6.3.2.2    rtdm_lock_irqrestore                . . . . . . . . . . . . . . . . . . . . . 28
6.3.2.3    rtdm_lock_irqsave                   . . . . . . . . . . . . . . . . . . . . 28
6.3.3    Function Documentation                . . . . . . . . . . . . . . . . . . . . . . . 29
6.3.3.1    rtdm_lock_get()                    . . . . . . . . . . . . . . . . . . . . . . . . 29
6.3.3.2    rtdm_lock_init()                   . . . . . . . . . . . . . . . . . . . . . . . . 29
6.3.3.3    rtdm_lock_put()                    . . . . . . . . . . . . . . . . . . . . . . . . 29
6.3.3.4    rtdm_lock_put_irqrestore()         . . . . . . . . . . . . . . . . . . . . . . . . 30
6.4    User-space driver core                   . . . . . . . . . . . . . . . . . . . . . . . . . . 31
6.4.1    Detailed Description                   . . . . . . . . . . . . . . . . . . . . . . . . . 32
6.4.2    Macro Definition Documentation         . . . . . . . . . . . . . . . . . . . . . . . . . 32
6.4.2.1    UDD_IRQ_CUSTOM                      . . . . . . . . . . . . . . . . . . . . . . . . . 33
6.4.2.2    UDD_IRQ_NONE                        . . . . . . . . . . . . . . . . . . . . . . . . . 33
6.4.2.3    UDD_MEM_LOGICAL                     . . . . . . . . . . . . . . . . . . . . . . . . . 33
6.4.2.4    UDD_MEM_NONE                        . . . . . . . . . . . . . . . . . . . . . . . . . 33
6.4.2.5    UDD_MEM_PHYS                        . . . . . . . . . . . . . . . . . . . . . . . . . 33
6.4.2.6    UDD_MEM_VIRTUAL                     . . . . . . . . . . . . . . . . . . . . . . . . . 34
6.4.2.7    UDD_RTIOC_IRQDIS                    . . . . . . . . . . . . . . . . . . . . . . . . . 34
6.4.2.8    UDD_RTIOC_IRQEN                     . . . . . . . . . . . . . . . . . . . . . . . . . 34
6.4.2.9    UDD_RTIOC_IRQSIG                    . . . . . . . . . . . . . . . . . . . . . . . . . 34
6.4.3    Function Documentation                . . . . . . . . . . . . . . . . . . . . . . . . . . 35
6.4.3.1    udd_disable_irq()                  . . . . . . . . . . . . . . . . . . . . . . . . . 35
6.4.3.2    udd_enable_irq()                   . . . . . . . . . . . . . . . . . . . . . . . . . 35
6.4.3.3    udd_get_device()                   . . . . . . . . . . . . . . . . . . . . . . . . . 36
6.4.3.4    udd_notify_event()                 . . . . . . . . . . . . . . . . . . . . . . . . . 37
6.4.3.5    udd_register_device()              . . . . . . . . . . . . . . . . . . . . . . . . . 37
6.4.3.6    udd_unregister_device()            . . . . . . . . . . . . . . . . . . . . . . . . . 38
6.5    Thread state flags                      . . . . . . . . . . . . . . . . . . . . . . . . . . 39
6.5.1    Detailed Description                   . . . . . . . . . . . . . . . . . . . . . . . . . 40
6.5.2    Macro Definition Documentation         . . . . . . . . . . . . . . . . . . . . . . . . . 40
6.5.2.1    XNHELD                             . . . . . . . . . . . . . . . . . . . . . . . . . 40
6.5.2.2 XNMIGRATE ......................................................... 40
6.5.2.3 XNPEND .......................................................... 40
6.5.2.4 XNREADY .......................................................... 41
6.5.2.5 XNSUSP ............................................................ 41
6.5.2.6 XNTRAPLB .......................................................... 41

6.6 Thread information flags ............................................. 42
6.6.1 Detailed Description ............................................... 42

6.7 CAN Devices ......................................................... 43
6.7.1 Detailed Description ............................................... 50
6.7.2 Macro Definition Documentation ................................. 53
6.7.2.1 CAN_CTRLMODE_3_SAMPLES ................................ 54
6.7.2.2 CAN_CTRLMODE_LISTENONLY ................................ 54
6.7.2.3 CAN_CTRLMODE_LOOPBACK .................................. 54
6.7.2.4 CAN_ERR_LOSTARB_UNSPEC ................................. 54
6.7.2.5 CAN_RAW_ERR_FILTER ....................................... 54
6.7.2.6 CAN_RAW_FILTER .............................................. 55
6.7.2.7 CAN_RAW_LOOPBACK ......................................... 56
6.7.2.8 CAN_RAW_RECV_own_MSGS .................................. 56
6.7.2.9 RTCAN_RTIOC_RCV_TIMEOUT ................................. 57
6.7.2.10 RTCAN_RTIOC_SND_TIMEOUT ................................. 57
6.7.2.11 RTCAN_RTIOC_TAKE_TIMESTAMP ............................ 58
6.7.2.12 SIOCGCANBAUDRATE ....................................... 59
6.7.2.13 SIOCGCANCTRLMODE ....................................... 59
6.7.2.14 SIOCGCANCUSTOMBITTIME ................................ 60
6.7.2.15 SIOCGCANSTATE ............................................. 60
6.7.2.16 SIOCGIFINDEX .............................................. 61
6.7.2.17 SIOCSCANBAUDRATE ....................................... 61
6.7.2.18 SIOCSCANCTRLMODE ....................................... 62
6.7.2.19 SIOCSCANCUSTOMBITTIME ................................ 63
6.7.2.20 SIOCSCANMODE .............................................. 63
6.12.2.8 XDDP_EVTDOWN ........................................... 81
6.12.2.9 XDDP_EVTIN ............................................. 81
6.12.2.10 XDDP_EVTNOBUF ....................................... 82
6.12.2.11 XDDP_EVTOUT .......................................... 82
6.12.2.12 XDDP_LABEL ............................................. 82
6.12.2.13 XDDP_MONITOR .......................................... 83
6.12.2.14 XDDP_POOLSZ ........................................... 84

