned long)
- #define **RTTST_RTIOC_RTDM_DEFER_CLOSE** _IOW(RTIOC_TYPE_TESTING, 0x40, unsigned long)

7.21.1 Detailed Description

Real-Time Driver Model for Xenomai, testing device profile header.

Note

Copyright (C) 2005 Jan Kiszka <jan.kiszka@web.de>

Xenomai is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

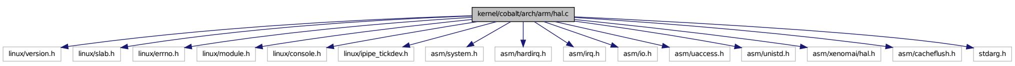
Xenomai is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with Xenomai; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

7.22 kernel/cobalt/arch/arm/hal.c File Reference

Adeos-based Real-Time Abstraction Layer for ARM.

Include dependency graph for hal.c:



Functions

- int rthal_timer_request (void(*tick_handler)(void), void(*mode_emul)(enum clock_event_mode mode, struct clock_event_device *cdev), int(*tick_emul)(unsigned long delay, struct clock_event_device *cdev), int cpu)

Grab the hardware timer.

- void rthal_timer_release (int cpu)

  Release the hardware timer.

7.22.1 Detailed Description

Adeos-based Real-Time Abstraction Layer for ARM. ARM port Copyright (C) 2005 Stelian Pop

ARM-specific HAL services.

7.22.2 Function Documentation

7.22.2.1 void rthal_timer_release (int *cpu*)

Release the hardware timer.

Releases the hardware timer, thus reverting the effect of a previous call to rthal_timer_request(). In case the timer hardware is shared with Linux, a periodic setup suitable for the Linux kernel will be reset.

Parameters

cpu The CPU number the timer was grabbed from.

Environments:

This service can be called from:

- Linux domain context.

7.22.2.2 int rthal_timer_request (void(*)(void) *tick_handler*, void(*)(enum clock_event_mode mode, struct clock_event_device *cdev) *mode_emul*, int(*)(unsigned long delay, struct clock_event_device *cdev) *tick_emul*, int *cpu*)

Grab the hardware timer.

rthal_timer_request() grabs and tunes the hardware timer in oneshot mode in order to clock the master time base. GENERIC_CLOCKEVENTS is required from the host kernel.

A user-defined routine is registered as the clock tick handler. This handler will always be invoked on behalf of the Xenomai domain for each incoming tick.

Host tick emulation is a way to share the clockchip hardware between Linux and Xenomai, when the former provides support for oneshot timing (i.e. high resolution timers and no-HZ scheduler ticking).

Parameters

tick_handler The address of the Xenomai tick handler which will process each incoming tick.

mode_emul The optional address of a callback to be invoked upon mode switch of the host tick device, notified by the Linux kernel.

tick_emul The optional address of a callback to be invoked upon setup of the next shot date for the host tick device, notified by the Linux kernel.

cpu The CPU number to grab the timer from.

Returns

a positive value is returned on success, representing the duration of a Linux periodic tick expressed as a count of nanoseconds; zero should be returned when the Linux kernel does not undergo periodic timing on the given CPU (e.g. oneshot mode). Otherwise:

- -EBUSY is returned if the hardware timer has already been grabbed. rthal_timer_request() must be issued before rthal_timer_request() is called again.

- -ENODEV is returned if the hardware timer cannot be used. This situation may occur after the kernel disabled the timer due to invalid calibration results; in such a case, such hardware is unusable for any timing duties.

Environments:

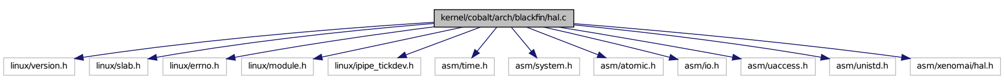
This service can be called from:

- Linux domain context.

7.23 kernel/cobalt/arch/blackfin/hal.c File Reference

Adeos-based Real-Time Abstraction Layer for the Blackfin architecture.

Include dependency graph for hal.c:



7.23.1 Detailed Description

Adeos-based Real-Time Abstraction Layer for the Blackfin architecture. Copyright (C) 2005-2006 Philippe Gerum.

Xenomai is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, Inc., 675 Mass Ave, Cambridge MA 02139, USA; either version 2 of the License, or (at your option) any later version.

Xenomai is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

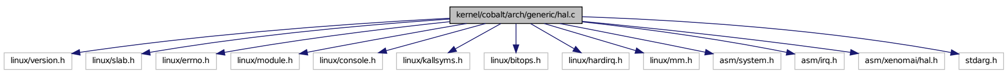
You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

Blackfin-specific HAL services.

7.24 kernel/cobalt/arch/generic/hal.c File Reference

Generic Real-Time HAL.

Include dependency graph for hal.c:



Functions

- int rthal_apc_alloc (const char *name, void(*handler)(void *cookie), void *cookie)

  Allocate an APC slot.

- void rthal_apc_free (int apc)

  Releases an APC slot.

7.24.1 Detailed Description

Generic Real-Time HAL. Copyright ©2005 Philippe Gerum.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, Inc., 675 Mass Ave, Cambridge MA 02139, USA; either version 2 of the License, or (at your option) any later version.

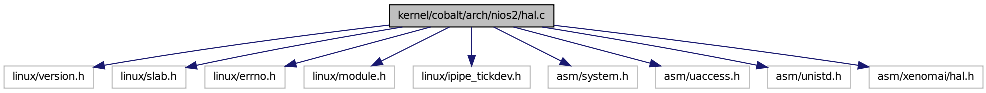
This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

7.25 kernel/cobalt/arch/nios2/hal.c File Reference

Adeos-based Real-Time Abstraction Layer for the NIOS2 architecture.

Include dependency graph for hal.c:



7.25.1 Detailed Description

Adeos-based Real-Time Abstraction Layer for the NIOS2 architecture. Copyright (C) 2009 Philippe Gerum.

Xenomai is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, Inc., 675 Mass Ave, Cambridge MA 02139, USA; either version 2 of the License, or (at your option) any later version.

Xenomai is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

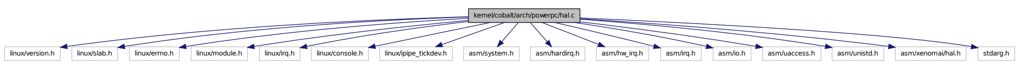
You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

NIOS2-specific HAL services.

7.26 kernel/cobalt/arch/powerpc/hal.c File Reference

Adeos-based Real-Time Abstraction Layer for PowerPC.

Include dependency graph for hal.c:



7.26.1 Detailed Description

Adeos-based Real-Time Abstraction Layer for PowerPC. Copyright (C) 2004-2006 Philippe Gerum.

64-bit PowerPC adoption copyright (C) 2005 Taneli Vähäkangas and Heikki Lindholm

Xenomai is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, Inc., 675 Mass Ave, Cambridge MA 02139, USA; either version 2 of the License, or (at your option) any later version.

Xenomai is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

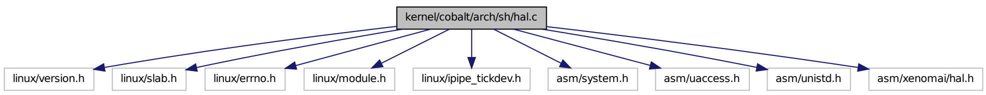
You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

PowerPC-specific HAL services.

7.27 kernel/cobalt/arch/sh/hal.c File Reference

Adeos-based Real-Time Abstraction Layer for the SuperH architecture.

Include dependency graph for hal.c:



7.27.1 Detailed Description

Adeos-based Real-Time Abstraction Layer for the SuperH architecture. Copyright (C) 2011 Philippe Gerum.

Xenomai is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, Inc., 675 Mass Ave, Cambridge MA 02139, USA; either version 2 of the License, or (at your option) any later version.

Xenomai is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

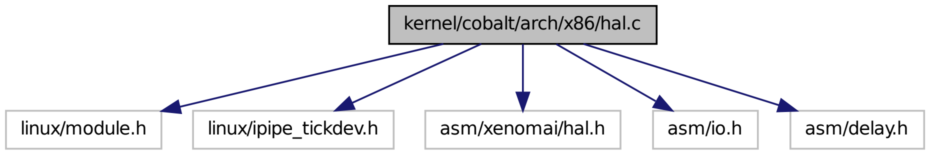
You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

SuperH-specific HAL services.

7.28 kernel/cobalt/arch/x86/hal.c File Reference

Adeos-based Real-Time Abstraction Layer for x86.

Include dependency graph for hal.c:



7.28.1 Detailed Description

Adeos-based Real-Time Abstraction Layer for x86. Common code of i386 and x86_64.

Copyright (C) 2007-2012 Philippe Gerum.

Xenomai is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, Inc., 675 Mass Ave, Cambridge MA 02139, USA; either version 2 of the License, or (at your option) any later version.

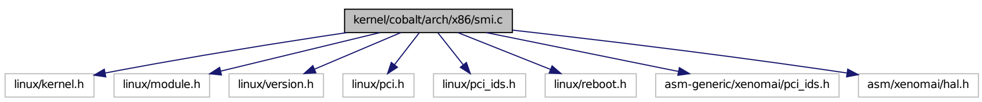
Xenomai is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

7.29 kernel/cobalt/arch/x86/smi.c File Reference

SMI workaround for x86.

