## Contents

1 Module Index .......................................................... 1
   1.1 Modules .......................................................... 1

2 File Index .................................................................. 3
   2.1 File List ............................................................ 3

3 Module Documentation ................................................. 5
   3.1 Thread cancellation. ............................................. 5
      3.1.1 Detailed Description ...................................... 5
      3.1.2 Function Documentation .................................. 6
         3.1.2.1 pthread_cancel ........................................ 6
         3.1.2.2 pthread_cleanup_pop ................................. 6
         3.1.2.3 pthread_cleanup_push ............................... 7
         3.1.2.4 pthread_setcancelstate ............................. 7
         3.1.2.5 pthread_setcanceltype ............................. 8
         3.1.2.6 pthread_testcancel ................................. 8
   3.2 Clocks and timers services. ..................................... 9
      3.2.1 Detailed Description ...................................... 9
      3.2.2 Function Documentation .................................. 10
         3.2.2.1 clock_getres ........................................... 10
         3.2.2.2 clock_gettime ......................................... 10
         3.2.2.3 clock_nanosleep ....................................... 11
         3.2.2.4 clock_settime ......................................... 11
         3.2.2.5 nanosleep ............................................... 12
         3.2.2.6 timer_create ........................................... 13
         3.2.2.7 timer_delete ........................................... 13
         3.2.2.8 timer_getoverrun ..................................... 14
         3.2.2.9 timer_gettime ......................................... 14
         3.2.2.10 timer_settime ........................................ 15
   3.3 Condition variables services. ................................. 16
      3.3.1 Detailed Description ...................................... 16
      3.3.2 Function Documentation .................................. 17
3.3.2.1  pthread_cond.Broadcast ........................................... 17
3.3.2.2  pthread_cond_destroy ........................................... 18
3.3.2.3  pthread_cond_init ............................................... 18
3.3.2.4  pthread_cond_signal ............................................ 19
3.3.2.5  pthread_cond_timedwait ........................................ 19
3.3.2.6  pthread_cond_wait .............................................. 20
3.3.2.7  pthread_condattr_destroy .................................... 21
3.3.2.8  pthread_condattr_getclock ................................... 21
3.3.2.9  pthread_condattr_getpshared ................................. 22
3.3.2.10 pthread_condattr_init ....................................... 22
3.3.2.11 pthread_condattr_setclock .................................. 23
3.3.2.12 pthread_condattr_setpshared ................................ 23

3.4 Interruptions management services. .................................. 25
  3.4.1 Detailed Description ............................................. 25
  3.4.2 Function Documentation ......................................... 25
    3.4.2.1  pthread_intr_attach_np .................................. 25
    3.4.2.2  pthread_intr_control_np ................................ 26
    3.4.2.3  pthread_intr_detach_np .................................. 27
    3.4.2.4  pthread_intr_wait_np .................................... 27

3.5 POSIX skin. ................................................................ 29
  3.5.1 Detailed Description ............................................. 29

3.6 Message queues services. ............................................. 30
  3.6.1 Detailed Description ............................................. 30
  3.6.2 Function Documentation ......................................... 30
    3.6.2.1  mq_close ....................................................... 30
    3.6.2.2  mq_getattr ................................................... 31
    3.6.2.3  mq_notify .................................................... 32
    3.6.2.4  mq_open ...................................................... 32
    3.6.2.5  mq_receive .................................................. 34
    3.6.2.6  mq_send ...................................................... 34
    3.6.2.7  mq_setattr ................................................... 35
    3.6.2.8  mq_timedreceive ........................................... 36
    3.6.2.9  mq_timedsend ............................................... 36
    3.6.2.10 mq_unlink ................................................... 37

3.7 Mutex services. ....................................................... 38
  3.7.1 Detailed Description ............................................. 38
  3.7.2 Function Documentation ......................................... 39
    3.7.2.1  pthread_mutex_destroy .................................. 39
    3.7.2.2  pthread_mutex_init ....................................... 39
    3.7.2.3  pthread_mutex_lock ....................................... 40
3.7.2.4 pthread_mutex_timedlock .............................................. 40
3.7.2.5 pthread_mutex_trylock ........................................... 41
3.7.2.6 pthread_mutex_unlock ............................................ 42
3.7.2.7 pthread_mutexattr_destroy ....................................... 42
3.7.2.8 pthread_mutexattr_getprotocol .................................. 43
3.7.2.9 pthread_mutexattr_getpshared .................................... 43
3.7.2.10 pthread_mutexattr_setprotocol ................................ 44
3.7.2.11 pthread_mutexattr_init ........................................... 44
3.7.2.12 pthread_mutexattr_setpshared .................................. 45
3.7.2.13 pthread_mutexattr_settype ...................................... 46
3.7.2.14 pthread_mutexattr_settype ...................................... 46

3.8 Threads scheduling services. .............................................. 47
3.8.1 Detailed Description .................................................. 47
3.8.2 Function Documentation .............................................. 48
  3.8.2.1 pthread_getschedparam .......................................... 48
  3.8.2.2 pthread_getschedparam_ex ...................................... 48
  3.8.2.3 pthread_setschedparam .......................................... 48
  3.8.2.4 pthread_setschedparam_ex ...................................... 49
  3.8.2.5 sched_get_priority_max .......................................... 50
  3.8.2.6 sched_get_priority_min .......................................... 50
  3.8.2.7 sched_rr_get_interval ........................................... 51
  3.8.2.8 sched_setconfig_np ............................................. 51
  3.8.2.9 sched_yield ..................................................... 52

3.9 Semaphores services. .................................................... 53
3.9.1 Detailed Description ................................................ 53
3.9.2 Function Documentation .............................................. 53
  3.9.2.1 sem_close ....................................................... 53
  3.9.2.2 sem_destroy .................................................... 54
  3.9.2.3 sem_getvalue ................................................... 54
  3.9.2.4 sem_init ....................................................... 55
  3.9.2.5 sem_open ....................................................... 55
  3.9.2.6 sem_post ....................................................... 56
  3.9.2.7 sem_timedwait .................................................. 57
  3.9.2.8 sem_trywait .................................................... 57
  3.9.2.9 sem_unlink ..................................................... 58
  3.9.2.10 sem_wait ....................................................... 58

3.10 Shared memory services. .............................................. 60
3.10.1 Detailed Description ................................................ 60
3.10.2 Function Documentation .............................................. 60
  3.10.2.1 close ........................................................ 60
3.10.2.2 ftruncate .................................................. 61
3.10.2.3 mmap ..................................................... 62
3.10.2.4 munmap .................................................... 63
3.10.2.5 shm_open .................................................. 63
3.10.2.6 shm_unlink ................................................. 65

3.11 Signals services .................................................. 66
3.11.1 Detailed Description ........................................... 66
3.11.2 Function Documentation ...................................... 67
3.11.2.1 pthread_kill .............................................. 67
3.11.2.2 pthread_sigmask ....................................... 67
3.11.2.3 pthread_sigqueue_np .................................. 68
3.11.2.4 sigaction .................................................. 68
3.11.2.5 sigaddset ................................................. 69
3.11.2.6 sigdelset .................................................. 70
3.11.2.7 sigemptyset .............................................. 70
3.11.2.8 sigfillset ................................................. 71
3.11.2.9 sigismember ............................................. 71
3.11.2.10 sigpending .............................................. 71
3.11.2.11 sigtimedwait .......................................... 72
3.11.2.12 sigwait .................................................. 72
3.11.2.13 sigwaitinfo ............................................. 73

3.12 Threads management services ..................................... 74
3.12.1 Detailed Description ........................................... 74
3.12.2 Enumeration Type Documentation ................................ 75
3.12.2.1 init_step ................................................. 75
3.12.3 Function Documentation ...................................... 75
3.12.3.1 pthread_create ........................................ 75
3.12.3.2 pthread_detach ........................................ 76
3.12.3.3 pthread_equal .......................................... 76
3.12.3.4 pthread_exit ............................................ 77
3.12.3.5 pthread_join ............................................. 77
3.12.3.6 pthread_make_periodic_np ............................. 78
3.12.3.7 pthread_self ............................................. 78
3.12.3.8 pthread_set_mode_np .................................. 78
3.12.3.9 pthread_set_name_np .................................. 79
3.12.3.10 pthread_wait_np ...................................... 80

3.13 Thread creation attributes ........................................ 81
3.13.1 Detailed Description ........................................... 82
3.13.2 Function Documentation ...................................... 82
3.13.2.1 pthread_attr_destroy .................................. 82
3.13.2.2 pthread_attr_getaffinity_np ........................................... 82
3.13.2.3 pthread_attr_getdetachstate ........................................... 83
3.13.2.4 pthread_attr_getfp_np .................................................. 83
3.13.2.5 pthread_attr_getinheritsched ......................................... 84
3.13.2.6 pthread_attr_getname_np .............................................. 84
3.13.2.7 pthread_attr_getschedparam .......................................... 85
3.13.2.8 pthread_attr_getschedparam_ex .................................... 85
3.13.2.9 pthread_attr_getschedpolicy ......................................... 86
3.13.2.10 pthread_attr_getscope ............................................... 86
3.13.2.11 pthread_attr_getstacksize ......................................... 87
3.13.2.12 pthread_attr_init ..................................................... 87
3.13.2.13 pthread_attr_setsaffinity_np ..................................... 88
3.13.2.14 pthread_attr_setdetachstate ...................................... 88
3.13.2.15 pthread_attr_setfp_np .............................................. 88
3.13.2.16 pthread_attr_setinheritsched .................................... 89
3.13.2.17 pthread_attr_setname_np .......................................... 89
3.13.2.18 pthread_attr_setschedparam ...................................... 90
3.13.2.19 pthread_attr_setschedparam_ex ................................ 90
3.13.2.20 pthread_attr_setschedpolicy .................................... 91
3.13.2.21 pthread_attr_setscope .............................................. 91
3.13.2.22 pthread_attr_setstacksize ....................................... 92

3.14 Thread-specific data. .......................................................... 93
  3.14.1 Detailed Description ....................................................... 93
  3.14.2 Function Documentation .................................................. 93
    3.14.2.1 pthread_getspecific ............................................. 93
    3.14.2.2 pthread_key_create ............................................ 94
    3.14.2.3 pthread_key_delete ............................................ 94
    3.14.2.4 pthread_setspecific ........................................... 94

4 File Documentation ................................................................. 97
  4.1 ksrc/skins/posix/syscall.c File Reference ................................ 97
    4.1.1 Detailed Description ................................................... 97

Index .......................................................... 98
Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

- POSIX skin.......................................................... 29
- Clocks and timers services................................. 9
- Condition variables services............................ 16
- Interruptions management services..................... 25
- Message queues services.................................. 30
- Mutex services...................................................... 38
- Semaphores services.......................................... 53
- Shared memory services..................................... 60
- Signals services.................................................. 66
- Threads management services............................ 74
  - Thread cancellation.......................................... 5
  - Threads scheduling services............................. 47
  - Thread creation attributes.............................. 81
- Thread-specific data........................................... 93
Chapter 2

File Index

2.1 File List

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

ksrc/skins/posix/apc.h .................................................................................. ??
ksrc/skins/posix/cancel.h ........................................................................... ??
ksrc/skins/posix/cb_lock.h ........................................................................... ??
ksrc/skins/posix/cond.h .............................................................................. ??
ksrc/skins/posix/internal.h ........................................................................... ??
ksrc/skins/posix/intr.h ................................................................................ ??
ksrc/skins/posix/mq.h .................................................................................. ??
ksrc/skins/posix/mutex.h ............................................................................ ??
ksrc/skins/posix/once.h .............................................................................. ??
ksrc/skins/posix/registry.h .......................................................................... ??
ksrc/skins/posix/sem.h ................................................................................ ??
ksrc/skins/posix/shm.h ................................................................................ ??
ksrc/skins/posix/sig.h .................................................................................. ??
ksrc/skins/posix/syscall.c .......................................................................... 97
ksrc/skins/posix/thread.h ............................................................................ ??
ksrc/skins/posix/timer.h ............................................................................. ??
ksrc/skins/posix/td.s ................................................................................... ??

This file is part of the Xenomai project.
Chapter 3

Module Documentation

3.1 Thread cancellation.

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.

- void pthread_testcancel (void)
  
  Test if a cancellation request is pending.

3.1.1 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.
3.1.2 Function Documentation

3.1.2.1 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.

3.1.2.2 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.
3.1.2.3  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 at 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

<table>
<thead>
<tr>
<th>routine</th>
<th>the cleanup routine to be registered;</th>
</tr>
</thead>
<tbody>
<tr>
<td>arg</td>
<td>the argument associated with this routine.</td>
</tr>
</tbody>
</table>

Valid contexts:

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

See Also

Specification.

3.1.2.4  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

<table>
<thead>
<tr>
<th>state</th>
<th>new cancelability state of the calling thread;</th>
</tr>
</thead>
<tbody>
<tr>
<td>oldstate_ptr</td>
<td>address where the old cancelability state will be stored on success.</td>
</tr>
</tbody>
</table>

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).
3.1.2.5 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_CANCELASYNCHRONOUS, meaning that cancellation requests are handled as soon as they are sent.

Parameters

<table>
<thead>
<tr>
<th>type</th>
<th>new cancelability type of the calling thread;</th>
</tr>
</thead>
<tbody>
<tr>
<td>oldtype_ptr</td>
<td>address where the old cancelability type will be stored on success.</td>
</tr>
</tbody>
</table>

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.

3.1.2.6 void pthread_testcancel ( void )

Test if a cancellation request is pending.

This function creates a cancellation point if the calling thread is a Xenomai POSIX skin thread (i.e. created with the pthread_create() service).

This function is a cancellation point. It has no effect if cancellation is disabled.

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

See Also
Specification.
3.2 Clocks and timers services.

Clocks and timers services.

Functions

- `int clock_getres (clockid_t clock_id, struct timespec *res)`
  - Get the resolution of the specified clock.
- `int clock_gettime (clockid_t clock_id, struct timespec *tp)`
  - Read the specified clock.
- `int clock_settime (clockid_t clock_id, const struct timespec *tp)`
  - Set the specified clock.
- `int clock_nanosleep (clockid_t clock_id, int flags, const struct timespec *rqtp, struct timespec *rmtp)`
  - Sleep some amount of time.
- `int nanosleep (const struct timespec *rqtp, struct timespec *rmtp)`
  - Sleep some amount of time.
- `int timer_create (clockid_t clockid, const struct sigevent *__restrict__ evp, timer_t *__restrict__ timerid)`
  - Create a timer object.
- `int timer_delete (timer_t timerid)`
  - Delete a timer object.
- `int timer_settime (timer_t timerid, int flags, const struct itimerspec *__restrict__ value, struct itimerspec *__restrict__ ovalue)`
  - Start or stop a timer.
- `int timer_gettime (timer_t timerid, struct itimerspec *__value)`
  - Get timer next expiration date and reload value.
- `int timer_getoverrun (timer_t timerid)`
  - Get expiration overruns count since the most recent timer expiration signal delivery.

3.2.1 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 system clock tick.

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 system clock tick, like the CLOCK_REALTIME clock.

Timer objects may be created with the `timer_create()` service using either of the two clocks, but the resolution of these timers is one system clock tick, as is the case for `clock_nanosleep()`.

Note

The duration of the POSIX clock tick depends on the active time base (configurable at compile-time with the constant `CONFIG_XENO_OPT_POSIX_PERIOD`, and at run-time with the `xenoposix` module parameter `tick_arg`). When the time base is aperiodic (which is the default) the system clock tick is one nanosecond.

See Also

- Specification.
3.2.2 Function Documentation

3.2.2.1 int clock_getres ( clockid_t clock_id, struct timespec * res )

Get the resolution of the specified clock.

This service returns, at the address res, if it is not NULL, the resolution of the clock clock_id.

For both CLOCK_REALTIME and CLOCK_MONOTONIC, this resolution is the duration of one system clock tick. No other clock is supported.

Parameters

<table>
<thead>
<tr>
<th>clock_id</th>
<th>clock identifier, either CLOCK_REALTIME or CLOCK_MONOTONIC;</th>
</tr>
</thead>
<tbody>
<tr>
<td>res</td>
<td>the address where the resolution of the specified clock will be stored on success.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>on success;</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>with errno set if:</td>
</tr>
<tr>
<td></td>
<td>- EINVAL, clock_id is invalid;</td>
</tr>
</tbody>
</table>

See Also

Specification.

3.2.2.2 int clock_gettime ( clockid_t clock_id, struct timespec * tp )

Read the specified clock.

This service returns, at the address tp the current value of the clock clock_id. If clock_id is:

- CLOCK_REALTIME, the clock value represents the amount of time since the Epoch, with a precision of one system clock tick;
- CLOCK_MONOTONIC, the clock value is given by an architecture-dependent high resolution counter, with a precision independent from the system clock tick duration.
- CLOCK_HOST_REALTIME, the clock value as seen by the host, typically Linux. Resolution and precision depend on the host, but it is guaranteed that both, host and Xenomai, see the same information.

Parameters

<table>
<thead>
<tr>
<th>clock_id</th>
<th>clock identifier, either CLOCK_REALTIME, CLOCK_MONOTONIC, or CLOCK_HOST_REALTIME;</th>
</tr>
</thead>
<tbody>
<tr>
<td>tp</td>
<td>the address where the value of the specified clock will be stored.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>on success;</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>with errno set if:</td>
</tr>
<tr>
<td></td>
<td>- EINVAL, clock_id is invalid.</td>
</tr>
</tbody>
</table>

See Also

Specification.
3.2.2.3 int clock_nanosleep ( clockid_t clock_id, int flags, const struct timespec *rqtp, struct timespec *rmtp )

Sleep some amount of time.

This service suspends the calling thread until the wakeup time specified by rqtp, or a signal is delivered to the caller. If the flag TIMER_ABSTIME is set in the flags argument, the wakeup time is specified as an absolute value of the clock clock_id. If the flag TIMER_ABSTIME is not set, the wakeup time is specified as a time interval.