6.12.3 Enumeration Type Documentation ................................................. 84

6.12.3.1 anonymous enum ........................................ 84

6.12.4 Function Documentation ......................................................... 85

6.12.4.1 bind__AF_RTIPC() ........................................ 85
6.12.4.2 close__AF_RTIPC() ........................................ 87
6.12.4.3 connect__AF_RTIPC() ...................................... 87
6.12.4.4 getpeername__AF_RTIPC() .................................. 88
6.12.4.5 getsockname__AF_RTIPC() .................................. 89
6.12.4.6 getsockopt__AF_RTIPC() .................................... 89
6.12.4.7 recvmsg__AF_RTIPC() ...................................... 89
6.12.4.8 sendmsg__AF_RTIPC() ...................................... 90
6.12.4.9 setsockopt__AF_RTIPC() .................................... 91
6.12.4.10 socket__AF_RTIPC() ...................................... 92

6.13 Asynchronous Procedure Calls ......................................................... 93

6.13.1 Detailed Description ......................................................... 93
6.13.2 Function Documentation ......................................................... 93

6.13.2.1 xnapc_alloc() ............................................ 93
6.13.2.2 xnapc_free() .............................................. 94
6.13.2.3 xnapc_schedule() .......................................... 94

6.14 In-kernel arithmetics ............................................................... 96

6.14.1 Detailed Description ......................................................... 96
6.14.2 Function Documentation ......................................................... 96

6.14.2.1 xnarch_generic_full_divmod64() ......................... 96
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.18.2.4</td>
<td><code>xnheap_init()</code></td>
<td>113</td>
</tr>
<tr>
<td>6.18.2.5</td>
<td><code>xnheap_set_name()</code></td>
<td>114</td>
</tr>
<tr>
<td>6.19</td>
<td>Cobalt</td>
<td>115</td>
</tr>
<tr>
<td>6.19.1</td>
<td>Detailed Description</td>
<td>115</td>
</tr>
<tr>
<td>6.20</td>
<td>Cobalt kernel</td>
<td>116</td>
</tr>
<tr>
<td>6.20.1</td>
<td>Detailed Description</td>
<td>117</td>
</tr>
<tr>
<td>6.20.1.1</td>
<td>Dual kernel service tags</td>
<td>117</td>
</tr>
<tr>
<td>6.21</td>
<td>Interrupt management</td>
<td>119</td>
</tr>
<tr>
<td>6.21.1</td>
<td>Detailed Description</td>
<td>119</td>
</tr>
<tr>
<td>6.21.2</td>
<td>Function Documentation</td>
<td>119</td>
</tr>
<tr>
<td>6.21.2.1</td>
<td><code>xnintr_affinity()</code></td>
<td>119</td>
</tr>
<tr>
<td>6.21.2.2</td>
<td><code>xnintr_attach()</code></td>
<td>120</td>
</tr>
<tr>
<td>6.21.2.3</td>
<td><code>xnintr_destroy()</code></td>
<td>121</td>
</tr>
<tr>
<td>6.21.2.4</td>
<td><code>xnintr_detach()</code></td>
<td>121</td>
</tr>
<tr>
<td>6.21.2.5</td>
<td><code>xnintr_disable()</code></td>
<td>122</td>
</tr>
<tr>
<td>6.21.2.6</td>
<td><code>xnintr_enable()</code></td>
<td>122</td>
</tr>
<tr>
<td>6.21.2.7</td>
<td><code>xnintr_init()</code></td>
<td>122</td>
</tr>
<tr>
<td>6.22</td>
<td>Locking services</td>
<td>125</td>
</tr>
<tr>
<td>6.22.1</td>
<td>Detailed Description</td>
<td>125</td>
</tr>
<tr>
<td>6.22.2</td>
<td>Macro Definition Documentation</td>
<td>125</td>
</tr>
<tr>
<td>6.22.2.1</td>
<td><code>splexit</code></td>
<td>125</td>
</tr>
<tr>
<td>6.22.2.2</td>
<td><code>splhigh</code></td>
<td>126</td>
</tr>
<tr>
<td>6.22.2.3</td>
<td><code>spltest</code></td>
<td>126</td>
</tr>
<tr>
<td>6.23</td>
<td>Lightweight key-to-object mapping service</td>
<td>127</td>
</tr>
<tr>
<td>6.23.1</td>
<td>Detailed Description</td>
<td>127</td>
</tr>
<tr>
<td>6.23.2</td>
<td>Function Documentation</td>
<td>127</td>
</tr>
<tr>
<td>6.23.2.1</td>
<td><code>xnmap_create()</code></td>
<td>128</td>
</tr>
<tr>
<td>6.23.2.2</td>
<td><code>xnmap_delete()</code></td>
<td>128</td>
</tr>
<tr>
<td>6.23.2.3</td>
<td><code>xnmap_enter()</code></td>
<td>129</td>
</tr>
<tr>
<td>6.23.2.4</td>
<td><code>xnmap_fetch()</code></td>
<td>129</td>
</tr>
</tbody>
</table>
6.23.2.5 xnmap_fetch_nocheck() .................................................. 130
6.23.2.6 xnmap_remove() .......................................................... 130