Include dependency graph for smi.c:



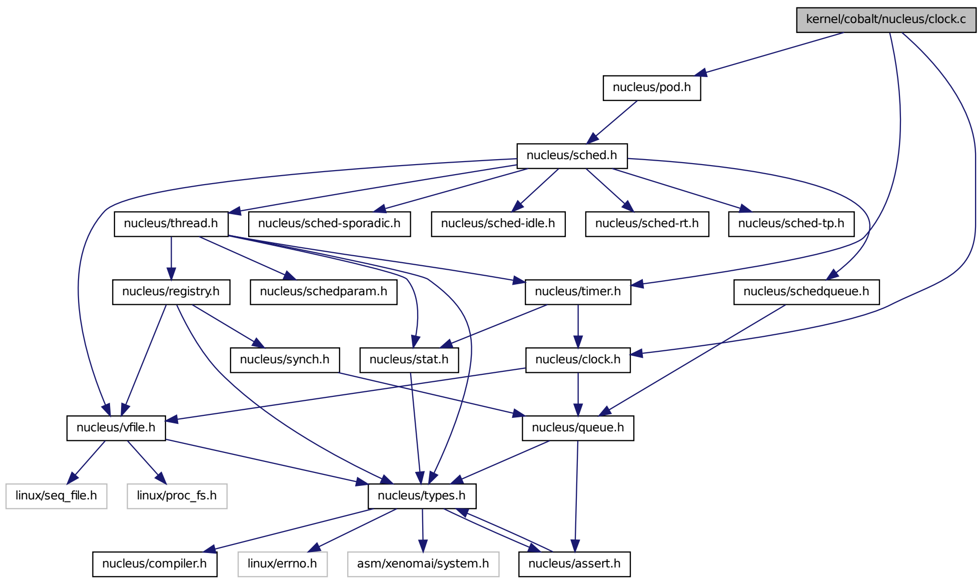
7.29.1 Detailed Description

SMI workaround for x86. Cut/Pasted from Vitor Angelo "smi" module. Adapted by Gilles Chanteperdrix <gilles.chanteperdrix@xenomai.org>.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, Inc., 675 Mass Ave, Cambridge MA 02139, USA; either version 2 of the License, or (at your option) any later version.

7.30 kernel/cobalt/nucleus/clock.c File Reference

Include dependency graph for clock.c:



Functions

- void xnclock_adjust (xnsticks_t delta)

  Adjust the clock time for the system.

7.30.1 Detailed Description

Note

Copyright (C) 2006-2011 Philippe Gerum <rpm@xenomai.org>.

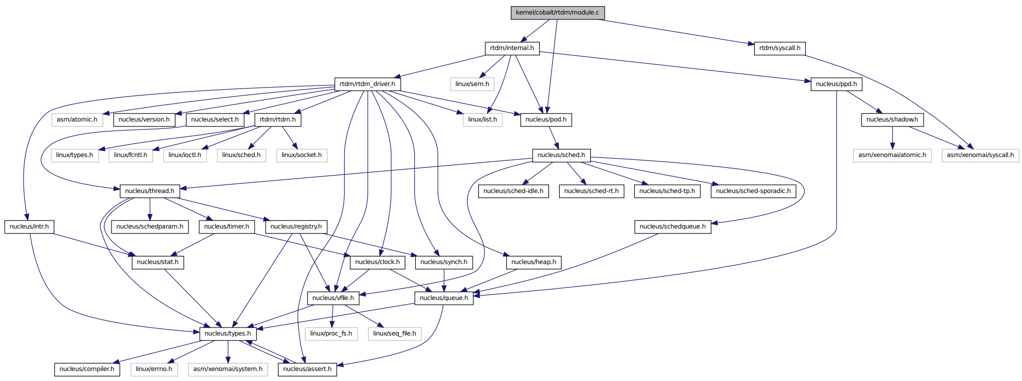
Xenomai is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with Xenomai; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

7.31 kernel/cobalt/rtdm/module.c File Reference

Real-Time Driver Model for Xenomai.

Include dependency graph for module.c:



7.31.1 Detailed Description

Real-Time Driver Model for Xenomai.

Note

Copyright (C) 2005, 2006 Jan Kiszka <jan.kiszka@web.de>
Copyright (C) 2005 Joerg Langenberg <joerg.langenberg@gmx.net>

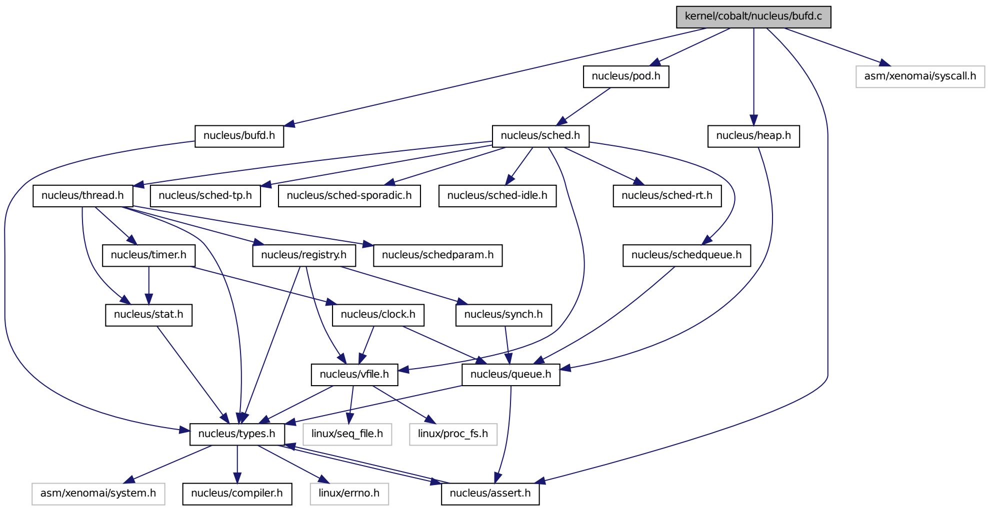
Xenomai is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

Xenomai is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with Xenomai; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

7.32 kernel/cobalt/nucleus/bufd.c File Reference

Include dependency graph for bufd.c:



Functions

- ssize_t xnbufd_copy_to_kmem (void ∗ptr, struct xnbufd ∗bufd, size_t len)

  Copy memory covered by a buffer descriptor to kernel memory.

- ssize_t xnbufd_copy_from_kmem (struct xnbufd ∗bufd, void ∗from, size_t len)

  Copy kernel memory to the area covered by a buffer descriptor.

- ssize_t xnbufd_unmap_uread (struct xnbufd ∗bufd)

  Finalize a buffer descriptor obtained from xnbufd_map_uread().

- ssize_t xnbufd_unmap_uwrite (struct xnbufd ∗bufd)

  Finalize a buffer descriptor obtained from xnbufd_map_uwrite().

- void xnbufd_invalidate (struct xnbufd ∗bufd)

  Invalidate a buffer descriptor.

- ssize_t xnbufd_unmap_kread (struct xnbufd ∗bufd)

  Finalize a buffer descriptor obtained from xnbufd_map_kread().

- ssize_t xnbufd_unmap_kwrite (struct xnbufd ∗bufd)

  Finalize a buffer descriptor obtained from xnbufd_map_kwrite().

7.32.1 Detailed Description

Note

Copyright (C) 2009 Philippe Gerum <rpm@xenomai.org>.

Xenomai is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

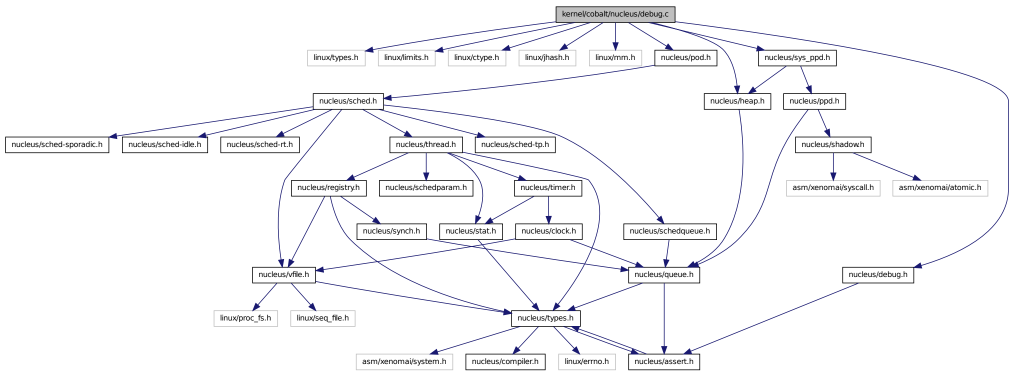
Xenomai is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with Xenomai; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

7.33 kernel/cobalt/nucleus/debug.c File Reference

Debug services.

Include dependency graph for debug.c:



7.33.1 Detailed Description

Debug services.

