Xenomai core

2.99.0

Generated by Doxygen 1.7.1

Thu Jan 5 2012 10:14:32

Contents

1 Module Index **1**

1.1 Modules 1

2 Data Structure Index **3**

2.1 Data Structures 3

3 File Index **5**

3.1 File List 5

4 Module Documentation **7**

4.1 Thread state flags. 7

4.1.1 Detailed Description 8

4.1.2 Define Documentation 8

4.1.2.1 XNHELD 8

4.1.2.2 XNLOCK 9

4.1.2.3 XNMIGRATE 9

4.1.2.4 XNPEND 9

4.1.2.5 XNREADY 9

4.1.2.6 XNSUSP 9

4.2 Thread information flags. 9

4.2.1 Detailed Description 10

4.3 Buffer descriptors. 10

4.3.1 Detailed Description 11

4.3.2 Function Documentation 13

4.3.2.1 xnbufd_copy_from_kmem 13

4.3.2.2 xnbufd_copy_to_kmem 14

4.3.2.3 xnbufd_invalidate 15

4.3.2.4 xnbufd_map_kread 15

4.3.2.5 xnbufd_map_kwrite 16

4.3.2.6 xnbufd_map_uread . . . 16
4.3.2.7 xnbufd_map_uwrite . . . 17
4.3.2.8 xnbufd_reset . . . 17
4.3.2.9 xnbufd_unmap_kread . . . 18
4.3.2.10 xnbufd_unmap_kwrite . . . 18
4.3.2.11 xnbufd_unmap_uread . . . 19
4.3.2.12 xnbufd_unmap_uwrite . . . 19
4.4 System clock services. . . . 20
4.4.1 Function Documentation . . . 20
4.4.1.1 xnclock_adjust . . . 20
4.5 Debugging services. . . . 21
4.6 Dynamic memory allocation services. . . . 21
4.6.1 Detailed Description . . . 22
4.6.2 Function Documentation . . . 22
4.6.2.1 xnheap_alloc . . . 22
4.6.2.2 xnheap_destroy . . . 23
4.6.2.3 xnheap_extend . . . 23
4.6.2.4 xnheap_free . . . 24
4.6.2.5 xnheap_init . . . 25
4.6.2.6 xnheap_schedule_free . . . 26
4.6.2.7 xnheap_set_label . . . 26
4.6.2.8 xnheap_test_and_free . . . 27
4.7 Interrupt management. . . . 27
4.7.1 Detailed Description . . . 28
4.7.2 Function Documentation . . . 28
4.7.2.1 xnintr_affinity . . . 28
4.7.2.2 xnintr_attach . . . 28
4.7.2.3 xnintr_destroy . . . 29
4.7.2.4 xnintr_detach . . . 30
4.7.2.5 xnintr_disable . . . 30
4.7.2.6 xnintr_enable . . . 31
4.7.2.7 xnintr_init . . . 31
4.8 Lightweight key-to-object mapping service . . . 33
4.8.1 Detailed Description . . . 33
4.8.2 Function Documentation . . . 33
4.8.2.1 xnmap_create . . . 33

4.8.2.2 xnmap_delete . . . 34
4.8.2.3 xnmap_enter . . . 35
4.8.2.4 xnmap_fetch . . . 35
4.8.2.5 xnmap_fetch_nocheck . . . 36
4.8.2.6 xnmap_remove . . . 36
4.9 Real-time pod services. . . . 37
4.9.1 Detailed Description . . . 39
4.9.2 Function Documentation . . . 39
4.9.2.1 __xnpod_reset_thread . . . 39
4.9.2.2 xnpod_abort_thread . . . 39
4.9.2.3 xnpod_add_hook . . . 40
4.9.2.4 xnpod_delete_thread . . . 41
4.9.2.5 xnpod_disable_timesource . . . 41
4.9.2.6 xnpod_dispatch_signals . . . 42
4.9.2.7 xnpod_enable_timesource . . . 42
4.9.2.8 xnpod_handle_exception . . . 43
4.9.2.9 xnpod_init . . . 43
4.9.2.10 xnpod_init_thread . . . 43
4.9.2.11 xnpod_migrate_thread . . . 45
4.9.2.12 xnpod_remove_hook . . . 45
4.9.2.13 xnpod_resume_thread . . . 46
4.9.2.14 xnpod_schedule . . . 47
4.9.2.15 xnpod_set_thread_mode . . . 48
4.9.2.16 xnpod_set_thread_periodic . . . 48
4.9.2.17 xnpod_set_thread_schedparam . . . 49
4.9.2.18 xnpod_set_thread_tslice . . . 50
4.9.2.19 xnpod_shutdown . . . 51
4.9.2.20 xnpod_start_thread . . . 51
4.9.2.21 xnpod_stop_thread . . . 53
4.9.2.22 xnpod_suspend_thread . . . 53
4.9.2.23 xnpod_unblock_thread . . . 54
4.9.2.24 xnpod_wait_thread_period . . . 55
4.9.2.25 xnpod_welcome_thread . . . 56
4.10 Registry services. . . . 56
4.10.1 Detailed Description . . . 57
4.10.2 Function Documentation . . . 57

4.10.2.1 xnregistry_bind . . . 57
4.10.2.2 xnregistry_enter . . . 58
4.10.2.3 xnregistry_fetch . . . 59
4.10.2.4 xnregistry_get . . . 60
4.10.2.5 xnregistry_put . . . 60
4.10.2.6 xnregistry_remove . . . 61
4.10.2.7 xnregistry_remove_safe . . . 61
4.11 File descriptors events multiplexing services. . . . 62
4.11.1 Detailed Description . . . 63
4.11.2 Function Documentation . . . 63
4.11.2.1 xnselect . . . 63
4.11.2.2 xnselect_bind . . . 64
4.11.2.3 xnselect_destroy . . . 65
4.11.2.4 xnselect_init . . . 65
4.11.2.5 xnselect_signal . . . 65
4.11.2.6 xnselector_destroy . . . 65
4.11.2.7 xnselector_init . . . 66
4.12 Real-time shadow services. . . . 66
4.12.1 Detailed Description . . . 66
4.12.2 Function Documentation . . . 66
4.12.2.1 xnshadow_harden . . . 66
4.12.2.2 xnshadow_map . . . 67
4.12.2.3 xnshadow_ppd_get . . . 68
4.12.2.4 xnshadow_relax . . . 68
4.13 Thread synchronization services. . . . 69
4.13.1 Detailed Description . . . 70
4.13.2 Function Documentation . . . 70
4.13.2.1 xnsynch_acquire . . . 70
4.13.2.2 xnsynch_clear_boost . . . 71
4.13.2.3 xnsynch_flush . . . 71
4.13.2.4 xnsynch_forget_sleeper . . . 72
4.13.2.5 xnsynch_init . . . 73
4.13.2.6 xnsynch_peek_pendq . . . 74
4.13.2.7 xnsynch_release . . . 74
4.13.2.8 xnsynch_release_all_ownerships . . . 75
4.13.2.9 xnsynch_requeue_sleeper . . . 75

4.13.2.10 xnsynch_sleep_on . . . 76
4.13.2.11 xnsynch_wakeup_one_sleeper . . . 76
4.13.2.12 xnsynch_wakeup_this_sleeper . . . 77
4.14 Timer services. . . . 78
4.14.1 Detailed Description . . . 79
4.14.2 Function Documentation . . . 79
4.14.2.1 xntimer_destroy . . . 79
4.14.2.2 xntimer_freeze . . . 79
4.14.2.3 xntimer_get_date . . . 80
4.14.2.4 xntimer_get_interval . . . 80
4.14.2.5 xntimer_get_overruns . . . 81
4.14.2.6 xntimer_get_timeout . . . 81
4.14.2.7 xntimer_init . . . 81
4.14.2.8 xntimer_start . . . 82
4.14.2.9 xntimer_stop . . . 83
4.14.2.10 xntimer_tick . . . 83
4.15 Virtual file services . . . 83
4.15.1 Detailed Description . . . 85
4.15.2 Function Documentation . . . 86
4.15.2.1 xnvfile_destroy . . . 86
4.15.2.2 xnvfile_get_blob . . . 86
4.15.2.3 xnvfile_get_integer . . . 86
4.15.2.4 xnvfile_get_string . . . 87
4.15.2.5 xnvfile_init_dir . . . 87
4.15.2.6 xnvfile_init_link . . . 88
4.15.2.7 xnvfile_init_regular . . . 88
4.15.2.8 xnvfile_init_snapshot . . . 89
4.15.3 Variable Documentation . . . 90
4.15.3.1 nkvfroot . . . 90
4.15.3.2 nkvfroot . . . 90
4.16 Sched . . . 90
4.16.1 Function Documentation . . . 91
4.16.1.1 xnsched_rotate . . . 91

5 Data Structure Documentation **93**
5.1 xnpod Struct Reference . . . 93
5.1.1 Detailed Description . . . 94

5.1.2 Field Documentation . . . 94
5.1.2.1 refcnt . . . 94
5.1.2.2 sched . . . 94
5.1.2.3 status . . . 94
5.1.2.4 tdeleteq . . . 94
5.1.2.5 threadq . . . 94
5.1.2.6 timerlck . . . 94
5.1.2.7 tstartq . . . 94
5.1.2.8 tswitchq . . . 95
5.2 xnsched Struct Reference . . . 95
5.2.1 Detailed Description . . . 95
5.2.2 Field Documentation . . . 95
5.2.2.1 curr . . . 95
5.2.2.2 htimer . . . 95
5.2.2.3 inesting . . . 95
5.2.2.4 lflags . . . 96
5.2.2.5 rootcb . . . 96
5.2.2.6 rt . . . 96
5.2.2.7 status . . . 96
5.3 xnthread_info Struct Reference . . . 96
5.3.1 Detailed Description . . . 97
5.3.2 Field Documentation . . . 97
5.3.2.1 affinity . . . 97
5.3.2.2 bprio . . . 97
5.3.2.3 cprio . . . 97
5.3.2.4 cpu . . . 97
5.3.2.5 ctxswitches . . . 97
5.3.2.6 exectime . . . 98
5.3.2.7 modeswitches . . . 98
5.3.2.8 name . . . 98
5.3.2.9 pagefaults . . . 98
5.3.2.10 relpoint . . . 98
5.3.2.11 state . . . 98
5.3.2.12 syscalls . . . 98
5.4 xnvfile_lock_ops Struct Reference . . . 98
5.4.1 Detailed Description . . . 99

5.4.2 Field Documentation . . . 99
5.4.2.1 get . . . 99
5.4.2.2 put . . . 99
5.5 xnvfile_regular_iterator Struct Reference . . . 99
5.5.1 Detailed Description . . . 100
5.5.2 Field Documentation . . . 100
5.5.2.1 pos . . . 100
5.5.2.2 private . . . 100
5.5.2.3 seq . . . 100
5.5.2.4 vfile . . . 100
5.6 xnvfile_regular_ops Struct Reference . . . 100
5.6.1 Detailed Description . . . 101
5.6.2 Field Documentation . . . 101
5.6.2.1 begin . . . 101
5.6.2.2 end . . . 101
5.6.2.3 next . . . 102
5.6.2.4 rewind . . . 102
5.6.2.5 show . . . 102
5.6.2.6 store . . . 103
5.7 xnvfile_rev_tag Struct Reference . . . 103
5.7.1 Detailed Description . . . 104
5.7.2 Field Documentation . . . 104
5.7.2.1 rev . . . 104
5.8 xnvfile_snapshot Struct Reference . . . 104
5.8.1 Detailed Description . . . 104
5.9 xnvfile_snapshot_iterator Struct Reference . . . 105
5.9.1 Detailed Description . . . 106
5.9.2 Field Documentation . . . 106
5.9.2.1 databuf . . . 106
5.9.2.2 endfn . . . 106
5.9.2.3 nrdata . . . 106
5.9.2.4 private . . . 106
5.9.2.5 seq . . . 106
5.9.2.6 vfile . . . 106
5.10 xnvfile_snapshot_ops Struct Reference . . . 106
5.10.1 Detailed Description . . . 107

5.10.2 Field Documentation . . . 107
5.10.2.1 begin . . . 107
5.10.2.2 end . . . 107
5.10.2.3 next . . . 108
5.10.2.4 rewind . . . 108
5.10.2.5 show . . . 109
5.10.2.6 store . . . 109

6 File Documentation **111**

6.1 include/cobalt/nucleus/bufd.h File Reference . . . 111
6.1.1 Detailed Description . . . 112
6.2 include/cobalt/nucleus/clock.h File Reference . . . 113
6.2.1 Detailed Description . . . 113
6.3 include/cobalt/nucleus/hostrt.h File Reference . . . 114
6.3.1 Detailed Description . . . 115
6.4 include/cobalt/nucleus/map.h File Reference . . . 116
6.4.1 Detailed Description . . . 117
6.5 include/cobalt/nucleus/pod.h File Reference . . . 117
6.5.1 Detailed Description . . . 119
6.6 include/cobalt/nucleus/registry.h File Reference . . . 120
6.6.1 Detailed Description . . . 121
6.7 include/cobalt/nucleus/sched-idle.h File Reference . . . 121
6.7.1 Detailed Description . . . 122
6.8 include/cobalt/nucleus/sched-rt.h File Reference . . . 122
6.8.1 Detailed Description . . . 122
6.9 include/cobalt/nucleus/sched-sporadic.h File Reference . . . 123
6.9.1 Detailed Description . . . 123
6.10 include/cobalt/nucleus/sched-tp.h File Reference . . . 123
6.10.1 Detailed Description . . . 124
6.11 include/cobalt/nucleus/sched.h File Reference . . . 124
6.11.1 Detailed Description . . . 125
6.12 include/cobalt/nucleus/select.h File Reference . . . 125
6.12.1 Detailed Description . . . 127
6.13 include/cobalt/nucleus/timer.h File Reference . . . 128
6.13.1 Detailed Description . . . 129
6.14 include/cobalt/nucleus/vdso.h File Reference . . . 129
6.14.1 Detailed Description . . . 130

6.15 include/cobalt/nucleus/vfile.h File Reference 130
6.15.1 Detailed Description 132
6.16 kernel/cobalt/nucleus/bufd.c File Reference 133
6.16.1 Detailed Description 134
6.17 kernel/cobalt/nucleus/clock.c File Reference 134
6.17.1 Detailed Description 135
6.18 kernel/cobalt/nucleus/debug.c File Reference 135
6.18.1 Detailed Description 135
6.19 kernel/cobalt/nucleus/heap.c File Reference 136
6.19.1 Detailed Description 137
6.20 kernel/cobalt/nucleus/intr.c File Reference 137
6.20.1 Detailed Description 138
6.21 kernel/cobalt/nucleus/map.c File Reference 139
6.21.1 Detailed Description 139
6.22 kernel/cobalt/nucleus/pod.c File Reference 140
6.22.1 Detailed Description 142
6.23 kernel/cobalt/nucleus/registry.c File Reference 142
6.23.1 Detailed Description 143
6.24 kernel/cobalt/nucleus/sched-idle.c File Reference 143
6.24.1 Detailed Description 144
6.25 kernel/cobalt/nucleus/sched-rt.c File Reference 144
6.25.1 Detailed Description 145
6.26 kernel/cobalt/nucleus/sched-sporadic.c File Reference 145
6.26.1 Detailed Description 146
6.27 kernel/cobalt/nucleus/sched-tp.c File Reference 146
6.27.1 Detailed Description 147
6.28 kernel/cobalt/nucleus/sched.c File Reference 148
6.28.1 Detailed Description 148
6.29 kernel/cobalt/nucleus/select.c File Reference 148
6.29.1 Detailed Description 149
6.30 kernel/cobalt/nucleus/shadow.c File Reference 150
6.30.1 Detailed Description 150
6.31 kernel/cobalt/nucleus/synch.c File Reference 151
6.31.1 Detailed Description 152
6.32 kernel/cobalt/nucleus/timer.c File Reference 153
6.32.1 Detailed Description 154

6.33 kernel/cobalt/nucleus/vfile.c File Reference 154
6.33.1 Detailed Description 155

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

Thread state flags. 7
Thread information flags. 9
Buffer descriptors. 10
System clock services. 20
Debugging services. 21
Dynamic memory allocation services. 21
Interrupt management. 27
Lightweight key-to-object mapping service 33
Real-time pod services. 37
Registry services. 56
File descriptors events multiplexing services. 62
Real-time shadow services. 66
Thread synchronization services. 69
Timer services. 78
Virtual file services 83
Sched 90

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

xnpod (Real-time pod descriptor) 93
xnsched (Scheduling information structure) 95
xnthread_info (Structure containing thread information) 96
xnvfile_lock_ops (Vfile locking operations) 98
xnvfile_regular_iterator (Regular vfile iterator) 99
xnvfile_regular_ops (Regular vfile operation descriptor) 100
xnvfile_rev_tag (Snapshot revision tag) 103
xnvfile_snapshot (Snapshot vfile descriptor) 104
xnvfile_snapshot_iterator (Snapshot-driven vfile iterator) 105
xnvfile_snapshot_ops (Snapshot vfile operation descriptor) 106

Chapter 3

File Index

3.1 File List

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

include/cobalt/nucleus/**assert.h** . . . . . **??**
include/cobalt/nucleus/**bheap.h** . . . . . **??**
include/cobalt/nucleus/bufd.h 111
include/cobalt/nucleus/clock.h 113
include/cobalt/nucleus/**compiler.h** . . . . . **??**
include/cobalt/nucleus/**debug.h** . . . . . **??**
include/cobalt/nucleus/**heap.h** . . . . . **??**
include/cobalt/nucleus/hostrt.h (Definitions for global semaphore heap shared objects) 114
include/cobalt/nucleus/**intr.h** . . . . . **??**
include/cobalt/nucleus/map.h 116
include/cobalt/nucleus/**pipe.h** . . . . . **??**
include/cobalt/nucleus/pod.h (Real-time pod interface header) 117
include/cobalt/nucleus/**ppd.h** . . . . . **??**
include/cobalt/nucleus/**queue.h** . . . . . **??**
include/cobalt/nucleus/registry.h (This file is part of the Xenomai project) 120
include/cobalt/nucleus/sched-idle.h (Definitions for the IDLE scheduling class) 121
include/cobalt/nucleus/sched-rt.h (Definitions for the RT scheduling class) 122
include/cobalt/nucleus/sched-sporadic.h (Definitions for the SSP scheduling class) . . . 123
include/cobalt/nucleus/sched-tp.h (Definitions for the TP scheduling class) 123
include/cobalt/nucleus/sched.h (Scheduler interface header) 124
include/cobalt/nucleus/**schedparam.h** . . . . . **??**
include/cobalt/nucleus/**schedqueue.h** . . . . . **??**
include/cobalt/nucleus/select.h (File descriptors events multiplexing header) 125
include/cobalt/nucleus/**seqlock.h** . . . . . **??**
include/cobalt/nucleus/**shadow.h** . . . . . **??**
include/cobalt/nucleus/**stat.h** . . . . . **??**
include/cobalt/nucleus/**synch.h** . . . . . **??**
include/cobalt/nucleus/**sys_ppd.h** . . . . . **??**
include/cobalt/nucleus/**system.h** . . . . . **??**
include/cobalt/nucleus/**thread.h** . . . . . **??**
include/cobalt/nucleus/timer.h 128
include/cobalt/nucleus/**trace.h** . . . . . **??**
include/cobalt/nucleus/**types.h** . . . . . **??**

include/cobalt/nucleus/vdso.h (Definitions for global semaphore heap shared objects) . 129
include/cobalt/nucleus/**version.h** . . . . . **??**
include/cobalt/nucleus/vfile.h (This file is part of the Xenomai project) 130
kernel/cobalt/nucleus/bufd.c 133
kernel/cobalt/nucleus/clock.c 134
kernel/cobalt/nucleus/debug.c (Debug services) 135
kernel/cobalt/nucleus/heap.c (Dynamic memory allocation services) 136
kernel/cobalt/nucleus/intr.c (Interrupt management) 137
kernel/cobalt/nucleus/map.c 139
kernel/cobalt/nucleus/pod.c (Real-time pod services) 140
kernel/cobalt/nucleus/registry.c (This file is part of the Xenomai project) 142
kernel/cobalt/nucleus/sched-idle.c (Idle scheduling class implementation (i.e. Linux placeholder)) 143
kernel/cobalt/nucleus/sched-rt.c (Common real-time scheduling class implementation (FIFO + RR)) 144
kernel/cobalt/nucleus/sched-sporadic.c (POSIX SCHED_SPORADIC scheduling class) . 145
kernel/cobalt/nucleus/sched-tp.c (Temporal partitioning (typical of IMA systems)) . . . 146
kernel/cobalt/nucleus/sched.c 148
kernel/cobalt/nucleus/select.c (File descriptors events multiplexing) 148
kernel/cobalt/nucleus/shadow.c (Real-time shadow services) 150
kernel/cobalt/nucleus/synch.c (Thread synchronization services) 151
kernel/cobalt/nucleus/timer.c 153
kernel/cobalt/nucleus/vfile.c (This file is part of the Xenomai project) 154

Chapter 4

Module Documentation

4.1 Thread state flags.

Bits reporting permanent or transient states of thread.

Defines

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

Thread is currently migrating to another CPU.

- #define XNHELD 0x00000800
  Thread is held to process emergency.
- #define XNBOOST 0x00001000
  Undergoes a PIP boost.
- #define XNDEBUG 0x00002000
  Hit a debugger breakpoint (shadow only).
- #define XNLOCK 0x00004000
  Holds the scheduler lock (i.e.
- #define XNRRB 0x00008000
  Undergoes a round-robin scheduling.
- #define XNASDI 0x00010000
  ASR are disabled.
- #define XNDEFCAN 0x00020000
  Deferred cancelability mode (self-set only).
- #define XNTRAPSW 0x00040000
  Trap execution mode switches.
- #define XNFPU 0x00080000
  Thread uses FPU.
- #define XNSHADOW 0x00100000
  Shadow thread.
- #define XNROOT 0x00200000
  Root thread (that is, Linux/IDLE).
- #define XNOTHER 0x00400000
  Non real-time shadow (prio=0).

4.1.1 Detailed Description

Bits reporting permanent or transient states of thread.

4.1.2 Define Documentation

4.1.2.1 #define XNHELD 0x00000800

Thread is held to process emergency.

Referenced by xnpod_resume_thread(), and xnpod_suspend_thread().

4.1.2.2 #define XNLOCK 0x00004000

Holds the scheduler lock (i.e.

not preemptible)

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

4.1.2.3 #define XNMIGRATE 0x00000400

Thread is currently migrating to another CPU.

Referenced by xnpod_delete_thread().

4.1.2.4 #define XNPEND 0x00000002

Sleep-wait for a resource.