6.24 Registry services ............................................................... 132

6.24.1 Detailed Description ....................................................... 132
6.24.2 Function Documentation .................................................. 132

6.24.2.1 xnregistry_bind() ....................................................... 132
6.24.2.2 xnregistry_enter() ...................................................... 133
6.24.2.3 xnregistry_lookup() .................................................... 134
6.24.2.4 xnregistry_remove() ................................................... 135
6.24.2.5 xnregistry_unlink() .................................................... 135

6.25 Driver programming interface ............................................. 136

6.25.1 Detailed Description ....................................................... 136

6.26 Driver to driver services .................................................... 137

6.26.1 Detailed Description ....................................................... 138
6.26.2 Function Documentation .................................................. 138

6.26.2.1 rtdm_accept() .......................................................... 138
6.26.2.2 rtdm_bind() ............................................................ 139
6.26.2.3 rtdm_close() ........................................................... 140
6.26.2.4 rtdm_connect() ........................................................ 140
6.26.2.5 rtdm_getpeername() ................................................... 141
6.26.2.6 rtdm_getsockname() .................................................. 142
6.26.2.7 rtdm_getsockopt() ...................................................... 143
6.26.2.8 rtdm_ioctl() ............................................................ 143
6.26.2.9 rtdm_listen() ........................................................... 144
6.26.2.10 rtdm_open() ............................................................ 145
6.26.2.11 rtdm_read() ........................................................... 146
6.26.2.12 rtdm_recv() ............................................................ 146
6.26.2.13 rtdm_recvfrom() ...................................................... 147
6.26.2.14 rtdm_recvmsg() ....................................................... 148
6.26.2.15 rtdm_send() ........................................................... 149
6.28.3.10 rtdm_recvmsg_handler() ........................................... 166
6.28.3.11 rtdm_select_handler() ............................................ 166
6.28.3.12 rtdm_sendmsg_handler() ........................................... 167
6.28.3.13 rtdm_socket_handler() ............................................ 167
6.28.3.14 rtdm_write_handler() ............................................. 168

6.29 Clock Services ............................................................. 169
   6.29.1 Detailed Description ................................................ 169
   6.29.2 Function Documentation ............................................ 169
      6.29.2.1 rtdm_clock_read() ............................................ 169
      6.29.2.2 rtdm_clock_read_monotonic() ................................ 170

6.30 Task Services ............................................................. 171
   6.30.1 Detailed Description ................................................ 172
   6.30.2 Typedef Documentation ............................................. 172
      6.30.2.1 rtdm_task_proc_t ............................................. 172
   6.30.3 Function Documentation ............................................ 172
      6.30.3.1 rtdm_task_busy_sleep() .................................... 173
      6.30.3.2 rtdm_task_busy_wait() ..................................... 173
      6.30.3.3 rtdm_task_current() ....................................... 174
      6.30.3.4 rtdm_task_destroy() ....................................... 174
      6.30.3.5 rtdm_task_init() ........................................... 175
      6.30.3.6 rtdm_task_join() ........................................... 175
      6.30.3.7 rtdm_task_set_period() ................................... 176
      6.30.3.8 rtdm_task_set_priority() ................................ 176
      6.30.3.9 rtdm_task_should_stop() .................................. 177
      6.30.3.10 rtdm_task_sleep() ......................................... 177
      6.30.3.11 rtdm_task_sleep_abs() ................................... 178
      6.30.3.12 rtdm_task_sleep_until() .................................. 178
      6.30.3.13 rtdm_task_unblock() ...................................... 179
      6.30.3.14 rtdm_task_wait_period() .................................. 179
      6.30.3.15 rtdm_wait_complete() .................................... 180
6.30.3.1 rtdm_wait_is_completed() .................................................. 180
6.30.3.17 rtdm_wait_prepare() ....................................................... 181

6.31 Timer Services ................................................................. 182

6.31.1 Detailed Description ....................................................... 182
6.31.2 Typedef Documentation ................................................... 182
  6.31.2.1 rtdm_timer_handler_t .................................................. 183
6.31.3 Enumeration Type Documentation ........................................ 183
  6.31.3.1 rtdm_timer_mode ....................................................... 183

6.31.4 Function Documentation .................................................. 183
  6.31.4.1 rtdm_timer_destroy() .................................................. 183
  6.31.4.2 rtdm_timer_init() ....................................................... 184
  6.31.4.3 rtdm_timer_start() ...................................................... 184
  6.31.4.4 rtdm_timer_start_in_handler() ...................................... 185
  6.31.4.5 rtdm_timer_stop() ...................................................... 185
  6.31.4.6 rtdm_timer_stop_in_handler() ...................................... 186

6.32 Synchronisation Services .................................................. 187

6.32.1 Detailed Description ....................................................... 188
6.32.2 Enumeration Type Documentation ........................................ 188
  6.32.2.1 rtdm_selecttype ....................................................... 188

6.32.3 Function Documentation .................................................. 189
  6.32.3.1 rtdm_for_each_waiter() .................................................. 189
  6.32.3.2 rtdm_for_each_waiter_safe() ......................................... 189
  6.32.3.3 rtdm_timedwait() ....................................................... 190
  6.32.3.4 rtdm_timedwait_condition() ......................................... 191
  6.32.3.5 rtdm_timedwait_condition_locked() .................................. 192
  6.32.3.6 rtdm_timedwait_locked() .............................................. 192
  6.32.3.7 rtdm_toseq_init() ....................................................... 193
  6.32.3.8 rtdm_wait() ............................................................. 194
  6.32.3.9 rtdm_wait_condition() ............................................... 195
  6.32.3.10 rtdm_wait_condition_locked() ..................................... 195
## 6.32.3.11 rtdm_wait_locked() .......................... 196
## 6.32.3.12 rtdm_waitqueue_broadcast() ............... 197
## 6.32.3.13 rtdm_waitqueue_destroy() ................... 197
## 6.32.3.14 rtdm_waitqueue_flush() ..................... 197
## 6.32.3.15 rtdm_waitqueue_init() ....................... 198
## 6.32.3.16 rtdm_waitqueue_lock() ....................... 198
## 6.32.3.17 rtdm_waitqueue_signal() ................. 199
## 6.32.3.18 rtdm_waitqueue_unlock() ................. 199
## 6.32.3.19 rtdm_waitqueue_wakeup() .................. 200