If this service is interrupted by a signal, the flag TIMER_ABSTIME is not set, and rmtp is not NULL, the time remaining until the specified wakeup time is returned at the address rmtp.

The resolution of this service is one system clock tick.

Parameters

| clock_id | clock identifier, either CLOCK_REALTIME or CLOCK_MONOTONIC. |
| flags | one of: |
| | • 0 meaning that the wakeup time rqtp is a time interval; |
| | • TIMER_ABSTIME, meaning that the wakeup time is an absolute value of the clock clock_id. |
| rqtp | address of the wakeup time. |
| rmtp | address where the remaining time before wakeup will be stored if the service is interrupted by a signal. |

Returns

0 on success;
an error number if:
• EPERM, the caller context is invalid;
• ENOTSUP, the specified clock is unsupported;
• EINVAL, the specified wakeup time is invalid;
• EINTR, this service was interrupted by a signal.

Valid contexts:
• Xenomai kernel-space thread,
• Xenomai user-space thread (switches to primary mode).

See Also

Specification.

Referenced by nanosleep().
Module Documentation

Parameters

<table>
<thead>
<tr>
<th>clock_id</th>
<th>the id of the clock to be set, only CLOCK_REALTIME is supported.</th>
</tr>
</thead>
<tbody>
<tr>
<td>tp</td>
<td>the address of a struct timespec specifying the new date.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>on success;</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>with errno set if:</td>
</tr>
<tr>
<td></td>
<td>• EINVAL, clock_id is not CLOCK_REALTIME;</td>
</tr>
<tr>
<td></td>
<td>• EINVAL, the date specified by tp is invalid.</td>
</tr>
</tbody>
</table>

See Also

Specification.

3.2.2.5 int nanosleep ( const struct timespec *rqtp, struct timespec *rmtt )

Sleep some amount of time.

This service suspends the calling thread until the wakeup time specified by rqtp, or a signal is delivered. The wakeup time is specified as a time interval. 

If this service is interrupted by a signal and rmtt is not NULL, the time remaining until the specified wakeup time is returned at the address rmtt.

The resolution of this service is one system clock tick.

Parameters

<table>
<thead>
<tr>
<th>rqtp</th>
<th>address of the wakeup time.</th>
</tr>
</thead>
<tbody>
<tr>
<td>rmtt</td>
<td>address where the remaining time before wakeup will be stored if the service is interrupted by a signal.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>on success;</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>with errno set if:</td>
</tr>
<tr>
<td></td>
<td>• EPERM, the caller context is invalid;</td>
</tr>
<tr>
<td></td>
<td>• EINVAL, the specified wakeup time is invalid;</td>
</tr>
<tr>
<td></td>
<td>• EINTR, this service was interrupted by a signal.</td>
</tr>
</tbody>
</table>

Valid contexts:

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

See Also

Specification.

References clock_nanosleep().
### 3.2.2.6 int timer_create ( clockid_t clockid, const struct sigevent *__restrict__ evp, timer_t *__restrict__ timerid )

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>clockid</code></td>
<td>clock used as a timing base;</td>
</tr>
<tr>
<td><code>evp</code></td>
<td>description of the asynchronous notification to occur when the timer expires;</td>
</tr>
<tr>
<td><code>timerid</code></td>
<td>address where the identifier of the created timer will be stored on success.</td>
</tr>
</tbody>
</table>

#### Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>on success;</td>
</tr>
<tr>
<td>-1</td>
<td>with <code>errno</code> set if:</td>
</tr>
<tr>
<td></td>
<td>• EINVAL, the clock <code>clockid</code> is invalid;</td>
</tr>
<tr>
<td></td>
<td>• EINVAL, the member <code>sigev_notify</code> of the <code>sigevent</code> structure at the address <code>evp</code> is not <code>SIGEV_SIGNAL</code>;</td>
</tr>
<tr>
<td></td>
<td>• EINVAL, the member <code>sigev_signo</code> of the <code>sigevent</code> structure is an invalid signal number;</td>
</tr>
<tr>
<td></td>
<td>• EAGAIN, the maximum number of timers was exceeded, recompile with a larger value.</td>
</tr>
</tbody>
</table>

#### See Also

- [Specification](#).

### 3.2.2.7 int timer_delete ( timer_t timerid )

Delete a timer object.

This service deletes the timer `timerid`.

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>timerid</code></td>
<td>identifier of the timer to be removed;</td>
</tr>
</tbody>
</table>

#### Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>on success;</td>
</tr>
<tr>
<td>-1</td>
<td>with <code>errno</code> set if:</td>
</tr>
<tr>
<td></td>
<td>• EINVAL, <code>timerid</code> is invalid;</td>
</tr>
<tr>
<td></td>
<td>• EPERM, the timer <code>timerid</code> does not belong to the current process.</td>
</tr>
</tbody>
</table>
3.2.2.8 int timer_getoverrun ( timer_t timerid )

Get expiration overruns count since the most recent timer expiration signal delivery.

This service returns `timerid` expiration overruns count since the most recent timer expiration signal delivery. If this count is more than `DELAYTIMER_MAX` expirations, `DELAYTIMER_MAX` is returned.

Parameters

| timerid | Timer identifier. |

Returns

- the overruns count 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.

3.2.2.9 int timer_gettime ( timer_t timerid, struct itimerspec *value )

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.
3.2.2.10 int timer_settime ( timer_t timerid, int flags, const struct itimerspec *__restrict__ value, struct itimerspec *__restrict__ ovalue )

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 system clock ticks (see note in section Clocks and timers services for details on the duration of the system tick).

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.
3.3 Condition variables services.

Condition variables services.

Functions

- **int pthread_cond_init (pthread_cond_t *cnd, const pthread_condattr_t *attr)**
  
  Initialize a condition variable.

- **int pthread_cond_destroy (pthread_cond_t *cnd)**
  
  Destroy a condition variable.

- **int pthread_cond_wait (pthread_cond_t *cnd, pthread_mutex_t *mx)**
  
  Wait on a condition variable.

- **int pthread_cond_timedwait (pthread_cond_t *cnd, pthread_mutex_t *mx, const struct timespec *abstime)**
  
  Wait a bounded time on a condition variable.

- **int pthread_cond_signal (pthread_cond_t *cnd)**
  
  Signal a condition variable.

- **int pthread_cond_broadcast (pthread_cond_t *cnd)**
  
  Broadcast a condition variable.

- **int pthread_condattr_init (pthread_condattr_t *attr)**
  
  Initialize a condition variable attributes object.

- **int pthread_condattr_destroy (pthread_condattr_t *attr)**
  
  Destroy a condition variable attributes object.

- **int pthread_condattr_getclock (const pthread_condattr_t *attr, clockid_t *clk_id)**
  
  Get the clock selection attribute from a condition variable attributes object.

- **int pthread_condattr_setclock (pthread_condattr_t *attr, clockid_t clk_id)**
  
  Set the clock selection attribute of a condition variable attributes object.

- **int pthread_condattr_getpshared (const pthread_condattr_t *attr, int *pshared)**
  
  Get the process-shared attribute from a condition variable attributes object.

- **int pthread_condattr_setpshared (pthread_condattr_t *attr, int pshared)**
  
  Set the process-shared attribute of a condition variable attributes object.

3.3.1 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()`).
3.3.2 Function Documentation

3.3.2.1 int pthread_cond_broadcast ( pthread_cond_t * cnd )

Broadcast a condition variable.
This service unblocks all threads blocked on the condition variable cnd.
Parameters

| cnd  | the condition variable to be signalled. |

Returns

0 on success,
an error number if:

- EINVAL, the condition variable is invalid;
- EPERM, the condition variable is not process-shared and does not belong to the current process.

See Also

Specification.

3.3.2.2 int pthread_cond_destroy ( pthread_cond_t *cnd )

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 success,
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.

3.3.2.3 int pthread_cond_init ( pthread_cond_t *cnd, const pthread_condattr_t *attr )

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
3.3 Condition variables services.

### pthread_cond_init

<table>
<thead>
<tr>
<th>cnd</th>
<th>the condition variable to be initialized;</th>
</tr>
</thead>
<tbody>
<tr>
<td>attr</td>
<td>the condition variable attributes object.</td>
</tr>
</tbody>
</table>

**Returns**

- 0 on success,
- 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.

#### 3.3.2.4 int pthread_cond_signal ( pthread_cond_t * cnd )

**Signal a condition variable.**

This service unblocks one thread blocked on the condition variable `cnd`.

If more than one thread is blocked on the specified condition variable, the highest priority thread is unblocked.

**Parameters**

| cnd | the condition variable to be signalled. |

**Returns**

- 0 on success,
- an error number if:
  - EINVAL, the condition variable is invalid;
  - EPERM, the condition variable is not process-shared and does not belong to the current process.

**See Also**

- Specification.

#### 3.3.2.5 int pthread_cond_timedwait ( pthread_cond_t * cnd, pthread_mutex_t * mx, const struct timespec * abstime )

**Wait a bounded time on a condition variable.**

This service is equivalent to `pthread_cond_wait()`, except that the calling thread remains blocked on the condition variable `cnd` only until the timeout specified by `abstime` expires.

The timeout `abstime` is expressed as an absolute value of the `clock` attribute passed to `pthread_cond_init()`. By default, `CLOCK_REALTIME` is used.
3.3.2.6 int pthread_cond_wait ( pthread_cond_t * cnd, pthread_mutex_t * mx )

Wait on a condition variable.

This service atomically unlocks the mutex `mx`, and block the calling thread until the condition variable `cnd` is signalled using `pthread_cond_signal()` or `pthread_cond_broadcast()`. When the condition is signaled, this service re-acquire the mutex before returning.

Spurious wakeups occur if a signal is delivered to the blocked thread, so, an application should not assume that the condition changed upon successful return from this service.

Even if the mutex `mx` is recursive and its recursion count is greater than one on entry, it is unlocked before blocking the caller, and the recursion count is restored once the mutex is re-acquired by this service before returning.

Once a thread is blocked on a condition variable, a dynamic binding is formed between the condition variable `cnd` and the mutex `mx`; if another thread calls this service specifying `cnd` as a condition variable but another mutex than `mx`, this service returns immediately with the EINVAL status.

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 mutex `mx` is re-acquired before the cancellation cleanup handlers are called.
3.3 Condition variables services.

Returns

- 0 on success,
- an error number if:
  - EPERM, the caller context is invalid;
  - EINVAL, the specified condition variable or mutex is invalid;
  - EPERM, the specified condition variable is not process-shared and does not belong to the current process;
  - EINVAL, another thread is currently blocked on `cnd` using another mutex than `mx`;
  - EPERM, the specified mutex is not owned by the caller.

Valid contexts:

- Xenomai kernel-space thread;
- Xenomai user-space thread (switches to primary mode).

See Also

- Specification.

3.3.2.7 `int pthread_condattr_destroy ( pthread_condattr_t * attr )`

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.

3.3.2.8 `int pthread_condattr_getclock ( const pthread_condattr_t * attr, clockid_t * clk_id )`

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` or `CLOCK_MONOTONIC`.
3.3.2.9 int pthread_condattr_getpshared ( const pthread_condattr_t * attr, int * pshared )

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

This service stores, at the address \texttt{pshared}, the value of the \texttt{pshared} attribute in the condition variable attributes object \texttt{attr}.

The \texttt{pshared} attribute may only be one of \texttt{PTHREAD_PROCESS_PRIVATE} or \texttt{PTHREAD_PROCESS-SHARED}. See \texttt{pthread_condattr_setpshared()} for the meaning of these two constants.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{attr}</td>
<td>an initialized condition variable attributes object.</td>
</tr>
<tr>
<td>\texttt{pshared}</td>
<td>address where the value of the \texttt{pshared} attribute will be stored on success.</td>
</tr>
</tbody>
</table>

Returns

0 on success,
an error number if:

- EINVAL, the \texttt{pshared} address is invalid;
- EINVAL, the condition variable attributes object \texttt{attr} is invalid.

See Also

\texttt{Specification}.

3.3.2.10 int pthread_condattr_init ( pthread_condattr_t * attr )

Initialize a condition variable attributes object.

This services initializes the condition variable attributes object \texttt{attr} with default values for all attributes. Default value are:

- for the \texttt{clock} attribute, \texttt{CLOCK_REALTIME};
- for the \texttt{pshared} attribute \texttt{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`.

3.3.2.11 `int pthread_condattr_setclock ( pthread_condattr_t * attr, clockid_t clk_id )`

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` or `CLOCK_MONOTONIC`. |

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`.

3.3.2.12 `int pthread_condattr_setpshared ( pthread_condattr_t * attr, int pshared )`

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.
3.4 Interruptions management services.

Functions

- int pthread_intr_attach_np (pthread_intr_t *intrp, unsigned irq, xnisr_t isr, xniack_t iack)
  
  Create and attach an interrupt object.

- int pthread_intr_detach_np (pthread_intr_t intr)
  
  Destroy an interrupt object.

- int pthread_intr_control_np (pthread_intr_t intr, int cmd)
  
  Control the state of an interrupt channel.

- int pthread_intr_wait_np (pthread_intr_t intr, const struct timespec *to)
  
  Wait for the next interruption.

3.4.1 Detailed Description

Interruptions management services. The services described here allow applications written using the POSIX skin to handle interrupts, either in kernel-space or in user-space.

Note however, that it is recommended to use the standardized driver API of the RTDM skin (see rtdm).

3.4.2 Function Documentation

3.4.2.1 int pthread_intr_attach_np ( pthread_intr_t *intrp, unsigned irq, xnisr_t isr, xniack_t iack )

Create and attach an interrupt object.

This service creates and attaches an interrupt object.

In kernel-space:

This service installs *isr* as the handler for the interrupt *irq*. If *iack* is not null it is a custom interrupt acknowledge routine.

When called upon reception of an interrupt, the *isr* function is passed the address of an underlying xnintr_t object, and should use the macro PTHREAD_IDESC() to get the pthread_intr_t object. The meaning of the *isr* and *iack* function and what they should return is explained in xnintr_init() documentation.

This service is a non-portable extension of the POSIX interface.

<table>
<thead>
<tr>
<th>intrp</th>
<th>address where the created interrupt object identifier will be stored on success;</th>
</tr>
</thead>
<tbody>
<tr>
<td>irq</td>
<td>IRQ channel;</td>
</tr>
<tr>
<td>isr</td>
<td>interrupt handling routine;</td>
</tr>
<tr>
<td>iack</td>
<td>if not NULL, optional interrupt acknowledge routine.</td>
</tr>
</tbody>
</table>

In user-space:
The prototype of this service is:

```c
int pthread_intr_attach_np (pthread_intr_t *intrp, unsigned irq, int mode);
```

This service causes the installation of a default interrupt handler which unblocks any Xenomai user-space interrupt server thread blocked in a call to `pthread_intr_wait_np()`, and returns a value depending on the `mode` parameter.

Parameters:

- `intrp` and `irq` have the same meaning as in kernel-space; `mode` is a bitwise OR of the following values:
  - PTHREAD_INOAUTOENA, meaning that the interrupt should not be automatically re-enabled.
  - PTHREAD_IPROPAGATE, meaning that the interrupt should be propagated to lower priority domains. In effect, PTHREAD_IPROPAGATE implies PTHREAD_INOAUTOENA 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.

This service is intended to be used in conjunction with the `pthread_intr_wait_np()` service.

The return values are identical in kernel-space and user-space.

Return values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>on success; with <code>errno</code> set if:</td>
</tr>
<tr>
<td>-1</td>
<td>- ENOSYS, kernel-space Xenomai POSIX skin was built without support for interrupts, use RTDM or enable CONFIG_XENO_OPT_POSIX_INTR in kernel configuration;</td>
</tr>
<tr>
<td></td>
<td>- ENOMEM, insufficient memory exists in the system heap to create the interrupt object, increase CONFIG_XENO_OPT_SYS_HEAP_SIZE;</td>
</tr>
<tr>
<td></td>
<td>- EINVAL, a low-level error occurred while attaching the interrupt;</td>
</tr>
<tr>
<td></td>
<td>- EBUSY, an interrupt handler was already registered for the irq line <code>irq</code>.</td>
</tr>
</tbody>
</table>

References `pthread_intr_detach_np()`.

3.4.2.2 int pthread_intr_control_np ( pthread_intr_t intr, int cmd )

Control the state of an interrupt channel.

This service allows to enable or disable an interrupt channel.

This service is a non-portable extension of the POSIX interface.

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>intr</code></td>
<td>identifier of the interrupt to be enabled or disabled.</td>
</tr>
<tr>
<td><code>cmd</code></td>
<td>one of PTHREAD_IENABLE or PTHREAD_IDISABLE.</td>
</tr>
</tbody>
</table>

Return values
### 3.4 Interruptions management services

<table>
<thead>
<tr>
<th>0</th>
<th>on success;</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>with <code>errno</code> set if:</td>
</tr>
<tr>
<td></td>
<td>- ENOSYS, kernel-space Xenomai POSIX skin was built without support for interrupts, use RTDM or enable <code>CONFIG_XENO_OPT_POSIX_INTR</code> in kernel configuration;</td>
</tr>
<tr>
<td></td>
<td>- EINVAL, the identifier <code>intr</code> or <code>cmd</code> is invalid;</td>
</tr>
<tr>
<td></td>
<td>- EPERM, the interrupt <code>intr</code> does not belong to the current process.</td>
</tr>
</tbody>
</table>

#### 3.4.2.3 `int pthread_intr_detach_np ( pthread_intr_t intr )`

Destroy an interrupt object.

This service destroys the interrupt object `intr`. The memory allocated for this object is returned to the system heap, so further references using the same object identifier are not guaranteed to fail.

If a user-space interrupt server is blocked in a call to `pthread_intr_wait_np()`, it is unblocked and the blocking service returns with an error of EIDRM.

This service is a non-portable extension of the POSIX interface.