Author

Philippe Gerum

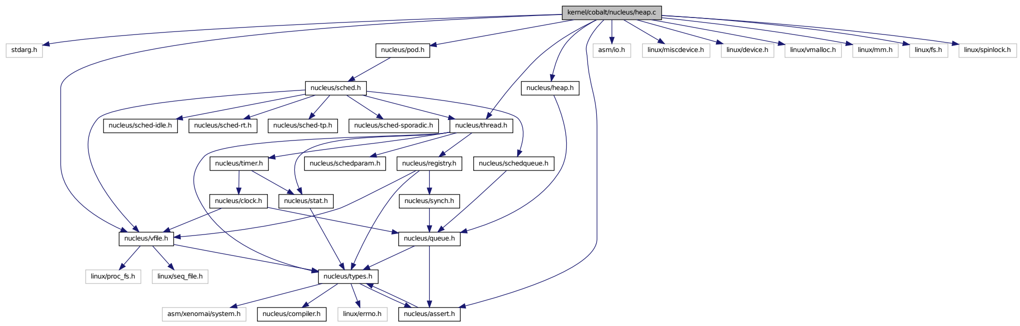
Copyright (C) 2010 Philippe Gerum <rpm@xenomai.org>.

Xenomai is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

7.34 kernel/cobalt/nucleus/heap.c File Reference

Dynamic memory allocation services.

Include dependency graph for heap.c:



Functions

- int xnheap_init (xnheap_t *heap, void *heapaddr, u_long heapsize, u_long pagesize)

  Initialize a memory heap.

- void xnheap_set_label (xnheap_t *heap, const char *label,...)

  Set the heap's label string.

- void xnheap_destroy (xnheap_t *heap, void(*flushfn)(xnheap_t *heap, void *extaddr, u_long extsize, void *cookie), void *cookie)

  Destroys a memory heap.

- void * xnheap_alloc (xnheap_t *heap, u_long size)

  Allocate a memory block from a memory heap.

- int xnheap_test_and_free (xnheap_t *heap, void *block, int(*ckfn)(void *block))

  Test and release a memory block to a memory heap.

- int xnheap_free (xnheap_t *heap, void *block)

  Release a memory block to a memory heap.

- int xnheap_extend (xnheap_t *heap, void *extaddr, u_long extsize)

  Extend a memory heap.

- void xnheap_schedule_free (xnheap_t *heap, void *block, xnholder_t *link)

    Schedule a memory block for release.

7.34.1 Detailed Description

Dynamic memory allocation services.

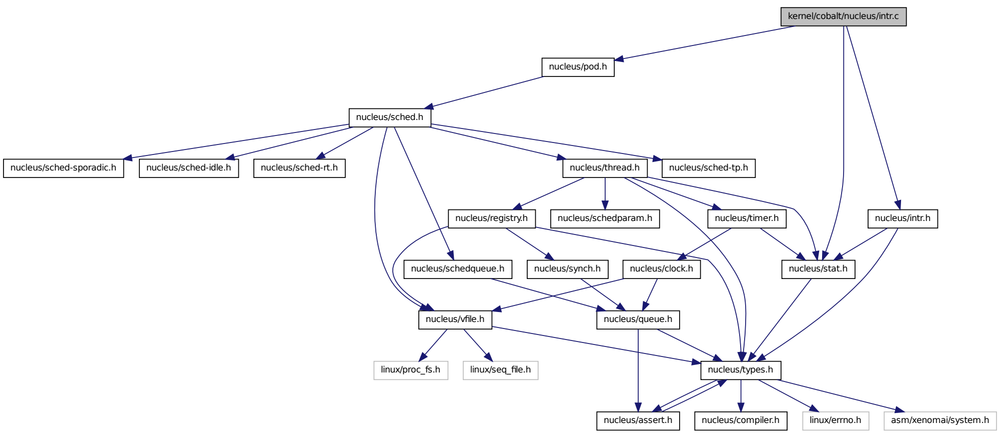
Author

Philippe Gerum

7.35 kernel/cobalt/nucleus/intr.c File Reference

Interrupt management.

Include dependency graph for intr.c:



Functions

- int xnintr_init (xnintr_t *intr, const char *name, unsigned irq, xnisr_t isr, xniack_t iack, xnflags_t flags)

  Initialize an interrupt object.

- int xnintr_destroy (xnintr_t *intr)

  Destroy an interrupt object.

- int xnintr_attach (xnintr_t *intr, void *cookie)

  Attach an interrupt object.

- int xnintr_detach (xnintr_t *intr)

  Detach an interrupt object.

- void xnintr_enable (xnintr_t *intr)

  Enable an interrupt object.

- void xnintr_disable (xnintr_t *intr)

  Disable an interrupt object.

- void xnintr_affinity (xnintr_t *intr, xnarch_cpumask_t cpumask)

  Set interrupt's processor affinity.

7.35.1 Detailed Description

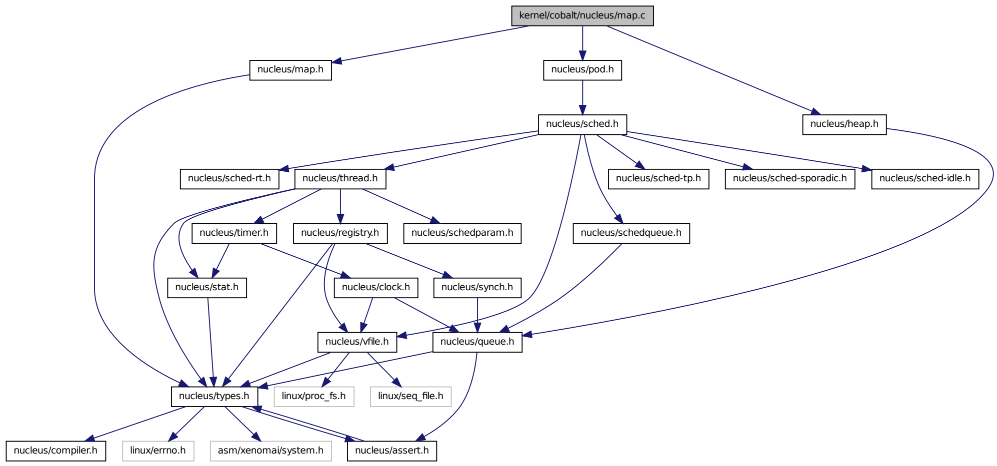
Interrupt management.

Author

Philippe Gerum

7.36 kernel/cobalt/nucleus/map.c File Reference

Include dependency graph for map.c:



Functions

- xnmap_t * xnmap_create (int nkeys, int reserve, int offset)

    Create a map.

- void xnmap_delete (xnmap_t *map)

    Delete a map.

- int xnmap_enter (xnmap_t *map, int key, void *objaddr)

    Index an object into a map.

- int xnmap_remove (xnmap_t *map, int key)

    Remove an object reference from a map.

7.36.1 Detailed Description

Note

Copyright (C) 2007 Philippe Gerum <rpm@xenomai.org>.

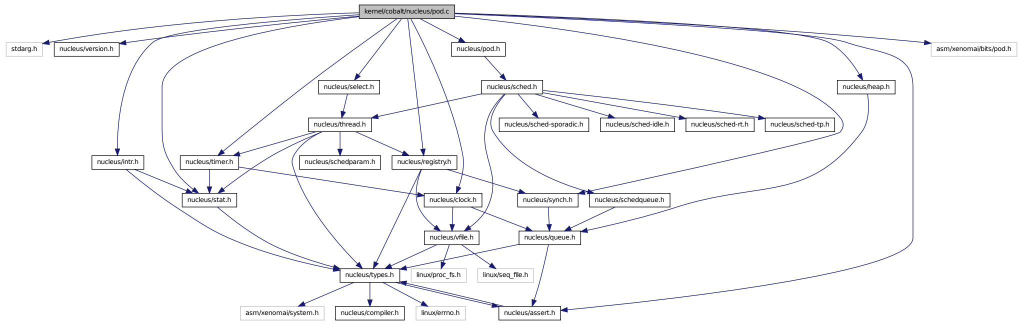
Xenomai is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

Xenomai is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

7.37 kernel/cobalt/nucleus/pod.c File Reference

Real-time pod services.

Include dependency graph for pod.c:



Functions

- int xnpod_init (void)

    Initialize the core pod.

- void xnpod_shutdown (int xtype)

    Shutdown the current pod.

- int xnpod_init_thread (struct xnthread *thread, const struct xnthread_init_attr *attr, struct xnsched_class *sched_class, const union xnsched_policy_param *sched_param)

    Initialize a new thread.

- int xnpod_start_thread (xnthread_t *thread, const struct xnthread_start_attr *attr)

    Initial start of a newly created thread.

- void __xnpod_reset_thread (struct xnthread *thread)

    Reset the thread.

- void xnpod_stop_thread (xnthread_t *thread)

    Stop a thread.

- xnflags_t xnpod_set_thread_mode (xnthread_t *thread, xnflags_t clrmask, xnflags_t setmask)

    Change a thread's control mode.

- void xnpod_delete_thread (xnthread_t *thread)

Delete a thread.

- void xnpod_abort_thread (xnthread_t *thread)

  Abort a thread.

- void xnpod_suspend_thread (xnthread_t *thread, xnflags_t mask, xnticks_t timeout, xntmode_t timeout_mode, struct xnsynch *wchan)

  Suspend a thread.

- void xnpod_resume_thread (xnthread_t *thread, xnflags_t mask)

  Resume a thread.

- int xnpod_unblock_thread (xnthread_t *thread)

  Unblock a thread.

- int xnpod_set_thread_schedparam (struct xnthread *thread, struct xnsched_class *sched_-class, const union xnsched_policy_param *sched_param)

  Change the base scheduling parameters of a thread.

- int xnpod_migrate_thread (int cpu)

  Migrate the current thread.

- void xnpod_dispatch_signals (void)

  Deliver pending asynchronous signals to the running thread.