### 6.33 Event Services ........................................ 201

#### 6.33.1 Detailed Description ............................ 201

#### 6.33.2 Function Documentation .......................... 201

##### 6.33.2.1 rtdm_event_clear() ......................... 201
##### 6.33.2.2 rtdm_event_destroy() ....................... 202
##### 6.33.2.3 rtdm_event_init() ......................... 202
##### 6.33.2.4 rtdm_event_pulse() ......................... 203
##### 6.33.2.5 rtdm_event_select() ....................... 203
##### 6.33.2.6 rtdm_event_signal() ....................... 204
##### 6.33.2.7 rtdm_event_timedwait() ................... 204
##### 6.33.2.8 rtdm_event_wait() ......................... 205

### 6.34 Semaphore Services .................................... 207

#### 6.34.1 Detailed Description ............................ 207

#### 6.34.2 Function Documentation .......................... 207

##### 6.34.2.1 rtdm_sem_destroy() ....................... 207
##### 6.34.2.2 rtdm_sem_down() ............................ 208
##### 6.34.2.3 rtdm_sem_init() ............................ 208
##### 6.34.2.4 rtdm_sem_select() ......................... 209
##### 6.34.2.5 rtdm_sem_timeddown() ..................... 209
##### 6.34.2.6 rtdm_sem_up() .............................. 210

### 6.35 Mutex services ........................................ 211
6.35.1 Detailed Description ........................................... 211
6.35.2 Function Documentation ....................................... 211
   6.35.2.1 rtdm_mutex_destroy() .................................. 211
   6.35.2.2 rtdm_mutex_init() ....................................... 212
   6.35.2.3 rtdm_mutex_lock() ....................................... 212
   6.35.2.4 rtdm_mutex_timedlock() ................................. 213
   6.35.2.5 rtdm_mutex_unlock() .................................... 213
6.36 Interrupt Management Services ................................. 215
   6.36.1 Detailed Description ...................................... 216
   6.36.2 Macro Definition Documentation .......................... 216
      6.36.2.1 rtdm_irq_get_arg ..................................... 216
   6.36.3 Typedef Documentation ................................... 216
      6.36.3.1 rtdm_irq_handler_t .................................. 217
   6.36.4 Function Documentation ................................... 218
      6.36.4.1 rtdm_irq_disable() .................................. 218
      6.36.4.2 rtdm_irq_enable() ................................... 218
      6.36.4.3 rtdm_irq_free() ...................................... 219
      6.36.4.4 rtdm_irq_request() .................................. 220
6.37 Non-Real-Time Signalling Services ............................ 221
   6.37.1 Detailed Description ...................................... 221
   6.37.2 Typedef Documentation ................................... 221
      6.37.2.1 rtdm_nrtsig_handler_t ............................... 221
   6.37.3 Function Documentation ................................... 222
      6.37.3.1 rtdm_nrtsig_destroy() ............................... 222
      6.37.3.2 rtdm_nrtsig_init() .................................. 222
      6.37.3.3 rtdm_nrtsig_pend() .................................. 223
      6.37.3.4 rtdm_schedule_nrt_work() ............................ 223
6.38 Utility Services .................................................. 224
   6.38.1 Detailed Description ...................................... 225
   6.38.2 Function Documentation ................................... 225
6.38.2.1 rtdm_copy_from_user() ................................................. 225
6.38.2.2 rtdm_copy_to_user() ................................................. 226
6.38.2.3 rtdm_free() .......................................................... 226
6.38.2.4 rtdm_in_rt_context() ................................................. 227
6.38.2.5 rtdm_iomap_to_user() ................................................. 227
6.38.2.6 rtdm_malloc() ......................................................... 228
6.38.2.7 rtdm_mmap_iomem() ................................................. 229
6.38.2.8 rtdm_mmap_kmem() ................................................... 229
6.38.2.9 rtdm_mmap_to_user() ................................................. 230
6.38.2.10 rtdm_mmap_vmem() .................................................. 231
6.38.2.11 rtdm_munmap() ........................................................ 232
6.38.2.12 rtdm_printk() ........................................................ 232
6.38.2.13 rtdm_printk_ratelimited() ....................................... 233
6.38.2.14 rtdm_ratelimit() ..................................................... 233
6.38.2.15 rtdm_read_user_ok() ............................................... 234
6.38.2.16 rtdm_rt_capable() .................................................. 234
6.38.2.17 rtdm_rwlock_user_ok() ............................................ 235
6.38.2.18 rtdm_safe_copy_from_user() ..................................... 235
6.38.2.19 rtdm_safe_copy_to_user() ....................................... 236
6.38.2.20 rtdm_strncpy_from_user() ....................................... 237

6.39 SCHED QUOTA scheduling policy ........................................... 238
   6.39.1 Detailed Description ................................................. 238

6.40 Thread scheduling control .................................................. 239
   6.40.1 Detailed Description ................................................. 239
   6.40.2 Function Documentation ............................................. 239
      6.40.2.1 xnsched_rotate() ............................................. 239
      6.40.2.2 xnsched_run() ................................................. 240