**Parameters**

| `intr` | identifier of the interrupt object to be destroyed. |

**Return values**

<table>
<thead>
<tr>
<th>0</th>
<th>on success;</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>with <code>errno</code> set if:</td>
</tr>
<tr>
<td></td>
<td>- ENOSYS, kernel-space Xenomai POSIX skin was built without support for interrupts, use RTDM or enable <code>CONFIG_XENO_OPT_POSIX_INTR</code> in kernel configuration;</td>
</tr>
<tr>
<td></td>
<td>- EINVAL, the interrupt object <code>intr</code> is invalid;</td>
</tr>
<tr>
<td></td>
<td>- EPERM, the interrupt <code>intr</code> does not belong to the current process.</td>
</tr>
</tbody>
</table>

Referenced by `pthread_intr_attach_np()`.

#### 3.4.2.4 `int pthread_intr_wait_np ( pthread_intr_t intr, const struct timespec *to )`

Wait for the next interruption.

This service is used by user-space interrupt server threads, to wait, if no interrupt is pending, for the next interrupt.

This service is a cancelation point. If a thread is canceled while blocked in a call to this service, no interruption notification is lost.

This service is a non-portable extension of the POSIX interface.

**Parameters**
Returns

the number of interrupt received on success;
-1 with *errno* set if:
  - ENOSYS, kernel-space Xenomai POSIX skin was built without support for interrupts, use RTDM or enable CONFIG_XENO_OPT_POSIX_INTR in kernel configuration;
  - EIDRM, the interrupt object was deleted;
  - EPERM, the interrupt *intr* does not belong to the current process;
  - ETIMEDOUT, the timeout specified by *to* expired;
  - EINTR, *pthread_intr_wait_np()* was interrupted by a signal.
3.5 POSIX skin.

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

Modules

- Clocks and timers services.
  - Clocks and timers services.
- Condition variables services.
  - Condition variables services.
- Interruptions management services.
  - Interruptions management services.
- Message queues services.
  - Message queues services.
- Mutex services.
  - Mutex services.
- Semaphores services.
  - Semaphores services.
- Shared memory services.
  - Shared memory services.
- Signals services.
  - Signals management services.
- Threads management services.
  - Threads management services.
- Thread-specific data.
  - Thread-specific data.

3.5.1 Detailed Description

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 native API services and POSIX services.

<table>
<thead>
<tr>
<th>Native API services</th>
<th>POSIX API services</th>
</tr>
</thead>
<tbody>
<tr>
<td>alarm</td>
<td>Clocks and timers services.</td>
</tr>
<tr>
<td>cond</td>
<td>Condition variables services.</td>
</tr>
<tr>
<td>event</td>
<td>no direct equivalence, see Condition variables services.</td>
</tr>
<tr>
<td>native_heap</td>
<td>Shared memory services.</td>
</tr>
<tr>
<td>interrupt</td>
<td>Interruptions management services.</td>
</tr>
<tr>
<td>mutex</td>
<td>Mutex services.</td>
</tr>
<tr>
<td>pipe</td>
<td>no direct equivalence, see Message queues services.</td>
</tr>
<tr>
<td>native_queue</td>
<td>Message queues services.</td>
</tr>
<tr>
<td>semaphore</td>
<td>Semaphores services.</td>
</tr>
<tr>
<td>task</td>
<td>Threads management services.</td>
</tr>
<tr>
<td>native_timer</td>
<td>Clocks and timers services.</td>
</tr>
</tbody>
</table>
3.6 Message queues services.

Message queues services.

Functions

- `mqd_t mq_open (const char *name, int oflags,...)`
  
  Open a message queue.

- `int mq_close (mqd_t fd)`

  Close a message queue.

- `int mq_unlink (const char *name)`

  Unlink a message queue.

- `int mq_send (mqd_t fd, const char *buffer, size_t len, unsigned prio)`

  Send a message to a message queue.

- `int mq_timedsend (mqd_t fd, const char *buffer, size_t len, unsigned prio, const struct timespec *abs_timeout)`

  Attempt, during a bounded time, to send a message to a message queue.

- `ssize_t mq_receive (mqd_t fd, char *buffer, size_t len, unsigned *priop)`

  Receive a message from a message queue.

- `ssize_t mq_timedreceive (mqd_t fd, char *restrict buffer, size_t len, unsigned *restrict priop, const struct timespec *restrict abs_timeout)`

  Attempt, during a bounded time, to receive a message from a message queue.

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

  Get the attributes object of a message queue.

- `int mq_setattr (mqd_t fd, const struct mq_attr *restrict attr, struct mq_attr *restrict oattr)`

  Set flags of a message queue.

- `int mq_notify (mqd_t fd, const struct sigevent *evp)`

  Register the current thread to be notified of message arrival at an empty message queue.

3.6.1 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()`.

3.6.2 Function Documentation

3.6.2.1 int `mq_close (mqd_t fd)`

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. |
3.6 Message queues services

Return values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>on success;</td>
</tr>
<tr>
<td>-1</td>
<td>with errno set if:</td>
</tr>
<tr>
<td></td>
<td>• EBADF, fd is an invalid message queue descriptor;</td>
</tr>
<tr>
<td></td>
<td>• EPERM, the caller context is invalid.</td>
</tr>
</tbody>
</table>

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.

3.6.2.2 int mq_getattr ( mqd_t fd, struct mq_attr * attr )

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fd</td>
<td>message queue descriptor;</td>
</tr>
<tr>
<td>attr</td>
<td>address where the message queue attributes will be stored on success.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>on success;</td>
</tr>
<tr>
<td>-1</td>
<td>with errno set if:</td>
</tr>
<tr>
<td></td>
<td>• EBADF, fd is not a valid descriptor.</td>
</tr>
</tbody>
</table>

See Also

Specification.
3.6.2.3  int mq_notify ( mqd_t fd, const struct sigevent * evp )

Register the current thread to be notified of message arrival at an empty message queue.

If `evp` is not `NULL` and is the address of a `sigevent` structure with the `sigev_notify` member set to `SIGEV_SIGNAL`, the current thread will be notified by a signal when a message is sent to the message queue `fd`, the queue is empty, and no thread is blocked in call to `mq_receive()` or `mq_timedreceive()`. After the notification, the thread is unregistered.

If `evp` is `NULL` or the `sigev_notify` member is `SIGEV_NONE`, the current thread is unregistered.

Only one thread may be registered at a time.

If the current thread is not a Xenomai POSIX skin thread (created with `pthread_create()`), this service fails.

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

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fd</code></td>
<td>message queue descriptor;</td>
</tr>
<tr>
<td><code>evp</code></td>
<td>pointer to an event notification structure.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>on success;</td>
</tr>
<tr>
<td>-1</td>
<td>with <code>errno</code> set if:</td>
</tr>
</tbody>
</table>

- EINVAL, `evp` is invalid;
- EPERM, the caller context is invalid;
- EBADF, `fd` is not a valid message queue descriptor;
- EBUSY, another thread is already registered.

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

See Also

Specification.

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

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 already 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

<table>
<thead>
<tr>
<th>name</th>
<th>name of the message queue to open;</th>
</tr>
</thead>
<tbody>
<tr>
<td>oflags</td>
<td>flags.</td>
</tr>
</tbody>
</table>

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](#).
3.6.2.5  ssize_t mq_receive ( mqd_t fd, char * buffer, size_t len, unsigned * priop )

Receive a message from a message queue.

If the message queue \texttt{fd} is not empty and if \texttt{len} is greater than the \texttt{mq_msgsize} of the message queue, this service copies, at the address \texttt{buffer}, the queued message with the highest priority.

If the queue is empty and the flag \texttt{O\_NONBLOCK} is not set for the descriptor \texttt{fd}, the calling thread is suspended until some message is sent to the queue. If the queue is empty and the flag \texttt{O\_NONBLOCK} is set for the descriptor \texttt{fd}, this service returns immediately a value of \texttt{-1} with \texttt{errno} set to \texttt{EAGAIN}.

Parameters

\begin{tabular}{|l|l|}
\hline
\texttt{fd} & the queue descriptor; \\
\texttt{buffer} & the address where the received message will be stored on success; \\
\texttt{len} & \texttt{buffer} length; \\
\texttt{priop} & address where the priority of the received message will be stored on success. \\
\hline
\end{tabular}

Returns

the message length, and copy a message at the address \texttt{buffer} on success;
-1 with no message unqueued and \texttt{errno} set if:
- \texttt{EBADF}, \texttt{fd} is not a valid descriptor open for reading;
- \texttt{EMSGSIZE}, the length \texttt{len} is lesser than the message queue \texttt{mq_msgsize} attribute;
- \texttt{EAGAIN}, the queue is empty, and the flag \texttt{O\_NONBLOCK} is set for the descriptor \texttt{fd};
- \texttt{EPERM}, the caller context is invalid;
- \texttt{EINTR}, the service was interrupted by a signal.

Valid contexts:

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

See Also

Specification.

3.6.2.6  int mq_send ( mqd_t fd, const char * buffer, size_t len, unsigned prio )

Send a message to a message queue.

If the message queue \texttt{fd} is not full, this service sends the message of length \texttt{len} pointed to by the argument \texttt{buffer}, with priority \texttt{prio}. A message with greater priority is inserted in the queue before a message with lower priority.

If the message queue is full and the flag \texttt{O\_NONBLOCK} is not set, the calling thread is suspended until the queue is not full. If the message queue is full and the flag \texttt{O\_NONBLOCK} is set, the message is not sent and the service returns immediately a value of \texttt{-1} with \texttt{errno} set to \texttt{EAGAIN}.

Parameters

\begin{tabular}{|l|l|}
\hline
\texttt{fd} & message queue descriptor; \\
\texttt{buffer} & pointer to the message to be sent; \\
\hline
\end{tabular}
Returns

0 and send a message on success;
-1 with no message sent and errno set if:

• EBADF, `fd` is not a valid message queue descriptor open for writing;
• EMSGSIZE, the message length `len` exceeds the `mq_msgsize` attribute of the message queue;
• EAGAIN, the flag O_NONBLOCK is set for the descriptor `fd` and the message queue is full;
• EPERM, the caller context is invalid;
• EINTR, the service was interrupted by a signal.

Valid contexts:

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

See Also

Specification.

3.6.2.7 int mq_setattr ( mqd_t fd, const struct mq_attr __restrict__ attr, struct mq_attr __restrict__ oattr )

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.
3.6.2.8 ssize_t mq_timedreceive ( mqd_t fd, char *__restrict__ buffer, size_t len, unsigned __restrict__ priop, const struct timespec *__restrict__ abs_timeout )

Attempt, during a bounded time, to receive a message from a message queue.

This service is equivalent to `mq_receive()`, except that if the flag `O_NONBLOCK` is not set for the descriptor `fd` and the message queue is empty, the calling thread is only suspended until the timeout `abs_timeout` expires.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fd</code></td>
<td>the queue descriptor;</td>
</tr>
<tr>
<td><code>buffer</code></td>
<td>the address where the received message will be stored on success;</td>
</tr>
<tr>
<td><code>len</code></td>
<td>buffer length;</td>
</tr>
<tr>
<td><code>priop</code></td>
<td>address where the priority of the received message will be stored on success.</td>
</tr>
<tr>
<td><code>abs_timeout</code></td>
<td>the timeout, expressed as an absolute value of the CLOCK_REALTIME clock.</td>
</tr>
</tbody>
</table>

Returns

the message length, and copy a message at the address `buffer` on success;
-1 with no message unqueued and `errno` set if:

- EBADF, `fd` is not a valid descriptor open for reading;
- EMSGSIZE, the length `len` is lesser than the message queue `mq_msgsize` attribute;
- EAGAIN, the queue is empty, and the flag `O_NONBLOCK` is set for the descriptor `fd`;
- EPERM, the caller context is invalid;
- EINTR, the service was interrupted by a signal;
- ETIMEDOUT, the specified timeout expired.

Valid contexts:

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

See Also

`Specification`.

3.6.2.9 int mq_timedsend ( mqd_t fd, const char *buffer, size_t len, unsigned prio, const struct timespec *abs_timeout )

Attempt, during a bounded time, to send a message to a message queue.

This service is equivalent to `mq_send()`, except that if the message queue is full and the flag `O_NONBLOCK` is not set for the descriptor `fd`, the calling thread is only suspended until the timeout specified by `abs_timeout` expires.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fd</code></td>
<td>message queue descriptor;</td>
</tr>
<tr>
<td><code>buffer</code></td>
<td>pointer to the message to be sent;</td>
</tr>
<tr>
<td><code>len</code></td>
<td>length of the message;</td>
</tr>
<tr>
<td><code>prio</code></td>
<td>priority of the message;</td>
</tr>
</tbody>
</table>
3.6.2.10 int mq_unlink ( const char * name )

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

<table>
<thead>
<tr>
<th>0</th>
<th>on success;</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>with errno set if:</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• EPERM, the caller context is invalid;</td>
</tr>
<tr>
<td></td>
<td>• ENAMETOOLONG, the length of the name argument exceeds 64 characters;</td>
</tr>
<tr>
<td></td>
<td>• ENOENT, the message queue does not exist.</td>
</tr>
</tbody>
</table>

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.
3.7 Mutex services.

Mutex services.

Functions

- int pthread_mutex_init (pthread_mutex_t *mx, const pthread_mutexattr_t *attr)
  Initialize a mutex.
- int pthread_mutex_destroy (pthread_mutex_t *mx)
  Destroy a mutex.
- int pthread_mutex_trylock (pthread_mutex_t *mx)
  Attempt to lock a mutex.
- int pthread_mutex_lock (pthread_mutex_t *mx)
  Lock a mutex.
- int pthread_mutex_timedlock (pthread_mutex_t *mx, const struct timespec *to)
  Attempt, during a bounded time, to lock a mutex.
- int pthread_mutex_unlock (pthread_mutex_t *mx)
  Unlock a mutex.
- int pthread_mutexattr_init (pthread_mutexattr_t *attr)
  Initialize a mutex attributes object.
- int pthread_mutexattr_destroy (pthread_mutexattr_t *attr)
  Destroy a mutex attributes object.
- int pthread_mutexattr_gettype (const pthread_mutexattr_t *attr, int *type)
  Get the mutex type attribute from a mutex attributes object.
- int pthread_mutexattr_settype (pthread_mutexattr_t *attr, int type)
  Set the mutex type attribute of a mutex attributes object.
- int pthread_mutexattr_getprotocol (const pthread_mutexattr_t *attr, int *proto)
  Get the protocol attribute from a mutex attributes object.
- int pthread_mutexattr_setprotocol (pthread_mutexattr_t *attr, int proto)
  Set the protocol attribute of a mutex attributes object.
- int pthread_mutexattr_getpshared (const pthread_mutexattr_t *attr, int *pshared)
  Get the process-shared attribute of a mutex attributes object.
- int pthread_mutexattr_setpshared (pthread_mutexattr_t *attr, int pshared)
  Set the process-shared attribute of a mutex attributes object.

3.7.1 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.
3.7.2 Function Documentation

3.7.2.1 int pthread_mutex_destroy ( pthread_mutex_t * mx )

Destroy a mutex.
This service destroys the mutex mx, if it is unlocked and not referenced by any condition variable. The mutex becomes invalid for all mutex services (they all return the EINVAL error) except pthread_mutex_init().

Parameters

| mx | the mutex to be destroyed. |

Returns

0 on success,
an error number if:
- EINVAL, the mutex mx is invalid;
- EPERM, the mutex is not process-shared and does not belong to the current process;
- EBUSY, the mutex is locked, or used by a condition variable.

See Also

Specification.

3.7.2.2 int pthread_mutex_init ( pthread_mutex_t * mx, const pthread_mutexattr_t * attr )

Initialize a mutex.
This services initializes the mutex mx, using the mutex attributes object attr. If attr is NULL, default attributes are used (see pthread_mutexattr_init()).

Parameters

| mx | the mutex to be initialized; |
| attr | the mutex attributes object. |

Returns

0 on success,
an error number if:
- EINVAL, the mutex attributes object attr is invalid or uninitialized;
- EBUSY, the mutex mx was already initialized;
- ENOMEM, insufficient memory exists in the system heap to initialize the mutex, increase CONFIG_XENO_OPT_SYS_HEAPSZ.
- EAGAIN, insufficient memory exists in the semaphore heap to initialize the mutex, increase CONFIG_XENO_OPT_GLOBAL_SEM_HEAPSZ for a process-shared mutex, or CONFIG_XENO_OPT_SEM_HEAPSZ for a process-private mutex.

See Also

Specification.
3.7.2.3 int pthread_mutex_lock ( pthread_mutex_t *mx )

Lock a mutex.

This service attempts to lock the mutex `mx`. If the mutex is free, it becomes locked. If it was locked by another thread than the current one, the current thread is suspended until the mutex is unlocked. If it was already locked by the current mutex, the behaviour of this service depends on the mutex type:

- for mutexes of the `PTHREAD_MUTEX_NORMAL` type, this service deadlocks;
- for mutexes of the `PTHREAD_MUTEX_ERRORCHECK` type, this service returns the EDEADLK error number;
- for mutexes of the `PTHREAD_MUTEX_RECURSIVE` type, this service increments the lock recursion count and returns 0.

Parameters

| mx   | the mutex to be locked. |

Returns

0 on success
an error number if:

- EPERM, the caller context is invalid;
- EINVAL, the mutex `mx` is invalid;
- EPERM, the mutex is not process-shared and does not belong to the current process;
- EDEADLK, the mutex is of the `PTHREAD_MUTEX_ERRORCHECK` type and was already locked by the current thread;
- EAGAIN, the mutex is of the `PTHREAD_MUTEX_RECURSIVE` type and the maximum number of recursive locks has been exceeded.

Valid contexts:

- Xenomai kernel-space thread;
- Xenomai user-space thread (switches to primary mode).

See Also

`Specification`.