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

4.1.2.5 #define XNREADY 0x00000008

Linked to the ready queue.

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

4.1.2.6 #define XNSUSP 0x00000001

Suspended.

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

4.2 Thread information flags.

Bits reporting events notified to the thread.

Defines

- #define XNTIMEO 0x00000001

  Woken up due to a timeout condition.

- #define XNRMID 0x00000002

  Pending on a removed resource.

- #define XNBREAK 0x00000004

  Forcibly awaken from a wait state.

- #define XNKICKED 0x00000008
  Forced out of primary mode (shadow only).
- #define XNWAKEN 0x00000010
  Thread waken up upon resource availability.
- #define XNROBBED 0x00000020
  Robbed from resource ownership.
- #define XNAFFSET 0x00000040
  CPU affinity changed from primary mode.
- #define XNPRIOSET 0x00000080
  Priority changed from primary mode.
- #define XNABORT 0x00000100
  Thread is being aborted.
- #define XNCANPND 0x00000200
  Cancellation request is pending.
- #define XNSWREP 0x00000400
  Mode switch already reported.

4.2.1 Detailed Description

Bits reporting events notified to the thread.

4.3 Buffer descriptors.

Files

- file bufd.h
- file bufd.c

Functions

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

- ssize_t xnbufd_unmap_uwrite (struct xnbufd *bufd)
  Finalize a buffer descriptor obtained from xnbufd_map_uwrite().
- static void xnbufd_map_kread (struct xnbufd *bufd, const void *ptr, size_t len)
  Initialize a buffer descriptor for reading from kernel memory.
- static void xnbufd_map_kwrite (struct xnbufd *bufd, void *ptr, size_t len)
  Initialize a buffer descriptor for writing to kernel memory.
- ssize_t xnbufd_unmap_kread (struct xnbufd *bufd)
  Finalize a buffer descriptor obtained from xnbufd_map_kread().
- ssize_t xnbufd_unmap_kwrite (struct xnbufd *bufd)
  Finalize a buffer descriptor obtained from xnbufd_map_kwrite().
- ssize_t xnbufd_copy_to_kmem (void *ptr, struct xnbufd *bufd, size_t len)
  Copy memory covered by a buffer descriptor to kernel memory.
- ssize_t xnbufd_copy_from_kmem (struct xnbufd *bufd, void *from, size_t len)
  Copy kernel memory to the area covered by a buffer descriptor.
- void xnbufd_invalidate (struct xnbufd *bufd)
  Invalidate a buffer descriptor.
- static void xnbufd_reset (struct xnbufd *bufd)
  Reset a buffer descriptor.

4.3.1 Detailed Description

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

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

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

The common usage patterns are as follows:

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

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

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

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

    return ret;
}

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

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

    return ret;
}

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

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

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

    return ret;
}
```

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

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

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

    xnbufd_map_kread(&bufd, ptr, len);
    ret = rt_bulk_write_inner(&bufd);
    xnbufd_unmap_kread(&bufd);

    return ret;
}
```