6.41 Synchronous I/O multiplexing ............................................. 241
   6.41.1 Detailed Description ................................................. 241
   6.41.2 Function Documentation ............................................. 242
6.41.2.1 xnselect() ........................................ 242
6.41.2.2 xnselect_bind() .................................. 242
6.41.2.3 xnselect_destroy() .............................. 243
6.41.2.4 xnselect_init() .................................. 244
6.41.2.5 xnselect_signal() ............................... 244
6.41.2.6 xnselector_destroy() ......................... 245
6.41.2.7 xnselector_init() ................................ 245

6.42 Thread synchronization services .............................. 246
  6.42.1 Detailed Description ................................ 246
  6.42.2 Function Documentation ......................... 246
    6.42.2.1 xnsynch_acquire() ......................... 247
    6.42.2.2 xnsynch_destroy() ....................... 247
    6.42.2.3 xnsynch_flush() .......................... 248
    6.42.2.4 xnsynch_init() ............................ 249
    6.42.2.5 xnsynch_peek_pendq() .................... 250
    6.42.2.6 xnsynch_release() ....................... 250
    6.42.2.7 xnsynch_sleep_on() ...................... 251
    6.42.2.8 xnsynch_try_acquire() ................... 252
    6.42.2.9 xnsynch_wakeup_one_sleeper() ............. 253
    6.42.2.10 xnsynch_wakeup_this_sleeper() ........... 253

6.43 Thread services ........................................ 255
  6.43.1 Detailed Description ................................ 256
  6.43.2 Function Documentation ......................... 256
    6.43.2.1 xnthread_cancel() ...................... 256
    6.43.2.2 xnthread_current() ...................... 257
    6.43.2.3 xnthread_from_task() .................... 257
    6.43.2.4 xnthread_harden() ....................... 258
    6.43.2.5 xnthread_init() ......................... 258
    6.43.2.6 xnthread_join() ......................... 259
    6.43.2.7 xnthread_map() ......................... 260
6.43.2.8 xnthread_migrate() ........................................... 261
6.43.2.9 xnthread_relax() ............................................ 262
6.43.2.10 xnthread_resume() .......................................... 262
6.43.2.11 xnthread_set_mode() ........................................ 263
6.43.2.12 xnthread_set_periodic() .................................... 264
6.43.2.13 xnthread_set_schedparam() .................................. 265
6.43.2.14 xnthread_set_slice() ....................................... 266
6.43.2.15 xnthread_start() ........................................... 267
6.43.2.16 xnthread_suspend() ......................................... 267
6.43.2.17 xnthread_test_cancel() ..................................... 269
6.43.2.18 xnthread_unblock() ......................................... 269
6.43.2.19 xnthread_wait_period() ..................................... 270

6.44 Timer services ..................................................... 271
6.44.1 Detailed Description ............................................ 271
6.44.2 Function Documentation ........................................ 272
  6.44.2.1 __xntimer_migrate() ......................................... 272
  6.44.2.2 program htick_shot() ....................................... 272
  6.44.2.3 switch_htick_mode() ...................................... 273
  6.44.2.4 xntimer_destroy() ......................................... 273
  6.44.2.5 xntimer_get_date() ........................................ 274
  6.44.2.6 xntimer_get_overruns() .................................... 274
  6.44.2.7 xntimer_get_timeout() .................................... 275
  6.44.2.8 xntimer_grab_hardware() .................................. 275
  6.44.2.9 xntimer_init() ............................................ 276
  6.44.2.10 xntimer_interval() ....................................... 277
  6.44.2.11 xntimer_release_hardware() ............................... 277
  6.44.2.12 xntimer_start() .......................................... 278
  6.44.2.13 xntimer_stop() ........................................... 278

6.45 Virtual file services ............................................. 280
6.45.1 Detailed Description ........................................... 281
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.50.2.1</td>
<td>a4l_buf_commit_absget()</td>
<td>297</td>
</tr>
<tr>
<td>6.50.2.2</td>
<td>a4l_buf_commit_absput()</td>
<td>298</td>
</tr>
<tr>
<td>6.50.2.3</td>
<td>a4l_buf_commit_get()</td>
<td>298</td>
</tr>
<tr>
<td>6.50.2.4</td>
<td>a4l_buf_commit_put()</td>
<td>299</td>
</tr>
<tr>
<td>6.50.2.5</td>
<td>a4l_buf_count()</td>
<td>300</td>
</tr>
<tr>
<td>6.50.2.6</td>
<td>a4l_buf_evt()</td>
<td>301</td>
</tr>
<tr>
<td>6.50.2.7</td>
<td>a4l_buf_get()</td>
<td>301</td>
</tr>
<tr>
<td>6.50.2.8</td>
<td>a4l_buf_prepare_absget()</td>
<td>302</td>
</tr>
<tr>
<td>6.50.2.9</td>
<td>a4l_buf_prepare_absput()</td>
<td>303</td>
</tr>
<tr>
<td>6.50.2.10</td>
<td>a4l_buf_prepare_get()</td>
<td>303</td>
</tr>
<tr>
<td>6.50.2.11</td>
<td>a4l_buf_prepare_put()</td>
<td>304</td>
</tr>
<tr>
<td>6.50.2.12</td>
<td>a4l_buf_put()</td>
<td>304</td>
</tr>
<tr>
<td>6.50.2.13</td>
<td>a4l_get_chan()</td>
<td>305</td>
</tr>
<tr>
<td>6.50.2.14</td>
<td>a4l_get_cmd()</td>
<td>305</td>
</tr>
</tbody>
</table>