3.7.2.4 int pthread_mutex_timedlock ( pthread_mutex_t *mx, const struct timespec *to )

Attempt, during a bounded time, to lock a mutex.

This service is equivalent to `pthread_mutex_lock()`, except that if the mutex `mx` is locked by another thread than the current one, this service only suspends the current thread until the timeout specified by `to` expires.

Parameters

<table>
<thead>
<tr>
<th>mx</th>
<th>the mutex to be locked;</th>
</tr>
</thead>
<tbody>
<tr>
<td>to</td>
<td>the timeout, expressed as an absolute value of the CLOCK_REALTIME clock.</td>
</tr>
</tbody>
</table>
Returns

0 on success;

an error number if:

- EPERM, the caller context is invalid;
- EINVAL, the mutex \( mx \) is invalid;
- EPERM, the mutex is not process-shared and does not belong to the current process;
- ETIMEDOUT, the mutex could not be locked and the specified timeout expired;
- EDEADLK, the mutex is of the \texttt{PTHREAD_MUTEX_ERRORCHECK} type and the mutex was already locked by the current thread;
- EAGAIN, the mutex is of the \texttt{PTHREAD_MUTEX_RECURSIVE} type and the maximum number of recursive locks has been exceeded.

Valid contexts:

- Xenomai kernel-space thread;
- Xenomai user-space thread (switches to primary mode).

See Also

- Specification.

3.7.2.5 \texttt{int pthread_mutex_trylock ( pthread_mutex_t * mx )}

Attempt to lock a mutex.

This service is equivalent to \texttt{pthread_mutex_lock()}, except that if the mutex \( mx \) is locked by another thread than the current one, this service returns immediately.

Parameters

| \( mx \) | the mutex to be locked. |

Returns

0 on success;

an error number if:

- EPERM, the caller context is invalid;
- EINVAL, the mutex is invalid;
- EPERM, the mutex is not process-shared and does not belong to the current process;
- EBUSY, the mutex was locked by another thread than the current one;
- EAGAIN, the mutex is recursive, and the maximum number of recursive locks has been exceeded.

Valid contexts:

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

See Also

- Specification.
3.7.2.6  int pthread_mutex_unlock ( pthread_mutex_t * mx )

Unlock a mutex.
This service unlocks the mutex mx. If the mutex is of the PTHREAD_MUTEX_RECURSIVE type and
the locking recursion count is greater than one, the lock recursion count is decremented and the mutex
remains locked.
Attempting to unlock a mutex which is not locked or which is locked by another thread than the current
one yields the EPERM error, whatever the mutex type attribute.
Parameters

| mx | the mutex to be released. |

Returns

0 on success;
an error number if:

- EPERM, the caller context is invalid;
- EINVAL, the mutex mx is invalid;
- EPERM, the mutex was not locked by the current thread.

Valid contexts:

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

See Also

Specification.

3.7.2.7  int pthread_mutexattr_destroy ( pthread_mutexattr_t * attr )

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.
3.7.2.8 int pthread_mutexattr_getprotocol ( const pthread_mutexattr_t * attr, int * proto )

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 protocol 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

<table>
<thead>
<tr>
<th>attr</th>
<th>an initialized mutex attributes object;</th>
</tr>
</thead>
<tbody>
<tr>
<td>proto</td>
<td>address where the value of the protocol attribute will be stored on success.</td>
</tr>
</tbody>
</table>

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.

3.7.2.9 int pthread_mutexattr_getpshared ( const pthread_mutexattr_t * attr, int * pshared )

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 pshared 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

<table>
<thead>
<tr>
<th>attr</th>
<th>an initialized mutex attributes object;</th>
</tr>
</thead>
<tbody>
<tr>
<td>pshared</td>
<td>address where the value of the pshared attribute will be stored on success.</td>
</tr>
</tbody>
</table>

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.

3.7.2.10 int pthread_mutexattr_gettype ( const pthread_mutexattr_t * attr, int * type )

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.
Module Documentation

Parameters

<table>
<thead>
<tr>
<th>attr</th>
<th>an initialized mutex attributes object,</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>address where the type attribute value will be stored on success.</td>
</tr>
</tbody>
</table>

Returns

0 on success,
an error number if:

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

See Also

Specification.

3.7.2.11 int pthread_mutexattr_init ( pthread_mutexattr_t *attr )

Initialize a mutex attributes object.
This service 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.

3.7.2.12 int pthread_mutexattr_setprotocol ( pthread_mutexattr_t *attr, int proto )

Set the protocol attribute of a mutex attributes object.
This service set the type attribute of the mutex attributes object attr.
3.7 Mutex services

Parameters

<table>
<thead>
<tr>
<th>attr</th>
<th>an initialized mutex attributes object,</th>
</tr>
</thead>
<tbody>
<tr>
<td>proto</td>
<td>value of the protocol attribute, may be one of:</td>
</tr>
<tr>
<td></td>
<td>• PTHREAD_PRIO_NONE, meaning that a mutex created with the attributes object attr will not follow any priority protocol;</td>
</tr>
<tr>
<td></td>
<td>• PTHREAD_PRIO_INHERIT, meaning that a mutex created with the attributes object attr, will follow the priority inheritance protocol.</td>
</tr>
</tbody>
</table>

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.

3.7.2.13 int pthread_mutexattr_setpshared ( pthread_mutexattr_t * attr, int pshared )

Set the process-shared attribute of a mutex attributes object.
This service set the pshared attribute of the mutex attributes object attr.

Parameters

<table>
<thead>
<tr>
<th>attr</th>
<th>an initialized mutex attributes object.</th>
</tr>
</thead>
<tbody>
<tr>
<td>pshared</td>
<td>value of the pshared attribute, may be one of:</td>
</tr>
<tr>
<td></td>
<td>• 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;</td>
</tr>
<tr>
<td></td>
<td>• 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.</td>
</tr>
</tbody>
</table>

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.
3.7.2.14  int pthread_mutexattr_settype ( pthread_mutexattr_t * attr, int type )

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.
3.8 Threads scheduling services.

Thread scheduling services.

Functions

- **int sched_get_priority_min**(int policy)
  
  Get minimum priority of the specified scheduling policy.

- **int sched_get_priority_max**(int policy)
  
  Get maximum priority of the specified scheduling policy.

- **int sched_rr_get_interval**(int pid, struct timespec *interval)
  
  Get the round-robin scheduling time slice.

- **int pthread_getschedparam**(pthread_t tid, int *pol, struct sched_param *par)
  
  Get the scheduling policy and parameters of the specified thread.

- **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.

- **int pthread_setschedparam**(pthread_t tid, int pol, const struct sched_param *par)
  
  Set the scheduling policy and parameters of the specified thread.

- **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.

- **int sched_yield**(void)
  
  Yield the processor.

- **int sched_setconfig_np**(int cpu, int policy, union sched_config *config, size_t len)
  
  Load CPU-specific scheduler settings for a given policy.

3.8.1 Detailed Description

Thread scheduling services. Xenomai POSIX skin supports the scheduling policies SCHED_FIFO, SCHED_RR, SCHED_SPORADIC, SCHED_TP and SCHED_OTHER.

The SCHED_OTHER policy is mainly useful for user-space non-realtime activities that need to synchronize with real-time activities.

The SCHED_RR policy is only effective if the time base is periodic (i.e. if configured with the compilation constant `CONFIG_XENO_OPT_POSIX_PERIOD` or the `xeno_nucleus` module parameter `tick_arg` set to a non null value). The SCHED_RR round-robin time slice is configured with the `xeno_posix` module parameter `time_slice`, as a count of system timer clock ticks.

The SCHED_SPORADIC policy provides a mean to schedule aperiodic or sporadic threads in periodic-based systems.

The SCHED_TP policy divides the scheduling time into a recurring global frame, which is itself divided into an arbitrary number of time partitions. Only threads assigned to the current partition are deemed runnable, and scheduled according to a FIFO-based rule within this partition. When completed, the current partition is advanced automatically to the next one by the scheduler, and the global time frame recurs from the first partition defined, when the last partition has ended.

The scheduling policy and priority of a thread is set when creating a thread, by using thread creation attributes (see `pthread_attr_setinheritsched()`, `pthread_attr_setschedpolicy()` and `pthread_attr_setschedparam()`), or when the thread is already running by using the service `pthread_setschedparam()`.

See Also

  Specification.
3.8.2 Function Documentation

3.8.2.1 int pthread_getschedparam ( pthread_t tid, int *pol, struct sched_param *par )

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

<table>
<thead>
<tr>
<th>tid</th>
<th>target thread;</th>
</tr>
</thead>
<tbody>
<tr>
<td>pol</td>
<td>address where the scheduling policy of tid is stored on success;</td>
</tr>
<tr>
<td>par</td>
<td>address where the scheduling parameters of tid is stored on success.</td>
</tr>
</tbody>
</table>

Returns

0 on success;
an error number if:
- ESRCH, tid is invalid.

See Also

Specification.

3.8.2.2 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.

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 or SCHED_TP parameters can be retrieved from this call.

Parameters

<table>
<thead>
<tr>
<th>tid</th>
<th>target thread;</th>
</tr>
</thead>
<tbody>
<tr>
<td>pol</td>
<td>address where the scheduling policy of tid is stored on success;</td>
</tr>
<tr>
<td>par</td>
<td>address where the scheduling parameters of tid is stored on success.</td>
</tr>
</tbody>
</table>

Returns

0 on success;
an error number if:
- ESRCH, tid is invalid.

See Also

Specification.

Referenced by pthread_create().

3.8.2.3 int pthread_setschedparam ( pthread_t tid, int pol, const struct sched_param *par )

Set the scheduling policy and parameters of the specified thread.
This service set the scheduling policy of the Xenomai POSIX skin thread \textit{tid} to the value \textit{pol}, and its scheduling parameters (i.e. its priority) to the value pointed to by \textit{par}.

When used in user-space, passing the current thread ID as \textit{tid} argument, this service turns the current thread into a Xenomai POSIX skin thread. If \textit{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 \texttt{pthread_setschedparam()} service, hereby causing the current thread to switch to secondary mode if it is Xenomai thread.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{tid}</td>
<td>target thread;</td>
</tr>
<tr>
<td>\textit{pol}</td>
<td>scheduling policy, one of SCHED_FIFO, SCHED_RR, SCHED_SPORADIC, SCHED_ED_TPT or SCHED_OTHER;</td>
</tr>
<tr>
<td>\textit{par}</td>
<td>scheduling parameters address.</td>
</tr>
</tbody>
</table>

### Returns

- 0 on success;
- an error number if:
  - ESRCH, \textit{tid} is invalid;
  - EINVAL, \textit{pol} or \textit{par} -> sched_priority is invalid;
  - EAGAIN, in user-space, insufficient memory exists in the system heap, increase CONFIG_XENO_OPT_SYS_HEAP_SZ;
  -EFAULT, in user-space, \textit{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 \texttt{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). \texttt{xeno_sigwinch_handler} prototype is:

```c
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 \texttt{xeno_sigwinch_handler}.

Referenced by \texttt{pthread_setschedparam_ex()}.

### 3.8.2.4 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.

This service is an extended version of \texttt{pthread_setschedparam()}, that supports Xenomai-specific or additional POSIX scheduling policies, which are not available with the host Linux environment.

Typically, a Xenomai thread policy can be set to SCHED_SPORADIC or SCHED_TP using this call.
Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>tid</strong></td>
<td>target thread;</td>
</tr>
<tr>
<td><strong>pol</strong></td>
<td>address where the scheduling policy of tid is stored on success;</td>
</tr>
<tr>
<td><strong>par</strong></td>
<td>address where the scheduling parameters of tid is stored on success.</td>
</tr>
</tbody>
</table>

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().

3.8.2.5 int sched_get_priority_max ( int policy )

Get maximum priority of the specified scheduling policy.

This service returns the maximum priority of the scheduling policy policy.

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>policy</strong></td>
<td>scheduling policy, one of SCHED_FIFO, SCHED_RR, SCHED_SPORADIC, SCHED_THP or SCHED_OTHER.</td>
</tr>
</tbody>
</table>

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>The maximum priority of policy on success;</td>
<td>-1 with errno set if:</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

See Also

Specification.

3.8.2.6 int sched_get_priority_min ( int policy )

Get minimum priority of the specified scheduling policy.

This service returns the minimum priority of the scheduling policy policy.

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>policy</strong></td>
<td>scheduling policy, one of SCHED_FIFO, SCHED_RR, SCHED_SPORADIC, SCHED_THP or SCHED_OTHER.</td>
</tr>
</tbody>
</table>
Return values

<table>
<thead>
<tr>
<th>The</th>
<th>minimum priority of policy on success;</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>with errno set if:</td>
</tr>
<tr>
<td></td>
<td>• EINVAL, policy is invalid.</td>
</tr>
</tbody>
</table>

See Also

Specification.

3.8.2.7 int sched_rr_get_interval ( int pid, struct timespec * interval )

Get the round-robin scheduling time slice.
This service returns the time quantum used by Xenomai POSIX skin SCHED_RR scheduling policy.
In kernel-space, this service only works if pid is zero, in user-space, round-robin scheduling policy is not supported, and this service not implemented.

Parameters

| pid       | must be zero;                        |
| interval  | address where the round-robin scheduling time quantum will be returned on success. |

Return values

| 0          | on success;                          |
| -1         | with errno set if:                   |
|           | • ESRCH, pid is invalid (not 0).     |

See Also

Specification.

3.8.2.8 int sched_setconfig_np ( int cpu, int policy, union sched_config * config, size_t len )

Load CPU-specific scheduler settings for a given policy.
Currently, this call only supports the SCHED_TP policy, for loading the temporal partitions. A configuration is strictly local to the target cpu, and may differ from other processors.

Parameters

| cpu        | processor to load the configuration of. |
| policy     | scheduling policy to which the configuration data applies. Currently, only SCHED_TP is valid. |
| p          | a pointer to the configuration data to load for cpu, applicable to policy. |

Settings applicable to SCHED_TP:

This call installs the temporal partitions for cpu.

- config.tp.windows should be a non-null set of time windows, defining the scheduling time slots for cpu. Each window defines its offset from the start of the global time frame (windows[].offset), a duration (windows[].duration), and the partition id it applies to (windows[].ptid).
Time windows must be strictly contiguous, i.e. windows[n].offset + windows[n].duration shall equal windows[n + 1].offset. If windows[].ptid is in the range [0..CONFIG_XENO_OPT_SCHED_TP_NRP-ART-1], SCHED_TP threads which belong to the partition being referred to may run for the duration of the time window.

Time holes may be defined using windows assigned to the pseudo partition #-1, during which no SCHED_TP threads may be scheduled.

- config.tp.nr_windows should define the number of elements present in the config.tp.windows[] array.

Parameters

| len | size of the configuration data (in bytes). |

Returns

- 0 on success;
- an error number if:
  - EINVAL, cpu is invalid, policy is different from SCHED_TP, SCHED_TP support is not compiled in (see CONFIG_XENO_OPT_SCHED_TP), len is zero, or p contains invalid parameters.
  - ENOMEM, lack of memory to perform the operation.

3.8.2.9 int sched_yield ( void )

Yield the processor.

This function move the current thread at the end of its priority group.

Return values

| 0 |

See Also

- Specification.
3.9 Semaphores services.

Semaphores services.

Functions

- int sem_init (sem_t *sm, int pshared, unsigned value)
  
  *Initialize an unnamed semaphore.*

- int sem_destroy (sem_t *sm)
  
  *Destroy an unnamed semaphore.*

- sem_t * sem_open (const char *name, int oflags,...)
  
  *Open a named semaphore.*

- int sem_close (sem_t *sm)
  
  *Close a named semaphore.*

- int sem_unlink (const char *name)
  
  *Unlink a named semaphore.*

- int sem_trywait (sem_t *sm)
  
  *Attempt to lock a semaphore.*

- int sem_wait (sem_t *sm)
  
  *Lock a semaphore.*

- int sem_timedwait (sem_t *sm, const struct timespec *abs_timeout)
  
  *Attempt, during a bounded time, to lock a semaphore.*

- int sem_post (sem_t *sm)
  
  *Unlock a semaphore.*

- int sem_getvalue (sem_t *sm, int *value)
  
  *Get the value of a semaphore.*

3.9.1 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 incrementated. The macro `SEM_VALUE_MAX` is defined to be this maximum value.

3.9.2 Function Documentation

3.9.2.1 int sem_close ( sem_t * 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

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>on success;</td>
</tr>
<tr>
<td>-1</td>
<td>with <em>errno</em> set if:</td>
</tr>
<tr>
<td></td>
<td>• EINVAL, the semaphore <em>sm</em> is invalid or is an unnamed semaphore.</td>
</tr>
</tbody>
</table>

See Also

- Specification.

### 3.9.2.2 int sem_destroy ( sem_t * sm )

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

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>on success;</td>
</tr>
<tr>
<td>-1</td>
<td>with <em>errno</em> set if:</td>
</tr>
<tr>
<td></td>
<td>• EINVAL, the semaphore <em>sm</em> is invalid or a named semaphore;</td>
</tr>
<tr>
<td></td>
<td>• EPERM, the semaphore <em>sm</em> is not process-shared and does not belong to the current process.</td>
</tr>
</tbody>
</table>

See Also

- Specification.

### 3.9.2.3 int sem_getvalue ( sem_t * sm, 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 locked, the value stored is zero.

Parameters

- **sm**: the semaphore to be closed.
- **value**: the address where the current count of the semaphore is stored.
3.9 Semaphores services.

A semaphore is a synchronization primitive used to control access to shared resources. It consists of a counter and a lock that allows multiple threads to have exclusive access to a resource.

### sem_init()

**Declaration:**