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

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

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

    return ret;
}
```

4.3.2 Function Documentation

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

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

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

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

- if *bufd* refers to a writable kernel area (i.e. see xnbufd_map_kwrite()), the copy is immediatly and fully performed with no restriction.

- if *bufd* refers to a writable user area (i.e. see xnbufd_map_uwrite()), the copy is performed only if that area lives in the currently active address space, and only if the caller may sleep Linux-wise to process any potential page fault which may arise while writing to that memory.

- if *bufd* refers to a user area which may not be immediately written to from the current context, the copy is postponed until xnbufd_unmap_uwrite() is invoked for *ubufd*, at which point the copy will take place. In such a case, the source memory is transferred to a carry over buffer allocated internally; this operation may lead to request dynamic memory from the nucleus heap if *len* is greater than 64 bytes.

Parameters

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

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

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

Returns

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

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

Environments:

This service can be called from:

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

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

Note

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

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

Copy memory covered by a buffer descriptor to kernel memory.

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

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

- if *bufd* refers to readable kernel area (i.e. see xnbufd_map_kread()), the copy is immediately and fully performed with no restriction.

- if *bufd* refers to a readable user area (i.e. see xnbufd_map_uread()), the copy is performed only if that area lives in the currently active address space, and only if the caller may sleep Linux-wise to process any potential page fault which may arise while reading from that memory.

- any attempt to read from *bufd* from a non-suitable context is considered as a bug, and will raise a panic assertion when the nucleus is compiled in debug mode.

Parameters

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

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

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

Returns

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

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

Environments:

This service can be called from:

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

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

Note

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

4.3.2.3 void xnbufd_invalidate (struct xnbufd * *bufd*)

Invalidate a buffer descriptor.

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

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

Parameters

bufd The address of the buffer descriptor to invalidate.

Environments:

This service can be called from:

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

Rescheduling: never.

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

Initialize a buffer descriptor for reading from kernel memory.

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

Parameters

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

ptr The start of the kernel buffer to map.

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

Environments:

This service can be called from:

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

Rescheduling: never.

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

Initialize a buffer descriptor for writing to kernel memory.

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

Parameters

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

ptr The start of the kernel buffer to map.

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

Environments:

This service can be called from:

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

Rescheduling: never.

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

Initialize a buffer descriptor for reading from user memory.

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

Parameters

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

ptr The start of the user buffer to map.

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

Environments:

This service can be called from:

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

Rescheduling: never.

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

Initialize a buffer descriptor for writing to user memory.

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

Parameters

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

ptr The start of the user buffer to map.

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

Environments:

This service can be called from:

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

Rescheduling: never.

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

Reset a buffer descriptor.

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

Parameters

bufd The address of the buffer descriptor to reset.

Environments:

This service can be called from:

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

Rescheduling: never.

4.3.2.9 ssize_t xnbufd_unmap_kread (struct xnbufd * *bufd*)

Finalize a buffer descriptor obtained from xnbufd_map_kread().

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

Parameters

bufd The address of the buffer descriptor to finalize.

Returns

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

Environments:

This service can be called from:

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

Rescheduling: never.

4.3.2.10 ssize_t xnbufd_unmap_kwrite (struct xnbufd * *bufd*)

Finalize a buffer descriptor obtained from xnbufd_map_kwrite().

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

Parameters

bufd The address of the buffer descriptor to finalize.

Returns

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

Environments:

This service can be called from:

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

Rescheduling: never.

4.3.2.11 ssize_t xnbufd_unmap_uread (struct xnbufd * *bufd*)

Finalize a buffer descriptor obtained from xnbufd_map_uread().

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

Parameters

bufd The address of the buffer descriptor to finalize.

Returns

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

Environments:

This service can be called from:

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

Rescheduling: never.

Note

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

4.3.2.12 ssize_t xnbufd_unmap_uwrite (struct xnbufd * *bufd*)

Finalize a buffer descriptor obtained from xnbufd_map_uwrite().

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

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

Parameters

bufd The address of the buffer descriptor to finalize.

Returns

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

Environments:

This service can be called from:

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

Rescheduling: never.

Note

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

4.4 System clock services.

Files

- file clock.h
- file clock.c

Functions

- void xnclock_adjust (xnsticks_t delta)

  Adjust the clock time for the system.

4.4.1 Function Documentation

4.4.1.1 void xnclock_adjust (xnsticks_t *delta*)

Adjust the clock time for the system.

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

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

Parameters

delta The adjustment of the system time expressed in ticks.

Note

This routine must be entered nklock locked, interrupts off.

Environments:

This service can be called from:

- Any kernel context.

Rescheduling: never.

4.5 Debugging services.

Files

- file debug.c

  Debug services.

4.6 Dynamic memory allocation services.

Files

- file heap.c

  Dynamic memory allocation services.

Functions

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

  Initialize a memory heap.

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

  Set the heap's label string.

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

  Destroys a memory heap.

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

  Allocate a memory block from a memory heap.

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

  Test and release a memory block to a memory heap.

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

  Release a memory block to a memory heap.

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

  Extend a memory heap.

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

  Schedule a memory block for release.

4.6.1 Detailed Description

Dynamic memory allocation services.

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

The data structures hierarchy is as follows:

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

4.6.2 Function Documentation

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

Allocate a memory block from a memory heap.

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

Parameters

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

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

Returns

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

Environments:

This service can be called from:

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

Rescheduling: never.

Referenced by xnshadow_map().

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

Destroys a memory heap.

Destroys a memory heap.

Parameters

heap The descriptor address of the destroyed heap.

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

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

Environments:

This service can be called from:

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

Rescheduling: never.

Referenced by xnpod_init(), and xnpod_shutdown().

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

Extend a memory heap.

Add a new extent to an existing memory heap.

Parameters

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

extaddr The address of the extent memory.

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

Returns

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

Environments:

This service can be called from:

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

Rescheduling: never.

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

Release a memory block to a memory heap.

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

Parameters

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

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

Returns

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

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

Environments:

This service can be called from:

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

Rescheduling: never.

References xnheap_test_and_free().

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

Initialize a memory heap.

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

Parameters

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

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

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

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

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

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

Parameters

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

Returns

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

- -EINVAL is returned whenever a parameter is invalid.

Environments:

This service can be called from:

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

Rescheduling: never.

Referenced by xnpod_init().

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

Schedule a memory block for release.

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

Parameters

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

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

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

Environments:

This service can be called from:

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

Rescheduling: never.

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

Set the heap's label string.

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

Parameters

heap The address of a heap descriptor.

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

Environments:

This service can be called from:

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

Rescheduling: never.

Referenced by xnpod_init().

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

Test and release a memory block to a memory heap.

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

Parameters

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

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

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

Returns

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

Environments:

This service can be called from:

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

Rescheduling: never.

Referenced by xnheap_free().

4.7 Interrupt management.

Files

- file intr.c

  Interrupt management.

Functions

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

  Initialize an interrupt object.

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

4.7.1 Detailed Description

Interrupt management.

4.7.2 Function Documentation

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

Set interrupt's processor affinity.

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

Parameters

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

cpumask The new processor affinity of the interrupt object.

Note

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

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

Attach an interrupt object.

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

Parameters

intr The descriptor address of the interrupt object to attach.

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

Returns

0 is returned on success. Otherwise:

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

Note

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

Environments:

This service can be called from:

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

Rescheduling: never.

Note

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

4.7.2.3 int xnintr_destroy (xnintr_t * *intr*)

Destroy an interrupt object.

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

Parameters

intr The descriptor address of the interrupt object to destroy.

Returns

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

Environments:

This service can be called from:

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

Rescheduling: never.

References xnintr_detach().

4.7.2.4 int xnintr_detach (xnintr_t * *intr*)

Detach an interrupt object.

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

Parameters

intr The descriptor address of the interrupt object to detach.

Returns

0 is returned on success. Otherwise:

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

Note

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

Environments:

This service can be called from:

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

Rescheduling: never.

Referenced by xnintr_destroy().

4.7.2.5 void xnintr_disable (xnintr_t * *intr*)

Disable an interrupt object.

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

Parameters

intr The descriptor address of the interrupt object to disable.

Environments:

This service can be called from:

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

Rescheduling: never.

4.7.2.6 void xnintr_enable (xnintr_t * *intr*)

Enable an interrupt object.

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

Parameters

intr The descriptor address of the interrupt object to enable.

Environments:

This service can be called from:

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

Rescheduling: never.

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

Initialize an interrupt object.

Associates an interrupt object with an IRQ line.

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

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

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

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

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

- XN_ISR_NOENABLE causes the nucleus to ask the real-time control layer _not_ to re-enable the IRQ line (read the following section). ipipe_end_irq() must be called to re-enable the IRQ line later.
- XN_ISR_PROPAGATE tells the nucleus to require the real-time control layer to forward the IRQ. For instance, this would cause the Adeos control layer to propagate the interrupt down the interrupt pipeline to other Adeos domains, such as Linux. This is the regular

way to share interrupts between the nucleus and the host system. In effect, XN_ISR_-PROPAGATE implies XN_ISR_NOENABLE since it would make no sense to re-enable the interrupt channel before the next domain down the pipeline has had a chance to process the propagated interrupt.

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

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

Parameters

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

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

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

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

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

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

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

Returns

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

Environments:

This service can be called from:

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

Rescheduling: never.

Referenced by xnpod_enable_timesource().

4.8 Lightweight key-to-object mapping service

Files

- file map.h
- file map.c

Functions

- xnmap_t * xnmap_create (int nkeys, int reserve, int offset)
  Create a map.
- void xnmap_delete (xnmap_t *map)
  Delete a map.
- int xnmap_enter (xnmap_t *map, int key, void *objaddr)
  Index an object into a map.
- int xnmap_remove (xnmap_t *map, int key)
  Remove an object reference from a map.
- static void * xnmap_fetch_nocheck (xnmap_t *map, int key)
  Search an object into a map - unchecked form.
- static void * xnmap_fetch (xnmap_t *map, int key)
  Search an object into a map.

4.8.1 Detailed Description

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

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

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

4.8.2 Function Documentation

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

Create a map.

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

Parameters

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

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

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

Returns

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

Environments:

This service can be called from:

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

Rescheduling: never.

4.8.2.2 void xnmap_delete (xnmap_t * *map*)

Delete a map.

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

Parameters

map The address of the map to delete.

Environments:

This service can be called from:

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

Rescheduling: never.

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

Index an object into a map.

Insert a new object into the given map.

Parameters

map The address of the map to insert into.

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

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

Returns

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

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

Environments:

This service can be called from:

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

Rescheduling: never.

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

Search an object into a map.

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

Parameters

map The address of the map to retrieve from.

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

Returns

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

Environments:

This service can be called from:

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

Rescheduling: never.

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

Search an object into a map - unchecked form.

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

Parameters

map The address of the map to retrieve from.

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

Returns

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

Environments:

This service can be called from:

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

Rescheduling: never.

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

Remove an object reference from a map.

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

Parameters

map The address of the map to remove from.

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

Returns

0 is returned on success. Otherwise:

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

Environments:

This service can be called from:

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

Rescheduling: never.

4.9 Real-time pod services.

Data Structures

- struct xnpod
  Real-time pod descriptor.

Files

- file pod.h
  Real-time pod interface header.
- file pod.c
  Real-time pod services.

Functions

- void __xnpod_reset_thread (struct xnthread *thread)
  Reset the thread.
- int xnpod_init (void)
  Initialize the core pod.
- int xnpod_enable_timesource (void)
  Activate the core time source.
- void xnpod_disable_timesource (void)
  Stop the core time source.

- void xnpod_shutdown (int xtype)

  Shutdown the current pod.

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

  Initialize a new thread.

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

  Initial start of a newly created thread.

- void xnpod_stop_thread (xnthread_t *thread)

  Stop a thread.

- void xnpod_delete_thread (xnthread_t *thread)

  Delete a thread.

- void xnpod_abort_thread (xnthread_t *thread)

  Abort a thread.

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

  Change a thread's control mode.

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

  Suspend a thread.

- void xnpod_resume_thread (xnthread_t *thread, xnflags_t mask)

  Resume a thread.

- int xnpod_unblock_thread (xnthread_t *thread)

  Unblock a thread.

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

  Change the base scheduling parameters of a thread.

- int xnpod_migrate_thread (int cpu)

  Migrate the current thread.

- void xnpod_dispatch_signals (void)

  Deliver pending asynchronous signals to the running thread.

- static void xnpod_schedule (void)

  Rescheduling procedure entry point.

- int xnpod_handle_exception (struct ipipe_trap_data *d)

  Exception handler.

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

  Make a thread periodic.

- int xnpod_wait_thread_period (unsigned long *overruns_r)

  Wait for the next periodic release point.

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

  Set thread time-slicing information.

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

  Install a nucleus hook.

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

  Remove a nucleus hook.

- void xnpod_welcome_thread (xnthread_t *thread, int imask)

  Thread prologue.

4.9.1 Detailed Description

Real-time pod services.