- void xnpod_welcome_thread (xnthread_t *thread, int imask)

  Thread prologue.

- int xnpod_add_hook (int type, void(*routine)(xnthread_t *))

  Install a nucleus hook.

- int xnpod_remove_hook (int type, void(*routine)(xnthread_t *))

  Remove a nucleus hook.

- int xnpod_handle_exception (struct ipipe_trap_data *d)

  Exception handler.

- int xnpod_enable_timesource (void)

  Activate the core time source.

- void xnpod_disable_timesource (void)

  Stop the core time source.

- int xnpod_set_thread_periodic (xnthread_t *thread, xnticks_t idate, xntmode_t timeout_-mode, xnticks_t period)

  Make a thread periodic.

- int xnpod_wait_thread_period (unsigned long *overruns_r)

  Wait for the next periodic release point.

- int xnpod_set_thread_tslice (struct xnthread *thread, xnticks_t quantum)

  Set thread time-slicing information.

7.37.1 Detailed Description

Real-time pod services.

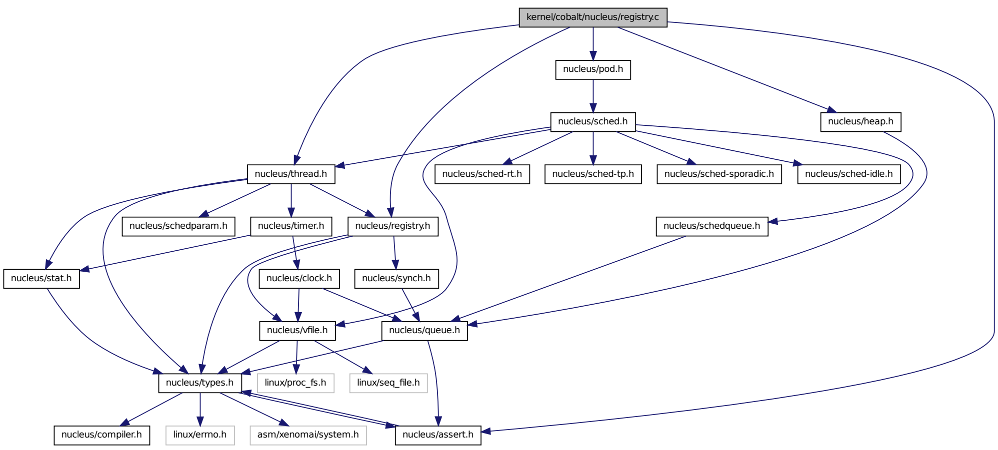
Author

Philippe Gerum

7.38 kernel/cobalt/nucleus/registry.c File Reference

This file is part of the Xenomai project.

Include dependency graph for registry.c:



Functions

- int xnregistry_enter (const char *key, void *objaddr, xnhandle_t *phandle, struct xnpnode *pnode)

  Register a real-time object.

- int xnregistry_bind (const char *key, xnticks_t timeout, int timeout_mode, xnhandle_t *phandle)

  Bind to a real-time object.

- int xnregistry_remove (xnhandle_t handle)

  Forcibly unregister a real-time object.

- int xnregistry_remove_safe (xnhandle_t handle, xnticks_t timeout)

  Unregister an idle real-time object.

- void * xnregistry_get (xnhandle_t handle)

  Find and lock a real-time object into the registry.

- u_long xnregistry_put (xnhandle_t handle)

  Unlock a real-time object from the registry.

- void * xnregistry_fetch (xnhandle_t handle)

  Find a real-time object into the registry.

7.38.1 Detailed Description

This file is part of the Xenomai project.

Note

Copyright (C) 2004 Philippe Gerum <rpm@xenomai.org>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

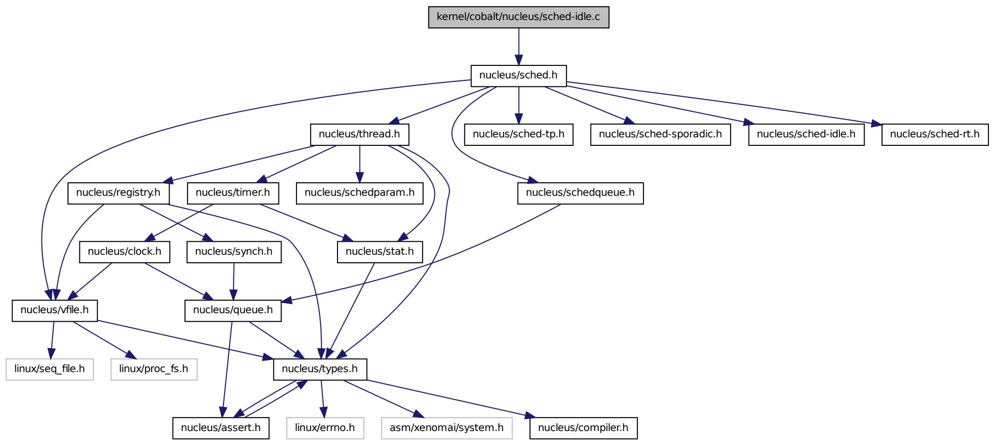
This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

7.39 kernel/cobalt/nucleus/sched-idle.c File Reference

Idle scheduling class implementation (i.e. Linux placeholder).

Include dependency graph for sched-idle.c:



7.39.1 Detailed Description

Idle scheduling class implementation (i.e. Linux placeholder).

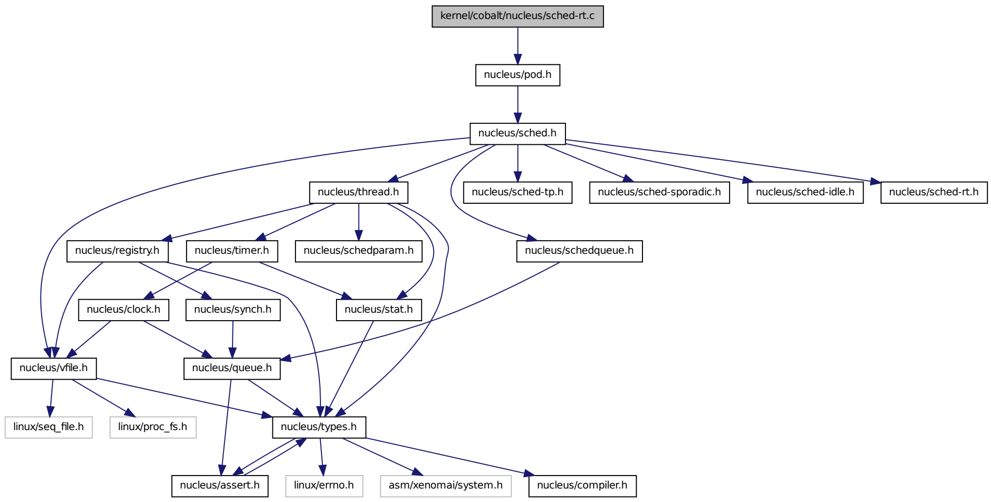
Author

Philippe Gerum Copyright (C) 2008 Philippe Gerum <rpm@xenomai.org>.

7.40 kernel/cobalt/nucleus/sched-rt.c File Reference

Common real-time scheduling class implementation (FIFO + RR).

Include dependency graph for sched-rt.c:



7.40.1 Detailed Description

Common real-time scheduling class implementation (FIFO + RR).

Author

Philippe Gerum Copyright (C) 2008 Philippe Gerum <rpm@xenomai.org>.

Xenomai is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

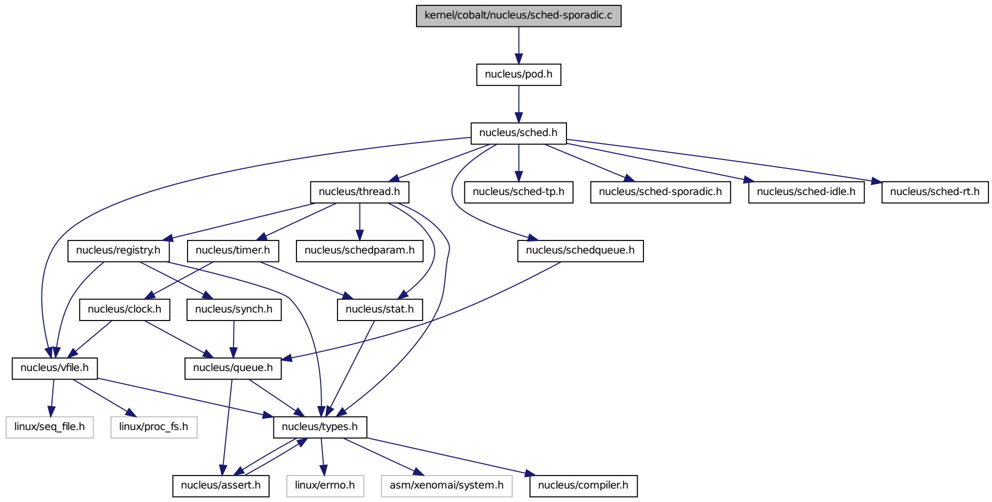
Xenomai is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with Xenomai; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

7.41 kernel/cobalt/nucleus/sched-sporadic.c File Reference

POSIX SCHED_SPORADIC scheduling class.

Include dependency graph for sched-sporadic.c:



7.41.1 Detailed Description

POSIX SCHED_SPORADIC scheduling class.