6.51 Interrupt management services | 306

6.51.1 Detailed Description | 306
6.51.2 Function Documentation | 306
6.51.2.1 a4l_free_irq() | 306
6.51.2.2 a4l_get_irq() | 307
6.51.2.3 a4l_request_irq() | 307

6.52 Misc services | 309

6.52.1 Detailed Description | 309
6.52.2 Function Documentation | 309
6.52.2.1 a4l_get_time() | 309

6.53 Clocks and timers | 310

6.53.1 Detailed Description | 310
6.53.2 Function Documentation | 311
6.53.2.1 clock_getres() | 311
6.53.2.2 clock_gettime() | 312
6.53.2.3 clock_nanosleep() | 313
<table>
<thead>
<tr>
<th>Section</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.53.2.4</td>
<td>clock_settime()</td>
</tr>
<tr>
<td>6.53.2.5</td>
<td>nanosleep()</td>
</tr>
<tr>
<td>6.53.2.6</td>
<td>timer_create()</td>
</tr>
<tr>
<td>6.53.2.7</td>
<td>timer_delete()</td>
</tr>
<tr>
<td>6.53.2.8</td>
<td>timer_getoverrun()</td>
</tr>
<tr>
<td>6.53.2.9</td>
<td>timer_gettime()</td>
</tr>
<tr>
<td>6.53.2.10</td>
<td>timer_settime()</td>
</tr>
<tr>
<td>6.54</td>
<td>Condition variables</td>
</tr>
<tr>
<td>6.54.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>6.54.2</td>
<td>Function Documentation</td>
</tr>
<tr>
<td>6.54.2.1</td>
<td>pthread_cond_broadcast()</td>
</tr>
<tr>
<td>6.54.2.2</td>
<td>pthread_cond_destroy()</td>
</tr>
<tr>
<td>6.54.2.3</td>
<td>pthread_cond_init()</td>
</tr>
<tr>
<td>6.54.2.4</td>
<td>pthread_cond_signal()</td>
</tr>
<tr>
<td>6.54.2.5</td>
<td>pthread_cond_timedwait()</td>
</tr>
<tr>
<td>6.54.2.6</td>
<td>pthread_cond_wait()</td>
</tr>
<tr>
<td>6.54.2.7</td>
<td>pthread_condattr_destroy()</td>
</tr>
<tr>
<td>6.54.2.8</td>
<td>pthread_condattr_getclock()</td>
</tr>
<tr>
<td>6.54.2.9</td>
<td>pthread_condattr_getpshared()</td>
</tr>
<tr>
<td>6.54.2.10</td>
<td>pthread_condattr_init()</td>
</tr>
<tr>
<td>6.54.2.11</td>
<td>pthread_condattr_setclock()</td>
</tr>
<tr>
<td>6.54.2.12</td>
<td>pthread_condattr_setpshared()</td>
</tr>
<tr>
<td>6.55</td>
<td>POSIX interface</td>
</tr>
<tr>
<td>6.55.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>6.56</td>
<td>Message queues</td>
</tr>
<tr>
<td>6.56.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>6.56.2</td>
<td>Function Documentation</td>
</tr>
<tr>
<td>6.56.2.1</td>
<td>mq_close()</td>
</tr>
<tr>
<td>6.56.2.2</td>
<td>mq_getattr()</td>
</tr>
<tr>
<td>6.56.2.3</td>
<td>mq_notify()</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>6.58.2.4</td>
<td>sched_get_priority_min_ex()</td>
</tr>
<tr>
<td>6.58.2.5</td>
<td>sched_getconfig_np()</td>
</tr>
<tr>
<td>6.58.2.6</td>
<td>sched_getscheduler()</td>
</tr>
<tr>
<td>6.58.2.7</td>
<td>sched_getscheduler_ex()</td>
</tr>
<tr>
<td>6.58.2.8</td>
<td>sched_setconfig_np()</td>
</tr>
<tr>
<td>6.58.2.9</td>
<td>sched_setscheduler()</td>
</tr>
<tr>
<td>6.58.2.10</td>
<td>sched_setscheduler_ex()</td>
</tr>
<tr>
<td>6.58.2.11</td>
<td>sched_yield()</td>
</tr>
<tr>
<td>6.59</td>
<td>Semaphores</td>
</tr>
<tr>
<td>6.59.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>6.59.2</td>
<td>Function Documentation</td>
</tr>
<tr>
<td>6.59.2.1</td>
<td>sem_close()</td>
</tr>
<tr>
<td>6.59.2.2</td>
<td>sem_destroy()</td>
</tr>
<tr>
<td>6.59.2.3</td>
<td>sem_init()</td>
</tr>
<tr>
<td>6.59.2.4</td>
<td>sem_post()</td>
</tr>
<tr>
<td>6.59.2.5</td>
<td>sem_timedwait()</td>
</tr>
<tr>
<td>6.59.2.6</td>
<td>sem_trywait()</td>
</tr>
<tr>
<td>6.59.2.7</td>
<td>sem_unlink()</td>
</tr>
<tr>
<td>6.59.2.8</td>
<td>sem_wait()</td>
</tr>
<tr>
<td>6.60</td>
<td>Thread management</td>
</tr>
<tr>
<td>6.60.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>6.60.2</td>
<td>Function Documentation</td>
</tr>
<tr>
<td>6.60.2.1</td>
<td>pthread_create()</td>
</tr>
<tr>
<td>6.60.2.2</td>
<td>pthread_join()</td>
</tr>
<tr>
<td>6.60.2.3</td>
<td>pthread_kill()</td>
</tr>
<tr>
<td>6.60.2.4</td>
<td>pthread_setmode_np()</td>
</tr>
<tr>
<td>6.60.2.5</td>
<td>pthread_setname_np()</td>
</tr>
<tr>
<td>6.61</td>
<td>Scheduling management</td>
</tr>
<tr>
<td>6.61.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>6.61.2</td>
<td>Function Documentation</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>6.70 Level 1 API</td>
<td>419</td>
</tr>
<tr>
<td>6.70.1 Detailed Description</td>
<td>419</td>
</tr>
<tr>
<td>6.71 Synchronous acquisition API</td>
<td>420</td>
</tr>
<tr>