4.9.2 Function Documentation

4.9.2.1 void __xnpod_reset_thread (struct xnthread * *thread*)

Reset the thread.

For internal use only.

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

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

Referenced by xnpod_stop_thread().

4.9.2.2 void xnpod_abort_thread (xnthread_t * *thread*)

Abort a thread.

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

Parameters

thread The descriptor address of the terminated thread.

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

Environments:

This service can be called from:

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

Rescheduling: possible if the current thread self-deletes.

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

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

Install a nucleus hook.

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

The scheduling is locked while a hook is executing.

Parameters

type Defines the kind of hook to install:

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

Parameters

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

Returns

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

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

Environments:

This service can be called from:

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

Rescheduling: never.

4.9.2.4 void xnpod_delete_thread (xnthread_t * *thread*)

Delete a thread.

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

Parameters

thread The descriptor address of the terminated thread.

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

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

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

Environments:

This service can be called from:

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

Rescheduling: possible if the current thread self-deletes.

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

Referenced by xnpod_abort_thread(), and xnpod_shutdown().

4.9.2.5 void xnpod_disable_timesource (void)

Stop the core time source.

Releases the hardware timer, and deactivates the system clock.

Environments:

This service can be called from:

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

Rescheduling: never.

References xntimer_freeze().

Referenced by xnpod_shutdown().

4.9.2.6 void xnpod_dispatch_signals (void)

Deliver pending asynchronous signals to the running thread.

For internal use only.

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

References XNASDI.

Referenced by xnpod_welcome_thread(), and xnshadow_harden().

4.9.2.7 int xnpod_enable_timesource (void)

Activate the core time source.

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

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

Returns

0 is returned on success. Otherwise:

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

Environments:

This service can be called from:

- Regular Linux kernel context.

Rescheduling: never.

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

Referenced by xnpod_init().

4.9.2.8 int xnpod_handle_exception (struct ipipe_trap_data * *d*)

Exception handler.

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

Parameters

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

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

4.9.2.9 int xnpod_init (void)

Initialize the core pod.

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

Returns

0 is returned on success. Otherwise:

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

Environments:

This service can be called from:

- Kernel module initialization code

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

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

Initialize a new thread.

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

Parameters

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

Warning

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

Parameters

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

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

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

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

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

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

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

Parameters

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

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

Returns

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

- -EINVAL is returned if *attr->flags* has invalid bits set.

- -ENOMEM is returned if not enough memory is available from the system heap to create the new thread's stack.

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

Environments:

This service can be called from:

- Kernel module initialization/cleanup code

- Kernel-based task
- User-space task

Rescheduling: never.

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

4.9.2.11 int xnpod_migrate_thread (int *cpu*)

Migrate the current thread.

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

Parameters

cpu The destination CPU.

Return values

0 if the thread could migrate ;

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

-EBUSY if the scheduler is locked.

References xnpod_schedule().

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

Remove a nucleus hook.

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

Parameters

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

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

Returns

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

Environments:

This service can be called from:

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

Rescheduling: never.

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

Resume a thread.

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

Parameters

thread The descriptor address of the resumed thread.

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

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

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

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

Environments:

This service can be called from:

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

Rescheduling: never.

References XNDELAY, XNHELD, XNPEND, XNREADY, xnsynch_forget_sleeper(), and xntimer_stop().

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

4.9.2.14 void xnpod_schedule (void) [inline, static]

Rescheduling procedure entry point.

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

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

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

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

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

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

Side-effects:

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

Environments:

This service can be called from:

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

Note

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

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

Referenced by xnpod_delete_thread(), xnpod_migrate_thread(), xnpod_shutdown(), xnpod_start_thread(), xnpod_stop_thread(), xnpod_suspend_thread(), xnregistry_enter(), xnregistry_put(), xnselect_bind(), and xnselect_destroy().

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

Change a thread's control mode.

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

Parameters

thread The descriptor address of the affected thread.

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

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

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

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

Environments:

This service can be called from:

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

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

References XNLOCK.

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

Make a thread periodic.

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

Parameters

thread The descriptor address of the affected thread. This thread is immediately delayed until the first periodic release point is reached.

idate The initial (absolute) date of the first release point, expressed in nanoseconds. The affected thread will be delayed by the first call to xnpod_wait_thread_period() until this point is reached. If *idate* is equal to XN_INFINITE, the current system date is used, and no initial delay takes place. In the latter case, *timeout_mode* is not considered and can have any valid value.

timeout_mode The mode of the *idate* parameter. It can either be set to XN_ABSOLUTE or XN_REALTIME with *idate* different from XN_INFINITE (see also xntimer_start()).

period The period of the thread, expressed in nanoseconds. As a side-effect, passing XN_INFINITE attempts to stop the thread's periodic timer; in the latter case, the routine always exits succesfully, regardless of the previous state of this timer.

Returns

0 is returned upon success. Otherwise:

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

Environments:

This service can be called from:

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

Rescheduling: none.

References xntimer_start(), and xntimer_stop().

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

Change the base scheduling parameters of a thread.

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

Parameters

thread The descriptor address of the affected thread.

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

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

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

Returns

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

Side-effects:

- This service does not call the rescheduling procedure but may affect the state of the runnable queue for the previous and new scheduling classes.
- Assigning the same scheduling class and parameters to a running or ready thread moves it to the end of the runnable queue, thus causing a manual round-robin.
- If the thread is a user-space shadow, this call propagates the request to the mated Linux task.

Environments:

This service can be called from:

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

Rescheduling: never.

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

Set thread time-slicing information.

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

Parameters

thread The descriptor address of the affected thread.

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

Returns

0 is returned upon success. Otherwise:

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

Environments:

This service can be called from:

- Any kernel context.

Rescheduling: never.

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

4.9.2.19 void xnpod_shutdown (int *xtype*)

Shutdown the current pod.

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

Parameters

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

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

Environments:

This service can be called from:

- Kernel module initialization/cleanup code

Rescheduling: never.

References xnheap_destroy(), xnpod_delete_thread(), xnpod_disable_timesource(), xnpod_schedule(), and XNROOT.

Referenced by xnpod_init().

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

Initial start of a newly created thread.

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

Parameters

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

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

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

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

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

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

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

- affinity: The processor affinity of this thread. Passing XNPOD_ALL_CPUS or an empty affinity set means "any cpu".

- entry: The address of the thread's body routine. In other words, it is the thread entry point.

- cookie: A user-defined opaque cookie the nucleus will pass to the emerging thread as the sole argument of its entry point.

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

Return values

0 if *thread* could be started ;

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

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

Environments:

This service can be called from:

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

- User-space task

Rescheduling: possible.

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

Referenced by xnshadow_map().

4.9.2.21 void xnpod_stop_thread (xnthread_t * *thread*)

Stop a thread.

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

Parameters

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

Environments:

This service can be called from:

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

Rescheduling: possible.

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

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

Suspend a thread.

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

Parameters

thread The descriptor address of the suspended thread.

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

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

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

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

Parameters

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

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

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

Note

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

Environments:

This service can be called from:

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

Rescheduling: possible if the current thread suspends itself.

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

Referenced by xnpod_abort_thread(), xnpod_handle_exception(), xnpod_init_thread(), xnpod_stop_thread(), xnpod_wait_thread_period(), xnshadow_map(), xnshadow_relax(), xnsynch_acquire(), and xnsynch_sleep_on().

4.9.2.23 int xnpod_unblock_thread (xnthread_t * *thread*)

Unblock a thread.

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

Parameters

thread The descriptor address of the unblocked thread.

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

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

Returns

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

Environments:

This service can be called from:

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

Rescheduling: never.

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

Referenced by __xnpod_reset_thread(), and xnpod_delete_thread().

4.9.2.24 int xnpod_wait_thread_period (unsigned long * *overruns_r*)

Wait for the next periodic release point.

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

Parameters

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

Returns

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

- -EWOULDBLOCK is returned if xnpod_set_thread_periodic() has not previously been called for the calling thread.

- -EINTR is returned if xnpod_unblock_thread() has been called for the waiting thread before the next periodic release point has been reached. In this case, the overrun counter is reset too.

- -ETIMEDOUT is returned if the timer has overrun, which indicates that one or more previous release points have been missed by the calling thread. If *overruns_r* is valid, the count of pending overruns is copied to the pointed memory location.

Environments:

This service can be called from:

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

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

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

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

Thread prologue.

For internal use only.

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

Entered with nklock locked, irqs off.

References XNLOCK, and xnpod_dispatch_signals().

4.10 Registry services.

Files

- file registry.h

  This file is part of the Xenomai project.

- file registry.c

  This file is part of the Xenomai project.

Functions

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

Register a real-time object.

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

  Bind to a real-time object.

- int xnregistry_remove (xnhandle_t handle)

  Forcibly unregister a real-time object.

- int xnregistry_remove_safe (xnhandle_t handle, xnticks_t timeout)

  Unregister an idle real-time object.

- void * xnregistry_get (xnhandle_t handle)

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

- u_long xnregistry_put (xnhandle_t handle)

  Unlock a real-time object from the registry.

- void * xnregistry_fetch (xnhandle_t handle)

  Find a real-time object into the registry.

4.10.1 Detailed Description

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

4.10.2 Function Documentation

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

Bind to a real-time object.

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

Parameters

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

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

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

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

Returns

0 is returned upon success. Otherwise:

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

Environments:

This service can be called from:

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

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

References XNBREAK, xnsynch_sleep_on(), and XNTIMEO.

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

Register a real-time object.

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

Parameters

key A valid NULL-terminated string by which the object will be indexed and later retrieved in the registry. Since it is assumed that such key is stored into the registered object, it will *not* be copied but only kept by reference in the registry. Pass an empty string if the object shall only occupy a registry slot for handle-based lookups.

objaddr An opaque pointer to the object to index by *key*.

phandle A pointer to a generic handle defined by the registry which will uniquely identify the indexed object, until the latter is unregistered using the xnregistry_remove() service.

pnode A pointer to an optional /proc node class descriptor. This structure provides the information needed to export all objects from the given class through the /proc filesystem, under the /proc/xenomai/registry entry. Passing NULL indicates that no /proc support is available for the newly registered object.

Returns

0 is returned upon success. Otherwise:

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

Environments:

This service can be called from:

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

Rescheduling: possible.

References xnpod_schedule(), and xnsynch_init().

4.10.2.3 void* xnregistry_fetch (xnhandle_t *handle*)

Find a real-time object into the registry.

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

Parameters

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

Returns

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

Environments:

This service can be called from:

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

Rescheduling: never.

4.10.2.4 void* xnregistry_get (xnhandle_t *handle*)

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

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

Parameters

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

Returns

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

Environments:

This service can be called from:

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

Rescheduling: never.

4.10.2.5 u_long xnregistry_put (xnhandle_t *handle*)

Unlock a real-time object from the registry.

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

Parameters

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

Returns

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

Environments:

This service can be called from:

- Kernel module initialization/cleanup code

- Interrupt service routine only if *handle* is different from XNOBJECT_SELF.
- Kernel-based thread

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

References xnpod_schedule(), and xnsynch_flush().

4.10.2.6 int xnregistry_remove (xnhandle_t *handle*)

Forcibly unregister a real-time object.

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

Parameters

handle The generic handle of the object to remove.

Returns

0 is returned upon success. Otherwise:

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

Environments:

This service can be called from:

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

Rescheduling: never.

Referenced by xnregistry_remove_safe().

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

Unregister an idle real-time object.

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

Parameters

handle The generic handle of the object to remove.

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

Returns

0 is returned upon success. Otherwise:

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

Environments:

This service can be called from:

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

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

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

4.11 File descriptors events multiplexing services.

Files

- file select.h

  file descriptors events multiplexing header.

- file select.c

  file descriptors events multiplexing.

Functions

- void xnselect_init (struct xnselect *select_block)

  Initialize a struct xnselect structure.

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

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

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

Signal a file descriptor state change.

- void xnselect_destroy (struct xnselect *select_block)

  Destroy the xnselect structure associated with a file descriptor.

- int xnselector_init (struct xnselector *selector)

  Initialize a selector structure.

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

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

- void xnselector_destroy (struct xnselector *selector)

  Destroy a selector block.

4.11.1 Detailed Description

File descriptors events multiplexing services.

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

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

- *XNSELECT_READ* meaning that a file descriptor is ready for reading;
- *XNSELECT_WRITE* meaning that a file descriptor is ready for writing;
- *XNSELECT_EXCEPT* meaning that a file descriptor received an exceptional event.

It works by defining two structures:

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

4.11.2 Function Documentation

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

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

Parameters

selector structure to check for pending events

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

in_fds the set of descriptors which events should be checked

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

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

timeout_mode the mode of *timeout*.

Return values

-EINVAL if *nfds* is negative;

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

-EINTR if *xnselect* was interrupted while waiting;

0 in case of timeout.

the number of file descriptors having received an event.

References XNBREAK, xnsynch_sleep_on(), and XNTIMEO.

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

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

Parameters

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

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

selector pointer to the selector structure;

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

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

state current state of the file descriptor>.