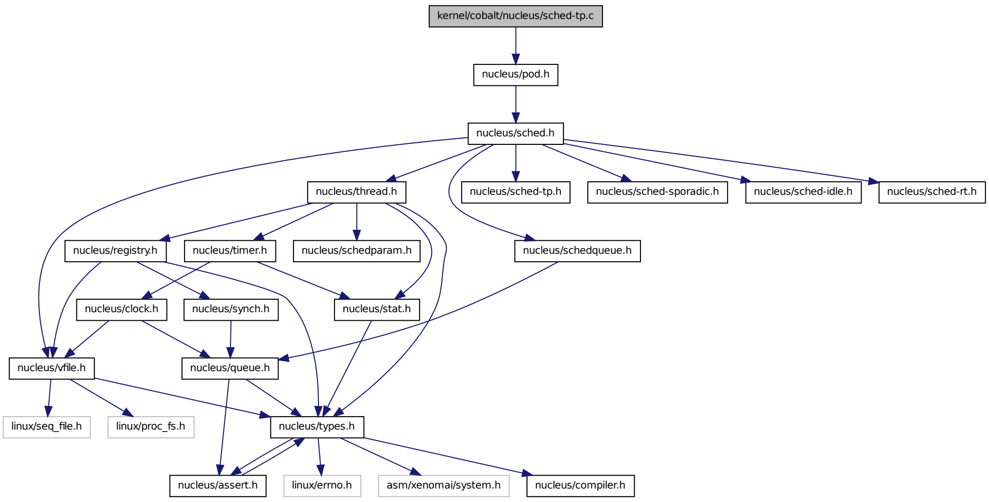
Author

Philippe Gerum Copyright (C) 2009 Philippe Gerum <rpm@xenomai.org>.

7.42 kernel/cobalt/nucleus/sched-tp.c File Reference

Temporal partitioning (typical of IMA systems).

Include dependency graph for sched-tp.c:



7.42.1 Detailed Description

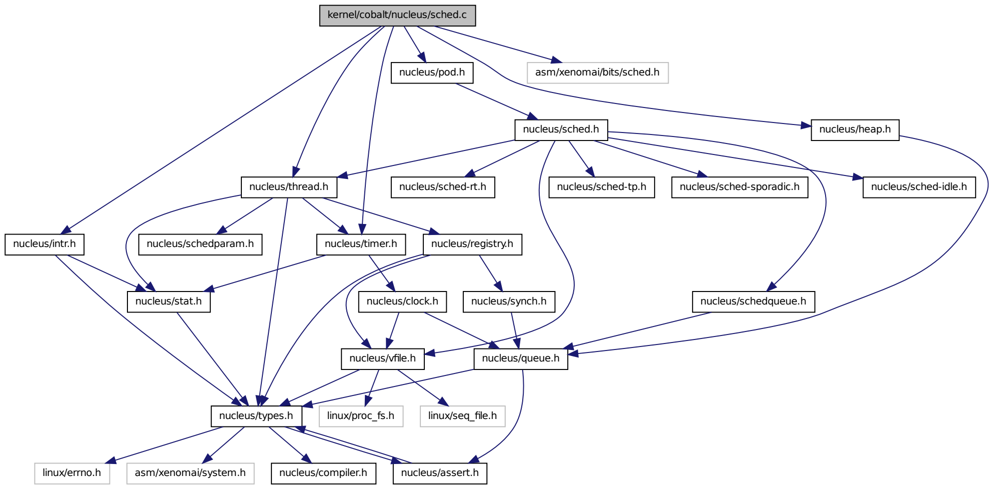
Temporal partitioning (typical of IMA systems).

Author

Philippe Gerum Copyright (C) 2008 Philippe Gerum <rpm@xenomai.org>.

7.43 kernel/cobalt/nucleus/sched.c File Reference

Include dependency graph for sched.c:



7.43.1 Detailed Description

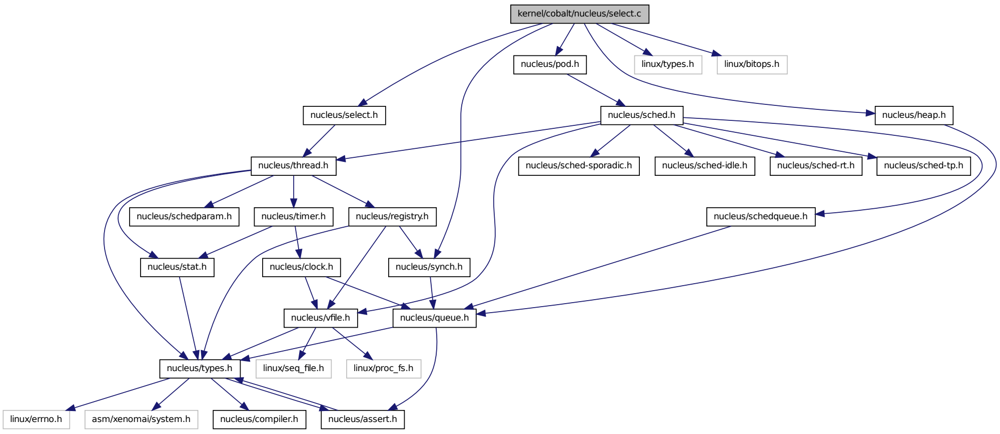
Author

Philippe Gerum

7.44 kernel/cobalt/nucleus/select.c File Reference

file descriptors events multiplexing.

Include dependency graph for select.c:



Functions

- void xnselect_init (struct xnselect *select_block)

  Initialize a struct xnselect structure.

- int xnselect_bind (struct xnselect *select_block, struct xnselect_binding *binding, struct xnselector *selector, unsigned type, unsigned index, unsigned state)

  Bind a file descriptor (represented by its xnselect structure) to a selector block.

- void xnselect_destroy (struct xnselect *select_block)

  Destroy the xnselect structure associated with a file descriptor.

- int xnselector_init (struct xnselector *selector)

  Initialize a selector structure.

- int xnselect (struct xnselector *selector, fd_set *out_fds[XNSELECT_MAX_TYPES], fd_set *in_fds[XNSELECT_MAX_TYPES], int nfds, xnticks_t timeout, xntmode_t timeout_mode)

  Check the state of a number of file descriptors, wait for a state change if no descriptor is ready.

- void xnselector_destroy (struct xnselector *selector)

  Destroy a selector block.

7.44.1 Detailed Description

file descriptors events multiplexing.

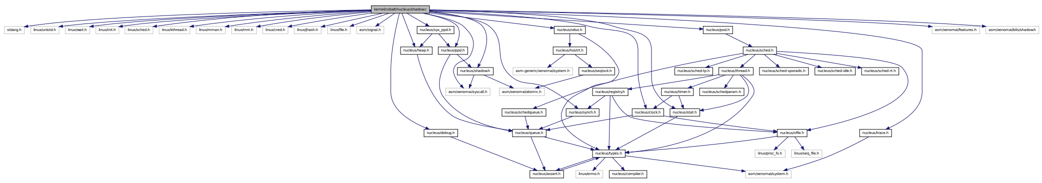
Author

Gilles Chanteperdrix

7.45 kernel/cobalt/nucleus/shadow.c File Reference

Real-time shadow services.

Include dependency graph for shadow.c:



Functions

- int xnshadow_harden (void)

  Migrate a Linux task to the Xenomai domain.

- void xnshadow_relax (int notify, int reason)

  Switch a shadow thread back to the Linux domain.

- int xnshadow_map (xnthread_t *thread, xncompletion_t __user *u_completion, unsigned long __user *u_window_offset)

  Create a shadow thread context.

- xnshadow_ppd_t * xnshadow_ppd_get (unsigned int muxid)

  Return the per-process data attached to the calling process.

7.45.1 Detailed Description

Real-time shadow services.

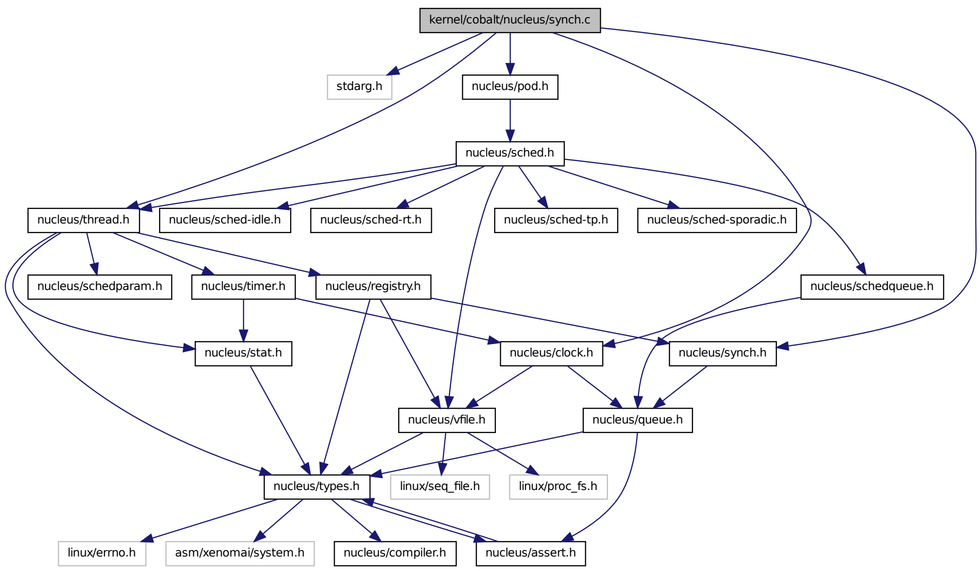
Author

Philippe Gerum

7.46 kernel/cobalt/nucleus/synch.c File Reference

Thread synchronization services.