```c
int sem_init ( sem_t ∗ sm, int pshared, unsigned value )
```

**Description:**

Initialize an unnamed semaphore.

This service initializes the semaphore `sm`, with the value `value`.

This service fails if `sm` is already initialized or is a named semaphore.

**Parameters**

- `sm`: the semaphore to be initialized;
- `pshared`: if zero, means that the new semaphore may only be used by threads in the same process as the thread calling `sem_init()`; if non zero, means that the new semaphore may be used by any thread that has access to the memory where the semaphore is allocated.
- `value`: the semaphore initial value.

**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.

### sem_open()

**Declaration:**

```c
sem_t ∗ sem_open ( const char ∗ name, int oflags, ... )
```

**Description:**

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

<table>
<thead>
<tr>
<th>name</th>
<th>the name of the semaphore to be created;</th>
</tr>
</thead>
<tbody>
<tr>
<td>oflags</td>
<td>flags.</td>
</tr>
</tbody>
</table>

### 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`.

#### 3.9.2.6 int sem_post ( sem_t *sm )

Unlock a semaphore.

This service unlocks the semaphore *sm*.

If no thread is currently blocked on this semaphore, its count is incremented, otherwise the highest priority thread is unblocked.

### Parameters

| sm      | the semaphore to be unlocked. |

### Return values

<table>
<thead>
<tr>
<th>0</th>
<th>on success;</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>with errno set if:</td>
</tr>
</tbody>
</table>

- `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`. |
3.9 Semaphores services.

See Also

Specification.

3.9.2.7 int sem_timedwait (sem_t * sm, const struct timespec * abs_timeout)

Attempt, during a bounded time, to lock a semaphore.

This service is equivalent to sem_wait(), except that the caller is only blocked until the timeout abs_timeout expires.

Parameters

<table>
<thead>
<tr>
<th>sm</th>
<th>the semaphore to be locked;</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs_timeout</td>
<td>the timeout, expressed as an absolute value of the CLOCK_REALTIME clock.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>on success;</th>
</tr>
</thead>
</table>
| -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 locked and the specified timeout expired.

Valid contexts:

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

See Also

Specification.

3.9.2.8 int sem_trywait (sem_t * sm)

Attempt to lock a semaphore.

This service is equivalent to sem_wait(), except that it returns immediately if the semaphore sm is currently locked, and that it is not a cancellation point.

Parameters

| sm  | the semaphore to be locked. |
Return values

<table>
<thead>
<tr>
<th>0</th>
<th>on success;</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>with errno set if:</td>
</tr>
<tr>
<td></td>
<td>• EINVAL, the specified semaphore is invalid or uninitialized;</td>
</tr>
<tr>
<td></td>
<td>• EPERM, the semaphore sm is not process-shared and does not belong to the current process;</td>
</tr>
<tr>
<td></td>
<td>• EAGAIN, the specified semaphore is currently locked.</td>
</tr>
</tbody>
</table>

See Also

*Specification.*

3.9.2.9 `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

<table>
<thead>
<tr>
<th>0</th>
<th>on success;</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>with errno set if:</td>
</tr>
<tr>
<td></td>
<td>• ENAMETOOLONG, the length of the <code>name</code> argument exceeds 64 characters;</td>
</tr>
<tr>
<td></td>
<td>• ENOENT, the named semaphore does not exist.</td>
</tr>
</tbody>
</table>

See Also

*Specification.*

3.9.2.10 `int sem_wait ( sem_t * sm )`

Lock a semaphore.

This service locks the semaphore `sm` if it is currently unlocked (i.e. if its value is greater than 0). If the semaphore is currently locked, the calling thread is suspended until the semaphore is unlocked, 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

<table>
<thead>
<tr>
<th></th>
<th>the semaphore to be locked.</th>
</tr>
</thead>
</table>

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>on success;</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>with \texttt{errno} set if:</td>
</tr>
</tbody>
</table>

- EPERM, the caller context is invalid;
- EINVAL, the semaphore is invalid or uninitialized;
- EPERM, the semaphore \texttt{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

\texttt{Specification}. 
3.10 Shared memory services.

Shared memory services.

Functions

- int **shm_open**(const char ∗name, int oflags, mode_t mode)
  
  Open a shared memory object.

- int **close**(int fd)
  
  Close a file descriptor.

- int **shm_unlink**(const char ∗name)
  
  Unlink a shared memory object.

- int **ftruncate**(int fd, off_t len)
  
  Truncate a file or shared memory object to a specified length.

- void ∗**mmap**(void ∗addr, size_t len, int prot, int flags, int fd, off_t off)
  
  Map pages of memory.

- int **munmap**(void ∗addr, size_t len)
  
  Unmap pages of memory.

3.10.1 Detailed Description

Shared memory services. Shared memory objects are memory regions that can be mapped into processes address space, allowing them to share these regions as well as to share them with kernel-space modules.

Shared memory are also the only mean by which anonymous POSIX skin synchronization objects (mutexes, condition variables or semaphores) may be shared between kernel-space modules and user-space processes, or between several processes.

3.10.2 Function Documentation

3.10.2.1 int close( int fd )

Close a file descriptor.

This service closes the file descriptor fd. In kernel-space, this service only works for file descriptors opened with **shm_open()**, i.e. shared memory objects. A shared memory object is only destroyed once all file descriptors are closed with this service, it is unlinked with the **shm_unlink()** service, and all mappings are unmapped with the **munmap()** service.

Parameters

| fd | file descriptor. |

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>on success;</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>with errno set if:</td>
</tr>
<tr>
<td></td>
<td>- EBADF, fd is not a valid file descriptor (in kernel-space, it was not obtained with <strong>shm_open()</strong>);</td>
</tr>
<tr>
<td></td>
<td>- EPERM, the caller context is invalid.</td>
</tr>
</tbody>
</table>
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.

Referenced by shm_open().

3.10.2.2 int ftruncate ( int fd, off_t len )

Truncate a file or shared memory object to a specified length.

When used in kernel-space, this service set to len the size of a shared memory object opened with the shm_open() service. In user-space this service falls back to Linux regular ftruncate service for file descriptors not obtained with shm_open(). When this service is used to increase the size of a shared memory object, the added space is zero-filled.

Shared memory are suitable for direct memory access (allocated in physically contiguous memory) if O_DIRECT was passed to shm_open.

Shared memory objects may only be resized if they are not currently mapped.

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>fd</td>
<td>file descriptor;</td>
</tr>
<tr>
<td>len</td>
<td>new length of the underlying file or shared memory object.</td>
</tr>
</tbody>
</table>

Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>on success;</td>
</tr>
<tr>
<td>-1</td>
<td>with errno set if:</td>
</tr>
</tbody>
</table>

- EBADF, fd is not a valid file descriptor;
- EPERM, the caller context is invalid;
- EINVAL, the specified length is invalid;
- EINVAL, the architecture can not honour the O_DIRECT flag;
- EINTR, this service was interrupted by a signal;
- EBUSY, fd is a shared memory object descriptor and the underlying shared memory is currently mapped;
- EFBIG, allocation of system memory failed.

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

See Also
- Specification.

Referenced by shm_open().

Generated on Thu Jul 7 2016 13:25:13 for Xenomai POSIX skin API by Doxygen
Map pages of memory.

This service allow shared memory regions to be accessed by the caller.

When used in kernel-space, this service returns the address of the offset `off` of the shared memory object underlying `fd`. The protection flags `prot`, are only checked for consistency with `fd` open flags, but memory protection is unsupported. An existing shared memory region exists before it is mapped, this service only increments a reference counter.

The only supported value for `flags` is `MAP_SHARED`.

When used in user-space, this service maps the specified shared memory region into the caller address-space. If `fd` is not a shared memory object descriptor (i.e. not obtained with `shm_open()`), this service falls back to the regular Linux `mmap` service.

### Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>addr</code></td>
<td>ignored.</td>
</tr>
<tr>
<td><code>len</code></td>
<td>size of the shared memory region to be mapped.</td>
</tr>
<tr>
<td><code>prot</code></td>
<td>protection bits, checked in kernel-space, but only useful in user-space, are a bitwise or of the following values:</td>
</tr>
<tr>
<td></td>
<td>• PROT_NONE, meaning that the mapped region can not be accessed;</td>
</tr>
<tr>
<td></td>
<td>• PROT_READ, meaning that the mapped region can be read;</td>
</tr>
<tr>
<td></td>
<td>• PROT_WRITE, meaning that the mapped region can be written;</td>
</tr>
<tr>
<td></td>
<td>• PROT_EXEC, meaning that the mapped region can be executed.</td>
</tr>
<tr>
<td><code>flags</code></td>
<td>only MAP_SHARED is accepted, meaning that the mapped memory region is shared.</td>
</tr>
<tr>
<td><code>fd</code></td>
<td>file descriptor, obtained with <code>shm_open()</code></td>
</tr>
<tr>
<td><code>off</code></td>
<td>offset in the shared memory region.</td>
</tr>
</tbody>
</table>

### Return values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>on success;</td>
</tr>
<tr>
<td><code>MAP_FAILED</code></td>
<td>with <code>errno</code> set if:</td>
</tr>
<tr>
<td></td>
<td>• EINVAL, <code>len</code> is null or <code>addr</code> is not a multiple of <code>PAGE_SIZE</code>;</td>
</tr>
<tr>
<td></td>
<td>• EBADF, <code>fd</code> is not a shared memory object descriptor (obtained with <code>shm_open()</code>);</td>
</tr>
<tr>
<td></td>
<td>• EPERM, the caller context is invalid;</td>
</tr>
<tr>
<td></td>
<td>• ENOTSUP, <code>flags</code> is not <code>MAP_SHARED</code>;</td>
</tr>
<tr>
<td></td>
<td>• EACCES, <code>fd</code> is not opened for reading or is not opened for writing and PROT_WRITE is set in <code>prot</code>;</td>
</tr>
<tr>
<td></td>
<td>• EINTR, this service was interrupted by a signal;</td>
</tr>
<tr>
<td></td>
<td>• ENXIO, the range [off;off+len) is invalid for the shared memory region specified by <code>fd</code>;</td>
</tr>
<tr>
<td></td>
<td>• EAGAIN, insufficient memory exists in the system heap to create the mapping, increase <code>CONFIG_XENO_OPT_SYS_HEAPSZ</code>.</td>
</tr>
</tbody>
</table>
3.10 Shared memory services.

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

See Also

Specification.

3.10.2.4 int munmap ( void *addr, size_t len )

Unmap pages of memory.

This service unmaps the shared memory region \([addr;addr+len)\) from the caller address-space. When called from kernel-space the memory region remain accessible as long as it exists, and this service only decrements a reference counter.

When called from user-space, if the region is not a shared memory region, this service falls back to the regular Linux `munmap()` service.

Parameters

<table>
<thead>
<tr>
<th>addr</th>
<th>start address of shared memory area;</th>
</tr>
</thead>
<tbody>
<tr>
<td>len</td>
<td>length of the shared memory area.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>on success;</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>with <code>errno</code> set if:</td>
</tr>
<tr>
<td></td>
<td>- EINVAL, <code>len</code> is null, <code>addr</code> is not a multiple of the page size or the range ([addr;addr+len)) is not a mapped region;</td>
</tr>
<tr>
<td></td>
<td>- ENXIO, <code>addr</code> is not the address of a shared memory area;</td>
</tr>
<tr>
<td></td>
<td>- EPERM, the caller context is invalid;</td>
</tr>
<tr>
<td></td>
<td>- EINTR, this service was interrupted by a signal.</td>
</tr>
</tbody>
</table>

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.

3.10.2.5 int shm_open ( const char *name, int oflags, mode_t mode )

Open a shared memory object.

This service establishes a connection between a shared memory object and a file descriptor. Further use of this descriptor will allow to dimension and map the shared memory into the calling context address space.

One of the following access mode should be set in `oflags`:
- O_RDONLY, meaning that the shared memory object may only be mapped with the PROT_READ flag;

- O_WRONLY, meaning that the shared memory object may only be mapped with the PROT_WRITE flag;

- O_RDWR, meaning that the shared memory object may be mapped with the PROT_READ | PROT_WRITE flag.

If no shared memory object named name exists, and oflags has the O_CREAT bit set, the shared memory object is created by this function.

If oflags has the two bits O_CREAT and O_EXCL set and the shared memory object already exists, this service fails.

If oflags has the bit O_TRUNC set, the shared memory exists and is not currently mapped, its size is truncated to 0.

If oflags has the bit O_DIRECT set, the shared memory will be suitable for direct memory access (allocated in physically contiguous memory).

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

<table>
<thead>
<tr>
<th>name</th>
<th>name of the shared memory object to open;</th>
</tr>
</thead>
<tbody>
<tr>
<td>oflags</td>
<td>flags.</td>
</tr>
<tr>
<td>mode</td>
<td>ignored.</td>
</tr>
</tbody>
</table>

Returns

-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 shared memory object already exists;
- ENOENT, the bit O_CREAT is not set in oflags and the shared memory object does not exist;
- ENOSPC, insufficient memory exists in the system heap to create the shared memory object, increase CONFIG_XENO_OPT_SYS_HEAP_SIZ;
- EPERM, the caller context is invalid;
- EINVAL, the O_TRUNC flag was specified and the shared memory object is currently mapped;
- EMFILE, too many descriptors are currently open.

Valid contexts:

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

See Also

Specification.

References close(), and ftruncate().

Generated on Thu Jul 7 2016 13:25:13 for Xenomai POSIX skin API by Doxygen
3.10.2.6 int shm_unlink ( const char * name )

Unlink a shared memory object.

This service unlinks the shared memory object named name. The shared memory object is not destroyed until every file descriptor obtained with the shm_open() service is closed with the close() service and all mappings done with mmap() are unmapped with munmap(). However, after a call to this service, the unlinked shared memory object may no longer be reached with the shm_open() service.

Parameters

| name | name of the shared memory object to be unlinked. |

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>on success;</th>
</tr>
</thead>
</table>
| -1 | with errno set if:

- EPERM, the caller context is invalid;
- ENAMETOOLONG, the length of the name argument exceeds 64 characters;
- ENOENT, the shared memory object 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.
3.11 Signals services.

Signals management services.

Functions

- int **sigemptyset**(sigset_t *set)
  
  *Initialize and empty a signal set.*

- int **sigfillset**(sigset_t *set)
  
  *Initialize and fill a signal set.*

- int **sigaddset**(sigset_t *set, int sig)
  
  *Add a signal to a signal set.*

- int **sigdelset**(sigset_t *set, int sig)
  
  *Delete a signal from a signal set.*

- int **sigismember**(const sigset_t *set, int sig)
  
  *Test for a signal in a signal set.*

- int **sigaction**(int sig, const struct sigaction *act, struct sigaction *oact)
  
  *Examine and change a signal action.*

- int **pthread_kill**(pthread_t thread, int sig)
  
  *Send a signal to a thread.*

- int **pthread_sigqueue_np**(pthread_t thread, int sig, union sigval value)
  
  *Queue a signal to a thread.*

- int **sigpending**(sigset_t *set)
  
  *Examine pending signals.*

- int **pthread_sigmask**(int how, const sigset_t *set, sigset_t *oset)
  
  *Examine and change the set of signals blocked by a thread.*

- int **sigwait**(const sigset_t *set, int *sig)
  
  *Wait for signals.*

- int **sigwaitinfo**(const sigset_t *__restrict__ set, siginfo_t *__restrict__ info)
  
  *Wait for signals.*

- int **sigtimedwait**(const sigset_t *__restrict__ set, siginfo_t *__restrict__ info, const struct timespec *__restrict__ timeout)
  
  *Wait during a bounded time for signals.*

3.11.1 Detailed Description

Signals management services. Signals are asynchronous notifications delivered to a process or thread. Such notifications occur as the result of an exceptional event or at the request of another process.

The services documented here are reserved to Xenomai kernel-space threads, user-space threads switch to secondary mode when handling signals, and use Linux regular signals services.

Xenomai POSIX skin signals are implemented as real-time signals, meaning that they are queued when posted several times to a thread before the first notification is handled, and that each signal carry additional data in a **siginfo_t** object. In order to ensure consistency with user-space signals, valid signals number range from 1 to SIGRTMAX, signals from SIGRTMIN to SIGRTMAX being higher priority than signals from 1 to SIGRTMIN-1. As a special case, signal 0 may be used with services **pthread_kill()** and **pthread_sigqueue_np()** to check if a thread exists, but entails no other action.

The action to be taken upon reception of a signal depends on the thread signal mask, (see **pthread_sigmask()**), and on the settings described by a **sigaction** structure (see **sigaction()**).
3.11.2 Function Documentation

3.11.2.1 int pthread_kill ( pthread_t thread, int sig )

Send a signal to a thread.
This service send the signal sig to the Xenomai POSIX skin thread thread (created with pthread_create()). If sig is zero, this service check for existence of the thread thread, but no signal is sent.

Parameters

<table>
<thead>
<tr>
<th>thread</th>
<th>thread identifier;</th>
</tr>
</thead>
<tbody>
<tr>
<td>sig</td>
<td>signal number.</td>
</tr>
</tbody>
</table>

Returns

0 on success;
an error number if:

- EINVAL, sig is an invalid signal number;
- EAGAIN, the maximum number of pending signals has been exceeded;
- ESRCH, thread is an invalid thread identifier.

See Also

Specification.

3.11.2.2 int pthread_sigmask ( int how, const sigset_t * set, sigset_t * oset )

Examine and change the set of signals blocked by a thread.
The signal mask of a thread is the set of signals that are blocked by this thread.
If oset is not NULL, this service stores, at the address oset the current signal mask of the calling thread.
If set is not NULL, this service sets the signal mask of the calling thread according to the value of how, as follow:

- if how is SIG_BLOCK, the signals in set are added to the calling thread signal mask;
- if how is SIG_SETMASK, the calling thread signal mask is set to set;
- if how is SIG_UNBLOCK, the signals in set are removed from the calling thread signal mask.

If some signals are unblocked by this service, they are handled before this service returns.

Parameters

<table>
<thead>
<tr>
<th>how</th>
<th>if set is not null, a value indicating how to interpret set;</th>
</tr>
</thead>
<tbody>
<tr>
<td>set</td>
<td>if not null, a signal set that will be used to modify the calling thread signal mask;</td>
</tr>
<tr>
<td>oset</td>
<td>if not null, address where the previous value of the calling thread signal mask will be stored on success.</td>
</tr>
</tbody>
</table>

Returns

0 on success;
an error number if:

- EPERM, the calling context is invalid;
- EINVAL, how is not SIG_BLOCK, SIG_UNBLOCK or SIG_SETMASK.
Valid contexts:
- Xenomai POSIX skin kernel-space thread.

See Also
- Specification.

3.11.2.3 int pthread_sigqueue_np ( pthread_t thread, int sig, union sigval value )

Queue a signal to a thread.
This service send the signal sig to the Xenomai POSIX skin thread thread (created with pthread_create()), with the value value. If sig is zero, this service check for existence of the thread thread, but no signal is sent.
This service is equivalent to the POSIX service sigqueue(), except that the signal is directed to a thread instead of being directed to a process.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>thread</td>
<td>thread identifier,</td>
</tr>
<tr>
<td>sig</td>
<td>signal number,</td>
</tr>
<tr>
<td>value</td>
<td>additional datum passed to thread with the signal sig.</td>
</tr>
</tbody>
</table>

Returns

- 0 on success;
- an error number if:
  - EINVAL, sig is an invalid signal number;
  - EAGAIN, the maximum number of pending signals has been exceeded;
  - ESRCH, thread is an invalid thread identifier.

See Also
- sigqueue() specification.

3.11.2.4 int sigaction ( int sig, const struct sigaction *act, struct sigaction *oact )

Examine and change a signal action.
The sigaction structure describes the actions to be taken upon signal delivery. A sigaction structure is associated with every signal, for the kernel-space as a whole.
If oact is not NULL, this service returns at the address oact, the current value of the sigaction structure associated with the signal sig.
If act is not NULL, this service set to the value pointed to by act, the sigaction structure associated with the signal sig.
The structure sigaction has the following members:

- sa_flags, is a bitwise OR of the flags;
  - SA_RESETHAND, meaning that the signal handler will be reset to SIG_GFL and SA_SIGIN-FO cleared upon reception of a signal,
  - SA_NODEFER, meaning that the signal handler will be called with the signal sig not masked when handling the signal sig,
– SA_SIGINFO, meaning that the member sa_sigaction of the sigaction structure will be used as a signal handler instead of sa_handler

• sa_mask, of type sigset_t, is the value to which the thread signals mask will be set during execution of the signal handler (sig is automatically added to this set if SA_NODEFER is not set in sa_flags);

• sa_handler, of type void (*)(int) is the signal handler which will be called upon signal delivery if SA_SIGINFO is not set in sa_flags, or one of SIG_IGN or SIG_DFL, meaning that the signal will be respectively ignored or handled with the default handler;

• sa_sigaction, of type void (*)(int, siginfo_t *, void *) is the signal handler which will be called upon signal delivery if SA_SIGINFO is set in sa_flags.

When using sa_handler as a signal handler, it is passed the number of the received signal, when using sa_sigaction, two additional arguments are passed:

• a pointer to a siginfo_t object, containing additional information about the received signal;

• a void pointer, always null in this implementation.

The following members of the siginfo_t structure are filled by this implementation:

• si_signo, the signal number;

• si_code, the provenance of the signal, one of:
  – SI_QUEUE, the signal was queued with pthread_sigqueue_np(),
  – SI_USER, the signal was queued with pthread_kill(),
  – SI_TIMER, the signal was queued by a timer (see timer_settime()),
  – SI_MESQ, the signal was queued by a message queue (see mq_notify());

• si_value, an additional datum, of type union sigval.

Parameters

<table>
<thead>
<tr>
<th>sig</th>
<th>a signal number;</th>
</tr>
</thead>
<tbody>
<tr>
<td>act</td>
<td>if not null, description of the action to be taken upon notification of the signal sig;</td>
</tr>
<tr>
<td>oact</td>
<td>if not null, address where the previous description of the signal action is stored on success.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>0</th>
<th>on success;</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>with errno set if:</td>
</tr>
</tbody>
</table>

  • EINVAL, sig is an invalid signal number;

  • ENOTSUP, the sa_flags member of act contains other flags than SA_RESETHAND, SA_NODEFER and SA_SIGINFO;

See Also

Specification.

3.11.2.5 int sigaddset ( sigset_t * set, int sig )

Add a signal to a signal set.

This service adds the signal number sig to the signal set pointed to by set.
### 3.11.2.6 int sigdelset ( sigset_t *set, int sig )

Delete a signal from a signal set.

This service remove the signal number `sig` from the signal set pointed to by `set`.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>set</code></td>
<td>address of a signal set;</td>
</tr>
<tr>
<td><code>sig</code></td>
<td>signal to be removed from <code>set</code>.</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>0</code></td>
<td>on success;</td>
</tr>
<tr>
<td><code>-1</code></td>
<td>with <code>errno</code> set if:</td>
</tr>
<tr>
<td></td>
<td>• <code>EINVAL</code>, <code>sig</code> is not a valid signal number.</td>
</tr>
</tbody>
</table>

**See Also**

*Specification.*

### 3.11.2.7 int sigemptyset ( sigset_t *set )

Initialize and empty a signal set.

This service initializes and empties the signal set pointed to by `set`.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>set</code></td>
<td>address of the signal set to be initialized.</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>0</code></td>
</tr>
</tbody>
</table>

**See Also**

*Specification.*
3.11.2.8 int sigfillset ( sigset_t * set )

Initialize and fill a signal set.
This service initializes and fills the signal set pointed to by set.

Parameters

| set | address of a signal set to be filled. |

Return values

| 0 |

See Also

Specification.

3.11.2.9 int sigismember ( const sigset_t * set, int sig )

Test for a signal in a signal set.
This service tests whether the signal number sig is member of the signal set pointed to by set.

Parameters

| set | address of a signal set; |
| sig | tested signal number. |

Return values

| 0 | on success; |
| -1 | with errno set if: |

- EINVAL, sig is not a valid signal number.

See Also

Specification.

3.11.2.10 int sigpending ( sigset_t * set )

Examine pending signals.
This service stores, at the address set, the set of signals that are currently blocked and have been received by the calling thread.

Parameters

| set | address where the set of blocked and received signals are stored on success. |

Return values

| 0 | on success; |
3.11.2.11 int sigtimedwait ( const sigset_t *__restrict__ set, siginfo_t *__restrict__ info, const struct timespec *__restrict__ timeout )

Wait during a bounded time for signals.

This service is equivalent to the `sigwaitinfo()` service, except that the calling thread is only blocked until the timeout specified by `timeout` expires.

Parameters

- `set`: set of signals to wait for;
- `info`: address where the received `siginfo_t` object will be stored on success;
- `timeout`: the timeout, expressed as a time interval.

Return values

- `0`: on success;
- `-1` with `errno` set if:
  - `EINVAL`, the specified timeout is invalid;
  - `EPERM`, the caller context is invalid;
  - `EINVAL`, a signal in `set` is not currently blocked;
  - `EAGAIN`, no signal was received and the specified timeout expired.

Valid contexts:

- Xenomai POSIX skin kernel-space thread.

See Also

`Specification`.

3.11.2.12 int sigwait ( const sigset_t * set, int * sig )

Wait for signals.

This service blocks a Xenomai kernel-space POSIX skin thread until a signal of the set `set` is received. If a signal in `set` is not currently blocked by the calling thread, this service returns immediately with an error. The signal received is stored at the address `sig`.

If a signal of the set `set` was already pending, it is cleared and this service returns immediately.

Signals are received in priority order, i.e. from `SIGRTMIN` to `SIGRTMAX`, then from 1 to `SIGRTMIN-1`. 

Valid contexts:

- Xenomai POSIX skin kernel-space thread.

See Also

`Specification`.
3.11 Signals services

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>set</td>
<td>set of signals to wait for;</td>
</tr>
<tr>
<td>sig</td>
<td>address where the received signal will be stored on success.</td>
</tr>
</tbody>
</table>

Returns
- 0 on success;
- an error number if:
  - EPERM, the caller context is invalid;
  - EINVAL, a signal in set is not currently blocked.

Valid contexts:
- Xenomai POSIX skin kernel-space thread.

See Also
- Specification.

3.11.2.13 int sigwaitinfo ( const sigset_t *__restrict__ set, siginfo_t *__restrict__ info )

Wait for signals.
This service is equivalent to the sigwait() service, except that it returns, at the address info, the siginfo_t object associated with the received signal instead of only returning the signal number.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>set</td>
<td>set of signals to wait for;</td>
</tr>
<tr>
<td>info</td>
<td>address where the received siginfo_t object will be stored on success.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>on success;</td>
</tr>
<tr>
<td>-1</td>
<td>with errno set if:</td>
</tr>
</tbody>
</table>
  - EPERM, the caller context is invalid;
  - EINVAL, a signal in set is not currently blocked.

Valid contexts:
- Xenomai POSIX skin kernel-space thread.

See Also
- Specification.
3.12 Threads management services.

Threads management services.

Modules

- **Thread cancellation.**
  - Thread cancellation.
- **Threads scheduling services.**
  - Thread scheduling services.
- **Thread creation attributes.**
  - Thread creation attributes.

Enumerations

- enum **init_step**
  - Execute an initialization routine.

Functions

- int **pthread_create**(pthread_t *tid, const pthread_attr_t *attr, void *(*start)(void *), void *arg)
  - Create a thread.
- int **pthread_detach**(pthread_t thread)
  - Detach a running thread.
- int **pthread_equal**(pthread_t t1, pthread_t t2)
  - Compare thread identifiers.
- void **pthread_exit**(void *value_ptr)
  - Terminate the current thread.
- int **pthread_join**(pthread_t thread, void **value_ptr)
  - Wait for termination of a specified thread.
- pthread_t **pthread_self**(void)
  - Get the identifier of the calling thread.
- int **pthread_make_periodic_np**(pthread_t thread, struct timespec *starttp, struct timespec *periodtp)
  - Make a thread periodic.
- int **pthread_wait_np**(unsigned long *overruns_r)
  - Wait for current thread next period.
- int **pthread_set_mode_np**(int clrmask, int setmask)
  - Set the mode of the current thread.
- int **pthread_set_name_np**(pthread_t thread, const char *name)
  - Set a thread name.

3.12.1 Detailed Description

Threads management services.

See Also

- Specification.
3.12.2 Enumeration Type Documentation

3.12.2.1 enum init_step

Execute an initialization routine.
This service may be used by libraries which need an initialization function to be called only once.
The function init_routine will only be called, with no argument, the first time this service is called specifying the address once.

Returns

0 on success;
an error number if:

- EINVAL, the object pointed to by once is invalid (it must have been initialized with PTHREAD_D_ONCE_INIT);
- EPERM, the caller context is invalid.

Valid contexts:

- Xenomai kernel-space thread.

See Also

Specification.

3.12.3 Function Documentation

3.12.3.1 int pthread_create ( pthread_t * tid, const pthread_attr_t * attr, void *( )( void * ) start, void * arg )

Create a thread.
This service creates a thread. The created thread may be used with all POSIX skin services.
The new thread runs the start routine, with the arg argument.
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. See Thread creation attributes. for a definition of thread creation attributes and their default values.
Returning from the start routine has the same effect as calling pthread_exit() with the return value.

Parameters

<table>
<thead>
<tr>
<th>tid</th>
<th>address where the identifier of the new thread will be stored on success:</th>
</tr>
</thead>
<tbody>
<tr>
<td>attr</td>
<td>thread attributes:</td>
</tr>
<tr>
<td>start</td>
<td>thread routine:</td>
</tr>
<tr>
<td>arg</td>
<td>thread routine argument.</td>
</tr>
</tbody>
</table>

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 inheritance 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:

```c
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().

3.12.3.2 int pthread_detach ( pthread_t thread )

Detach a running thread.

This service detaches a joinable thread. A detached thread is a thread which control block is automatically reclaimed when it terminates. The control block of a joinable thread, on the other hand, is only reclaimed when joined with the service pthread_join().

If some threads are currently blocked in the pthread_join() service with thread as a target, they are unblocked and pthread_join() returns EINVAL.

Parameters

| thread | target thread. |

Returns

0 on success;
an error number if:

- ESRCH, thread is an invalid thread identifier;
- EINVAL, thread is not joinable.

See Also

Specification.

3.12.3.3 int pthread_equal ( pthread_t t1, pthread_t t2 )

Compare thread identifiers.

This service compare the thread identifiers t1 and t2. No attempt is made to check the threads for existence. In order to check if a thread exists, the pthread_kill() service should be used with the signal number 0.
3.12 Threads management services.

Parameters

| t1 | thread identifier; |
| t2 | other thread identifier. |

Returns

a non zero value if the thread identifiers are equal;
0 otherwise.

See Also

Specification.

3.12.3.4 void pthread_exit ( void * value_ptr )

Terminate the current thread.

This service terminate the current thread with the return value value_ptr. If the current thread is joinable, the return value is returned to any thread joining the current thread with the pthread_join() service.

When a thread terminates, cancellation cleanup handlers are executed in the reverse order that they were pushed. Then, thread-specific data destructors are executed.

Parameters

| value_ptr | thread return value. |

See Also

Specification.

3.12.3.5 int pthread_join ( pthread_t thread, void ** value_ptr )

Wait for termination of a specified thread.

If the thread thread is running and joinable, this service blocks the calling thread until the thread thread terminates or detaches. In this case, the calling context must be a blockable context (i.e. a Xenomai thread without the scheduler locked) or the root thread (i.e. a module initialization or cleanup routine). When thread terminates, the calling thread is unblocked and its return value is stored at the address value_ptr.

If, on the other hand, the thread thread has already finished execution, its return value is stored at the address value_ptr and this service returns immediately. In this case, this service may be called from any context.

This service is a cancelation point for POSIX skin threads: if the calling thread is canceled while blocked in a call to this service, the cancelation request is honored and thread remains joinable.

Multiple simultaneous calls to pthread_join() specifying the same running target thread block all the callers until the target thread terminates.

Parameters

| thread | identifier of the thread to wait for; |
value_ptr | address where the target thread return value will be stored on success.

Returns

0 on success;
an error number if:

- ESRCH, thread is invalid;
- EDEADLK, attempting to join the calling thread;
- EINVAL, thread is detached;
- EPERM, the caller context is invalid.

Valid contexts, if this service has to block its caller:

- Xenomai kernel-space thread;
- kernel module initialization or cleanup routine;
- Xenomai user-space thread (switches to primary mode).

See Also

Specification.

3.12.3.6 int pthread_make_periodic_np ( pthread_t thread, struct timespec *starttp, struct timespec *periodtp )

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

<table>
<thead>
<tr>
<th>thread</th>
<th>thread identifier. This thread is immediately delayed until the first periodic release point is reached.</th>
</tr>
</thead>
<tbody>
<tr>
<td>starttp</td>
<td>start time, expressed as an absolute value of the CLOCK_REALTIME clock. The affected thread will be delayed until this point is reached.</td>
</tr>
<tr>
<td>periodtp</td>
<td>period, expressed as a time interval.</td>
</tr>
</tbody>
</table>

Returns

0 on success;
an error number if:

- ESRCH, thread is invalid;
- ETIMEDOUT, the start time has already passed.

Rescheduling: always, until the starttp start time has been reached.

3.12.3.7 pthread_t pthread_self ( void )

Get the identifier of the calling thread.

This service returns the identifier of the calling thread.
3.12 Threads management services.

Returns

identifier of the calling thread;
NULL if the calling thread is not a POSIX skin thread.

See Also

Specification.

3.12.3.8 int pthread_set_mode_np ( int clrmask, int setmask )

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 by the scheduler until the scheduler is unlocked;
- PTHREAD_RPIOFF, when set, prevents the root Linux thread from inheriting the priority of the calling thread, when this thread is running in secondary mode;
- PTHREAD_WARNSW, when set, cause the signal SIGXCPU to be sent to the current thread, whenever it involuntary switches to secondary mode;
- PTHREAD_PRIMARY, cause the migration of the current thread to primary mode.

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

<table>
<thead>
<tr>
<th>clrmask</th>
<th>set of bits to be cleared;</th>
</tr>
</thead>
<tbody>
<tr>
<td>setmask</td>
<td>set of bits to be set.</td>
</tr>
</tbody>
</table>

Returns

0 on success;
an error number if:
- EINVAL, some bit in clrmask or setmask is invalid.

3.12.3.9 int pthread_set_name_np ( pthread_t thread, const char * name )

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; |
Returns

0 on success;
an error number if:

- ESRCH, thread is invalid.

3.12.3.10 int pthread_wait_np ( unsigned long *overruns_r )

Wait for current thread next period.
If it is periodic, this service blocks the calling thread until the next period elapses.
This service is a cancelation point for POSIX skin threads.
This service is a non-portable extension of the POSIX interface.

Parameters

| overruns_r | address where the overruns count is returned in case of overrun. |

Returns

0 on success;
an error number if:

- EPERM, the calling context is invalid;
- EWOULDBLOCK, the calling thread is not periodic;
- EINTR, this service was interrupted by a signal;
- ETIMEDOUT, at least one overrun occurred.

Valid contexts:

- Xenomai kernel-space thread;
- Xenomai user-space thread (switches to primary mode).
3.13 Thread creation attributes.