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

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

Return values

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

0 otherwise.

References xnpod_schedule().

4.11.2.3 void xnselect_destroy (struct xnselect * *select_block*)

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

Any binding with a *xnselector* block is destroyed.

Parameters

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

References xnpod_schedule().

4.11.2.4 void xnselect_init (struct xnselect * *select_block*)

Initialize a *struct xnselect* structure.

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

Parameters

select_block pointer to the xnselect structure to be initialized

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

Signal a file descriptor state change.

Parameters

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

state new value of the state.

Return values

1 if rescheduling is needed;

0 otherwise.

4.11.2.6 void xnselector_destroy (struct xnselector * *selector*)

Destroy a selector block.

All bindings with file descriptor are destroyed.

Parameters

selector the selector block to be destroyed

Referenced by xnpod_delete_thread().

4.11.2.7 int xnselector_init (struct xnselector * *selector*)

Initialize a selector structure.

Parameters

selector The selector structure to be initialized.

Return values

0

References xnsynch_init().

4.12 Real-time shadow services.

Files

- file shadow.c

  Real-time shadow services.

Functions

- int xnshadow_harden (void)

  Migrate a Linux task to the Xenomai domain.

- void xnshadow_relax (int notify, int reason)

  Switch a shadow thread back to the Linux domain.

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

  Create a shadow thread context.

- xnshadow_ppd_t * xnshadow_ppd_get (unsigned int muxid)

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

4.12.1 Detailed Description

Real-time shadow services.

4.12.2 Function Documentation

4.12.2.1 int xnshadow_harden (void)

Migrate a Linux task to the Xenomai domain.

For internal use only.

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

Environments:

This service can be called from:

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

Rescheduling: always.

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

Referenced by xnshadow_map().

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

Create a shadow thread context.

For internal use only.

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

Parameters

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

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

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

Returns

0 is returned on success. Otherwise:

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

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

- -EINVAL is returned if the thread control block does not bear the XNSHADOW bit.
- -EBUSY is returned if either the current Linux task or the associated shadow thread is already involved in a shadow mapping.

Environments:

This service can be called from:

- Regular user-space process.

Rescheduling: always.

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

4.12.2.3 xnshadow_ppd_t* xnshadow_ppd_get (unsigned int *muxid*)

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

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

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

Parameters

muxid the skin muxid.

Returns

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

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

Switch a shadow thread back to the Linux domain.

For internal use only.

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

Parameters

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

reason The reason to report along with the SIGDEBUG signal.

Environments:

This service can be called from:

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

Rescheduling: always.

Note

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

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

Referenced by xnpod_handle_exception(), and xnshadow_harden().

4.13 Thread synchronization services.

Files

- file synch.c

  Thread synchronization services.

Functions

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

  Initialize a synchronization object.

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

  Sleep on an ownerless synchronization object.

- struct xnthread * xnsynch_wakeup_one_sleeper (struct xnsynch *synch)

  Give the resource ownership to the next waiting thread.

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

  Give the resource ownership to a given waiting thread.

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

  Acquire the ownership of a synchronization object.

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

  Clear the priority boost.

- void xnsynch_requeue_sleeper (struct xnthread *thread)

  Change a sleeper's priority.

- struct xnthread * xnsynch_peek_pendq (struct xnsynch *synch)

  Access the thread leading a synch object wait queue.

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

  Unblock all waiters pending on a resource.

- void xnsynch_forget_sleeper (struct xnthread *thread)

  Abort a wait for a resource.

- void xnsynch_release_all_ownerships (struct xnthread *thread)

  Release all ownerships.

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

  Give the resource ownership to the next waiting thread.

4.13.1 Detailed Description

Thread synchronization services.

4.13.2 Function Documentation

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

Acquire the ownership of a synchronization object.

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

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

Parameters

synch The descriptor address of the synchronization object to acquire.

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

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

Returns

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

Environments:

This service can be called from:

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

Rescheduling: possible.

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

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

Clear the priority boost.

For internal use only.

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

Parameters

synch The descriptor address of the synchronization object.

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

Note

This routine must be entered nklock locked, interrupts off.

References XNBOOST.

Referenced by xnsynch_flush(), and xnsynch_forget_sleeper().

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

Unblock all waiters pending on a resource.

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

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

Parameters

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

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

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

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

Returns

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

Side-effects:

- The effective priority of the previous resource owner might be lowered to its base priority value as a consequence of the priority inheritance boost being cleared.
- The synchronization object is no more owned by any thread.

Environments:

This service can be called from:

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

Rescheduling: never.

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

Referenced by xnregistry_put().

4.13.2.4 void xnsynch_forget_sleeper (struct xnthread * *thread*)

Abort a wait for a resource.

For internal use only.

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

Parameters

thread The descriptor address of the affected thread.

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

Note

This routine must be entered nklock locked, interrupts off.

References XNPEND, and xnsynch_clear_boost().

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

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

Initialize a synchronization object.

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

This object has built-in support for priority inheritance.

Parameters

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

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

- XNSYNCH_PRIO causes the threads waiting for the resource to pend in priority order. Otherwise, FIFO ordering is used (XNSYNCH_FIFO).
- XNSYNCH_OWNER indicates that the synchronization object shall track its owning thread (required if XNSYNCH_PIP is selected). Note that setting this flag implies the use xnsynch_acquire and xnsynch_release instead of xnsynch_sleep_on and xnsynch_wakeup_one_sleeper/xnsynch_wakeup_this_sleeper.
- XNSYNCH_PIP causes the priority inheritance mechanism to be automatically activated when a priority inversion is detected among threads using this object. Otherwise, no priority inheritance takes place upon priority inversion (XNSYNCH_NOPIP).
- XNSYNCH_DREORD (Disable REORDering) tells the nucleus that the wait queue should not be reordered whenever the priority of a blocked thread it holds is changed. If this flag is not specified, changing the priority of a blocked thread using xnpod_set_thread_schedparam() will cause this object's wait queue to be reordered according to the new priority level, provided the synchronization object makes the waiters wait by priority order on the awaited resource (XNSYNCH_PRIO).

Parameters

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

Environments:

This service can be called from:

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

Rescheduling: never.

Referenced by xnregistry_enter(), and xnselector_init().

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

Access the thread leading a synch object wait queue.

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

Parameters

synch The descriptor address of the target synchronization object.

Returns

The descriptor address of the unblocked thread.

Environments:

This service can be called from:

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

Rescheduling: never.

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

Give the resource ownership to the next waiting thread.

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

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

Parameters

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

owner The descriptor address of the current owner.

Returns

The descriptor address of the unblocked thread.

Side-effects:

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

Environments:

This service can be called from:

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

Rescheduling: never.

References XNOTHER.

Referenced by xnsynch_release_all_ownerships().

4.13.2.8 void xnsynch_release_all_ownerships (struct xnthread * *thread*)

Release all ownerships.

For internal use only.

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

Parameters

thread The descriptor address of the affected thread.

Note

This routine must be entered nklock locked, interrupts off.

References xnsynch_release().

Referenced by __xnpod_reset_thread(), and xnpod_delete_thread().

4.13.2.9 void xnsynch_requeue_sleeper (struct xnthread * *thread*)

Change a sleeper's priority.

For internal use only.

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

Parameters

thread The descriptor address of the affected thread.

Note

This routine must be entered nklock locked, interrupts off.

References XNBOOST.

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

Sleep on an ownerless synchronization object.

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

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

Parameters

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

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

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

Returns

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

Environments:

This service can be called from:

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

Rescheduling: always.

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

Referenced by xnregistry_bind(), xnregistry_remove_safe(), and xnselect().

4.13.2.11 struct xnthread* xnsynch_wakeup_one_sleeper (struct xnsynch * *synch*) [read]

Give the resource ownership to the next waiting thread.

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

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

Parameters

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

Returns

The descriptor address of the unblocked thread.

Side-effects:

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

Environments:

This service can be called from:

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

Rescheduling: never.

References XNPEND, and xnpod_resume_thread().

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

Give the resource ownership to a given waiting thread.

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

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

Parameters

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

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

Returns

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

Side-effects:

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

Environments:

This service can be called from:

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

Rescheduling: never.

References XNPEND, and xnpod_resume_thread().

4.14 Timer services.

Files

- file timer.h
- file timer.c

Functions

- int xntimer_start (xntimer_t *timer, xnticks_t value, xnticks_t interval, xntmode_t mode)
  Arm a timer.
- xnticks_t xntimer_get_date (xntimer_t *timer)
  Return the absolute expiration date.
- xnticks_t xntimer_get_timeout (xntimer_t *timer)
  Return the relative expiration date.
- xnticks_t xntimer_get_interval (xntimer_t *timer)
  Return the timer interval value.
- static void xntimer_stop (xntimer_t *timer)
  Disarm a timer.
- void xntimer_tick (void)
  Process a timer tick.
- void xntimer_init (xntimer_t *timer, void(*handler)(xntimer_t *timer))
  Initialize a timer object.
- void xntimer_destroy (xntimer_t *timer)
  Release a timer object.
- unsigned long xntimer_get_overruns (xntimer_t *timer, xnticks_t now)
  Get the count of overruns for the last tick.

- void xntimer_freeze (void)

  Freeze all timers (from every time bases).

4.14.1 Detailed Description

The Xenomai timer facility always operate the timer hardware in oneshot mode, regardless of the time base in effect. Periodic timing is obtained through a software emulation, using cascading timers.

The timer object stores time as a count of CPU ticks (e.g. TSC values).

4.14.2 Function Documentation

4.14.2.1 void xntimer_destroy (xntimer_t * *timer*)

Release a timer object.

Destroys a timer. After it has been destroyed, all resources associated with the timer have been released. The timer is automatically deactivated before deletion if active on entry.

Parameters

timer The address of a valid timer descriptor.

Environments:

This service can be called from:

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

Rescheduling: never.

References xntimer_stop().

Referenced by xnpod_delete_thread().

4.14.2.2 void xntimer_freeze (void)

Freeze all timers (from every time bases).

For internal use only.

This routine deactivates all active timers atomically.

Environments:

This service can be called from:

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

Rescheduling: never.

References xnsched::status.

Referenced by xnpod_disable_timesource().

4.14.2.3 xnticks_t xntimer_get_date (xntimer_t * *timer*)

Return the absolute expiration date.

Return the next expiration date of a timer as an absolute count of nanoseconds.

Parameters

timer The address of a valid timer descriptor.

Returns

The expiration date in nanoseconds. The special value XN_INFINITE is returned if *timer* is currently disabled.

Environments:

This service can be called from:

- Any kernel context.

Rescheduling: never.

4.14.2.4 xnticks_t xntimer_get_interval (xntimer_t * *timer*)

Return the timer interval value.

Return the timer interval value in nanoseconds.

Parameters

timer The address of a valid timer descriptor.

Returns

The duration of a period in nanoseconds. The special value XN_INFINITE is returned if *timer* is currently disabled or one shot.

Environments:

This service can be called from:

- Any kernel context.

Rescheduling: never.

4.14.2.5 unsigned long xntimer_get_overruns (xntimer_t * *timer*, xnticks_t *now*)

Get the count of overruns for the last tick.

This service returns the count of pending overruns for the last tick of a given timer, as measured by the difference between the expected expiry date of the timer and the date *now* passed as argument.

Parameters

timer The address of a valid timer descriptor.

now current date (in the monotonic time base)

Returns

the number of overruns of *timer* at date *now*

Referenced by xnpod_wait_thread_period().

4.14.2.6 xnticks_t xntimer_get_timeout (xntimer_t * *timer*)

Return the relative expiration date.

This call returns the count of nanoseconds remaining until the timer expires.

Parameters

timer The address of a valid timer descriptor.

Returns

The count of nanoseconds until expiry. The special value XN_INFINITE is returned if *timer* is currently disabled. It might happen that the timer expires when this service runs (even if the associated handler has not been fired yet); in such a case, 1 is returned.

Environments:

This service can be called from:

- Any kernel context.

Rescheduling: never.

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

Initialize a timer object.

Creates a timer. When created, a timer is left disarmed; it must be started using xntimer_start() in order to be activated.

Parameters

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

handler The routine to call upon expiration of the timer.

There is no limitation on the number of timers which can be created/active concurrently.

Environments:

This service can be called from:

- Any kernel context.

Rescheduling: never.

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

Arm a timer.

Activates a timer so that the associated timeout handler will be fired after each expiration time. A timer can be either periodic or one-shot, depending on the reload value passed to this routine. The given timer must have been previously initialized.

Parameters

timer The address of a valid timer descriptor.

value The date of the initial timer shot, expressed in nanoseconds.

interval The reload value of the timer. It is a periodic interval value to be used for reprogramming the next timer shot, expressed in nanoseconds. If *interval* is equal to XN_INFINITE, the timer will not be reloaded after it has expired.

mode The timer mode. It can be XN_RELATIVE if *value* shall be interpreted as a relative date, XN_ABSOLUTE for an absolute date based on the monotonic clock of the related time base (as returned my xnclock_read_monotonic()), or XN_REALTIME if the absolute date is based on the adjustable real-time clock (obtained from xnclock_read()).

Returns

0 is returned upon success, or -ETIMEDOUT if an absolute date in the past has been given.

Environments:

This service can be called from:

- Any kernel context.

Rescheduling: never.

Note

Must be called with nklock held, IRQs off.

Referenced by xnpod_enable_timesource(), xnpod_set_thread_periodic(), xnpod_set_thread_tslice(), and xnpod_suspend_thread().

4.14.2.9 int xntimer_stop (xntimer_t * *timer*) [inline, static]

Disarm a timer.

This service deactivates a timer previously armed using xntimer_start(). Once disarmed, the timer can be subsequently re-armed using the latter service.

Parameters

timer The address of a valid timer descriptor.

Environments:

This service can be called from:

- Any kernel context.

Rescheduling: never.

Note

Must be called with nklock held, IRQs off.

Referenced by xnpod_resume_thread(), xnpod_set_thread_periodic(), xnpod_set_thread_tslice(), and xntimer_destroy().

4.14.2.10 void xntimer_tick (void)

Process a timer tick.

For internal use only.

This routine informs all active timers that the clock has been updated by processing the outstanding timer list. Elapsed timer actions will be fired.

Environments:

This service can be called from:

- Interrupt service routine, nklock locked, interrupts off

Rescheduling: never.

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

4.15 Virtual file services

Data Structures

- struct xnvfile_lock_ops

  Vfile locking operations.

- struct xnvfile_regular_ops
  Regular vfile operation descriptor.
- struct xnvfile_regular_iterator
  Regular vfile iterator.
- struct xnvfile_snapshot_ops
  Snapshot vfile operation descriptor.
- struct xnvfile_rev_tag
  Snapshot revision tag.
- struct xnvfile_snapshot
  Snapshot vfile descriptor.
- struct xnvfile_snapshot_iterator
  Snapshot-driven vfile iterator.

Files

- file vfile.h
  This file is part of the Xenomai project.

Functions

- int xnvfile_init_snapshot (const char *name, struct xnvfile_snapshot *vfile, struct xnvfile_-directory *parent)
  Initialize a snapshot-driven vfile.
- int xnvfile_init_regular (const char *name, struct xnvfile_regular *vfile, struct xnvfile_-directory *parent)
  Initialize a regular vfile.
- int xnvfile_init_dir (const char *name, struct xnvfile_directory *vdir, struct xnvfile_directory *parent)
  Initialize a virtual directory entry.
- int xnvfile_init_link (const char *from, const char *to, struct xnvfile_link *vlink, struct xnvfile_directory *parent)
  Initialize a virtual link entry.
- void xnvfile_destroy (struct xnvfile *vfile)
  Removes a virtual file entry.
- ssize_t xnvfile_get_blob (struct xnvfile_input *input, void *data, size_t size)
  Read in a data bulk written to the vfile.

- ssize_t xnvfile_get_string (struct xnvfile_input *input, char *s, size_t maxlen)
  Read in a C-string written to the vfile.

- ssize_t xnvfile_get_integer (struct xnvfile_input *input, long *valp)
  Evaluate the string written to the vfile as a long integer.

Variables

- struct xnvfile_directory nkvfroot
  Xenomai vfile root directory.

- struct xnvfile_directory nkvfroot
  Xenomai vfile root directory.

4.15.1 Detailed Description

Virtual files provide a mean to export Xenomai object states to user-space, based on common kernel interfaces. This encapsulation is aimed at:

- supporting consistent collection of very large record-based output, without encurring latency peaks for undergoing real-time activities.

- in the future, hiding discrepancies between linux kernel releases, regarding the proper way to export kernel object states to userland, either via the /proc interface or by any other mean.

This virtual file implementation offers record-based read support based on seq_files, single-buffer write support, directory and link handling, all visible from the /proc namespace.

The vfile support exposes four filesystem object types:

- snapshot-driven file (struct xnvfile_snapshot). This is commonly used to export real-time object states via the /proc filesystem. To minimize the latency involved in protecting the vfile routines from changes applied by real-time code on such objects, a snapshot of the data to output is first taken under proper locking, before the collected data is formatted and sent out in a lockless manner.

Because a large number of records may have to be output, the data collection phase is not strictly atomic as a whole, but only protected at record level. The vfile implementation can be notified of updates to the underlying data set, and restart the collection from scratch until the snapshot is fully consistent.

- regular sequential file (struct xnvfile_regular). This is basically an encapsulated sequential file object as available from the host kernel (i.e. seq_file), with a few additional features to make it more handy in a Xenomai environment, like implicit locking support and shortened declaration for simplest, single-record output.