Include dependency graph for synch.c:



Functions

- void xnsynch_init (struct xnsynch *synch, xnflags_t flags, xnarch_atomic_t *fastlock)

  Initialize a synchronization object.

- xnflags_t xnsynch_sleep_on (struct xnsynch *synch, xnticks_t timeout, xntmode_t timeout_-mode)

Sleep on an ownerless synchronization object.

- struct xnthread * xnsynch_wakeup_one_sleeper (struct xnsynch *synch)

  Give the resource ownership to the next waiting thread.

- struct xnpholder * xnsynch_wakeup_this_sleeper (struct xnsynch *synch, struct xnpholder *holder)

  Give the resource ownership to a given waiting thread.

- xnflags_t xnsynch_acquire (struct xnsynch *synch, xnticks_t timeout, xntmode_t timeout_mode)

  Acquire the ownership of a synchronization object.

- static void xnsynch_clear_boost (struct xnsynch *synch, struct xnthread *owner)

  Clear the priority boost.

- void xnsynch_requeue_sleeper (struct xnthread *thread)

  Change a sleeper's priority.

- struct xnthread * xnsynch_peek_pendq (struct xnsynch *synch)

  Access the thread leading a synch object wait queue.

- int xnsynch_flush (struct xnsynch *synch, xnflags_t reason)

  Unblock all waiters pending on a resource.

- void xnsynch_forget_sleeper (struct xnthread *thread)

  Abort a wait for a resource.

- void xnsynch_release_all_ownerships (struct xnthread *thread)

  Release all ownerships.

7.46.1 Detailed Description

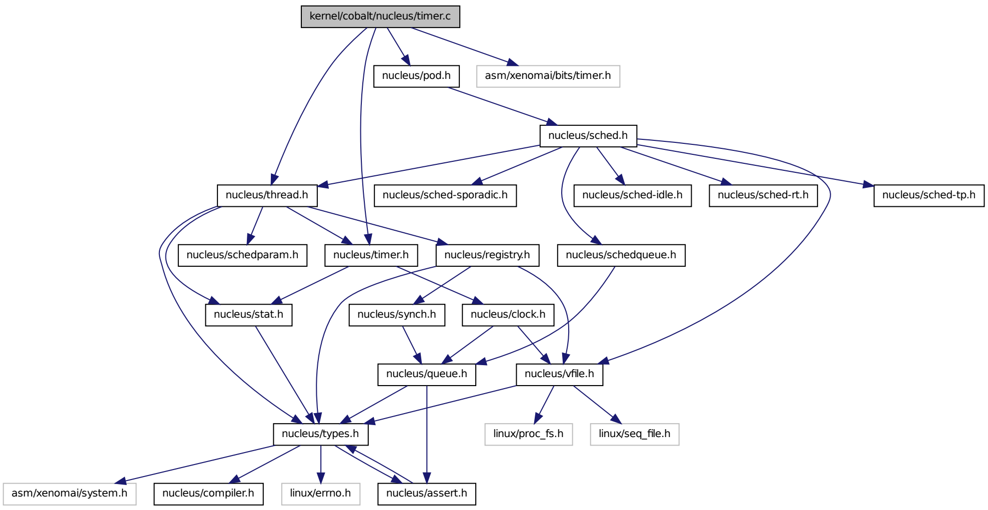
Thread synchronization services.

Author

Philippe Gerum

7.47 kernel/cobalt/nucleus/timer.c File Reference

Include dependency graph for timer.c:



Functions

- int xntimer_start (xntimer_t *timer, xnticks_t value, xnticks_t interval, xntmode_t mode)

  Arm a timer.
- xnticks_t xntimer_get_date (xntimer_t *timer)

  Return the absolute expiration date.
- xnticks_t xntimer_get_timeout (xntimer_t *timer)

  Return the relative expiration date.
- xnticks_t xntimer_get_interval (xntimer_t *timer)

  Return the timer interval value.
- void xntimer_tick (void)

  Process a timer tick.
- void xntimer_init (xntimer_t *timer, void(*handler)(xntimer_t *timer))

  Initialize a timer object.
- void xntimer_destroy (xntimer_t *timer)

  Release a timer object.
- unsigned long xntimer_get_overruns (xntimer_t *timer, xnticks_t now)

  Get the count of overruns for the last tick.

- void xntimer_freeze (void)

  Freeze all timers (from every time bases).

7.47.1 Detailed Description

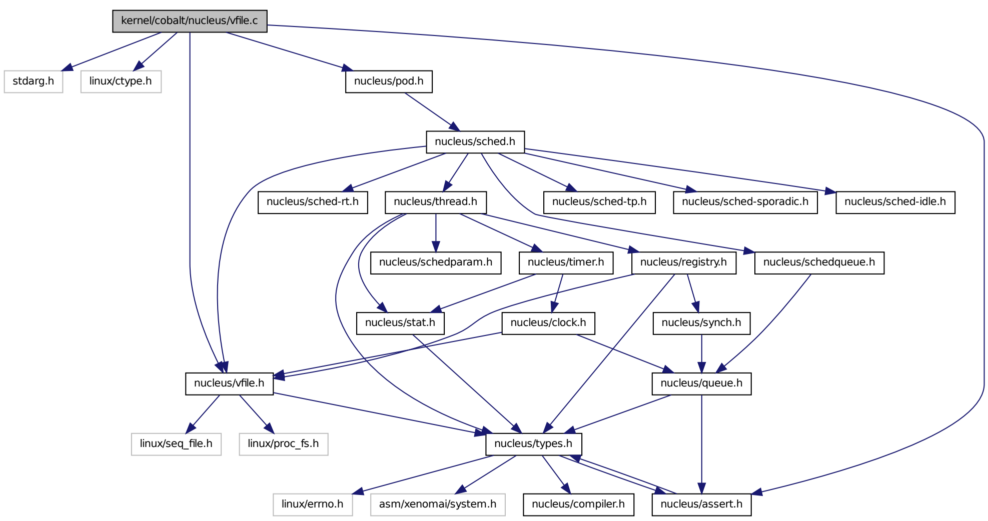
Note

Copyright (C) 2001,2002,2003,2007 Philippe Gerum <rpm@xenomai.org>. Copyright (C) 2004 Gilles Chanteperdrix <gilles.chanteperdrix@xenomai.org>

7.48 kernel/cobalt/nucleus/vfile.c File Reference

This file is part of the Xenomai project.

Include dependency graph for vfile.c:



Functions

- int xnvfile_init_snapshot (const char *name, struct xnvfile_snapshot *vfile, struct xnvfile_directory *parent)

  Initialize a snapshot-driven vfile.

- int xnvfile_init_regular (const char *name, struct xnvfile_regular *vfile, struct xnvfile_directory *parent)

  Initialize a regular vfile.

- int xnvfile_init_dir (const char *name, struct xnvfile_directory *vdir, struct xnvfile_directory *parent)

  Initialize a virtual directory entry.

- int xnvfile_init_link (const char *from, const char *to, struct xnvfile_link *vlink, struct xnvfile_directory *parent)

  Initialize a virtual link entry.

- void xnvfile_destroy (struct xnvfile *vfile)

  Removes a virtual file entry.

- ssize_t xnvfile_get_blob (struct xnvfile_input *input, void *data, size_t size)

  Read in a data bulk written to the vfile.

- ssize_t xnvfile_get_string (struct xnvfile_input *input, char *s, size_t maxlen)

  Read in a C-string written to the vfile.

- ssize_t xnvfile_get_integer (struct xnvfile_input *input, long *valp)

  Evaluate the string written to the vfile as a long integer.

Variables

- struct xnvfile_directory nkvfroot

  Xenomai vfile root directory.

7.48.1 Detailed Description

This file is part of the Xenomai project.

Note

Copyright (C) 2010 Philippe Gerum <rpm@xenomai.org>

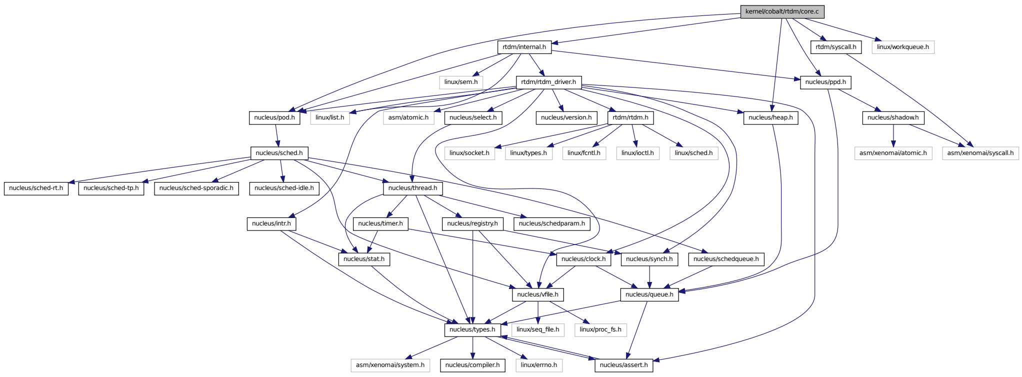
This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

7.49 kernel/cobalt/rtdm/core.c File Reference

Real-Time Driver Model for Xenomai, device operation multiplexing.