Functions

- `int pthread_attr_init (pthread_attr_t *attr)`
  Initialize a thread attributes object.
- `int pthread_attr_destroy (pthread_attr_t *attr)`
  Destroy a thread attributes object.
- `int pthread_attr_getdetachstate (const pthread_attr_t *attr, int *detachstate)`
  Get detachstate attribute.
- `int pthread_attr_setdetachstate (pthread_attr_t *attr, int detachstate)`
  Set detachstate attribute.
- `int pthread_attr_getstacksize (const pthread_attr_t *attr, size_t *stacksize)`
  Get stacksize attribute.
- `int pthread_attr_setstacksize (pthread_attr_t *attr, size_t stacksize)`
  Set stacksize attribute.
- `int pthread_attr_getinheritscached (const pthread_attr_t *attr, int *inheritscached)`
  Get inheritscached attribute.
- `int pthread_attr_setinheritscached (pthread_attr_t *attr, int inheritscached)`
  Set inheritscached attribute.
- `int pthread_attr_getschedpolicy (const pthread_attr_t *attr, int *policy)`
  Get schedpolicy attribute.
- `int pthread_attr_setschedpolicy (pthread_attr_t *attr, int policy)`
  Set schedpolicy attribute.
- `int pthread_attr_getschedparam (const pthread_attr_t *attr, struct sched_param *par)`
  Get schedparam attribute.
- `int pthread_attr_getschedparam_ex (const pthread_attr_t *attr, struct sched_param_ex *par)`
  Get schedparam_ex extended attribute.
- `int pthread_attr_setschedparam (pthread_attr_t *attr, const struct sched_param *par)`
  Set schedparam attribute.
- `int pthread_attr_setschedparam_ex (pthread_attr_t *attr, const struct sched_param_ex *par)`
  Set extended schedparam_ex attribute.
- `int pthread_attr_getscope (const pthread_attr_t *attr, int *scope)`
  Get contention scope attribute.
- `int pthread_attr_setscope (pthread_attr_t *attr, int scope)`
  Set contention scope attribute.
- `int pthread_attr_getname_np (const pthread_attr_t *attr, const char **name)`
  Get name attribute.
- `int pthread_attr_setname_np (pthread_attr_t *attr, const char *name)`
  Set name attribute.
- `int pthread_attr_getfp_np (const pthread_attr_t *attr, int *fp)`
  Get the floating point attribute.
- `int pthread_attr_setfp_np (pthread_attr_t *attr, int fp)`
  Set the floating point attribute.
- `int pthread_attr_getaffinity_np (const pthread_attr_t *attr, xnarch_cpumask_t *mask)`
  Get the processor affinity attribute.
- `int pthread_attr_setaffinity_np (pthread_attr_t *attr, xnarch_cpumask_t *mask)`
  Set the processor affinity attribute.
Thread creation attributes. The services described in this section allow to set the attributes of a `pthread_attr_t` object, passed to the `pthread_create()` service in order to set the attributes of a created thread.

A `pthread_attr_t` object has to be initialized with `pthread_attr_init()` first, which sets attributes to their default values, i.e. in kernel-space:

- `detachstate` to `PTHREAD_CREATE_JOINABLE`,
- `stacksize` to `PTHREAD_STACK_MIN`,
- `inheritsched` to `PTHREAD_EXPLICIT_SCHED`,
- `schedpolicy` to `SCHED_OTHER`,
- `name` to `NULL` (only available in kernel-space),
- scheduling priority to the minimum,
- floating-point hardware enabled (only available in kernel-space),
- processor affinity set to all available processors (only available as a thread attribute in kernel-space).

In user-space, the attributes and their defaults values are those documented by the underlying threading library (LinuxThreads or NPTL).

### 3.13.2 Function Documentation

#### 3.13.2.1 int pthread_attr_destroy ( pthread_attr_t * attr )

Destroy a thread attributes object.

This service invalidates the attribute object pointed to by `attr`. The object becomes invalid for all services (they all return EINVAL) except `pthread_attr_init()`.

See Also

- Specification.

#### 3.13.2.2 int pthread_attr_getaffinity_np ( const pthread_attr_t * attr, xnarch_cpumask_t * mask )

Get the processor affinity attribute.

This service stores, at the address `mask`, the value of the `affinity` attribute in the attribute object `attr`.

The `affinity` attributes is a bitmask where bits set indicate processor where a thread created with the attribute `attr` may run. The least significant bit corresponds to the first logical processor.

This service is a non-portable extension of the POSIX interface.

**Parameters**

<table>
<thead>
<tr>
<th>attr</th>
<th>attribute object;</th>
</tr>
</thead>
<tbody>
<tr>
<td>mask</td>
<td>address where the value of the <code>affinity</code> attribute will be stored on success.</td>
</tr>
</tbody>
</table>
3.13 Thread creation attributes.

Returns

0 on success;
an error number if:

- EINVAL, \textit{attr} is invalid.

Valid contexts:

- kernel module initialization or cleanup routine;
- Xenomai kernel-space thread.

3.13.2.3 \textbf{int pthread_attr_getdetachstate ( const pthread_attr_t * attr, int * detachstate )}

Get detachstate attribute.

This service returns, at the address \textit{detachstate}, the value of the \textit{detachstate} attribute in the thread attribute object \textit{attr}.

Valid values of this attribute are PTHREAD_CREATE_JOINABLE and PTHREAD_CREATE_DETACHED. A detached thread is a thread which control block is automatically reclaimed when it terminates. The control block of a joinable thread, on the other hand, is only reclaimed when joined with the service \textit{pthread_join()}.

A thread that was created joinable may be detached after creation by using the \textit{pthread_detach()} service.

Parameters

<table>
<thead>
<tr>
<th>attr</th>
<th>attribute object</th>
</tr>
</thead>
<tbody>
<tr>
<td>detachstate</td>
<td>address where the value of the detachstate attribute will be stored on success.</td>
</tr>
</tbody>
</table>

Returns

0 on success;
an error number if:

- EINVAL, \textit{attr} is invalid;

See Also

- Specification.

3.13.2.4 \textbf{int pthread_attr_getfp_np ( const pthread_attr_t * attr, int * fp )}

Get the floating point attribute.

This service returns, at the address \textit{fp}, the value of the \textit{fp} attribute in the attribute object \textit{attr}.

The \textit{fp} attribute is a boolean attribute indicating whether a thread created with the attribute \textit{attr} may use floating-point hardware.

This service is a non-portable extension of the POSIX interface.

Parameters

<table>
<thead>
<tr>
<th>attr</th>
<th>attribute object;</th>
</tr>
</thead>
<tbody>
<tr>
<td>fp</td>
<td>address where the value of the \textit{fp} attribute will be stored on success.</td>
</tr>
</tbody>
</table>

Generated on Thu Jul 7 2016 13:25:13 for Xenomai POSIX skin API by Doxygen
Returns

0 on success;
an error number if:

- EINVAL, attr is invalid.

Valid contexts:

- kernel module initialization or cleanup routine;
- Xenomai kernel-space thread.

3.13.2.5 int pthread_attr_getinheritsched ( const pthread_attr_t * attr, int * inheritsched )

Get inheritsched attribute.

This service returns at the address inheritsched the value of the inheritsched attribute in the attribute object attr.

Threads created with this attribute set to PTHREAD_INHERIT_SCHED will use the same scheduling policy and priority as the thread calling pthread_create(). Threads created with this attribute set to PTHREAD_EXPLICIT_SCHED will use the value of the schedpolicy attribute as scheduling policy, and the value of the schedparam attribute as scheduling priority.

Parameters

<table>
<thead>
<tr>
<th>attr</th>
<th>attribute object;</th>
</tr>
</thead>
<tbody>
<tr>
<td>inheritsched</td>
<td>address where the value of the inheritsched attribute will be stored on success.</td>
</tr>
</tbody>
</table>

Returns

0 on success;
an error number if:

- EINVAL, attr is invalid.

See Also

Specification.

3.13.2.6 int pthread_attr_getname_np ( const pthread_attr_t * attr, const char ** name )

Get name attribute.

This service stores, at the address name, the value of the name attribute in the attribute object attr.

The name attribute is the name under which a thread created with the attribute object attr will appear under /proc/xenomai/sched.

The name returned by this function is only valid until the name is changed with pthread_attr_setname_np() or the attr object is destroyed with pthread_attr_destroy().

If name is NULL, a unique default name will be used.

This service is a non-portable extension of the POSIX interface.

Parameters
3.13 Thread creation attributes.

<table>
<thead>
<tr>
<th>attr</th>
<th>attribute object;</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>address where the value of the name attribute will be stored on success.</td>
</tr>
</tbody>
</table>

Returns

0 on success;
an error number if:
   - EINVAL, attr is invalid.

Valid contexts:
   - kernel module initialization or cleanup routine;
   - Xenomai kernel-space thread.

3.13.2.7 int pthread_attr_getschedparam ( const pthread_attr_t * attr, struct sched_param * par )

Get schedparam attribute.

This service stores, at the address par, the limited form of the schedparam attribute in the attribute object attr.

The limited form only defines the sched_priority member, that is sufficient to hold the scheduling parameter for SCHED_FIFO, SCHED_RR and SCHED_OTHER class members. Threads created with attr will use the value of this attribute as a scheduling priority if the attribute inheritedsched is set to PTHREAD_EXPLICIT_SCHED. Valid priorities range from 1 to 99.

pthread_attr_getschedparam_ex() should be used to retrieve the parameters for extended scheduling classes, such as SCHED_SPORADIC or SCHED_TP.

Parameters

<table>
<thead>
<tr>
<th>attr</th>
<th>attribute object;</th>
</tr>
</thead>
<tbody>
<tr>
<td>par</td>
<td>address where the value of the schedparam attribute will be stored on success.</td>
</tr>
</tbody>
</table>

Returns

0 on success;
an error number if:
   - EINVAL, attr is invalid.

See Also

Specification.

3.13.2.8 int pthread_attr_getschedparam_ex ( const pthread_attr_t * attr, struct sched_param_ex * par )

Get schedparam_ex extended attribute.

This service is an extended version of pthread_attr_getschedparam(), that also supports Xenomai-specific or additional POSIX scheduling policies, which are not available with the host Linux environment.

Typically, SCHED_SPORADIC or SCHED_TP parameters can be retrieved from this call.
Parameters

<table>
<thead>
<tr>
<th>attr</th>
<th>attribute object;</th>
</tr>
</thead>
<tbody>
<tr>
<td>par</td>
<td>address where the value of the extended schedparam_ex attribute will be stored on success.</td>
</tr>
</tbody>
</table>

Returns

- 0 on success;
- an error number if:
  - EINVAL, attr is invalid.

See Also

*Specification.*

3.13.2.9  int pthread_attr_getschedpolicy ( const pthread_attr_t * attr, int * policy )

Get schedpolicy attribute.

This service stores, at the address policy, the value of the policy attribute in the attribute object attr.

Threads created with the attribute object attr use the value of this attribute as scheduling policy if the inheritsched attribute is set to PTHREAD_EXPLICIT_SCHED. The value of this attribute is one of SCHED_FIFO, SCHED_RR, SCHED_SPORADIC, SCHED_TP or SCHED_OTHER.

Parameters

<table>
<thead>
<tr>
<th>attr</th>
<th>attribute object;</th>
</tr>
</thead>
<tbody>
<tr>
<td>policy</td>
<td>address where the value of the policy attribute in the attribute object attr will be stored on success.</td>
</tr>
</tbody>
</table>

Returns

- 0 on success;
- an error number if:
  - EINVAL, attr is invalid.

See Also

*Specification.*

3.13.2.10 int pthread_attr_getscope ( const pthread_attr_t * attr, int * scope )

Get contention scope attribute.

This service stores, at the address scope, the value of the scope attribute in the attribute object attr.

The scope attribute represents the scheduling contention scope of threads created with the attribute object attr. This implementation only supports the value PTHREAD_SCOPE_SYSTEM.

Parameters
3.13 Thread creation attributes.

| attr | attribute object; |
| scope | address where the value of the scope attribute will be stored on success. |

Returns

- 0 on success;
- an error number if:
  - EINVAL, attr is invalid.

See Also

- Specification.

3.13.2.11 int pthread_attr_getStackSize ( const pthread_attr_t * attr, size_t * stacksize )

Get stacksize attribute.

This service stores, at the address stacksize, the value of the stacksize attribute in the attribute object attr.

The stacksize attribute is used as the stack size of the threads created using the attribute object attr.

Parameters

| attr | attribute object; |
| stacksize | address where the value of the stacksize attribute will be stored on success. |

Returns

- 0 on success;
- an error number if:
  - EINVAL, attr is invalid.

See Also

- Specification.

3.13.2.12 int pthread_attr_init ( pthread_attr_t * attr )

Initialize a thread attributes object.

This service initializes the thread creation attributes structure pointed to by attr. Attributes are set to their default values (see Thread creation attributes.).

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

Parameters

| attr | address of the thread attributes object to initialize. |

Returns

- 0.

See Also

- Specification.
3.13.2.13 int pthread_attr_setaffinity_np ( pthread_attr_t * attr, xnarch_cpumask_t mask )

Set the processor affinity attribute.
This service sets to mask, the value of the affinity attribute in the attribute object attr.
The affinity attributes is a bitmask where bits set indicate processor where a thread created with the attribute attr may run. The least significant bit corresponds to the first logical processor.
This service is a non-portable extension of the POSIX interface.

Parameters

<table>
<thead>
<tr>
<th>attr</th>
<th>attribute object;</th>
</tr>
</thead>
<tbody>
<tr>
<td>mask</td>
<td>address where the value of the affinity attribute will be stored on success.</td>
</tr>
</tbody>
</table>

Returns

0 on success;
an error number if:
  - EINVAL, attr is invalid.

Valid contexts:

- kernel module initialization or cleanup routine;
- Xenomai kernel-space thread.

3.13.2.14 int pthread_attr_setdetachstate ( pthread_attr_t * attr, int detachstate )

Set detachstate attribute.
This service sets to detachstate the value of the detachstate attribute in the attribute object attr.
Valid values of this attribute are PTHREAD_CREATE_JOINABLE and PTHREAD_CREATE_DETACHED. A detached thread is a thread which control block is automatically reclaimed when it terminates. The control block of a joinable thread, on the other hand, is only reclaimed when joined with the service pthread_join().
A thread that was created joinable may be detached after creation by using the pthread_detach() service.

Parameters

<table>
<thead>
<tr>
<th>attr</th>
<th>attribute object;</th>
</tr>
</thead>
<tbody>
<tr>
<td>detachstate</td>
<td>value of the detachstate attribute.</td>
</tr>
</tbody>
</table>

Returns

0 on success;
an error number if:
  - EINVAL, the attribute object attr is invalid

See Also

Specification.

3.13.2.15 int pthread_attr_setfp_np ( pthread_attr_t * attr, int fp )

Set the floating point attribute.
This service set to *fp*, the value of the *fp* attribute in the attribute object *attr*.

The *fp* attribute is a boolean attribute indicating whether a thread created with the attribute *attr* may use floating-point hardware.

This service is a non-portable extension of the POSIX interface.

### Parameters

<table>
<thead>
<tr>
<th>attr</th>
<th>attribute object;</th>
</tr>
</thead>
<tbody>
<tr>
<td>fp</td>
<td>value of the <em>fp</em> attribute.</td>
</tr>
</tbody>
</table>

### Returns

- 0 on success;
- an error number if:
  - EINVAL, *attr* is invalid.

### Valid contexts:

- kernel module initialization or cleanup routine;
- Xenomai kernel-space thread.

### See Also

- Specification.

### 3.13.2.16 `int pthread_attr_setinheritsched ( pthread_attr_t * attr, int inheritsched )`  

Set inheritsched attribute.

This service set to *inheritsched* the value of the *inheritsched* attribute in the attribute object *attr*.

Threads created with this attribute set to PTHREAD_INHERIT_SCHED will use the same scheduling policy and priority as the thread calling `pthread_create()`. Threads created with this attribute set to PTHREAD_EXPLICIT_SCHED will use the value of the schedpolicy attribute as scheduling policy, and the value of the schedparam attribute as scheduling priority.

### Parameters

<table>
<thead>
<tr>
<th>attr</th>
<th>attribute object;</th>
</tr>
</thead>
<tbody>
<tr>
<td>inheritsched</td>
<td>value of the <em>inheritsched</em> attribute, PTHREAD_INHERIT_SCHED or PTHREAD_EXPLICIT_SCHED.</td>
</tr>
</tbody>
</table>

### Returns

- 0 on success;
- an error number if:
  - EINVAL, *attr* or *inheritsched* is invalid.

### See Also

- Specification.

### 3.13.2.17 `int pthread_attr_setname_np ( pthread_attr_t * attr, const char * name )`  

Set name attribute.

This service set to *name*, the value of the *name* attribute in the attribute object *attr*.

The *name* attribute is the name under which a thread created with the attribute object *attr* will appear under /proc/xenomai/sched.

If *name* is NULL, a unique default name will be used.

This service is a non-portable extension of the POSIX interface.
Parameters

<table>
<thead>
<tr>
<th>attr</th>
<th>attribute object;</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>value of the name attribute.</td>
</tr>
</tbody>
</table>

Returns

0 on success;
an error number if:

- EINVAL, attr is invalid;
- ENOMEM, insufficient memory exists in the system heap to duplicate the name string, increase CONFIG_XENO_OPT_SYS_HEAPSZ.

Valid contexts:

- kernel module initialization or cleanup routine;
- Xenomai kernel-space thread.

3.13.2.18   int pthread_attr_setschedparam ( pthread_attr_t * attr, const struct sched_param * par )

Set schedparam attribute.
This service set to par, the limited form of the schedparam attribute in the attribute object attr.
The limited form only defines the sched_priority member, that is sufficient to hold the scheduling parameter for SCHED_FIFO, SCHED_RR and SCHED_OTHER class members. Threads created with attr will use the value of this attribute as a scheduling priority if the attribute inheritsched is set to PTHREAD_EXPLICIT_SCHED. Valid priorities range from 1 to 99.

Parameters

<table>
<thead>
<tr>
<th>attr</th>
<th>attribute object;</th>
</tr>
</thead>
<tbody>
<tr>
<td>par</td>
<td>value of the schedparam attribute.</td>
</tr>
</tbody>
</table>

Returns

0 on success;
an error number if:

- EINVAL, attr or par is invalid.

See Also

Specification.