- virtual link (struct xnvfile_link). This is a symbolic link feature integrated with the vfile semantics. The link target is computed dynamically at creation time from a user-given helper routine.

- virtual directory (struct xnvfile_directory). A directory object, which can be used to create a hierarchy for ordering a set of vfile objects.

4.15.2 Function Documentation

4.15.2.1 void xnvfile_destroy (struct xnvfile * *vfile*)

Removes a virtual file entry.

Parameters

vfile A pointer to the virtual file descriptor to remove.

References nkvfroot.

4.15.2.2 ssize_t xnvfile_get_blob (struct xnvfile_input * *input*, void * *data*, size_t *size*)

Read in a data bulk written to the vfile.

When writing to a vfile, the associated store() handler from the snapshot-driven vfile or regular vfile is called, with a single argument describing the input data. xnvfile_get_blob() retrieves this data as an untyped binary blob, and copies it back to the caller's buffer.

Parameters

input A pointer to the input descriptor passed to the store() handler.

data The address of the destination buffer to copy the input data to.

size The maximum number of bytes to copy to the destination buffer. If *size* is larger than the actual data size, the input is truncated to *size*.

Returns

The number of bytes read and copied to the destination buffer upon success. Otherwise, a negative error code is returned:

- -EFAULT indicates an invalid source buffer address.

Referenced by xnvfile_get_integer(), and xnvfile_get_string().

4.15.2.3 ssize_t xnvfile_get_integer (struct xnvfile_input * *input*, long * *valp*)

Evaluate the string written to the vfile as a long integer.

When writing to a vfile, the associated store() handler from the snapshot-driven vfile or regular vfile is called, with a single argument describing the input data. xnvfile_get_integer() retrieves and interprets this data as a long integer, and copies the resulting value back to *valp*.

The long integer can be expressed in decimal, octal or hexadecimal bases depending on the prefix found.

Parameters

input A pointer to the input descriptor passed to the store() handler.

valp The address of a long integer variable to receive the value.

Returns

The number of characters read while evaluating the input as a long integer upon success. Otherwise, a negative error code is returned:

- -EINVAL indicates a parse error on the input stream; the written text cannot be evaluated as a long integer.
- -EFAULT indicates an invalid source buffer address.

References xnvfile_get_blob().

4.15.2.4 ssize_t xnvfile_get_string (struct xnvfile_input * *input*, char * *s*, size_t *maxlen*)

Read in a C-string written to the vfile.

When writing to a vfile, the associated store() handler from the snapshot-driven vfile or regular vfile is called, with a single argument describing the input data. xnvfile_get_string() retrieves this data as a null-terminated character string, and copies it back to the caller's buffer.

Parameters

input A pointer to the input descriptor passed to the store() handler.

s The address of the destination string buffer to copy the input data to.

maxlen The maximum number of bytes to copy to the destination buffer, including the ending null character. If *maxlen* is larger than the actual string length, the input is truncated to *maxlen*.

Returns

The number of characters read and copied to the destination buffer upon success. Otherwise, a negative error code is returned:

- -EFAULT indicates an invalid source buffer address.

References xnvfile_get_blob().

4.15.2.5 int xnvfile_init_dir (const char * *name*, struct xnvfile_directory * *vdir*, struct xnvfile_directory * *parent*)

Initialize a virtual directory entry.

Parameters

name The name which should appear in the pseudo-filesystem, identifying the vdir entry.

vdir A pointer to the virtual directory descriptor to initialize.

parent A pointer to a virtual directory descriptor standing for the parent directory of the new vdir. If NULL, the /proc root directory will be used. /proc/xenomai is mapped on the globally available *nkvfroot* vdir.

Returns

0 is returned on success. Otherwise:

- -ENOMEM is returned if the virtual directory entry cannot be created in the /proc hierarchy.

4.15.2.6 int xnvfile_init_link (const char * *from*, const char * *to*, struct xnvfile_link * *vlink*, struct xnvfile_directory * *parent*)

Initialize a virtual link entry.

Parameters

from The name which should appear in the pseudo-filesystem, identifying the vlink entry.

to The target file name which should be referred to symbolically by *name*.

vlink A pointer to the virtual link descriptor to initialize.

parent A pointer to a virtual directory descriptor standing for the parent directory of the new vlink. If NULL, the /proc root directory will be used. /proc/xenomai is mapped on the globally available *nkvfroot* vdir.

Returns

0 is returned on success. Otherwise:

- -ENOMEM is returned if the virtual link entry cannot be created in the /proc hierarchy.

4.15.2.7 int xnvfile_init_regular (const char * *name*, struct xnvfile_regular * *vfile*, struct xnvfile_directory * *parent*)

Initialize a regular vfile.

Parameters

name The name which should appear in the pseudo-filesystem, identifying the vfile entry.

vfile A pointer to a vfile descriptor to initialize from. The following fields in this structure should be filled in prior to call this routine:

- .privsz is the size (in bytes) of the private data area to be reserved in the vfile iterator. A NULL value indicates that no private area should be reserved.

- entry.lockops is a pointer to a locking descriptor", defining the lock and unlock operations for the vfile. This pointer may be left to NULL, in which case no locking will be applied.

- .ops is a pointer to an operation descriptor.

Parameters

parent A pointer to a virtual directory descriptor; the vfile entry will be created into this directory. If NULL, the /proc root directory will be used. /proc/xenomai is mapped on the globally available *nkvfroot* vdir.

Returns

0 is returned on success. Otherwise:

- -ENOMEM is returned if the virtual file entry cannot be created in the /proc hierarchy.

4.15.2.8 int xnvfile_init_snapshot (const char * *name*, struct xnvfile_snapshot * *vfile*, struct xnvfile_directory * *parent*)

Initialize a snapshot-driven vfile.

Parameters

name The name which should appear in the pseudo-filesystem, identifying the vfile entry.

vfile A pointer to a vfile descriptor to initialize from. The following fields in this structure should be filled in prior to call this routine:

- .privsz is the size (in bytes) of the private data area to be reserved in the vfile iterator. A NULL value indicates that no private area should be reserved.

- .datasz is the size (in bytes) of a single record to be collected by the next() handler from the operation descriptor.

- .tag is a pointer to a mandatory vfile revision tag structure (struct xnvfile_rev_tag). This tag will be monitored for changes by the vfile core while collecting data to output, so that any update detected will cause the current snapshot data to be dropped, and the collection to restart from the beginning. To this end, any change to the data which may be part of the collected records, should also invoke xnvfile_touch() on the associated tag.

- entry.lockops is a pointer to a locking descriptor", defining the lock and unlock operations for the vfile. This pointer may be left to NULL, in which case the operations on the nucleus lock (i.e. nklock) will be used internally around calls to data collection handlers (see operation descriptor).

- .ops is a pointer to an operation descriptor.

Parameters

parent A pointer to a virtual directory descriptor; the vfile entry will be created into this directory. If NULL, the /proc root directory will be used. /proc/xenomai is mapped on the globally available *nkvfroot* vdir.

Returns

0 is returned on success. Otherwise:

- -ENOMEM is returned if the virtual file entry cannot be created in the /proc hierarchy.

References xnvfile_snapshot_ops::store.

4.15.3 Variable Documentation

4.15.3.1 struct xnvfile_directory nkvfroot

Xenomai vfile root directory.

This vdir maps the /proc/xenomai directory. It can be used to create a hierarchy of Xenomai-related vfiles under this root.

Referenced by xnvfile_destroy().

4.15.3.2 struct xnvfile_directory nkvfroot

Xenomai vfile root directory.

This vdir maps the /proc/xenomai directory. It can be used to create a hierarchy of Xenomai-related vfiles under this root.

Referenced by xnvfile_destroy().

4.16 Sched

Data Structures

- struct xnsched

  Scheduling information structure.

Files

- file sched.h

  Scheduler interface header.

- file sched-idle.c

  Idle scheduling class implementation (i.e. Linux placeholder).

- file sched-rt.c

  Common real-time scheduling class implementation (FIFO + RR).

- file sched-sporadic.c

  POSIX SCHED_SPORADIC scheduling class.

- file sched-tp.c

  Temporal partitioning (typical of IMA systems).

- file sched.c

Typedefs

- typedef struct xnsched xnsched_t

  Scheduling information structure.

Functions

- static void xnsched_rotate (struct xnsched *sched, struct xnsched_class *sched_class, const union xnsched_policy_param *sched_param)

  Rotate a scheduler runqueue.

4.16.1 Function Documentation

4.16.1.1 void xnsched_rotate (struct xnsched * *sched*, struct xnsched_class * *sched_class*, const union xnsched_policy_param * *sched_param*) [inline, static]

Rotate a scheduler runqueue.

The specified scheduling class is requested to rotate its runqueue for the given scheduler. Rotation is performed according to the scheduling parameter specified by *sched_param*.

Note

The nucleus supports round-robin scheduling for the members of the RT class.

Parameters

sched The per-CPU scheduler hosting the target scheduling class.

sched_class The scheduling class which should rotate its runqueue.

sched_param The scheduling parameter providing rotation information to the specified scheduling class.

Environments:

This service should be called from:

- Kernel-based task
- Interrupt service routine
- User-space task (primary mode only)

Rescheduling: never.

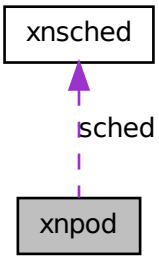
Chapter 5

Data Structure Documentation

5.1 xnpod Struct Reference

Real-time pod descriptor.

Collaboration diagram for xnpod:



Data Fields

- xnflags_t status
- xnsched_t sched [XNARCH_NR_CPUS]
- xnqueue_t threadq
- xnqueue_t tstartq
- xnqueue_t tswitchq
- xnqueue_t tdeleteq
- atomic_counter_t timerlck
- int refcnt

5.1.1 Detailed Description

Real-time pod descriptor. The source of all Xenomai magic.

5.1.2 Field Documentation

5.1.2.1 int xnpod::refcnt

Reference count.
Referenced by xnpod_init().

5.1.2.2 xnsched_t xnpod::sched[XNARCH_NR_CPUS]

Per-cpu scheduler slots.
Referenced by xnpod_init().

5.1.2.3 xnflags_t xnpod::status

Status bitmask.
Referenced by xnpod_init().

5.1.2.4 xnqueue_t xnpod::tdeleteq

Thread delete hook queue.
Referenced by xnpod_init().

5.1.2.5 xnqueue_t xnpod::threadq

All existing threads.
Referenced by xnpod_init().

5.1.2.6 atomic_counter_t xnpod::timerlck

Timer lock depth.
Referenced by xnpod_init().

5.1.2.7 xnqueue_t xnpod::tstartq

Thread start hook queue.
Referenced by xnpod_init().

5.1.2.8 xnqueue_t xnpod::tswitchq

Thread switch hook queue.

Referenced by xnpod_init().

The documentation for this struct was generated from the following file:

- include/cobalt/nucleus/pod.h

5.2 xnsched Struct Reference

Scheduling information structure.

Data Fields

- xnflags_t status
- xnflags_t lflags
- struct xnthread * curr
- struct xnsched_rt rt
- volatile unsigned inesting
- struct xntimer htimer
- struct xnthread rootcb

5.2.1 Detailed Description

Scheduling information structure.

5.2.2 Field Documentation

5.2.2.1 struct xnthread* xnsched::curr

Current thread.

Referenced by xnpod_delete_thread(), and xnpod_suspend_thread().

5.2.2.2 struct xntimer xnsched::htimer

Host timer.

Referenced by xnpod_enable_timesource(), and xntimer_tick().

5.2.2.3 volatile unsigned xnsched::inesting

Interrupt nesting level.

5.2.2.4 xnflags_t xnsched::lflags

Scheduler specific local flags bitmask.

Referenced by xnpod_schedule(), and xntimer_tick().

5.2.2.5 struct xnthread xnsched::rootcb

Root thread control block.

Referenced by xnpod_init().

5.2.2.6 struct xnsched_rt xnsched::rt

Context of built-in real-time class.

5.2.2.7 xnflags_t xnsched::status

Scheduler specific status bitmask.

Referenced by xnpod_delete_thread(), xnpod_schedule(), xntimer_freeze(), and xntimer_tick().

The documentation for this struct was generated from the following file:

- include/cobalt/nucleus/sched.h

5.3 xnthread_info Struct Reference

Structure containing thread information.

Data Fields

- unsigned long state
  Thread state,.
- int bprio
  Base priority.
- int cprio
  Current priority.
- int cpu
  CPU the thread currently runs on.
- unsigned long affinity
  Thread's CPU affinity.
- unsigned long long relpoint
  Time of next release.

- unsigned long long exectime
  Execution time in primary mode in nanoseconds.
- unsigned long modeswitches
  Number of primary->secondary mode switches.
- unsigned long ctxswitches
  Number of context switches.
- unsigned long pagefaults
  Number of triggered page faults.
- unsigned long syscalls
  Number of Xenomai syscalls.
- char name [XNOBJECT_NAME_LEN]
  Symbolic name assigned at creation.

5.3.1 Detailed Description

Structure containing thread information.

5.3.2 Field Documentation

5.3.2.1 unsigned long xnthread_info::affinity

Thread's CPU affinity.

5.3.2.2 int xnthread_info::bprio

Base priority.

5.3.2.3 int xnthread_info::cprio

Current priority.
May change through Priority Inheritance.

5.3.2.4 int xnthread_info::cpu

CPU the thread currently runs on.

5.3.2.5 unsigned long xnthread_info::ctxswitches

Number of context switches.

5.3.2.6 unsigned long long xnthread_info::exectime

Execution time in primary mode in nanoseconds.

5.3.2.7 unsigned long xnthread_info::modeswitches

Number of primary->secondary mode switches.

5.3.2.8 char xnthread_info::name[XNOBJECT_NAME_LEN]

Symbolic name assigned at creation.

5.3.2.9 unsigned long xnthread_info::pagefaults

Number of triggered page faults.

5.3.2.10 unsigned long long xnthread_info::relpoint

Time of next release.

5.3.2.11 unsigned long xnthread_info::state

Thread state,.

See also

Thread state flags.

5.3.2.12 unsigned long xnthread_info::syscalls

Number of Xenomai syscalls.

The documentation for this struct was generated from the following file:

- include/cobalt/nucleus/thread.h

5.4 xnvfile_lock_ops Struct Reference

Vfile locking operations.

Data Fields

- int(* get)(struct xnvfile *vfile)
- void(* put)(struct xnvfile *vfile)

5.4.1 Detailed Description

Vfile locking operations.

This structure describes the operations to be provided for implementing locking support on vfiles. They apply to both snapshot-driven and regular vfiles.

5.4.2 Field Documentation

5.4.2.1 int(* xnvfile_lock_ops::get)(struct xnvfile *vfile)

This handler should grab the desired lock.

Parameters

vfile A pointer to the virtual file which needs locking.

Returns

zero should be returned if the call succeeds. Otherwise, a negative error code can be returned; upon error, the current vfile operation is aborted, and the user-space caller is passed back the error value.

5.4.2.2 void(* xnvfile_lock_ops::put)(struct xnvfile *vfile)

This handler should release the lock previously grabbed by the get() handler.

Parameters

vfile A pointer to the virtual file which currently holds the lock to release.

The documentation for this struct was generated from the following file:

- include/cobalt/nucleus/vfile.h

5.5 xnvfile_regular_iterator Struct Reference

Regular vfile iterator.

Data Fields

- loff_t pos

  Current record position while iterating.

- struct seq_file * seq

  Backlink to the host sequential file supporting the vfile.

- struct xnvfile_regular * vfile

  Backlink to the vfile being read.

- char private [0]
  Start of private area.

5.5.1 Detailed Description

Regular vfile iterator.

This structure defines an iterator over a regular vfile.

5.5.2 Field Documentation

5.5.2.1 loff_t xnvfile_regular_iterator::pos

Current record position while iterating.

5.5.2.2 char xnvfile_regular_iterator::private[0]

Start of private area.

Use xnvfile_iterator_priv() to address it.

5.5.2.3 struct seq_file* xnvfile_regular_iterator::seq

Backlink to the host sequential file supporting the vfile.

5.5.2.4 struct xnvfile_regular* xnvfile_regular_iterator::vfile

Backlink to the vfile being read.

The documentation for this struct was generated from the following file:

- include/cobalt/nucleus/vfile.h

5.6 xnvfile_regular_ops Struct Reference

Regular vfile operation descriptor.

Data Fields

- int(* rewind)(struct xnvfile_regular_iterator *it)
- void *(* begin)(struct xnvfile_regular_iterator *it)
- void *(* next)(struct xnvfile_regular_iterator *it)
- void(* end)(struct xnvfile_regular_iterator *it)
- int(* show)(struct xnvfile_regular_iterator *it, void *data)
- ssize_t(* store)(struct xnvfile_input *input)

5.6.1 Detailed Description

Regular vfile operation descriptor.

This structure describes the operations available with a regular vfile. It defines handlers for sending back formatted kernel data upon a user-space read request, and for obtaining user data upon a user-space write request.

5.6.2 Field Documentation

5.6.2.1 void*(* xnvfile_regular_ops::begin)(struct xnvfile_regular_iterator *it)

This handler should prepare for iterating over the records upon a read request, starting from the specified position.

Parameters

it A pointer to the current vfile iterator. On entry, it->pos is set to the (0-based) position of the first record to output. This handler may be called multiple times with different position requests.

Returns

A pointer to the first record to format and output, to be passed to the show() handler as its *data* parameter, if the call succeeds. Otherwise:

- NULL in case no record is available, in which case the read operation will terminate immediately with no output.
- VFILE_SEQ_START, a special value indicating that the show() handler should receive a NULL data pointer first, in order to output a header.
- ERR_PTR(errno), where errno is a negative error code; upon error, the current operation will be aborted immediately.

Note

This handler is optional; if none is given in the operation descriptor (i.e. NULL value), the show() handler() will be called only once for a read operation, with a NULL *data* parameter. This particular setting is convenient for simple regular vfiles having a single, fixed record to output.

5.6.2.2 void(* xnvfile_regular_ops::end)(struct xnvfile_regular_iterator *it)

This handler is called after all records have been output.

Parameters

it A pointer to the current vfile iterator.

Note

This handler is optional and the pointer may be NULL.

5.6.2.3 void*(* xnvfile_regular_ops::next)(struct xnvfile_regular_iterator *it)

This handler should return the address of the next record to format and output by the show() handler".

Parameters

it A pointer to the current vfile iterator. On entry, it->pos is set to the (0-based) position of the next record to output.

Returns

A pointer to the next record to format and output, to be passed to the show() handler as its *data* parameter, if the call succeeds. Otherwise:

- NULL in case no record is available, in which case the read operation will terminate immediately with no output.
- ERR_PTR(errno), where errno is a negative error code; upon error, the current operation will be aborted immediately.

Note

This handler is optional; if none is given in the operation descriptor (i.e. NULL value), the read operation will stop after the first invocation of the show() handler.

5.6.2.4 int(* xnvfile_regular_ops::rewind)(struct xnvfile_regular_iterator *it)

This handler is called only once, when the virtual file is opened, before the begin() handler is invoked.

Parameters

it A pointer to the vfile iterator which will be used to read the file contents.

Returns

Zero should be returned upon success. Otherwise, a negative error code aborts the operation, and is passed back to the reader.

Note

This handler is optional. It should not be used to allocate resources but rather to perform consistency checks, since no closure call is issued in case the open sequence eventually fails.

5.6.2.5 int(* xnvfile_regular_ops::show)(struct xnvfile_regular_iterator *it, void *data)

This handler should format and output a record.

xnvfile_printf(), xnvfile_write(), xnvfile_puts() and xnvfile_putc() are available to format and/or emit the output. All routines take the iterator argument *it* as their first parameter.

Parameters

it A pointer to the current vfile iterator.

data A pointer to the record to format then output. The first call to the handler may receive a NULL *data* pointer, depending on the presence and/or return of a hander; the show handler should test this special value to output any header that fits, prior to receiving more calls with actual records.

Returns

zero if the call succeeds, also indicating that the handler should be called for the next record if any. Otherwise:

- A negative error code. This will abort the output phase, and return this status to the reader.
- VFILE_SEQ_SKIP, a special value indicating that the current record should be skipped and will not be output.

5.6.2.6 ssize_t(* xnvfile_regular_ops::store)(struct xnvfile_input *input)

This handler receives data written to the vfile, likely for updating some kernel setting, or triggering any other action which fits. This is the only handler which deals with the write-side of a vfile. It is called when writing to the /proc entry of the vfile from a user-space process.

The input data is described by a descriptor passed to the handler, which may be subsequently passed to parsing helper routines. For instance, xnvfile_get_string() will accept the input descriptor for returning the written data as a null-terminated character string. On the other hand, xnvfile_get_integer() will attempt to return a long integer from the input data.

Parameters

input A pointer to an input descriptor. It refers to an opaque data from the handler's standpoint.

Returns

the number of bytes read from the input descriptor if the call succeeds. Otherwise, a negative error code. Return values from parsing helper routines are commonly passed back to the caller by the store() handler.

Note

This handler is optional, and may be omitted for read-only vfiles.

The documentation for this struct was generated from the following file:

- include/cobalt/nucleus/vfile.h

5.7 xnvfile_rev_tag Struct Reference

Snapshot revision tag.

Data Fields

- int rev

  Current revision number.

5.7.1 Detailed Description

Snapshot revision tag.

This structure defines a revision tag to be used with snapshot-driven vfiles.

5.7.2 Field Documentation

5.7.2.1 int xnvfile_rev_tag::rev

Current revision number.

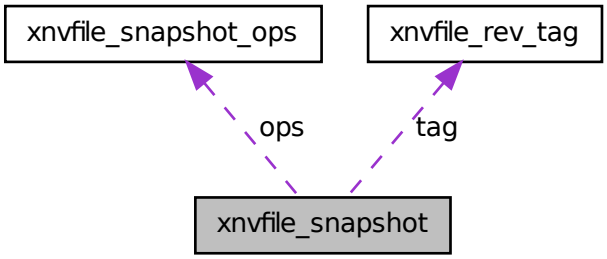
The documentation for this struct was generated from the following file:

- include/cobalt/nucleus/vfile.h

5.8 xnvfile_snapshot Struct Reference

Snapshot vfile descriptor.

Collaboration diagram for xnvfile_snapshot:



5.8.1 Detailed Description

Snapshot vfile descriptor.

This structure describes a snapshot-driven vfile. Reading from such a vfile involves a preliminary data collection phase under lock protection, and a subsequent formatting and output phase of the collected data records. Locking is done in a way that does not increase worst-case latency, regardless of the number of records to be collected for output.

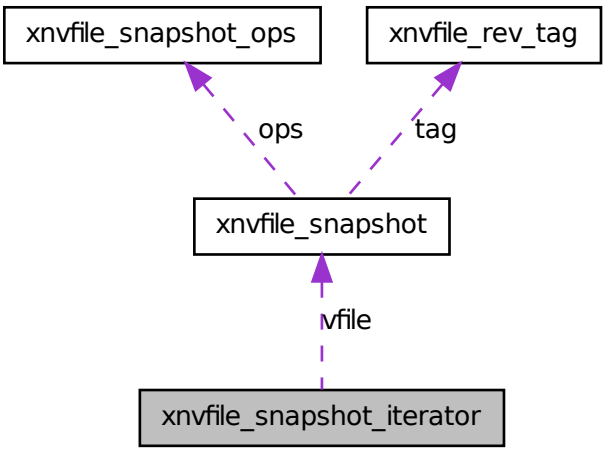
The documentation for this struct was generated from the following file:

- include/cobalt/nucleus/vfile.h

5.9 xnvfile_snapshot_iterator Struct Reference

Snapshot-driven vfile iterator.

Collaboration diagram for xnvfile_snapshot_iterator:



Data Fields

- int nrdata

  Number of collected records.

- caddr_t databuf

  Address of record buffer.

- struct seq_file * seq

  Backlink to the host sequential file supporting the vfile.

- struct xnvfile_snapshot * vfile

  Backlink to the vfile being read.

- void(* endfn)(struct xnvfile_snapshot_iterator *it, void *buf)

  Buffer release handler.

- char private [0]

  Start of private area.

5.9.1 Detailed Description

Snapshot-driven vfile iterator.

This structure defines an iterator over a snapshot-driven vfile.

5.9.2 Field Documentation

5.9.2.1 caddr_t xnvfile_snapshot_iterator::databuf

Address of record buffer.

5.9.2.2 void(* xnvfile_snapshot_iterator::endfn)(struct xnvfile_snapshot_iterator *it, void *buf)

Buffer release handler.

5.9.2.3 int xnvfile_snapshot_iterator::nrdata

Number of collected records.

5.9.2.4 char xnvfile_snapshot_iterator::private[0]

Start of private area.

Use xnvfile_iterator_priv() to address it.

5.9.2.5 struct seq_file* xnvfile_snapshot_iterator::seq

Backlink to the host sequential file supporting the vfile.

5.9.2.6 struct xnvfile_snapshot* xnvfile_snapshot_iterator::vfile

Backlink to the vfile being read.

The documentation for this struct was generated from the following file:

- include/cobalt/nucleus/vfile.h

5.10 xnvfile_snapshot_ops Struct Reference

Snapshot vfile operation descriptor.

Data Fields

- int(* rewind)(struct xnvfile_snapshot_iterator *it)
- void *(* begin)(struct xnvfile_snapshot_iterator *it)

- void(* end)(struct xnvfile_snapshot_iterator *it, void *buf)
- int(* next)(struct xnvfile_snapshot_iterator *it, void *data)
- int(* show)(struct xnvfile_snapshot_iterator *it, void *data)
- ssize_t(* store)(struct xnvfile_input *input)

5.10.1 Detailed Description

Snapshot vfile operation descriptor.

This structure describes the operations available with a snapshot-driven vfile. It defines handlers for returning a printable snapshot of some Xenomai object contents upon a user-space read request, and for updating this object upon a user-space write request.

5.10.2 Field Documentation

5.10.2.1 void*(* xnvfile_snapshot_ops::begin)(struct xnvfile_snapshot_iterator *it)

This handler should allocate the snapshot buffer to hold records during the data collection phase. When specified, all records collected via the next() handler" will be written to a cell from the memory area returned by begin().

Parameters

it A pointer to the current snapshot iterator.

Returns

A pointer to the record buffer, if the call succeeds. Otherwise:

- NULL in case of allocation error. This will abort the data collection, and return -ENOMEM to the reader.
- VFILE_SEQ_EMPTY, a special value indicating that no record will be output. In such a case, the next() handler will not be called, and the data collection will stop immediately. However, the show() handler will still be called once, with a NULL data pointer (i.e. header display request).

Note

This handler is optional; if none is given, an internal allocation depending on the value returned by the rewind() handler can be obtained.

5.10.2.2 void(* xnvfile_snapshot_ops::end)(struct xnvfile_snapshot_iterator *it, void *buf)

This handler releases the memory buffer previously obtained from begin(). It is usually called after the snapshot data has been output by show(), but it may also be called before rewinding the vfile after a revision change, to release the dropped buffer.

Parameters

it A pointer to the current snapshot iterator.

buf A pointer to the buffer to release.

Note

This routine is optional and the pointer may be NULL. It is not needed upon internal buffer allocation; see the description of the rewind() handler".

5.10.2.3 int(* xnvfile_snapshot_ops::next)(struct xnvfile_snapshot_iterator *it, void *data)

This handler fetches the next record, as part of the snapshot data to be sent back to the reader via the show().

Parameters

it A pointer to the current snapshot iterator.

data A pointer to the record to fill in.

Returns

a strictly positive value, if the call succeeds and leaves a valid record into *data*, which should be passed to the show() handler() during the formatting and output phase. Otherwise:

- A negative error code. This will abort the data collection, and return this status to the reader.
- VFILE_SEQ_SKIP, a special value indicating that the current record should be skipped. In such a case, the *data* pointer is not advanced to the next position before the next() handler is called anew.

Note

This handler is called with the vfile lock held. Before each invocation of this handler, the vfile core checks whether the revision tag has been touched, in which case the data collection is restarted from scratch. A data collection phase succeeds whenever all records can be fetched via the next() handler, while the revision tag remains unchanged, which indicates that a consistent snapshot of the object state was taken.

5.10.2.4 int(* xnvfile_snapshot_ops::rewind)(struct xnvfile_snapshot_iterator *it)

This handler (re-)initializes the data collection, moving the seek pointer at the first record. When the file revision tag is touched while collecting data, the current reading is aborted, all collected data dropped, and the vfile is eventually rewound.

Parameters

it A pointer to the current snapshot iterator. Two useful information can be retrieved from this iterator in this context:

- it->vfile is a pointer to the descriptor of the virtual file being rewound.
- xnvfile_iterator_priv(it) returns a pointer to the private data area, available from the descriptor, which size is vfile->privsz. If the latter size is zero, the returned pointer is meaningless and should not be used.

Returns

A negative error code aborts the data collection, and is passed back to the reader. Otherwise:

- a strictly positive value is interpreted as the total number of records which will be returned by the next() handler during the data collection phase. If no begin() handler is provided in the operation descriptor, this value is used to allocate the snapshot buffer internally. The size of this buffer would then be vfile->datasz * value.
- zero leaves the allocation to the begin() handler if present, or indicates that no record is to be output in case such handler is not given.

Note

This handler is optional; a NULL value indicates that nothing needs to be done for rewinding the vfile. It is called with the vfile lock held.

5.10.2.5 int(* xnvfile_snapshot_ops::show)(struct xnvfile_snapshot_iterator *it, void *data)

This handler should format and output a record from the collected data.

xnvfile_printf(), xnvfile_write(), xnvfile_puts() and xnvfile_putc() are available to format and/or emit the output. All routines take the iterator argument *it* as their first parameter.

Parameters

it A pointer to the current snapshot iterator.

data A pointer to the record to format then output. The first call to the handler is always passed a NULL *data* pointer; the show handler should test this special value to output any header that fits, prior to receiving more calls with actual records.

Returns

zero if the call succeeds, also indicating that the handler should be called for the next record if any. Otherwise:

- A negative error code. This will abort the output phase, and return this status to the reader.
- VFILE_SEQ_SKIP, a special value indicating that the current record should be skipped and will not be output.

5.10.2.6 ssize_t(* xnvfile_snapshot_ops::store)(struct xnvfile_input *input)

This handler receives data written to the vfile, likely for updating the associated Xenomai object's state, or triggering any other action which fits. This is the only handler which deals with the write-side of a vfile. It is called when writing to the /proc entry of the vfile from a user-space process.

The input data is described by a descriptor passed to the handler, which may be subsequently passed to parsing helper routines. For instance, xnvfile_get_string() will accept the input descriptor for returning the written data as a null-terminated character string. On the other hand, xnvfile_get_integer() will attempt to return a long integer from the input data.

Parameters

input A pointer to an input descriptor. It refers to an opaque data from the handler's standpoint.

Returns

the number of bytes read from the input descriptor if the call succeeds. Otherwise, a negative error code. Return values from parsing helper routines are commonly passed back to the caller by the store() handler.

Note

This handler is optional, and may be omitted for read-only vfiles.

Referenced by xnvfile_init_snapshot().

The documentation for this struct was generated from the following file:

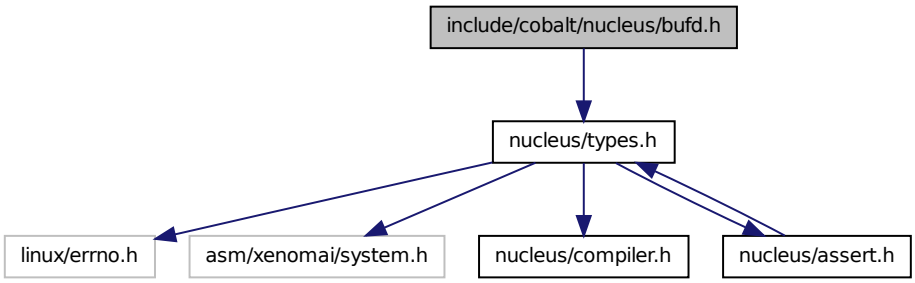
- include/cobalt/nucleus/vfile.h

Chapter 6

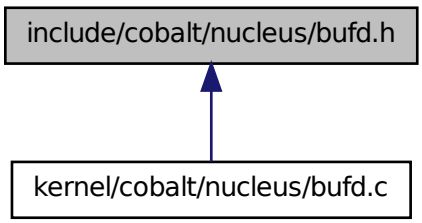
File Documentation

6.1 include/cobalt/nucleus/bufd.h File Reference

Include dependency graph for bufd.h:



This graph shows which files directly or indirectly include this file:



Functions

- static void xnbufd_map_uread (struct xnbufd *bufd, const void __user *ptr, size_t len)
  Initialize a buffer descriptor for reading from user memory.
- static void xnbufd_map_uwrite (struct xnbufd *bufd, void __user *ptr, size_t len)
  Initialize a buffer descriptor for writing to user memory.
- ssize_t xnbufd_unmap_uread (struct xnbufd *bufd)
  Finalize a buffer descriptor obtained from xnbufd_map_uread().
- ssize_t xnbufd_unmap_uwrite (struct xnbufd *bufd)
  Finalize a buffer descriptor obtained from xnbufd_map_uwrite().
- static void xnbufd_map_kread (struct xnbufd *bufd, const void *ptr, size_t len)
  Initialize a buffer descriptor for reading from kernel memory.
- static void xnbufd_map_kwrite (struct xnbufd *bufd, void *ptr, size_t len)
  Initialize a buffer descriptor for writing to kernel memory.
- ssize_t xnbufd_unmap_kread (struct xnbufd *bufd)
  Finalize a buffer descriptor obtained from xnbufd_map_kread().
- ssize_t xnbufd_unmap_kwrite (struct xnbufd *bufd)
  Finalize a buffer descriptor obtained from xnbufd_map_kwrite().
- ssize_t xnbufd_copy_to_kmem (void *ptr, struct xnbufd *bufd, size_t len)
  Copy memory covered by a buffer descriptor to kernel memory.
- ssize_t xnbufd_copy_from_kmem (struct xnbufd *bufd, void *from, size_t len)
  Copy kernel memory to the area covered by a buffer descriptor.
- void xnbufd_invalidate (struct xnbufd *bufd)
  Invalidate a buffer descriptor.
- static void xnbufd_reset (struct xnbufd *bufd)
  Reset a buffer descriptor.

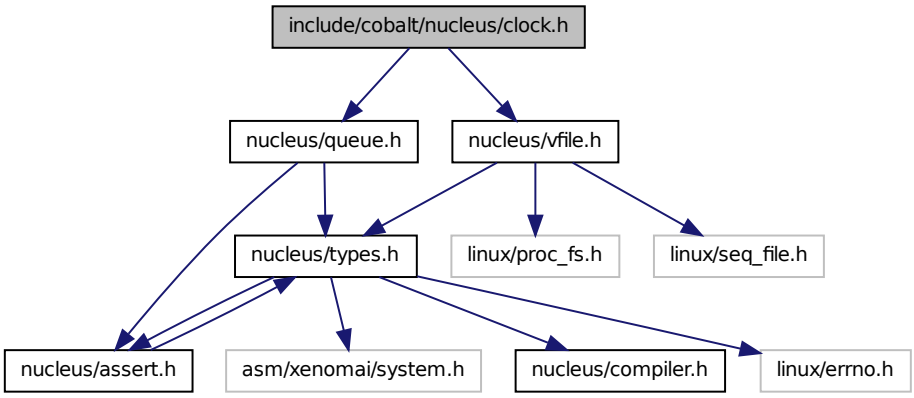
6.1.1 Detailed Description

Note

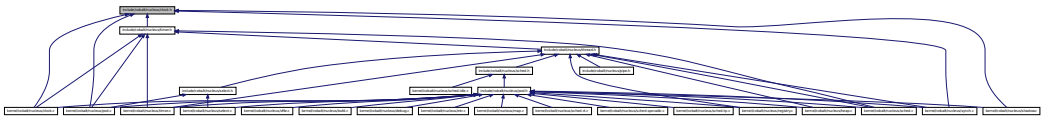
You should have received a copy of the GNU General Public License along with Xenomai; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

6.2 include/cobalt/nucleus/clock.h File Reference

Include dependency graph for clock.h:



This graph shows which files directly or indirectly include this file:



Functions

- void xnclock_adjust (xnsticks_t delta)

  Adjust the clock time for the system.

6.2.1 Detailed Description

Note

Copyright (C) 2006,2007 Philippe Gerum <rpm@xenomai.org>.

Xenomai is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

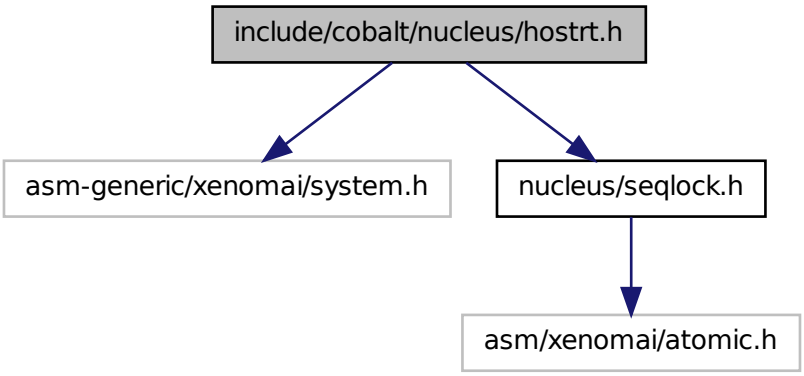
Xenomai is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with Xenomai; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

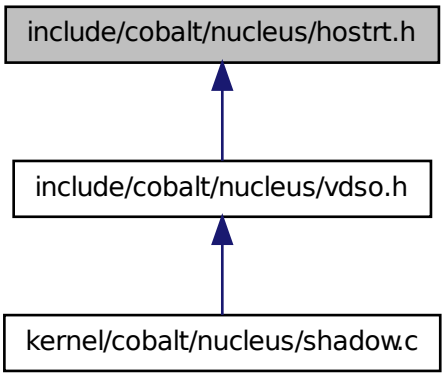
6.3 include/cobalt/nucleus/hostrt.h File Reference

Definitions for global semaphore heap shared objects.

Include dependency graph for hostrt.h:



This graph shows which files directly or indirectly include this file:



6.3.1 Detailed Description

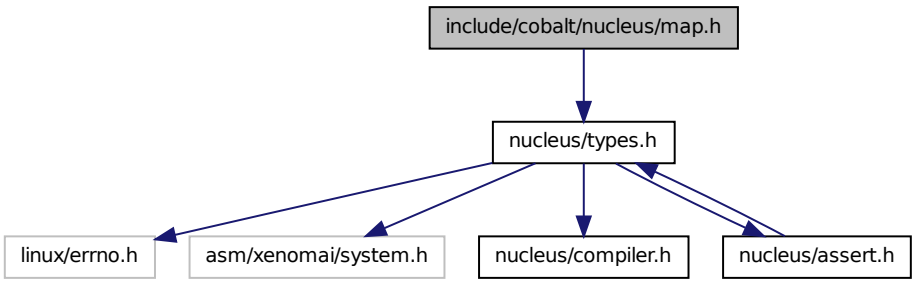
Definitions for global semaphore heap shared objects.

Author

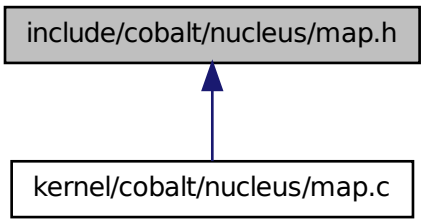
Wolfgang Mauerer

6.4 include/cobalt/nucleus/map.h File Reference

Include dependency graph for map.h:



This graph shows which files directly or indirectly include this file:



Functions

- xnmap_t * xnmap_create (int nkeys, int reserve, int offset)
  Create a map.
- void xnmap_delete (xnmap_t *map)
  Delete a map.
- int xnmap_enter (xnmap_t *map, int key, void *objaddr)
  Index an object into a map.
- int xnmap_remove (xnmap_t *map, int key)
  Remove an object reference from a map.

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

  Search an object into a map - unchecked form.

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

  Search an object into a map.

6.4.1 Detailed Description

Note

Copyright (C) 2007 Philippe Gerum <rpm@xenomai.org>.

Xenomai is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

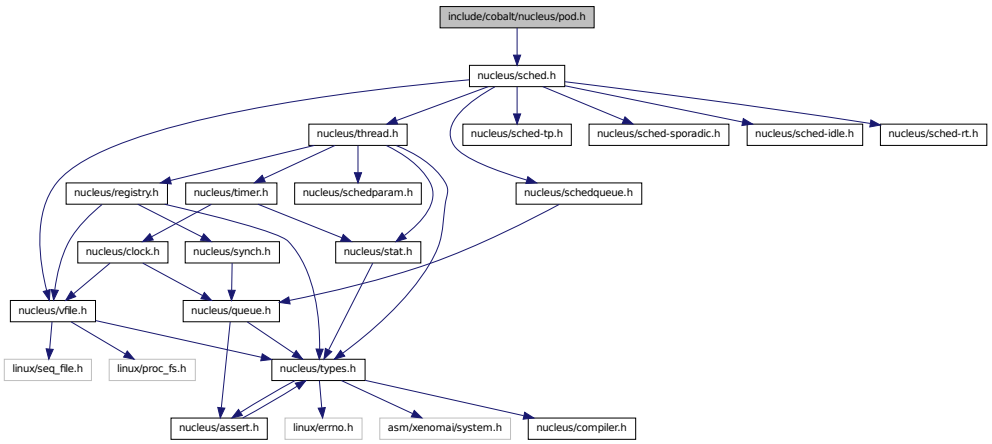
Xenomai is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with Xenomai; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

6.5 include/cobalt/nucleus/pod.h File Reference

Real-time pod interface header.

Include dependency graph for pod.h:



This graph shows which files directly or indirectly include this file:

Data Structures

- struct xnpod

  Real-time pod descriptor.

Functions

- void __xnpod_reset_thread (struct xnthread *thread)

  Reset the thread.

- int xnpod_init (void)

  Initialize the core pod.

- int xnpod_enable_timesource (void)

  Activate the core time source.

- void xnpod_disable_timesource (void)

  Stop the core time source.

- void xnpod_shutdown (int xtype)

  Shutdown the current pod.

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

  Initialize a new thread.

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

  Initial start of a newly created thread.

- void xnpod_stop_thread (xnthread_t *thread)

  Stop a thread.

- void xnpod_delete_thread (xnthread_t *thread)

  Delete a thread.

- void xnpod_abort_thread (xnthread_t *thread)

  Abort a thread.

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

  Change a thread's control mode.

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

  Suspend a thread.

- void xnpod_resume_thread (xnthread_t *thread, xnflags_t mask)

  Resume a thread.

- int xnpod_unblock_thread (xnthread_t *thread)

  Unblock a thread.

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

  Change the base scheduling parameters of a thread.

- int xnpod_migrate_thread (int cpu)

  Migrate the current thread.

- void xnpod_dispatch_signals (void)

  Deliver pending asynchronous signals to the running thread.

- static void xnpod_schedule (void)

  Rescheduling procedure entry point.

- int xnpod_handle_exception (struct ipipe_trap_data *d)

  Exception handler.

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

  Make a thread periodic.

- int xnpod_wait_thread_period (unsigned long *overruns_r)

  Wait for the next periodic release point.

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

  Set thread time-slicing information.

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

  Install a nucleus hook.

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

  Remove a nucleus hook.

6.5.1 Detailed Description

Real-time pod interface header.

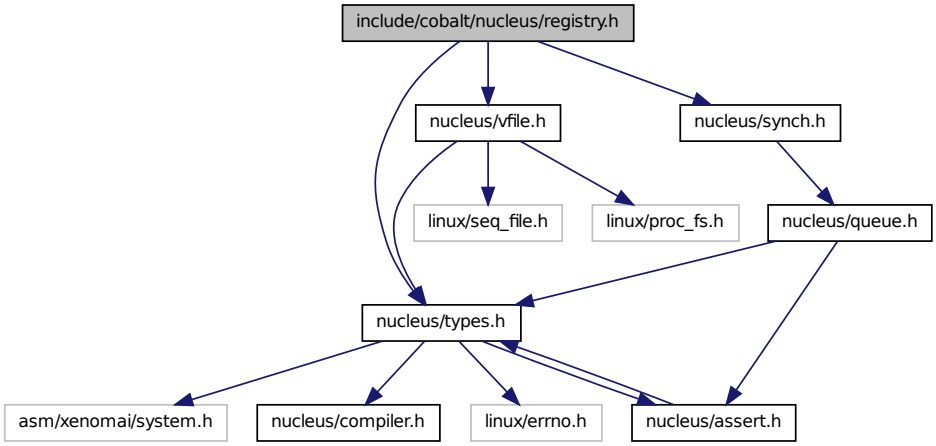
Author

Philippe Gerum

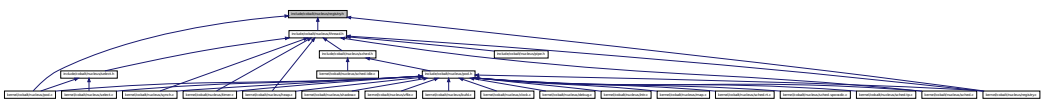
6.6 include/cobalt/nucleus/registry.h File Reference

This file is part of the Xenomai project.

Include dependency graph for registry.h:



This graph shows which files directly or indirectly include this file:



Functions

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

  Register a real-time object.

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

  Bind to a real-time object.

- int xnregistry_remove (xnhandle_t handle)

  Forcibly unregister a real-time object.

- int xnregistry_remove_safe (xnhandle_t handle, xnticks_t timeout)

  Unregister an idle real-time object.

- void * xnregistry_get (xnhandle_t handle)

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

- void * xnregistry_fetch (xnhandle_t handle)

  Find a real-time object into the registry.

- u_long xnregistry_put (xnhandle_t handle)

  Unlock a real-time object from the registry.

6.6.1 Detailed Description

This file is part of the Xenomai project.

Note

Copyright (C) 2004 Philippe Gerum <rpm@xenomai.org>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

6.7 include/cobalt/nucleus/sched-idle.h File Reference

Definitions for the IDLE scheduling class.

This graph shows which files directly or indirectly include this file:

6.7.1 Detailed Description

Definitions for the IDLE scheduling class.

Author

Philippe Gerum

6.8 include/cobalt/nucleus/sched-rt.h File Reference

Definitions for the RT scheduling class.

This graph shows which files directly or indirectly include this file:

6.8.1 Detailed Description

Definitions for the RT scheduling class.

Author

Philippe Gerum

Xenomai is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with Xenomai; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

6.9 include/cobalt/nucleus/sched-sporadic.h File Reference

Definitions for the SSP scheduling class.

This graph shows which files directly or indirectly include this file:

6.9.1 Detailed Description

Definitions for the SSP scheduling class.

Author

Philippe Gerum

Copyright (C) 2009 Philippe Gerum <rpm@xenomai.org>.

Xenomai is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

Xenomai is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with Xenomai; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

6.10 include/cobalt/nucleus/sched-tp.h File Reference

Definitions for the TP scheduling class.

This graph shows which files directly or indirectly include this file:

6.10.1 Detailed Description

Definitions for the TP scheduling class.

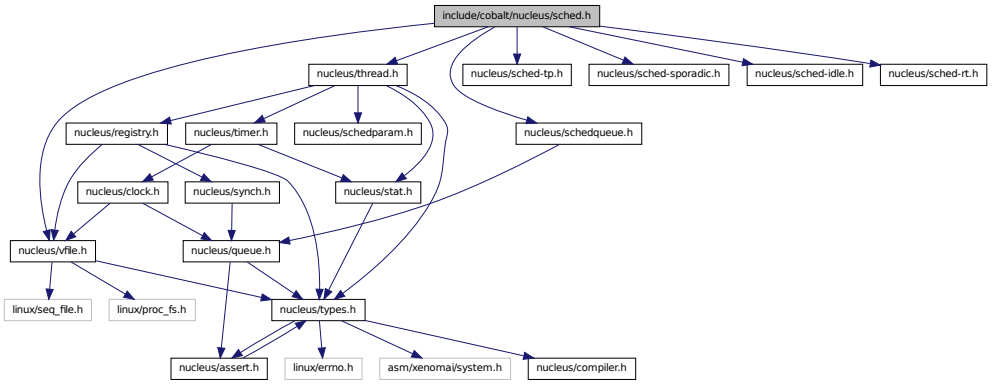
Author

Philippe Gerum

6.11 include/cobalt/nucleus/sched.h File Reference

Scheduler interface header.

Include dependency graph for sched.h:



This graph shows which files directly or indirectly include this file:

Data Structures

- struct xnsched

Scheduling information structure.

Typedefs

- typedef struct xnsched xnsched_t

  Scheduling information structure.

Functions

- static void xnsched_rotate (struct xnsched *sched, struct xnsched_class *sched_class, const union xnsched_policy_param *sched_param)

  Rotate a scheduler runqueue.

6.11.1 Detailed Description

Scheduler interface header.

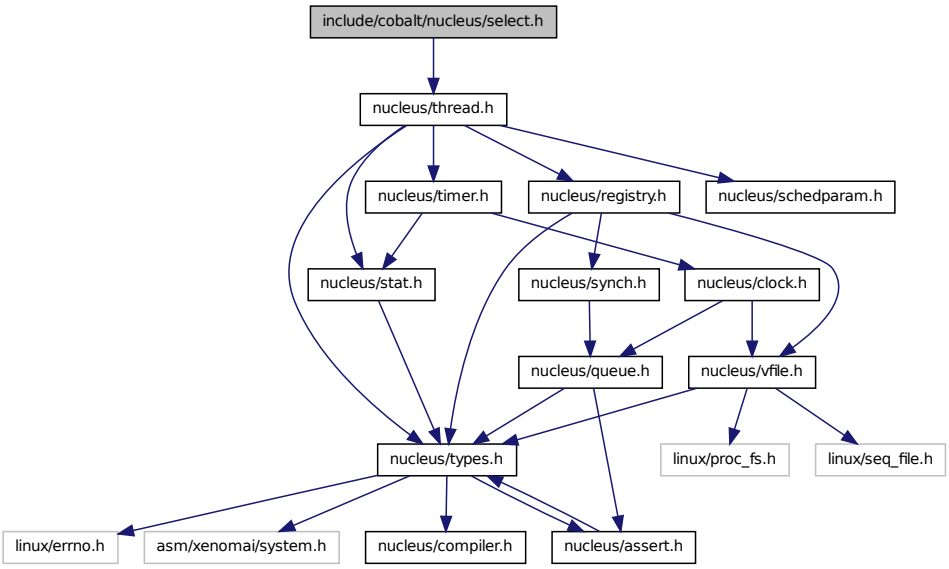
Author

Philippe Gerum

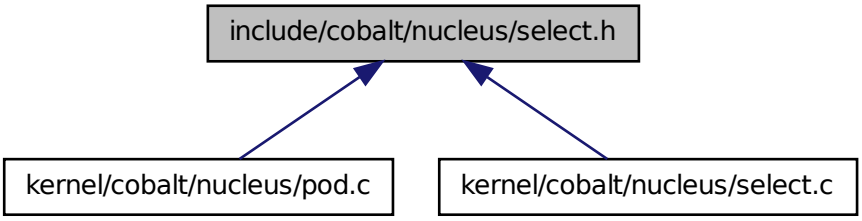
6.12 include/cobalt/nucleus/select.h File Reference

file descriptors events multiplexing header.

Include dependency graph for select.h:



This graph shows which files directly or indirectly include this file:



Functions

- void xnselect_init (struct xnselect *select_block)

  Initialize a struct xnselect structure.

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

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

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

  Signal a file descriptor state change.

- void xnselect_destroy (struct xnselect *select_block)

  Destroy the xnselect structure associated with a file descriptor.

- int xnselector_init (struct xnselector *selector)

  Initialize a selector structure.

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

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

- void xnselector_destroy (struct xnselector *selector)

  Destroy a selector block.

6.12.1 Detailed Description

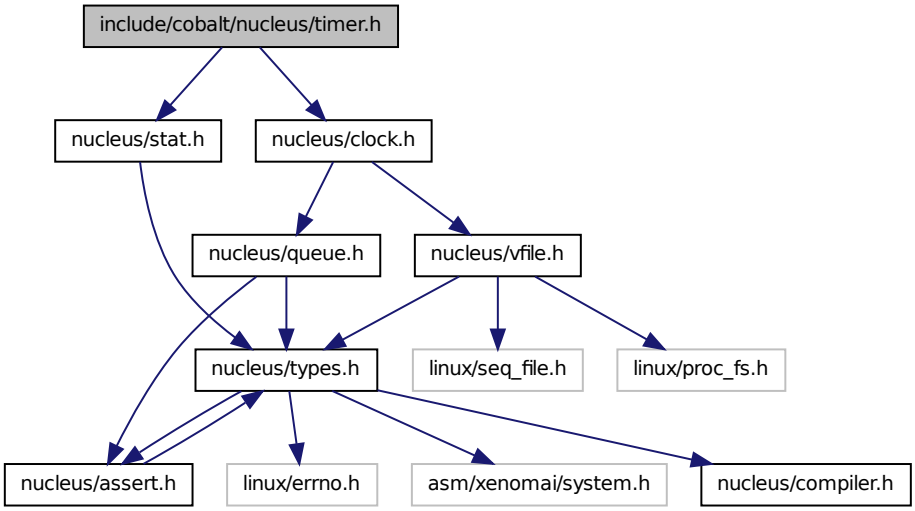
file descriptors events multiplexing header.

Author

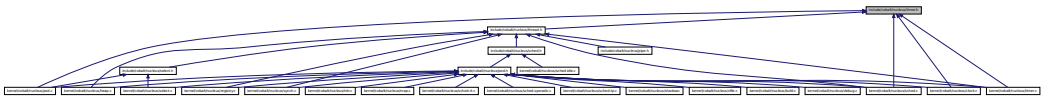
Gilles Chanteperdrix

6.13 include/cobalt/nucleus/timer.h File Reference

Include dependency graph for timer.h:



This graph shows which files directly or indirectly include this file:



Functions

- void xntimer_destroy (xntimer_t *timer)
  Release a timer object.
- int xntimer_start (xntimer_t *timer, xnticks_t value, xnticks_t interval, xntmode_t mode)
  Arm a timer.
- xnticks_t xntimer_get_date (xntimer_t *timer)
  Return the absolute expiration date.
- xnticks_t xntimer_get_timeout (xntimer_t *timer)
  Return the relative expiration date.
- xnticks_t xntimer_get_interval (xntimer_t *timer)
  Return the timer interval value.

- static void xntimer_stop (xntimer_t *timer)

  Disarm a timer.

- unsigned long xntimer_get_overruns (xntimer_t *timer, xnticks_t now)

  Get the count of overruns for the last tick.

- void xntimer_freeze (void)

  Freeze all timers (from every time bases).

- void xntimer_tick (void)

  Process a timer tick.

6.13.1 Detailed Description

Note

Copyright (C) 2001,2002,2003 Philippe Gerum <rpm@xenomai.org>.

Xenomai is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

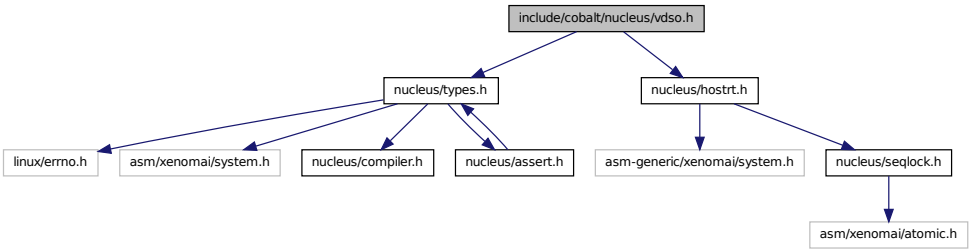
Xenomai is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with Xenomai; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

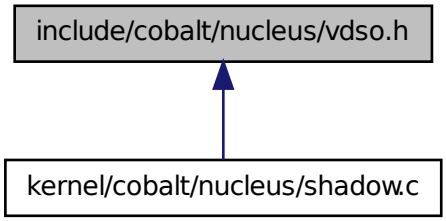
6.14 include/cobalt/nucleus/vdso.h File Reference

Definitions for global semaphore heap shared objects.

Include dependency graph for vdso.h:



This graph shows which files directly or indirectly include this file:



6.14.1 Detailed Description

Definitions for global semaphore heap shared objects.

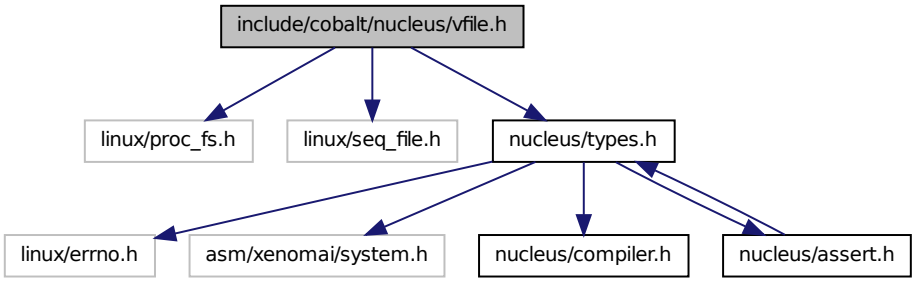
Author

Wolfgang Mauerer

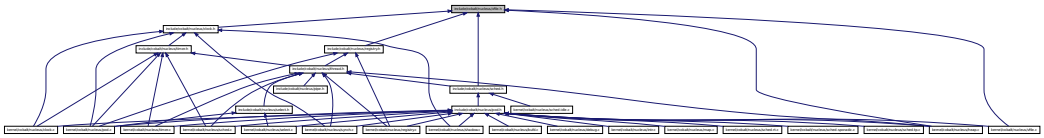
6.15 include/cobalt/nucleus/vfile.h File Reference

This file is part of the Xenomai project.

Include dependency graph for vfile.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct xnvfile_lock_ops

  Vfile locking operations.

- struct xnvfile_regular_ops

  Regular vfile operation descriptor.

- struct xnvfile_regular_iterator

  Regular vfile iterator.

- struct xnvfile_snapshot_ops

  Snapshot vfile operation descriptor.

- struct xnvfile_rev_tag

  Snapshot revision tag.

- struct xnvfile_snapshot

  Snapshot vfile descriptor.

- struct xnvfile_snapshot_iterator

  Snapshot-driven vfile iterator.

Functions

- int xnvfile_init_snapshot (const char *name, struct xnvfile_snapshot *vfile, struct xnvfile_-directory *parent)

  Initialize a snapshot-driven vfile.

- int xnvfile_init_regular (const char *name, struct xnvfile_regular *vfile, struct xnvfile_-directory *parent)

  Initialize a regular vfile.

- int xnvfile_init_dir (const char *name, struct xnvfile_directory *vdir, struct xnvfile_directory *parent)

  Initialize a virtual directory entry.

- int xnvfile_init_link (const char *from, const char *to, struct xnvfile_link *vlink, struct xnvfile_directory *parent)

  Initialize a virtual link entry.

- void xnvfile_destroy (struct xnvfile *vfile)

  Removes a virtual file entry.

- ssize_t xnvfile_get_blob (struct xnvfile_input *input, void *data, size_t size)

  Read in a data bulk written to the vfile.

- ssize_t xnvfile_get_string (struct xnvfile_input *input, char *s, size_t maxlen)

  Read in a C-string written to the vfile.

- ssize_t xnvfile_get_integer (struct xnvfile_input *input, long *valp)

  Evaluate the string written to the vfile as a long integer.

Variables

- struct xnvfile_directory nkvfroot

  Xenomai vfile root directory.

6.15.1 Detailed Description

This file is part of the Xenomai project.

Note

Copyright (C) 2010 Philippe Gerum <rpm@xenomai.org>

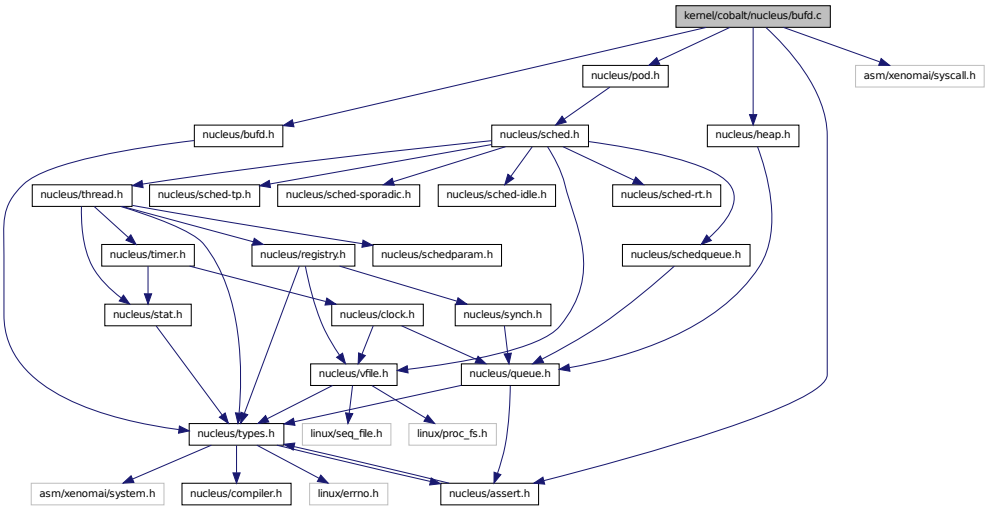
This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

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.

6.16 kernel/cobalt/nucleus/bufd.c File Reference

Include dependency graph for bufd.c:



Functions

- ssize_t xnbufd_copy_to_kmem (void *ptr, struct xnbufd *bufd, size_t len)
  Copy memory covered by a buffer descriptor to kernel memory.
- ssize_t xnbufd_copy_from_kmem (struct xnbufd *bufd, void *from, size_t len)
  Copy kernel memory to the area covered by a buffer descriptor.
- ssize_t xnbufd_unmap_uread (struct xnbufd *bufd)
  Finalize a buffer descriptor obtained from xnbufd_map_uread().
- ssize_t xnbufd_unmap_uwrite (struct xnbufd *bufd)
  Finalize a buffer descriptor obtained from xnbufd_map_uwrite().
- void xnbufd_invalidate (struct xnbufd *bufd)
  Invalidate a buffer descriptor.
- ssize_t xnbufd_unmap_kread (struct xnbufd *bufd)
  Finalize a buffer descriptor obtained from xnbufd_map_kread().
- ssize_t xnbufd_unmap_kwrite (struct xnbufd *bufd)