Include dependency graph for core.c:



Functions

- struct rtdm_dev_context * rtdm_context_get (int fd)

  Retrieve and lock a device context.

- int rtdm_select_bind (int fd, rtdm_selector_t *selector, enum rtdm_selecttype type, unsigned fd_index)

  Bind a selector to specified event types of a given file descriptor.

- void rtdm_context_lock (struct rtdm_dev_context *context)

  Increment context reference counter.

- void rtdm_context_unlock (struct rtdm_dev_context *context)

  Decrement context reference counter.

- void rtdm_context_put (struct rtdm_dev_context *context)

  Release a device context obtained via rtdm_context_get().

- int rtdm_open (const char *path, int oflag,...)

  Open a device.

- int rtdm_socket (int protocol_family, int socket_type, int protocol)

  Create a socket.

- int rtdm_close (int fd)

Close a device or socket.

- int rtdm_ioctl (int fd, int request,...)

  Issue an IOCTL.
- ssize_t rtdm_read (int fd, void *buf, size_t nbyte)

  Read from device.
- ssize_t rtdm_write (int fd, const void *buf, size_t nbyte)

  Write to device.
- ssize_t rtdm_recvmsg (int fd, struct msghdr *msg, int flags)

  Receive message from socket.
- ssize_t rtdm_recvfrom (int fd, void *buf, size_t len, int flags, struct sockaddr *from, socklen_t *fromlen)

  Receive message from socket.
- ssize_t rtdm_recv (int fd, void *buf, size_t len, int flags)

  Receive message from socket.
- ssize_t rtdm_sendmsg (int fd, const struct msghdr *msg, int flags)

  Transmit message to socket.
- ssize_t rtdm_sendto (int fd, const void *buf, size_t len, int flags, const struct sockaddr *to, socklen_t tolen)

  Transmit message to socket.
- ssize_t rtdm_send (int fd, const void *buf, size_t len, int flags)

  Transmit message to socket.
- int rtdm_bind (int fd, const struct sockaddr *my_addr, socklen_t addrlen)

  Bind to local address.
- int rtdm_connect (int fd, const struct sockaddr *serv_addr, socklen_t addrlen)

  Connect to remote address.
- int rtdm_listen (int fd, int backlog)

  Listen for incomming connection requests.
- int rtdm_accept (int fd, struct sockaddr *addr, socklen_t *addrlen)

  Accept a connection requests.
- int rtdm_shutdown (int fd, int how)

  Shut down parts of a connection.
- int rtdm_getsockopt (int fd, int level, int optname, void *optval, socklen_t *optlen)

  Get socket option.
- int rtdm_setsockopt (int fd, int level, int optname, const void *optval, socklen_t optlen)

Set socket option.

- int rtdm_getsockname (int fd, struct sockaddr *name, socklen_t *namelen)

  Get local socket address.

- int rtdm_getpeername (int fd, struct sockaddr *name, socklen_t *namelen)

  Get socket destination address.

- int rt_dev_open (const char *path, int oflag,...)

  Open a device.

- int rt_dev_socket (int protocol_family, int socket_type, int protocol)

  Create a socket.

- int rt_dev_close (int fd)

  Close a device or socket.

- int rt_dev_ioctl (int fd, int request,...)

  Issue an IOCTL.

- ssize_t rt_dev_read (int fd, void *buf, size_t nbyte)

  Read from device.

- ssize_t rt_dev_write (int fd, const void *buf, size_t nbyte)

  Write to device.

- ssize_t rt_dev_recvmsg (int fd, struct msghdr *msg, int flags)

  Receive message from socket.

- ssize_t rt_dev_recvfrom (int fd, void *buf, size_t len, int flags, struct sockaddr *from, socklen_t *fromlen)

  Receive message from socket.

- ssize_t rt_dev_recv (int fd, void *buf, size_t len, int flags)

  Receive message from socket.

- ssize_t rt_dev_sendmsg (int fd, const struct msghdr *msg, int flags)

  Transmit message to socket.

- ssize_t rt_dev_sendto (int fd, const void *buf, size_t len, int flags, const struct sockaddr *to, socklen_t tolen)

  Transmit message to socket.

- ssize_t rt_dev_send (int fd, const void *buf, size_t len, int flags)

  Transmit message to socket.

- int rt_dev_bind (int fd, const struct sockaddr *my_addr, socklen_t addrlen)

  Bind to local address.

- int rt_dev_connect (int fd, const struct sockaddr *serv_addr, socklen_t addrlen)

Connect to remote address.

- int rt_dev_listen (int fd, int backlog)

  Listen for incomming connection requests.

- int rt_dev_accept (int fd, struct sockaddr *addr, socklen_t *addrlen)

  Accept a connection requests.

- int rt_dev_shutdown (int fd, int how)

  Shut down parts of a connection.

- int rt_dev_getsockopt (int fd, int level, int optname, void *optval, socklen_t *optlen)

  Get socket option.

- int rt_dev_setsockopt (int fd, int level, int optname, const void *optval, socklen_t optlen)

  Set socket option.

- int rt_dev_getsockname (int fd, struct sockaddr *name, socklen_t *namelen)

  Get local socket address.

- int rt_dev_getpeername (int fd, struct sockaddr *name, socklen_t *namelen)

  Get socket destination address.

7.49.1 Detailed Description

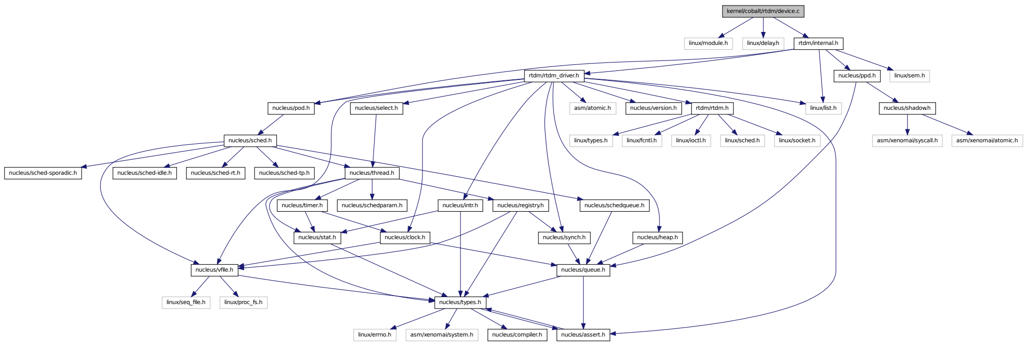
Real-Time Driver Model for Xenomai, device operation multiplexing.

Note

7.50 kernel/cobalt/rtdm/device.c File Reference

Real-Time Driver Model for Xenomai, device management.

Include dependency graph for device.c:



Functions

- int rtdm_dev_register (struct rtdm_device *device)

  Register a RTDM device.

- int rtdm_dev_unregister (struct rtdm_device *device, unsigned int poll_delay)

  Unregisters a RTDM device.

7.50.1 Detailed Description

Real-Time Driver Model for Xenomai, device management.

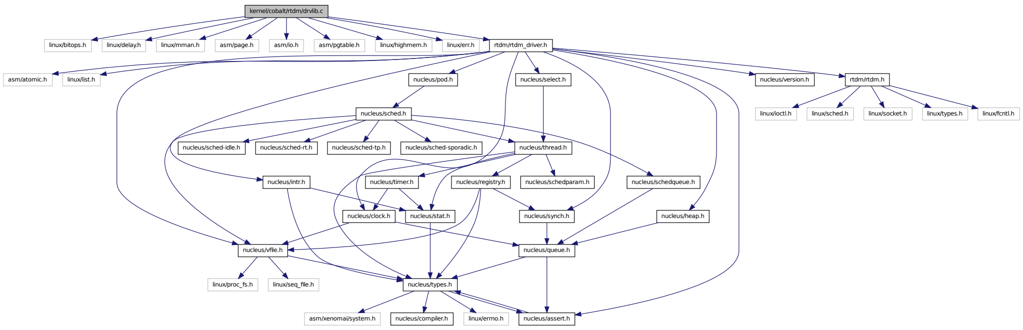
Note

Copyright (C) 2005 Jan Kiszka <jan.kiszka@web.de>
Copyright (C) 2005 Joerg Langenberg <joerg.langenberg@gmx.net>

7.51 kernel/cobalt/rtdm/drvlib.c File Reference

Real-Time Driver Model for Xenomai, driver library.

Include dependency graph for drvlib.c:



Functions

- nanosecs_abs_t rtdm_clock_read (void)

    Get system time.

- nanosecs_abs_t rtdm_clock_read_monotonic (void)

    Get monotonic time.

- int rtdm_task_init (rtdm_task_t *task, const char *name, rtdm_task_proc_t task_proc, void *arg, int priority, nanosecs_rel_t period)

    Intialise and start a real-time task.

- void rtdm_task_destroy (rtdm_task_t *task)

    Destroy a real-time task.

- void rtdm_task_set_priority (rtdm_task_t *task, int priority)

    Adjust real-time task priority.

- int rtdm_task_set_period (rtdm_task_t *task, nanosecs_rel_t period)

    Adjust real-time task period.

- int rtdm_task_wait_period (void)

    Wait on next real-time task period.

- int rtdm_task_unblock (rtdm_task_t *task)

    Activate a blocked real-time task.