3.13.2.19   int pthread_attr_setschedparam_ex ( pthread_attr_t * attr, const struct sched_param_ex * par )

Set extended schedparam_ex attribute.
This service is an extended version of pthread_attr_setschedparam(), that also supports Xenomai-specific or additional POSIX scheduling policies, which are not available with the host Linux environment.
Typically, SCHED_SPORADIC or SCHED_TP parameters can be set using this call.
3.13 Thread creation attributes.

Parameters

<table>
<thead>
<tr>
<th>attr</th>
<th>attribute object;</th>
</tr>
</thead>
<tbody>
<tr>
<td>par</td>
<td>value of the schedparam attribute.</td>
</tr>
</tbody>
</table>

Returns

0 on success;
an error number if:

- EINVAL, attr or par is invalid.

See Also

Specification.

3.13.2.20 int pthread_attr_setschedpolicy ( pthread_attr_t * attr, int policy )

Set schedpolicy attribute.

This service set to policy the value of the policy attribute in the attribute object attr.

Threads created with the attribute object attr use the value of this attribute as scheduling policy if the inheritsched attribute is set to PTHREAD_EXPLICIT_SCHED. The value of this attribute is one of SCHED_FIFO, SCHED_RR, SCHED_SPORADIC, SCHED_TP or SCHED_OTHER.

Parameters

<table>
<thead>
<tr>
<th>attr</th>
<th>attribute object;</th>
</tr>
</thead>
<tbody>
<tr>
<td>policy</td>
<td>value of the policy attribute.</td>
</tr>
</tbody>
</table>

Returns

0 on success;
an error number if:

- EINVAL, attr or policy is invalid.

See Also

Specification.

3.13.2.21 int pthread_attr_setscope ( pthread_attr_t * attr, int scope )

Set contention scope attribute.

This service set to scope the value of the scope attribute in the attribute object attr.

The scope attribute represents the scheduling contention scope of threads created with the attribute object attr. This implementation only supports the value PTHREAD_SCOPE_SYSTEM.

Parameters

| attr   | attribute object; |

See Also

Specification.
Returns

0 on success;
an error number if:

- ENOTSUP, `scope` is an unsupported value of the `scope` attribute.
- EINVAL, `attr` is invalid.

See Also

- Specification.

3.13.22 `int pthread_attr_setstacksize ( pthread_attr_t *attr, size_t stacksize )`

Set stacksize attribute.

This service set to `stacksize`, the value of the `stacksize` attribute in the attribute object `attr`.

The `stacksize` attribute is used as the stack size of the threads created using the attribute object `attr`.

The minimum value for this attribute is PTHREAD_STACK_MIN.

Parameters

<table>
<thead>
<tr>
<th>attr</th>
<th>attribute object;</th>
</tr>
</thead>
<tbody>
<tr>
<td>stacksize</td>
<td>value of the <code>stacksize</code> attribute.</td>
</tr>
</tbody>
</table>

Returns

0 on success;
an error number if:

- EINVAL, `attr` or `stacksize` is invalid.

See Also

- Specification.
3.14 Thread-specific data.

Thread-specific data.

Functions

- `int pthread_key_create (pthread_key_t *key, void (*)(void *))`
  
  Create a thread-specific data key.

- `int pthread_setspecific (pthread_key_t key, const void *value)`
  
  Associate a thread-specific value with the specified key.

- `void * pthread_getspecific (pthread_key_t key)`
  
  Get the thread-specific value bound to the specified key.

- `int pthread_key_delete (pthread_key_t key)`
  
  Delete a thread-specific data key.

3.14.1 Detailed Description

Thread-specific data. Programs often need global or static variables that have different values in different threads. Since threads share one memory space, this cannot be achieved with regular variables. Thread-specific data is the POSIX threads answer to this need.

Each thread possesses a private memory block, the thread-specific data area, or TSD area for short. This area is indexed by TSD keys. The TSD area associates values of type `void *` to TSD keys. TSD keys are common to all threads, but the value associated with a given TSD key can be different in each thread.

When a thread is created, its TSD area initially associates `NULL` with all keys.

The services documented here are valid in kernel-space context; when called in user-space, the underlying Linux threading library (LinuxThreads or NPTL) services are used.

3.14.2 Function Documentation

3.14.2.1 `void * pthread_getspecific ( pthread_key_t key )`

Get the thread-specific value bound to the specified key.

This service returns the value associated, for the calling thread, with the key `key`.

Parameters

| key | TSD key, obtained with `pthread_key_create()` |

Returns

- the value associated with `key`;
- `NULL` if the context is invalid.

Valid contexts:

- Xenomai POSIX skin kernel-space thread.

See Also

- `Specification`
3.14.2.2 int pthread_key_create ( pthread_key_t *key, void(*)(void *) destructor )

Create a thread-specific data key.
This service create a TSD key. The NULL value is associated for all threads with the new key and the new key is returned at the address key. If destructor is not null, it is executed when a thread is terminated as long as the datum associated with the key is not NULL, up to PTHREAD_DESTRUCTOR_ITERATIONS times.

Parameters

<table>
<thead>
<tr>
<th>key</th>
<th>address where the new key will be stored on success;</th>
</tr>
</thead>
<tbody>
<tr>
<td>destructor</td>
<td>function to be invoked when a thread terminates and has a non NULL value associated with the new key.</td>
</tr>
</tbody>
</table>

Returns

0 on success;
an error number if:
- EAGAIN, the total number of keys PTHREAD_KEYS_MAX TSD has been exceeded;
- ENOMEM, insufficient memory exists in the system heap to create a new key, increase CONFIG_XENO_OPT_SYS_HEAPSZ.

See Also

Specification.

3.14.2.3 int pthread_key_delete ( pthread_key_t key )

Delete a thread-specific data key.
This service deletes the TSD key key. Note that the key destructor function is not called, so, if any thread has a value associated with key that is a pointer to dynamically allocated memory, the application has to manage to free that memory by other means.

Parameters

| key | the TSD key to be destroyed. |

Returns

0 on success;
an error number if:
- EINVAL, key is invalid.

See Also

Specification.

3.14.2.4 int pthread_setspecific ( pthread_key_t key, const void *value )

Associate a thread-specific value with the specified key.
This service associates, for the calling thread, the value value to the key key.
### Parameters

<table>
<thead>
<tr>
<th>key</th>
<th>TSD key, obtained with <code>pthread_key_create()</code>;</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>value.</td>
</tr>
</tbody>
</table>

### Returns

- 0 on success;
- an error number if:
  - EPERM, the caller context is invalid;
  - EINVAL, key is invalid.

### Valid contexts:

- Xenomai POSIX skin kernel-space thread.

### See Also

- Specification.
Chapter 4

File Documentation

4.1 ksrc/skins/posix/syscall.c File Reference

This file is part of the Xenomai project.

#include <linux/err.h>
#include <asm/xenomai/wrappers.h>
#include <nucleus/jhash.h>
#include <nucleus/ppd.h>
#include <nucleus/sys_ppd.h>
#include <posix/syscall.h>
#include <posix/posix.h>
#include <posix/thread.h>
#include <posix/mutex.h>
#include <posix/cond.h>
#include <posix/mq.h>
#include <posix/intr.h>
#include <posix/registry.h>
#include <posix/sem.h>
#include <posix/shm.h>
#include <posix/timer.h>

4.1.1 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

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.
Index

clock_getres
  Clocks and timers services., 10

clock_gettime
  Clocks and timers services., 10

clock_nanosleep
  Clocks and timers services., 10

clock_settime
  Clocks and timers services., 11

Clocks and timers services., 9
  clock_getres, 10
  clock_gettime, 10
  clock_nanosleep, 10
  clock_settime, 11
  nanosleep, 12
  timer_create, 12
  timer_delete, 13
  timer_getoverrun, 14
  timer_gettime, 14
  timer_settime, 14

close
  Shared memory services., 60

Condition variables services., 16
  pthread_cond_broadcast, 17
  pthread_cond_destroy, 18
  pthread_cond_init, 18
  pthread_cond_signal, 19
  pthread_cond_timedwait, 19
  pthread_cond_wait, 20
  pthread_condattr_destroy, 21
  pthread_condattr_getclock, 21
  pthread_condattr_getpshared, 22
  pthread_condattr_init, 22
  pthread_condattr_setclock, 23
  pthread_condattr_setpshared, 23

ftruncate
  Shared memory services., 61

init_step
  Threads management services., 75

Interruptions management services., 25
  pthread_intr_attach_np, 25
  pthread_intr_control_np, 26
  pthread_intr_detach_np, 27
  pthread_intr_wait_np, 27

ksrc/skins/posix/syscall.c, 97

Message queues services., 30
  mq_close, 30
  mq_getattr, 31
  mq_notify, 31
  mq_open, 32
  mq_receive, 33
  mq_send, 34
  mq_setattr, 35
  mq_timedreceive, 35
  mq_timedsend, 36
  mq_unlink, 37

mmap
  Shared memory services., 61

mq_close
  Message queues services., 30

mq_getattr
  Message queues services., 31

mq_notify
  Message queues services., 31

mq_open
  Message queues services., 32

mq_receive
  Message queues services., 33

mq_send
  Message queues services., 34

mq_setattr
  Message queues services., 35

mq_timedreceive
  Message queues services., 35

mq_timedsend
  Message queues services., 36

mq_unlink
  Message queues services., 37

munmap
  Shared memory services., 63

Mutex services., 38
  pthread_mutex_destroy, 39
  pthread_mutex_init, 39
  pthread_mutex_lock, 39
  pthread_mutex_timedlock, 40
  pthread_mutex_trylock, 41
  pthread_mutex_unlock, 41
  pthread_mutexattr_destroy, 42
  pthread_mutexattr_getprotocol, 42
  pthread_mutexattr_getpshared, 43
  pthread_mutexattr_gettype, 43
  pthread_mutexattr_init, 44
  pthread_mutexattr_setprotocol, 44
  pthread_mutexattr_setpshared, 45
  pthread_mutexattr_settype, 45

nanosleep
Clocks and timers services., 12
POSIX skin., 29
pthread_attr_destroy
Thread creation attributes., 82
pthread_attr_getaffinity_np
Thread creation attributes., 82
pthread_attr_getdetachstate
Thread creation attributes., 83
pthread_attr_getfp_np
Thread creation attributes., 83
pthread_attr_getinheritsched
Thread creation attributes., 84
pthread_attr_getname_np
Thread creation attributes., 84
pthread_attr_getsched
Thread creation attributes., 85
pthread_attr_getschedparam
Thread creation attributes., 85
pthread_attr_getschedpolicy
Thread creation attributes., 86
pthread_attr_getscope
Thread creation attributes., 86
pthread_attr_getstacksize
Thread creation attributes., 87
pthread_attr_init
Thread creation attributes., 87
pthread_attr_setaffinity_np
Thread creation attributes., 87
pthread_attr_setdetachstate
Thread creation attributes., 88
pthread_attr_setfp_np
Thread creation attributes., 88
pthread_attr_setinheritsched
Thread creation attributes., 89
pthread_attr_setname_np
Thread creation attributes., 89
pthread_attr_setschedparam
Thread creation attributes., 90
pthread_attr_setschedpolicy
Thread creation attributes., 91
pthread_attr_setscope
Thread creation attributes., 91
pthread_attr_setstacksize
Thread creation attributes., 92
pthread_cancel
Thread cancellation., 6
pthread_cleanup_pop
Thread cancellation., 6
pthread_cleanup_push
Thread cancellation., 6
pthread_cond_broadcast
Condition variables services., 17
pthread_cond_destroy
Condition variables services., 18
pthread_cond_init
Condition variables services., 18
pthread_cond_signal
Condition variables services., 19
pthread_cond_timedwait
Condition variables services., 19
pthread_cond_wait
Condition variables services., 20
pthread_condattr_destroy
Condition variables services., 21
pthread_condattr_getclock
Condition variables services., 21
pthread_condattr_getpshared
Condition variables services., 22
pthread_condattr_init
Condition variables services., 22
pthread_condattr_setclock
Condition variables services., 23
pthread_condattr_setpshared
Condition variables services., 23
pthread_create
Threads management services., 75
pthread_detach
Threads management services., 76
pthread_equal
Threads management services., 76
pthread_exit
Threads management services., 77
pthread_getschedparam
Threads scheduling services., 48
pthread_getschedparam_ex
Threads scheduling services., 48
pthread_getspecific
Thread-specific data., 93
pthread_intr_attach_np
Interruptions management services., 25
pthread_intr_control_np
Interruptions management services., 26
pthread_intr_detach_np
Interruptions management services., 27
pthread_intr_wait_np
Interruptions management services., 27
pthread_join
Threads management services., 77
pthread_key_create
Thread-specific data., 93
pthread_key_delete
Thread-specific data., 94
pthread_kill
Signals services., 67
pthread_make_periodic_np
Threads management services., 78
pthread_mutex_destroy
Mutex services., 39
pthread_mutex_init
Mutex services., 39
pthread_mutex_lock
Mutex services., 39
pthread_mutex_timedlock
Mutex services., 40
pthread_mutex_trylock
Mutex services., 41
pthread_mutex_unlock
Mutex services., 41
pthread_mutexattr_destroy
Mutex services., 42
pthread_mutexattr_getprotocol
Mutex services., 42
pthread_mutexattr_getpshared
Mutex services., 43
pthread_mutexattr_gettype
Mutex services., 43
pthread_mutexattr_init
Mutex services., 44
pthread_mutexattr_setprotocol
Mutex services., 44
pthread_mutexattr_setpshared
Mutex services., 45
pthread_mutexattr_settype
Mutex services., 45
pthread_self
Threads management services., 78
pthread_set_mode_np
Threads management services., 79
pthread_set_name_np
Threads management services., 79
pthread_setcancelstate
Thread cancellation., 7
pthread_setcanceltype
Thread cancellation., 8
pthread_setschedparam
Threads scheduling services., 48
pthread_setschedparam_ex
Threads scheduling services., 49
pthread_setspecific
Thread-specific data., 94
pthread_sigmask
Signals services., 67
pthread_sigqueue_np
Signals services., 68
pthread_testcancel
Thread cancellation., 8
pthread_wait_np
Threads management services., 80
sched_get_priority_max
Threads scheduling services., 50
sched_get_priority_min
Threads scheduling services., 50
sched_rr_get_interval
Threads scheduling services., 51
sched_setconfig_np
Threads scheduling services., 51
sched_yield
Threads scheduling services., 52
sem_close
Semaphores services., 53
sem_destroy
Semaphores services., 54
sem_getvalue
Semaphores services., 54
sem_init
Semaphores services., 55
sem_open
Semaphores services., 55
sem_post
Semaphores services., 56
sem_timedwait
Semaphores services., 57
sem_trywait
Semaphores services., 57
sem_unlink
Semaphores services., 58
sem_wait
Semaphores services., 58
Shared memory services., 60
close, 60
tfruncate, 61
mmap, 61
munmap, 63
shm_open, 63
shm_unlink, 64
shm_open
Shared memory services., 63
shm_unlink
Shared memory services., 64
sigaction
Signals services., 68
sigaddset
Signals services., 69
sigdelset
Signals services., 70
sigemptyset
Signals services., 70
sigfillset
Signals services., 70
sigismember
Signals services., 71
Signals services., 66
pthread_kill, 67
pthread_sigmask, 67
pthread_sigqueue_np, 68
sigaction, 68
sigaddset, 69
sigdelset, 70
sigemptyset, 70
sigfillset, 70
sigismember, 71
sigpending, 71
sigtimedwait, 72
sigwait, 72
sigwaitinfo, 73

sigpending
  Signals services., 71
sigtimedwait
  Signals services., 72
sigwait
  Signals services., 72
sigwaitinfo
  Signals services., 73

Thread cancellation., 5
  pthread_cancel, 6
  pthread_cleanup_pop, 6
  pthread_cleanup_push, 6
  pthread_setcancelstate, 7
  pthread_setcanceltype, 8
  pthread_testcancel, 8

Thread creation attributes., 81
  pthread_attr_destroy, 82
  pthread_attr_getaffinity_np, 82
  pthread_attr_getdetachstate, 83
  pthread_attr_getfp_np, 83
  pthread_attr_getinheritsched, 84
  pthread_attr_getname_np, 84
  pthread_attr_getschedparam, 85
  pthread_attr_getschedparam_ex, 85
  pthread_attr_getschedpolicy, 86
  pthread_attr_getscope, 86
  pthread_attr_getstacksize, 87
  pthread_attr_init, 87
  pthread_attr_setaffinity_np, 87
  pthread_attr_setdetachstate, 88
  pthread_attr_setfp_np, 88
  pthread_attr_setinheritsched, 89
  pthread_attr_setname_np, 89
  pthread_attr_setschedparam, 90
  pthread_attr_setschedparam_ex, 90
  pthread_attr_setschedpolicy, 91
  pthread_attr_setscope, 91
  pthread_attr_setstacksize, 92

Thread-specific data., 93
  pthread_getspecific, 93
  pthread_key_create, 93
  pthread_key_delete, 94
  pthread_setspecific, 94

Threads management services., 74
  init_step, 75
  pthread_create, 75
  pthread_detach, 76
  pthread_equal, 76
  pthread_exit, 77
  pthread_join, 77
  pthread_make_periodic_np, 78
  pthread_self, 78
  pthread_set_mode_np, 79
  pthread_set_name_np, 79
  pthread_wait_np, 80

Threads scheduling services., 47
  pthread_getschedparam, 48
  pthread_getschedparam_ex, 48
  pthread_setschedparam, 48
  pthread_setschedparam_ex, 49
  sched_get_priority_max, 50
  sched_get_priority_min, 50
  sched_rr_get_interval, 51
  sched_setconfig_np, 51
  sched_yield, 52

timer_create
  Clocks and timers services., 12

timer_delete
  Clocks and timers services., 13

timer_getoverrun
  Clocks and timers services., 14

timer_gettime
  Clocks and timers services., 14

timer_settime
  Clocks and timers services., 14