<td>6.71.1 Detailed Description</td>
<td>422</td>
</tr>
<tr>
<td>6.71.2 Function Documentation</td>
<td>422</td>
</tr>
<tr>
<td>6.71.2.1 a4l_snd_insn()</td>
<td>422</td>
</tr>
<tr>
<td>6.71.2.2 a4l_snd_insnlist()</td>
<td>423</td>
</tr>
<tr>
<td>6.72 Level 2 API</td>
<td>424</td>
</tr>
<tr>
<td>6.72.1 Detailed Description</td>
<td>424</td>
</tr>
<tr>
<td>6.73 Synchronous acquisition API</td>
<td>425</td>
</tr>
<tr>
<td>6.73.1 Detailed Description</td>
<td>425</td>
</tr>
<tr>
<td>6.73.2 Function Documentation</td>
<td>425</td>
</tr>
<tr>
<td>6.73.2.1 a4l_config_subd()</td>
<td>425</td>
</tr>
<tr>
<td>6.73.2.2 a4l_sync_dio()</td>
<td>426</td>
</tr>
<tr>
<td>6.73.2.3 a4l_sync_read()</td>
<td>427</td>
</tr>
<tr>
<td>6.73.2.4 a4l_sync_write()</td>
<td>427</td>
</tr>
<tr>
<td>6.74 Analogy user API</td>
<td>429</td>
</tr>
<tr>
<td>6.74.1 Detailed Description</td>
<td>429</td>
</tr>
<tr>
<td>6.75 Level 0 API</td>
<td>430</td>
</tr>
<tr>
<td>6.75.1 Detailed Description</td>
<td>430</td>
</tr>
<tr>
<td>6.76 Basic Syscall API</td>
<td>431</td>
</tr>
<tr>
<td>6.76.1 Detailed Description</td>
<td>431</td>
</tr>
<tr>
<td>6.76.2 Function Documentation</td>
<td>431</td>
</tr>
<tr>
<td>6.76.2.1 a4l_sys_close()</td>
<td>431</td>
</tr>
<tr>
<td>6.76.2.2 a4l_sys_open()</td>
<td>432</td>
</tr>
<tr>
<td>6.76.2.3 a4l_sys_read()</td>
<td>432</td>
</tr>
<tr>
<td>6.76.2.4 a4l_sys_write()</td>
<td>433</td>
</tr>
<tr>
<td>6.77 Attach / detach Syscall API</td>
<td>434</td>
</tr>
<tr>
<td>6.77.1 Detailed Description</td>
<td>434</td>
</tr>
<tr>
<td>6.77.2 Function Documentation</td>
<td>434</td>
</tr>
</tbody>
</table>
6.77 Alarm services .................................................. 437
  6.78.1 Detailed Description ..................................... 437
  6.78.2 Function Documentation ................................ 437
    6.78.2.1 rt_alarm_create() .................................. 438
    6.78.2.2 rt_alarm_delete() .................................. 438
    6.78.2.3 rt_alarm_inquire() ................................ 439
    6.78.2.4 rt_alarm_start() .................................. 440
    6.78.2.5 rt_alarm_stop() ................................... 440
6.79 Buffer services ............................................. 442
  6.79.1 Detailed Description ..................................... 443
  6.79.2 Macro Definition Documentation ....................... 443
    6.79.2.1 B_PRIO ........................................... 443
  6.79.3 Function Documentation ................................ 443
    6.79.3.1 rt_buffer_bind() .................................. 443
    6.79.3.2 rt_buffer_clear() .................................. 444
    6.79.3.3 rt_buffer_create() ................................ 445
    6.79.3.4 rt_buffer_delete() ................................ 446
    6.79.3.5 rt_buffer_inquire() ................................ 446
    6.79.3.6 rt_buffer_read() .................................. 447
    6.79.3.7 rt_buffer_read_timed() ............................... 447
    6.79.3.8 rt_buffer_read_until() ............................. 449
    6.79.3.9 rt_buffer_unbind() ................................ 449
    6.79.3.10 rt_buffer_write() ................................ 450
    6.79.3.11 rt_buffer_write_timed() ........................... 450
    6.79.3.12 rt_buffer_write_until() ......................... 451
6.80 Condition variable services ......................... 453
  6.80.1 Detailed Description ..................................... 454
6.80.2 Function Documentation ........................................ 454
   6.80.2.1 rt_cond_bind() ........................................ 454
   6.80.2.2 rt_cond_broadcast() .................................. 455
   6.80.2.3 rt_cond_create() ...................................... 455
   6.80.2.4 rt_cond_delete() ...................................... 456
   6.80.2.5 rt_cond_inquire() .................................... 457
   6.80.2.6 rt_cond_signal() ...................................... 457
   6.80.2.7 rt_cond_unbind() ..................................... 458
   6.80.2.8 rt_cond_wait() ........................................ 458
   6.80.2.9 rt_cond_wait_timed() ................................ 459
   6.80.2.10 rt_cond_wait_until() ................................. 460