- rtdm_task_t * rtdm_task_current (void)

    Get current real-time task.

- int rtdm_task_sleep (nanosecs_rel_t delay)

    Sleep a specified amount of time.

- int rtdm_task_sleep_until (nanosecs_abs_t wakeup_time)

    Sleep until a specified absolute time.

- int rtdm_task_sleep_abs (nanosecs_abs_t wakeup_time, enum rtdm_timer_mode mode)

  Sleep until a specified absolute time.

- void rtdm_task_join_nrt (rtdm_task_t *task, unsigned int poll_delay)

  Wait on a real-time task to terminate.

- void rtdm_task_busy_sleep (nanosecs_rel_t delay)

  Busy-wait a specified amount of time.

- int rtdm_timer_init (rtdm_timer_t *timer, rtdm_timer_handler_t handler, const char *name)

  Initialise a timer.

- void rtdm_timer_destroy (rtdm_timer_t *timer)

  Destroy a timer.

- int rtdm_timer_start (rtdm_timer_t *timer, nanosecs_abs_t expiry, nanosecs_rel_t interval, enum rtdm_timer_mode mode)

  Start a timer.

- void rtdm_timer_stop (rtdm_timer_t *timer)

  Stop a timer.

- int rtdm_timer_start_in_handler (rtdm_timer_t *timer, nanosecs_abs_t expiry, nanosecs_rel_t interval, enum rtdm_timer_mode mode)

  Start a timer from inside a timer handler.

- void rtdm_timer_stop_in_handler (rtdm_timer_t *timer)

  Stop a timer from inside a timer handler.

- int rtdm_irq_request (rtdm_irq_t *irq_handle, unsigned int irq_no, rtdm_irq_handler_t handler, unsigned long flags, const char *device_name, void *arg)

  Register an interrupt handler.

- int rtdm_irq_free (rtdm_irq_t *irq_handle)

  Release an interrupt handler.

- int rtdm_irq_enable (rtdm_irq_t *irq_handle)

  Enable interrupt line.

- int rtdm_irq_disable (rtdm_irq_t *irq_handle)

  Disable interrupt line.

- int rtdm_nrtsig_init (rtdm_nrtsig_t *nrt_sig, rtdm_nrtsig_handler_t handler, void *arg)

  Register a non-real-time signal handler.

- void rtdm_nrtsig_destroy (rtdm_nrtsig_t *nrt_sig)

  Release a non-realtime signal handler.

- void rtdm_nrtsig_pend (rtdm_nrtsig_t *nrt_sig)

  Trigger non-real-time signal.

- int rtdm_mmap_to_user (rtdm_user_info_t *user_info, void *src_addr, size_t len, int prot, void **pptr, struct vm_operations_struct *vm_ops, void *vm_private_data)

  Map a kernel memory range into the address space of the user.

- int rtdm_iomap_to_user (rtdm_user_info_t *user_info, phys_addr_t src_addr, size_t len, int prot, void **pptr, struct vm_operations_struct *vm_ops, void *vm_private_data)

  Map an I/O memory range into the address space of the user.

- int rtdm_munmap (rtdm_user_info_t *user_info, void *ptr, size_t len)

  Unmap a user memory range.

- void rtdm_printk (const char *format,...)

  Real-time safe message printing on kernel console.

- void * rtdm_malloc (size_t size)

  Allocate memory block in real-time context.

- void rtdm_free (void *ptr)

  Release real-time memory block.

- int rtdm_read_user_ok (rtdm_user_info_t *user_info, const void __user *ptr, size_t size)

  Check if read access to user-space memory block is safe.

- int rtdm_rw_user_ok (rtdm_user_info_t *user_info, const void __user *ptr, size_t size)

  Check if read/write access to user-space memory block is safe.

- int rtdm_copy_from_user (rtdm_user_info_t *user_info, void *dst, const void __user *src, size_t size)

  Copy user-space memory block to specified buffer.

- int rtdm_safe_copy_from_user (rtdm_user_info_t *user_info, void *dst, const void __user *src, size_t size)

  Check if read access to user-space memory block and copy it to specified buffer.

- int rtdm_copy_to_user (rtdm_user_info_t *user_info, void __user *dst, const void *src, size_t size)

  Copy specified buffer to user-space memory block.

- int rtdm_safe_copy_to_user (rtdm_user_info_t *user_info, void __user *dst, const void *src, size_t size)

  Check if read/write access to user-space memory block is safe and copy specified buffer to it.

- int rtdm_strncpy_from_user (rtdm_user_info_t *user_info, char *dst, const char __user *src, size_t count)

  Copy user-space string to specified buffer.

- int rtdm_in_rt_context (void)

Test if running in a real-time task.

- int rtdm_rt_capable (rtdm_user_info_t *user_info)
  Test if the caller is capable of running in real-time context.

Timeout Sequence Management

- void rtdm_toseq_init (rtdm_toseq_t *timeout_seq, nanosecs_rel_t timeout)
  Initialise a timeout sequence.

Event Services

- void rtdm_event_init (rtdm_event_t *event, unsigned long pending)
  Initialise an event.
- void rtdm_event_destroy (rtdm_event_t *event)
  Destroy an event.
- void rtdm_event_pulse (rtdm_event_t *event)
  Signal an event occurrence to currently listening waiters.
- void rtdm_event_signal (rtdm_event_t *event)
  Signal an event occurrence.
- int rtdm_event_wait (rtdm_event_t *event)
  Wait on event occurrence.
- int rtdm_event_timedwait (rtdm_event_t *event, nanosecs_rel_t timeout, rtdm_toseq_t *timeout_seq)
  Wait on event occurrence with timeout.
- void rtdm_event_clear (rtdm_event_t *event)
  Clear event state.
- int rtdm_event_select_bind (rtdm_event_t *event, rtdm_selector_t *selector, enum rtdm_selecttype type, unsigned fd_index)
  Bind a selector to an event.

Semaphore Services

- void rtdm_sem_init (rtdm_sem_t *sem, unsigned long value)
  Initialise a semaphore.
- void rtdm_sem_destroy (rtdm_sem_t *sem)
  Destroy a semaphore.
- int rtdm_sem_down (rtdm_sem_t *sem)
  Decrement a semaphore.
- int rtdm_sem_timeddown (rtdm_sem_t *sem, nanosecs_rel_t timeout, rtdm_toseq_t *timeout_seq)

Decrement a semaphore with timeout.

- void rtdm_sem_up (rtdm_sem_t *sem)
  Increment a semaphore.
- int rtdm_sem_select_bind (rtdm_sem_t *sem, rtdm_selector_t *selector, enum rtdm_-selecttype type, unsigned fd_index)
  Bind a selector to a semaphore.

Mutex Services

- void rtdm_mutex_init (rtdm_mutex_t *mutex)
  Initialise a mutex.
- void rtdm_mutex_destroy (rtdm_mutex_t *mutex)
  Destroy a mutex.
- void rtdm_mutex_unlock (rtdm_mutex_t *mutex)
  Release a mutex.
- int rtdm_mutex_lock (rtdm_mutex_t *mutex)
  Request a mutex.
- int rtdm_mutex_timedlock (rtdm_mutex_t *mutex, nanosecs_rel_t timeout, rtdm_toseq_t *timeout_seq)
  Request a mutex with timeout.

7.51.1 Detailed Description

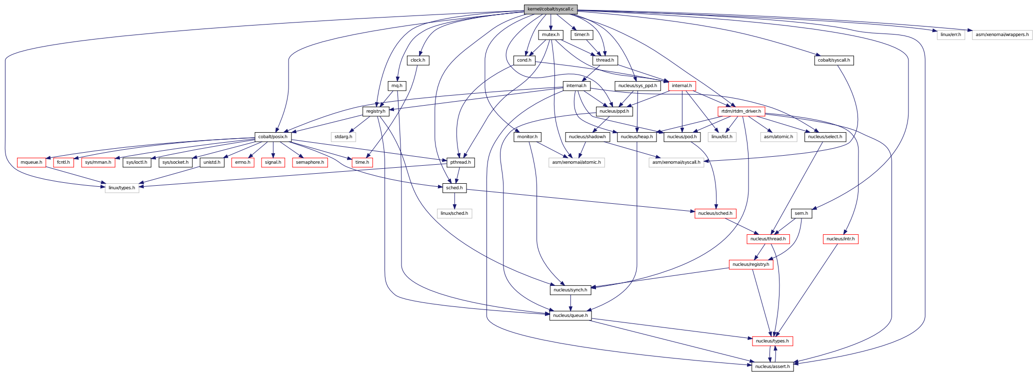
Real-Time Driver Model for Xenomai, driver library.

Note

7.52 kernel/cobalt/syscall.c File Reference

This file is part of the Xenomai project.

Include dependency graph for syscall.c:



7.52.1 Detailed Description

This file is part of the Xenomai project. Copyright (C) 2005 Philippe Gerum <rpm@xenomai.org> Copyright (C) 2005 Gilles Chanteperdrix <gilles.chanteperdrix@xenomai.org>

Chapter 8

Example Documentation

8.1 bufp-label.c

8.2 bufp-readwrite.c

8.3 cross-link.c

8.4 iddp-label.c

8.5 iddp-sendrecv.c

8.6 rtcan_rtt.c

8.7 rtcanconfig.c

8.8 rtcanrecv.c

8.9 rtcansend.c

8.10 xddp-echo.c

8.11 xddp-label.c

8.12 xddp-stream.c

Index