Finalize a buffer descriptor obtained from xnbufd_map_kwrite().

6.16.1 Detailed Description

Note

Copyright (C) 2009 Philippe Gerum <rpm@xenomai.org>.

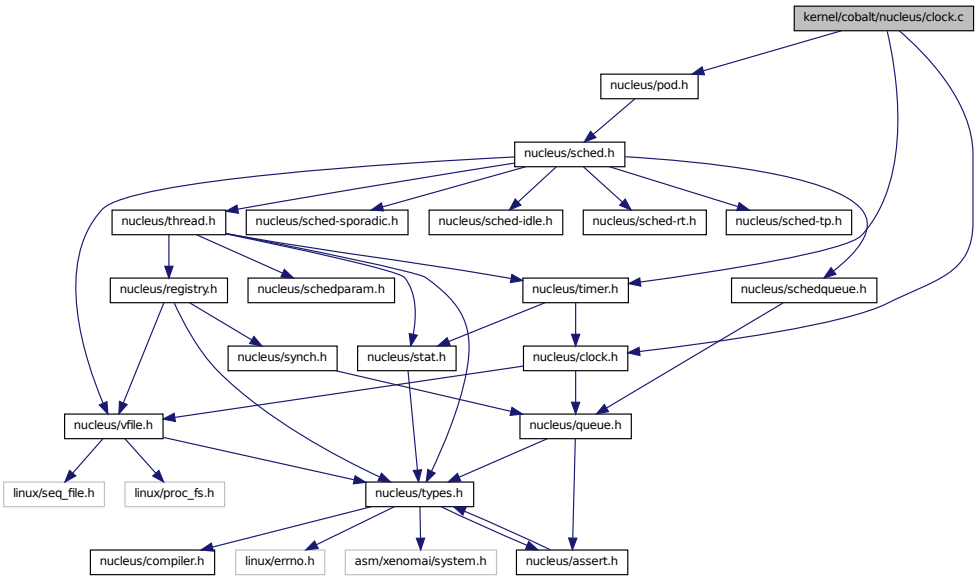
Xenomai is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

Xenomai is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with Xenomai; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

6.17 kernel/cobalt/nucleus/clock.c File Reference

Include dependency graph for clock.c:



Functions

- void xnclock_adjust (xnsticks_t delta)

  Adjust the clock time for the system.

6.17.1 Detailed Description

Note

Copyright (C) 2006-2011 Philippe Gerum <rpm@xenomai.org>.

Xenomai is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

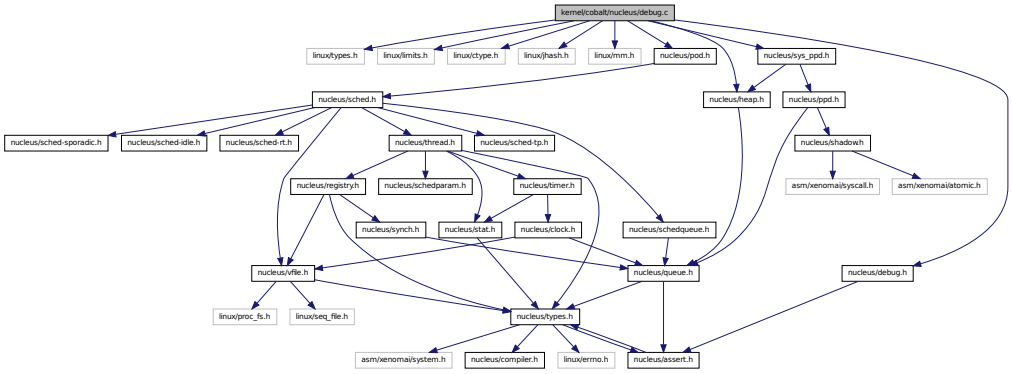
Xenomai is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with Xenomai; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

6.18 kernel/cobalt/nucleus/debug.c File Reference

Debug services.

Include dependency graph for debug.c:



6.18.1 Detailed Description

Debug services.

Author

Philippe Gerum

Copyright (C) 2010 Philippe Gerum <rpm@xenomai.org>.

Xenomai is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

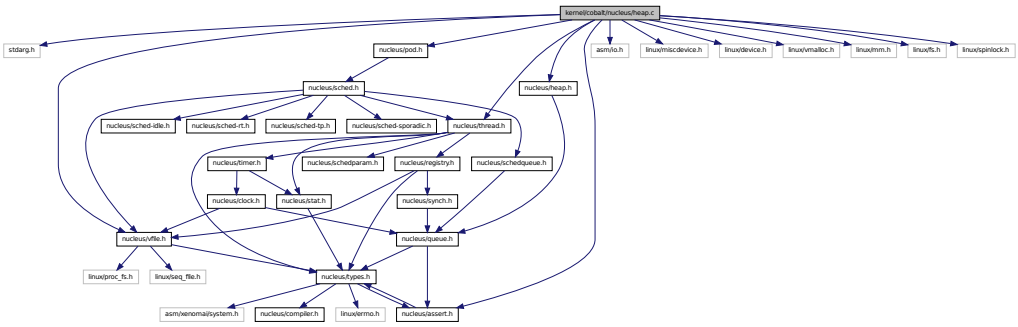
Xenomai is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with Xenomai; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

6.19 kernel/cobalt/nucleus/heap.c File Reference

Dynamic memory allocation services.

Include dependency graph for heap.c:



Functions

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

  Initialize a memory heap.

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

  Set the heap's label string.

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

  Destroys a memory heap.

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

  Allocate a memory block from a memory heap.

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

  Test and release a memory block to a memory heap.

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

  Release a memory block to a memory heap.

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

  Extend a memory heap.

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

  Schedule a memory block for release.

6.19.1 Detailed Description

Dynamic memory allocation services.

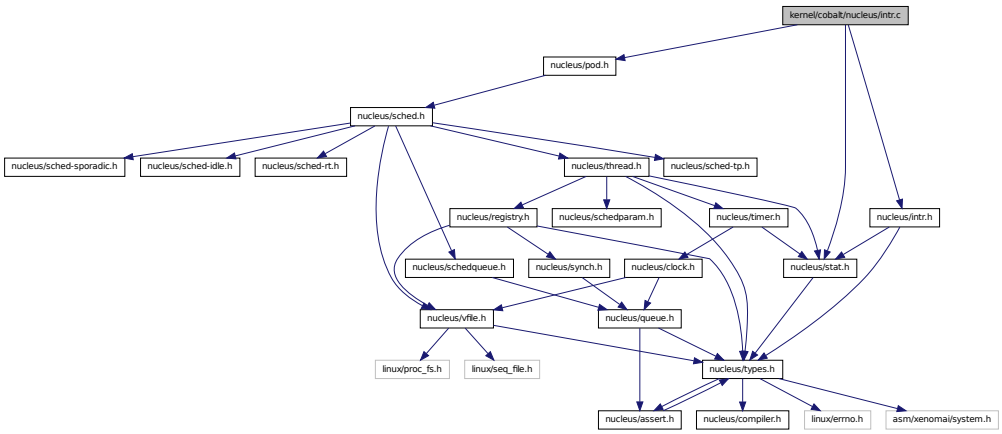
Author

Philippe Gerum

6.20 kernel/cobalt/nucleus/intr.c File Reference

Interrupt management.

Include dependency graph for intr.c:



Functions

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

  Initialize an interrupt object.

- int xnintr_destroy (xnintr_t *intr)

  Destroy an interrupt object.

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

  Attach an interrupt object.

- int xnintr_detach (xnintr_t *intr)

  Detach an interrupt object.

- void xnintr_enable (xnintr_t *intr)

  Enable an interrupt object.

- void xnintr_disable (xnintr_t *intr)

  Disable an interrupt object.

- void xnintr_affinity (xnintr_t *intr, xnarch_cpumask_t cpumask)

  Set interrupt's processor affinity.

6.20.1 Detailed Description

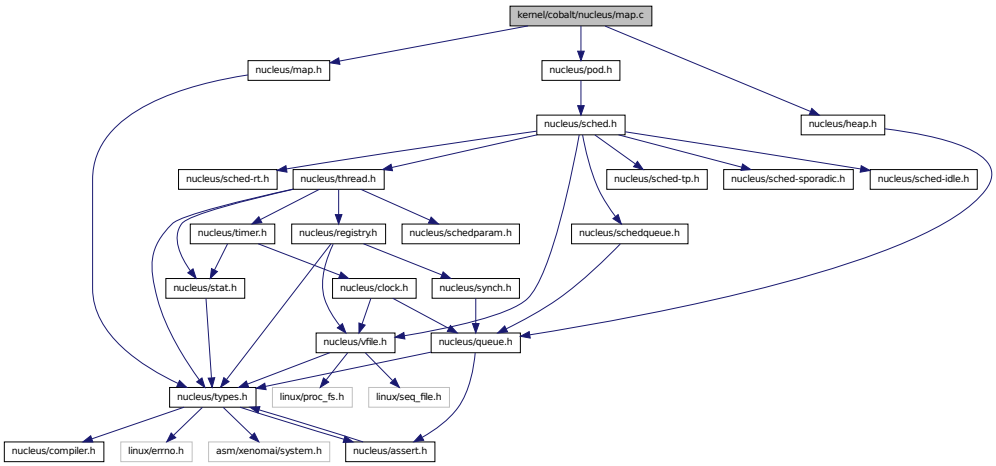
Interrupt management.

Author

Philippe Gerum

6.21 kernel/cobalt/nucleus/map.c File Reference

Include dependency graph for map.c:



Functions

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

  Create a map.

- void xnmap_delete (xnmap_t *map)

  Delete a map.

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

  Index an object into a map.

- int xnmap_remove (xnmap_t *map, int key)

  Remove an object reference from a map.

6.21.1 Detailed Description

Note

Copyright (C) 2007 Philippe Gerum <rpm@xenomai.org>.

Xenomai is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

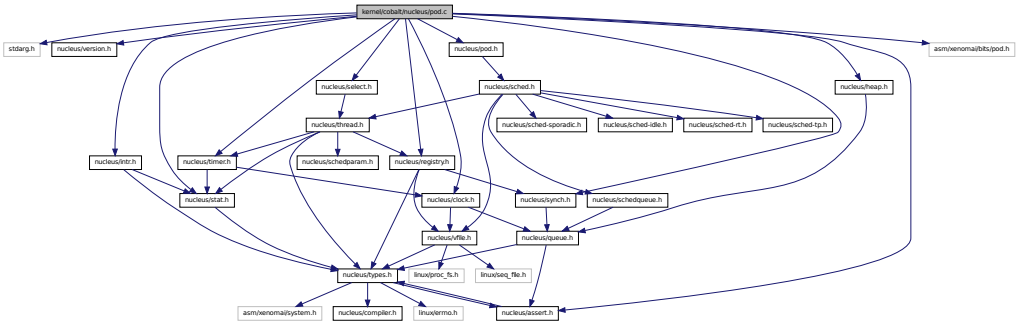
Xenomai is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with Xenomai; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

6.22 kernel/cobalt/nucleus/pod.c File Reference

Real-time pod services.

Include dependency graph for pod.c:



Functions

- int xnpod_init (void)

  Initialize the core pod.

- void xnpod_shutdown (int xtype)

  Shutdown the current pod.

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

  Initialize a new thread.

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

  Initial start of a newly created thread.

- void __xnpod_reset_thread (struct xnthread *thread)

  Reset the thread.

- void xnpod_stop_thread (xnthread_t *thread)

  Stop a thread.

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

  Change a thread's control mode.

- void xnpod_delete_thread (xnthread_t *thread)

Delete a thread.

- void xnpod_abort_thread (xnthread_t *thread)

  Abort a thread.

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

  Suspend a thread.

- void xnpod_resume_thread (xnthread_t *thread, xnflags_t mask)

  Resume a thread.

- int xnpod_unblock_thread (xnthread_t *thread)

  Unblock a thread.

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

  Change the base scheduling parameters of a thread.

- int xnpod_migrate_thread (int cpu)

  Migrate the current thread.

- void xnpod_dispatch_signals (void)

  Deliver pending asynchronous signals to the running thread.

- void xnpod_welcome_thread (xnthread_t *thread, int imask)

  Thread prologue.

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

  Install a nucleus hook.

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

  Remove a nucleus hook.

- int xnpod_handle_exception (struct ipipe_trap_data *d)

  Exception handler.

- int xnpod_enable_timesource (void)

  Activate the core time source.

- void xnpod_disable_timesource (void)

  Stop the core time source.

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

  Make a thread periodic.

- int xnpod_wait_thread_period (unsigned long *overruns_r)

  Wait for the next periodic release point.

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

  Set thread time-slicing information.

6.22.1 Detailed Description

Real-time pod services.

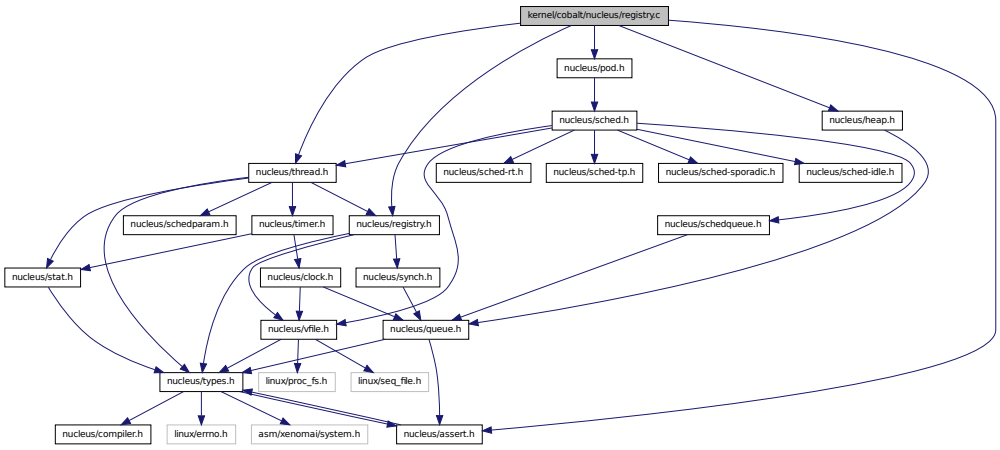
Author

Philippe Gerum

6.23 kernel/cobalt/nucleus/registry.c File Reference

This file is part of the Xenomai project.

Include dependency graph for registry.c:



Functions

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

  Register a real-time object.

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

  Bind to a real-time object.

- int xnregistry_remove (xnhandle_t handle)

  Forcibly unregister a real-time object.

- int xnregistry_remove_safe (xnhandle_t handle, xnticks_t timeout)

  Unregister an idle real-time object.

- void * xnregistry_get (xnhandle_t handle)

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

- u_long xnregistry_put (xnhandle_t handle)

  Unlock a real-time object from the registry.

- void * xnregistry_fetch (xnhandle_t handle)

  Find a real-time object into the registry.

6.23.1 Detailed Description

This file is part of the Xenomai project.

Note

Copyright (C) 2004 Philippe Gerum <rpm@xenomai.org>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

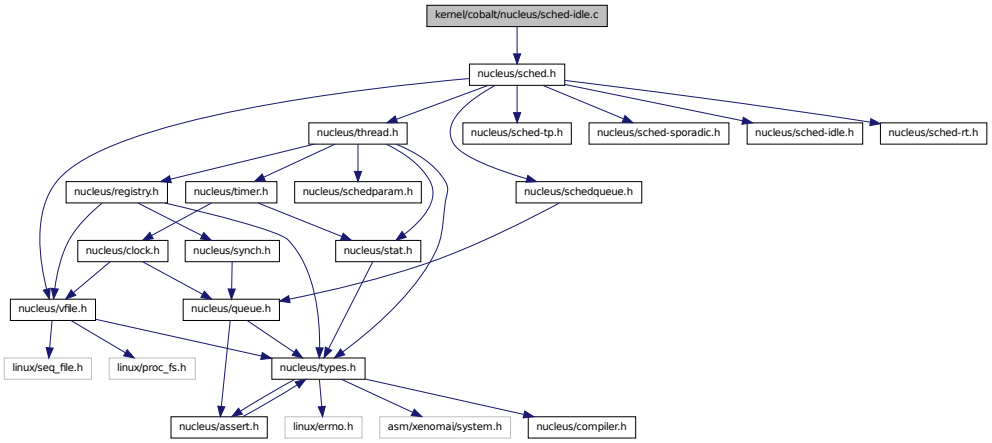
This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

6.24 kernel/cobalt/nucleus/sched-idle.c File Reference

Idle scheduling class implementation (i.e. Linux placeholder).

Include dependency graph for sched-idle.c:



6.24.1 Detailed Description

Idle scheduling class implementation (i.e. Linux placeholder).

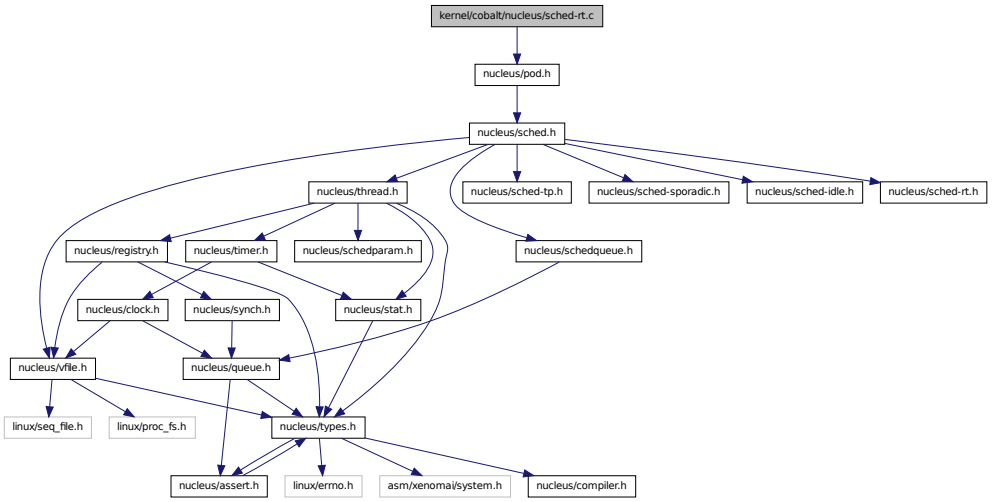
Author

Philippe Gerum Copyright (C) 2008 Philippe Gerum <rpm@xenomai.org>.

6.25 kernel/cobalt/nucleus/sched-rt.c File Reference

Common real-time scheduling class implementation (FIFO + RR).

Include dependency graph for sched-rt.c:



6.25.1 Detailed Description

Common real-time scheduling class implementation (FIFO + RR).

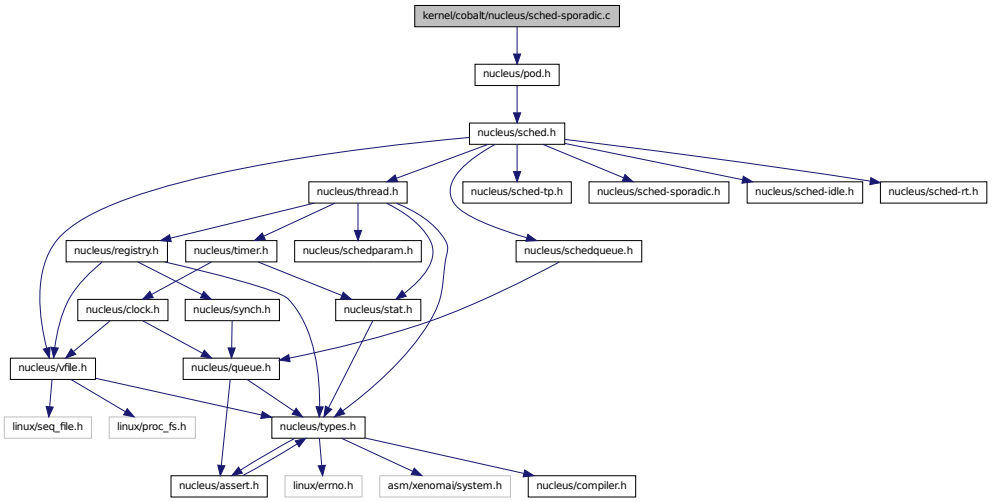
Author

Philippe Gerum Copyright (C) 2008 Philippe Gerum <rpm@xenomai.org>.

6.26 kernel/cobalt/nucleus/sched-sporadic.c File Reference

POSIX SCHED_SPORADIC scheduling class.

Include dependency graph for sched-sporadic.c:



6.26.1 Detailed Description

POSIX SCHED_SPORADIC scheduling class.

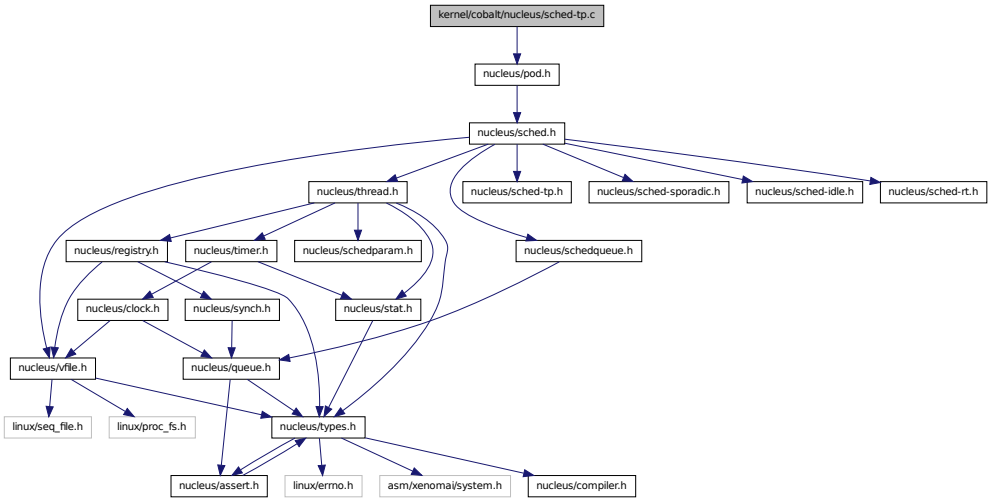
Author

Philippe Gerum Copyright (C) 2009 Philippe Gerum <rpm@xenomai.org>.

6.27 kernel/cobalt/nucleus/sched-tp.c File Reference

Temporal partitioning (typical of IMA systems).

Include dependency graph for sched-tp.c:



6.27.1 Detailed Description

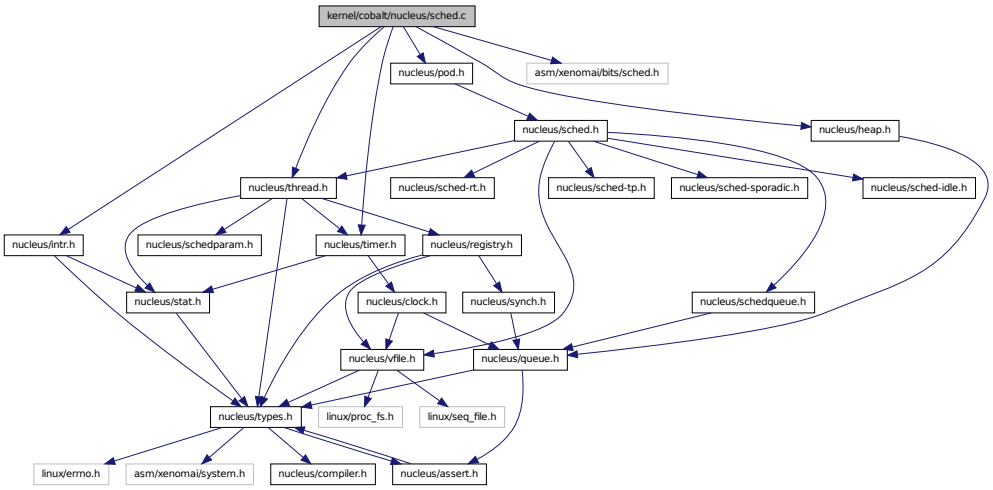
Temporal partitioning (typical of IMA systems).

Author

Philippe Gerum Copyright (C) 2008 Philippe Gerum <rpm@xenomai.org>.

6.28 kernel/cobalt/nucleus/sched.c File Reference

Include dependency graph for sched.c:



6.28.1 Detailed Description

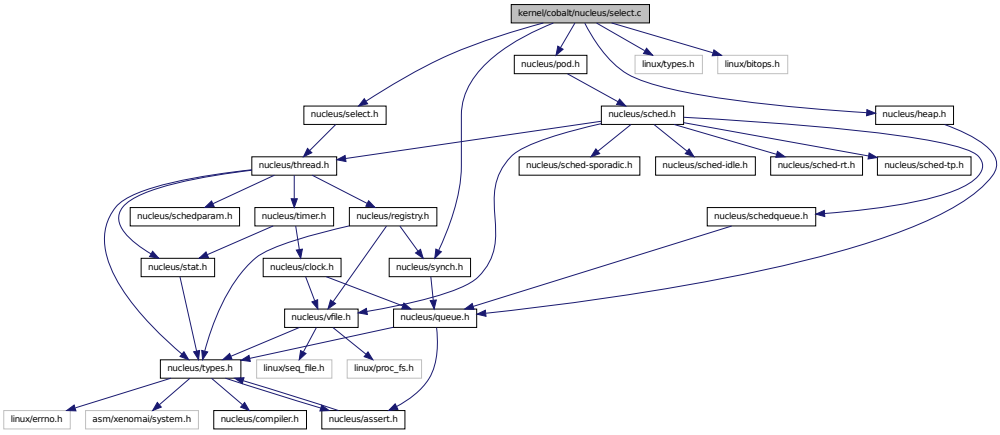
Author

Philippe Gerum

6.29 kernel/cobalt/nucleus/select.c File Reference

file descriptors events multiplexing.

Include dependency graph for select.c:



Functions

- void xnselect_init (struct xnselect *select_block)

  Initialize a struct xnselect structure.

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

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

- void xnselect_destroy (struct xnselect *select_block)

  Destroy the xnselect structure associated with a file descriptor.

- int xnselector_init (struct xnselector *selector)

  Initialize a selector structure.

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

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

- void xnselector_destroy (struct xnselector *selector)

  Destroy a selector block.

6.29.1 Detailed Description

file descriptors events multiplexing.

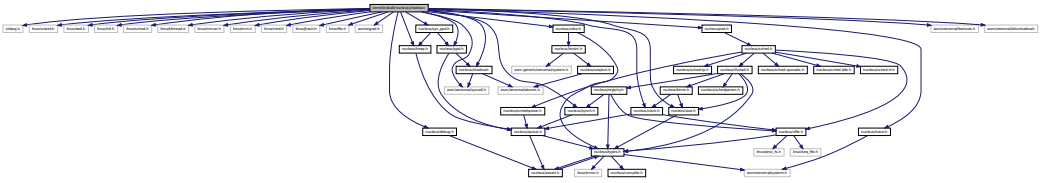
Author

Gilles Chanteperdrix

6.30 kernel/cobalt/nucleus/shadow.c File Reference

Real-time shadow services.

Include dependency graph for shadow.c:



Functions

- int xnshadow_harden (void)

  Migrate a Linux task to the Xenomai domain.

- void xnshadow_relax (int notify, int reason)

  Switch a shadow thread back to the Linux domain.

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

  Create a shadow thread context.

- xnshadow_ppd_t * xnshadow_ppd_get (unsigned int muxid)

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

6.30.1 Detailed Description

Real-time shadow services.

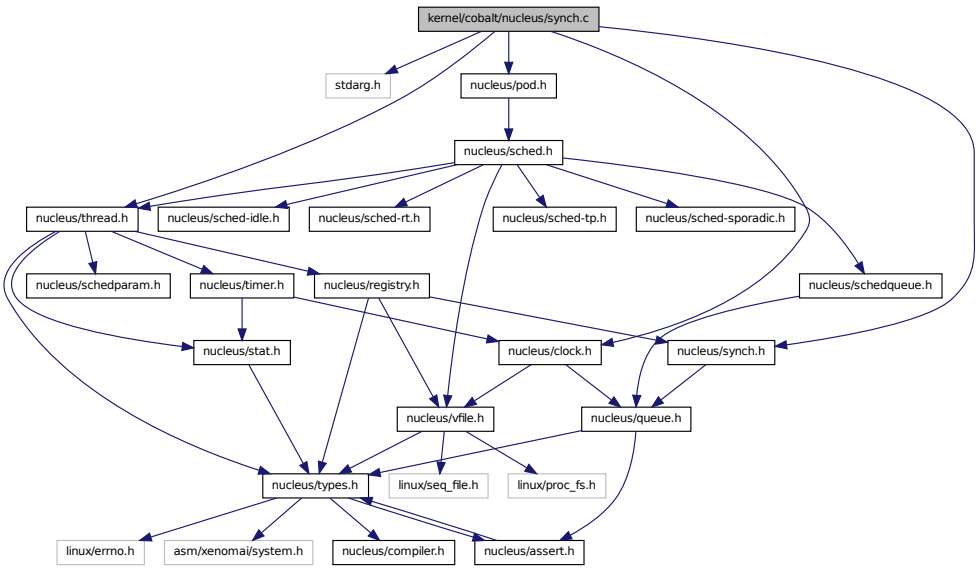
Author

Philippe Gerum

6.31 kernel/cobalt/nucleus/synch.c File Reference

Thread synchronization services.

Include dependency graph for synch.c:



Functions

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

  Initialize a synchronization object.

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

Sleep on an ownerless synchronization object.

- struct xnthread * xnsynch_wakeup_one_sleeper (struct xnsynch *synch)

  Give the resource ownership to the next waiting thread.

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

  Give the resource ownership to a given waiting thread.

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

  Acquire the ownership of a synchronization object.

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

  Clear the priority boost.

- void xnsynch_requeue_sleeper (struct xnthread *thread)

  Change a sleeper's priority.

- struct xnthread * xnsynch_peek_pendq (struct xnsynch *synch)

  Access the thread leading a synch object wait queue.

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

  Unblock all waiters pending on a resource.

- void xnsynch_forget_sleeper (struct xnthread *thread)

  Abort a wait for a resource.

- void xnsynch_release_all_ownerships (struct xnthread *thread)

  Release all ownerships.

6.31.1 Detailed Description

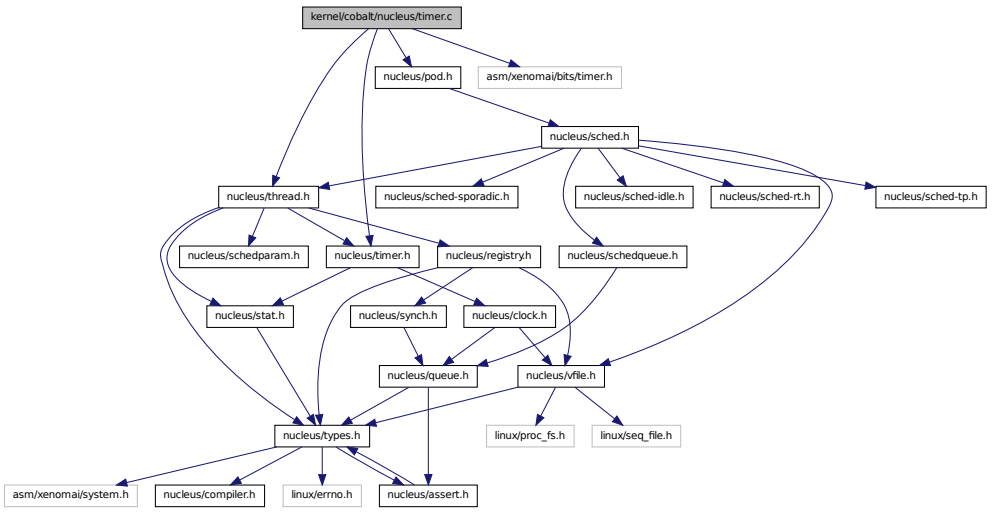
Thread synchronization services.

Author

Philippe Gerum

6.32 kernel/cobalt/nucleus/timer.c File Reference

Include dependency graph for timer.c:



Functions

- int xntimer_start (xntimer_t *timer, xnticks_t value, xnticks_t interval, xntmode_t mode)
  Arm a timer.
- xnticks_t xntimer_get_date (xntimer_t *timer)
  Return the absolute expiration date.
- xnticks_t xntimer_get_timeout (xntimer_t *timer)
  Return the relative expiration date.
- xnticks_t xntimer_get_interval (xntimer_t *timer)
  Return the timer interval value.
- void xntimer_tick (void)
  Process a timer tick.
- void xntimer_init (xntimer_t *timer, void(*handler)(xntimer_t *timer))
  Initialize a timer object.
- void xntimer_destroy (xntimer_t *timer)
  Release a timer object.
- unsigned long xntimer_get_overruns (xntimer_t *timer, xnticks_t now)
  Get the count of overruns for the last tick.

- void xntimer_freeze (void)

  Freeze all timers (from every time bases).

6.32.1 Detailed Description

Note

Copyright (C) 2001,2002,2003,2007 Philippe Gerum <rpm@xenomai.org>. Copyright (C) 2004 Gilles Chanteperdrix <gilles.chanteperdrix@xenomai.org>

Xenomai is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

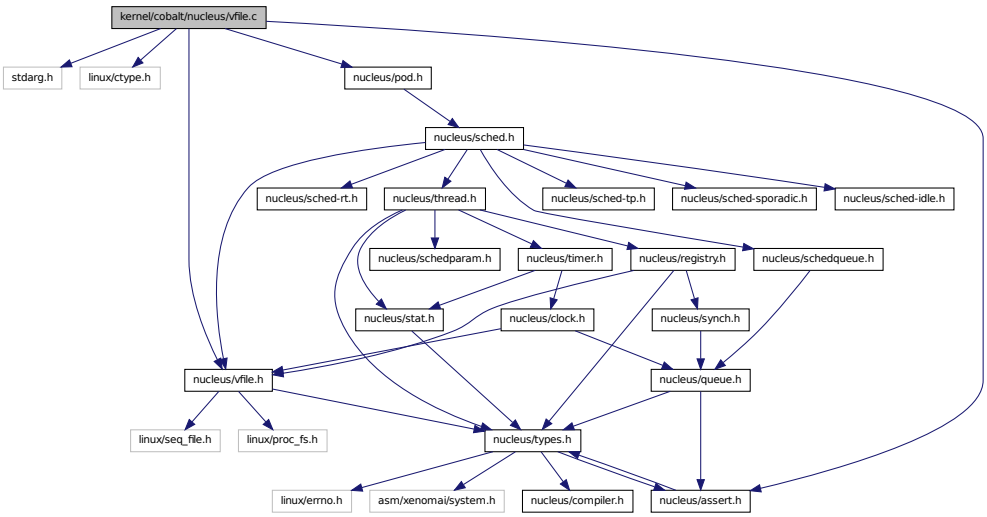
Xenomai is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with Xenomai; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

6.33 kernel/cobalt/nucleus/vfile.c File Reference

This file is part of the Xenomai project.

Include dependency graph for vfile.c:



Functions

- int xnvfile_init_snapshot (const char *name, struct xnvfile_snapshot *vfile, struct xnvfile_-directory *parent)

  Initialize a snapshot-driven vfile.

- int xnvfile_init_regular (const char *name, struct xnvfile_regular *vfile, struct xnvfile_-directory *parent)

  Initialize a regular vfile.

- int xnvfile_init_dir (const char *name, struct xnvfile_directory *vdir, struct xnvfile_directory *parent)

  Initialize a virtual directory entry.

- int xnvfile_init_link (const char *from, const char *to, struct xnvfile_link *vlink, struct xnvfile_directory *parent)

  Initialize a virtual link entry.

- void xnvfile_destroy (struct xnvfile *vfile)

  Removes a virtual file entry.

- ssize_t xnvfile_get_blob (struct xnvfile_input *input, void *data, size_t size)

  Read in a data bulk written to the vfile.

- ssize_t xnvfile_get_string (struct xnvfile_input *input, char *s, size_t maxlen)

  Read in a C-string written to the vfile.

- ssize_t xnvfile_get_integer (struct xnvfile_input *input, long *valp)

  Evaluate the string written to the vfile as a long integer.

Variables

- struct xnvfile_directory nkvfroot

  Xenomai vfile root directory.

6.33.1 Detailed Description

This file is part of the Xenomai project.

Note

Copyright (C) 2010 Philippe Gerum <rpm@xenomai.org>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

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

__xnpod_reset_thread
    pod, 39

affinity
    xnthread_info, 97

begin
    xnvfile_regular_ops, 101
    xnvfile_snapshot_ops, 107
bprio
    xnthread_info, 97
bufd
    xnbufd_copy_from_kmem, 13
    xnbufd_copy_to_kmem, 14
    xnbufd_invalidate, 15
    xnbufd_map_kread, 15
    xnbufd_map_kwrite, 16
    xnbufd_map_uread, 16
    xnbufd_map_uwrite, 17
    xnbufd_reset, 17
    xnbufd_unmap_kread, 17
    xnbufd_unmap_kwrite, 18
    xnbufd_unmap_uread, 18
    xnbufd_unmap_uwrite, 19
Buffer descriptors., 10

clock
    xnclock_adjust, 20
cprio
    xnthread_info, 97
cpu
    xnthread_info, 97
ctxswitches
    xnthread_info, 97
curr
    xnsched, 95

databuf
    xnvfile_snapshot_iterator, 106
Debugging services., 21
Dynamic memory allocation services., 21

end
    xnvfile_regular_ops, 101
    xnvfile_snapshot_ops, 107
endfn
    xnvfile_snapshot_iterator, 106
exectime
    xnthread_info, 97

File descriptors events multiplexing services., 62

get
    xnvfile_lock_ops, 99

heap
    xnheap_alloc, 22
    xnheap_destroy, 23
    xnheap_extend, 23
    xnheap_free, 24
    xnheap_init, 24
    xnheap_schedule_free, 25
    xnheap_set_label, 26
    xnheap_test_and_free, 26
htimer
    xnsched, 95

include/cobalt/nucleus/bufd.h, 111
include/cobalt/nucleus/clock.h, 113
include/cobalt/nucleus/hostrt.h, 114
include/cobalt/nucleus/map.h, 116
include/cobalt/nucleus/pod.h, 117
include/cobalt/nucleus/registry.h, 120
include/cobalt/nucleus/sched-idle.h, 121
include/cobalt/nucleus/sched-rt.h, 122
include/cobalt/nucleus/sched-sporadic.h, 123
include/cobalt/nucleus/sched-tp.h, 123
include/cobalt/nucleus/sched.h, 124
include/cobalt/nucleus/select.h, 125
include/cobalt/nucleus/timer.h, 128
include/cobalt/nucleus/vdso.h, 129
include/cobalt/nucleus/vfile.h, 130
inesting
    xnsched, 95
Interrupt management., 27
intr
    xnintr_affinity, 28
    xnintr_attach, 28
    xnintr_destroy, 29
    xnintr_detach, 29

xnintr_disable, 30
xnintr_enable, 30
xnintr_init, 31

kernel/cobalt/nucleus/bufd.c, 133
kernel/cobalt/nucleus/clock.c, 134
kernel/cobalt/nucleus/debug.c, 135
kernel/cobalt/nucleus/heap.c, 136
kernel/cobalt/nucleus/intr.c, 137
kernel/cobalt/nucleus/map.c, 139
kernel/cobalt/nucleus/pod.c, 140
kernel/cobalt/nucleus/registry.c, 142
kernel/cobalt/nucleus/sched-idle.c, 143
kernel/cobalt/nucleus/sched-rt.c, 144
kernel/cobalt/nucleus/sched-sporadic.c, 145
kernel/cobalt/nucleus/sched-tp.c, 146
kernel/cobalt/nucleus/sched.c, 148
kernel/cobalt/nucleus/select.c, 148
kernel/cobalt/nucleus/shadow.c, 150
kernel/cobalt/nucleus/synch.c, 151
kernel/cobalt/nucleus/timer.c, 153
kernel/cobalt/nucleus/vfile.c, 154

lflags
xnsched, 95
Lightweight key-to-object mapping service, 33

map
xnmap_create, 33
xnmap_delete, 34
xnmap_enter, 34
xnmap_fetch, 35
xnmap_fetch_nocheck, 36
xnmap_remove, 36
modeswitches
xnthread_info, 98

name
xnthread_info, 98
next
xnvfile_regular_ops, 101
xnvfile_snapshot_ops, 108
nkvfroot
vfile, 90
nrdata
xnvfile_snapshot_iterator, 106
nucleus_state_flags
XNHELD, 8
XNLOCK, 8
XNMIGRATE, 9
XNPEND, 9
XNREADY, 9
XNSUSP, 9

pagefaults
xnthread_info, 98
pod
__xnpod_reset_thread, 39
xnpod_abort_thread, 39
xnpod_add_hook, 40
xnpod_delete_thread, 41
xnpod_disable_timesource, 41
xnpod_dispatch_signals, 42
xnpod_enable_timesource, 42
xnpod_handle_exception, 42
xnpod_init, 43
xnpod_init_thread, 43
xnpod_migrate_thread, 45
xnpod_remove_hook, 45
xnpod_resume_thread, 45
xnpod_schedule, 46
xnpod_set_thread_mode, 48
xnpod_set_thread_periodic, 48
xnpod_set_thread_schedparam, 49
xnpod_set_thread_tslice, 50
xnpod_shutdown, 51
xnpod_start_thread, 51
xnpod_stop_thread, 53
xnpod_suspend_thread, 53
xnpod_unblock_thread, 54
xnpod_wait_thread_period, 55
xnpod_welcome_thread, 56
pos
xnvfile_regular_iterator, 100
private
xnvfile_regular_iterator, 100
xnvfile_snapshot_iterator, 106
put
xnvfile_lock_ops, 99

Real-time pod services., 37
Real-time shadow services., 66
refcnt
xnpod, 94
registry
xnregistry_bind, 57
xnregistry_enter, 58
xnregistry_fetch, 59
xnregistry_get, 59
xnregistry_put, 60
xnregistry_remove, 61
xnregistry_remove_safe, 61
Registry services., 56
relpoint
xnthread_info, 98
rev
xnvfile_rev_tag, 104
rewind
xnvfile_regular_ops, 102

xnvfile_snapshot_ops, 108
rootcb
xnsched, 96
rt
xnsched, 96

Sched, 90
sched
xnpod, 94
xnsched_rotate, 91
select
xnselect, 63
xnselect_bind, 64
xnselect_destroy, 64
xnselect_init, 65
xnselect_signal, 65
xnselector_destroy, 65
xnselector_init, 65
seq
xnvfile_regular_iterator, 100
xnvfile_snapshot_iterator, 106
shadow
xnshadow_harden, 66
xnshadow_map, 67
xnshadow_ppd_get, 68
xnshadow_relax, 68
show
xnvfile_regular_ops, 102
xnvfile_snapshot_ops, 109
state
xnthread_info, 98
status
xnpod, 94
xnsched, 96
store
xnvfile_regular_ops, 103
xnvfile_snapshot_ops, 109
synch
xnsynch_acquire, 70
xnsynch_clear_boost, 71
xnsynch_flush, 71
xnsynch_forget_sleeper, 72
xnsynch_init, 72
xnsynch_peek_pendq, 73
xnsynch_release, 74
xnsynch_release_all_ownerships, 75
xnsynch_requeue_sleeper, 75
xnsynch_sleep_on, 75
xnsynch_wakeup_one_sleeper, 76
xnsynch_wakeup_this_sleeper, 77
syscalls
xnthread_info, 98
System clock services., 20

tdeleteq
xnpod, 94
Thread information flags., 9
Thread state flags., 7
Thread synchronization services., 69
threadq
xnpod, 94
timer
xntimer_destroy, 79
xntimer_freeze, 79
xntimer_get_date, 80
xntimer_get_interval, 80
xntimer_get_overruns, 80
xntimer_get_timeout, 81
xntimer_init, 81
xntimer_start, 82
xntimer_stop, 82
xntimer_tick, 83
Timer services., 78
timerlck
xnpod, 94
tstartq
xnpod, 94
tswitchq
xnpod, 94

vfile
nkvfroot, 90
xnvfile_destroy, 86
xnvfile_get_blob, 86
xnvfile_get_integer, 86
xnvfile_get_string, 87
xnvfile_init_dir, 87
xnvfile_init_link, 88
xnvfile_init_regular, 88
xnvfile_init_snapshot, 89
xnvfile_regular_iterator, 100
xnvfile_snapshot_iterator, 106
Virtual file services, 83

xnbufd_copy_from_kmem
bufd, 13
xnbufd_copy_to_kmem
bufd, 14
xnbufd_invalidate
bufd, 15
xnbufd_map_kread
bufd, 15
xnbufd_map_kwrite
bufd, 16
xnbufd_map_uread
bufd, 16
xnbufd_map_uwrite
bufd, 17

xnbufd_reset
bufd, 17
xnbufd_unmap_kread
bufd, 17
xnbufd_unmap_kwrite
bufd, 18
xnbufd_unmap_uread
bufd, 18
xnbufd_unmap_uwrite
bufd, 19
xnclock_adjust
clock, 20
xnheap_alloc
heap, 22
xnheap_destroy
heap, 23
xnheap_extend
heap, 23
xnheap_free
heap, 24
xnheap_init
heap, 24
xnheap_schedule_free
heap, 25
xnheap_set_label
heap, 26
xnheap_test_and_free
heap, 26
XNHELD
nucleus_state_flags, 8
xnintr_affinity
intr, 28
xnintr_attach
intr, 28
xnintr_destroy
intr, 29
xnintr_detach
intr, 29
xnintr_disable
intr, 30
xnintr_enable
intr, 30
xnintr_init
intr, 31
XNLOCK
nucleus_state_flags, 8
xnmap_create
map, 33
xnmap_delete
map, 34
xnmap_enter
map, 34
xnmap_fetch
map, 35
xnmap_fetch_nocheck
map, 36
xnmap_remove
map, 36
XNMIGRATE
nucleus_state_flags, 9
XNPEND
nucleus_state_flags, 9
xnpod, 93
refcnt, 94
sched, 94
status, 94
tdeleteq, 94
threadq, 94
timerlck, 94
tstartq, 94
tswitchq, 94
xnpod_abort_thread
pod, 39
xnpod_add_hook
pod, 40
xnpod_delete_thread
pod, 41
xnpod_disable_timesource
pod, 41
xnpod_dispatch_signals
pod, 42
xnpod_enable_timesource
pod, 42
xnpod_handle_exception
pod, 42
xnpod_init
pod, 43
xnpod_init_thread
pod, 43
xnpod_migrate_thread
pod, 45
xnpod_remove_hook
pod, 45
xnpod_resume_thread
pod, 45
xnpod_schedule
pod, 46
xnpod_set_thread_mode
pod, 48
xnpod_set_thread_periodic
pod, 48
xnpod_set_thread_schedparam
pod, 49
xnpod_set_thread_tslice
pod, 50
xnpod_shutdown
pod, 51
xnpod_start_thread

pod, 51
xnpod_stop_thread
pod, 53
xnpod_suspend_thread
pod, 53
xnpod_unblock_thread
pod, 54
xnpod_wait_thread_period
pod, 55
xnpod_welcome_thread
pod, 56
XNREADY
nucleus_state_flags, 9
xnregistry_bind
registry, 57
xnregistry_enter
registry, 58
xnregistry_fetch
registry, 59
xnregistry_get
registry, 59
xnregistry_put
registry, 60
xnregistry_remove
registry, 61
xnregistry_remove_safe
registry, 61
xnsched, 95
curr, 95
htimer, 95
inesting, 95
lflags, 95
rootcb, 96
rt, 96
status, 96
xnsched_rotate
sched, 91
xnselect
select, 63
xnselect_bind
select, 64
xnselect_destroy
select, 64
xnselect_init
select, 65
xnselect_signal
select, 65
xnselector_destroy
select, 65
xnselector_init
select, 65
xnshadow_harden
shadow, 66
xnshadow_map
shadow, 67
xnshadow_ppd_get
shadow, 68
xnshadow_relax
shadow, 68
XNSUSP
nucleus_state_flags, 9
xnsynch_acquire
synch, 70
xnsynch_clear_boost
synch, 71
xnsynch_flush
synch, 71
xnsynch_forget_sleeper
synch, 72
xnsynch_init
synch, 72
xnsynch_peek_pendq
synch, 73
xnsynch_release
synch, 74
xnsynch_release_all_ownerships
synch, 75
xnsynch_requeue_sleeper
synch, 75
xnsynch_sleep_on
synch, 75
xnsynch_wakeup_one_sleeper
synch, 76
xnsynch_wakeup_this_sleeper
synch, 77
xnthread_info, 96
affinity, 97
bprio, 97
cprio, 97
cpu, 97
ctxswitches, 97
exectime, 97
modeswitches, 98
name, 98
pagefaults, 98
relpoint, 98
state, 98
syscalls, 98
xntimer_destroy
timer, 79
xntimer_freeze
timer, 79
xntimer_get_date
timer, 80
xntimer_get_interval
timer, 80
xntimer_get_overruns
timer, 80

xntimer_get_timeout
timer, 81
xntimer_init
timer, 81
xntimer_start
timer, 82
xntimer_stop
timer, 82
xntimer_tick
timer, 83
xnvfile_destroy
vfile, 86
xnvfile_get_blob
vfile, 86
xnvfile_get_integer
vfile, 86
xnvfile_get_string
vfile, 87
xnvfile_init_dir
vfile, 87
xnvfile_init_link
vfile, 88
xnvfile_init_regular
vfile, 88
xnvfile_init_snapshot
vfile, 89
xnvfile_lock_ops, 98
get, 99
put, 99
xnvfile_regular_iterator, 99
pos, 100
private, 100
seq, 100
vfile, 100
xnvfile_regular_ops, 100
begin, 101
end, 101
next, 101
rewind, 102
show, 102
store, 103
xnvfile_rev_tag, 103
rev, 104
xnvfile_snapshot, 104
xnvfile_snapshot_iterator, 105
databuf, 106
endfn, 106
nrdata, 106
private, 106
seq, 106
vfile, 106
xnvfile_snapshot_ops, 106
begin, 107
end, 107
next, 108
rewind, 108
show, 109
store, 109