6.81 Event flag group services ...................................... 461
   6.81.1 Detailed Description ..................................... 462
   6.81.2 Macro Definition Documentation .......................... 462
      6.81.2.1 EV_ANY ............................................ 462
      6.81.2.2 EV_PRIO ................................ .......... 462
   6.81.3 Function Documentation ................................... 462
      6.81.3.1 rt_event_bind() ................................... 462
      6.81.3.2 rt_event_clear() ................................... 463
      6.81.3.3 rt_event_create() ................................ 464
      6.81.3.4 rt_event_delete() ................................ 465
      6.81.3.5 rt_event_inquire() ................................ 465
      6.81.3.6 rt_event_signal() ................................ 466
      6.81.3.7 rt_event_unbind() ................................ 467
      6.81.3.8 rt_event_wait() .................................... 468
      6.81.3.9 rt_event_wait_timed() ............................. 468
      6.81.3.10 rt_event_wait_until() ......................... 470

6.82 Heap management services .................................... 471
   6.82.1 Detailed Description .................................... 472
   6.82.2 Macro Definition Documentation .......................... 472
6.85.3 Function Documentation ................................................. 489
  6.85.3.1 rt_pipe_bind() ................................................... 489
  6.85.3.2 rt_pipe_create() ............................................... 490
  6.85.3.3 rt_pipe_delete() ............................................... 491
  6.85.3.4 rt_pipe_read() ................................................ 492
  6.85.3.5 rt_pipe_read_timed() ........................................ 493
  6.85.3.6 rt_pipe_read_until() ........................................ 494
  6.85.3.7 rt_pipe_stream() ............................................... 494
  6.85.3.8 rt_pipe_unbind() ............................................ 495
  6.85.3.9 rt_pipe_write() .............................................. 495

6.86 Message queue services .................................................. 497
  6.86.1 Detailed Description ............................................... 498
  6.86.2 Macro Definition Documentation .................................. 498
  6.86.2.1 Q_PRIO ......................................................... 498

6.86.3 Function Documentation ............................................... 498
  6.86.3.1 rt_queue_alloc() .............................................. 498
  6.86.3.2 rt_queue_bind() .............................................. 499
  6.86.3.3 rt_queue_create() ........................................... 500
  6.86.3.4 rt_queue_delete() ........................................... 501
  6.86.3.5 rt_queue_flush() ............................................ 502
  6.86.3.6 rt_queue_free() .............................................. 502
  6.86.3.7 rt_queue_inquire() ........................................... 503
  6.86.3.8 rt_queue_read() ............................................. 503
  6.86.3.9 rt_queue_read_timed() ....................................... 504
  6.86.3.10 rt_queue_read_until() ..................................... 505
  6.86.3.11 rt_queue_receive() ........................................ 505
  6.86.3.12 rt_queue_receive_timed() .................................. 506
  6.86.3.13 rt_queue_receive_until() .................................. 507
  6.86.3.14 rt_queue_send() ............................................ 507
  6.86.3.15 rt_queue_unbind() ......................................... 508
6.88.3.9 rt_task_reply() .................................................. 528
6.88.3.10 rt_task_resume() ............................................. 529
6.88.3.11 rt_task_same() ................................................... 530
6.88.3.12 rt_task_self() ................................................... 531
6.88.3.13 rt_task_send() .................................................. 531
6.88.3.14 rt_task_send_timed() ........................................ 532
6.88.3.15 rt_task_send_until() .......................................... 533
6.88.3.16 rt_task_set_affinity() ........................................ 534
6.88.3.17 rt_task_set_mode() ............................................ 535
6.88.3.18 rt_task_set_periodic() ........................................ 536
6.88.3.19 rt_task_set_priority() ........................................ 537
6.88.3.20 rt_task_shadow() ............................................... 538
6.88.3.21 rt_task_sleep() ................................................ 539
6.88.3.22 rt_task_sleep_until() .......................................... 540
6.88.3.23 rt_task_slice() ............................................... 541
6.88.3.24 rt_task_spawn() .............................................. 542
6.88.3.25 rt_task_start() ............................................... 543
6.88.3.26 rt_task_suspend() ............................................ 544
6.88.3.27 rt_task_unbind() ............................................. 544
6.88.3.28 rt_task_unblock() ............................................ 545
6.88.3.29 rt_task_wait_period() ....................................... 545
6.88.3.30 rt_task_yield() ............................................... 546

6.89 Timer management services ......................................... 547
6.89.1 Detailed Description ............................................. 547
6.89.2 Typedef Documentation ........................................... 547
6.89.2.1 RT_TIMER_INFO .................................................. 548
6.89.3 Function Documentation .......................................... 548
6.89.3.1 rt_timer_inquire() ............................................ 548
6.89.3.2 rt_timer_ns2ticks() ............................................ 548
6.89.3.3 rt_timer_read() ............................................... 549
### 6.89.4 rt_timer_spin() ........................................ 549
### 6.89.5 rt_timer_ticks2ns() .................................... 550
### 6.90 VxWorks® emulator ...................................... 551
### 6.91 pSOS® emulator ........................................... 552
### 6.92 Transition Kit ............................................ 553
   - 6.92.1 Detailed Description .................................. 553
   - 6.92.2 Function Documentation ............................. 553
     - 6.92.2.1 COMPAT__rt_alarm_create() ..................... 553
     - 6.92.2.2 COMPAT__rt_event_clear() ...................... 554
     - 6.92.2.3 COMPAT__rt_event_create() .................... 555
     - 6.92.2.4 COMPAT__rt_event_signal() ..................... 556
     - 6.92.2.5 COMPAT__rt_pipe_create() ..................... 557
     - 6.92.2.6 COMPAT__rt_task_create() ..................... 558
     - 6.92.2.7 COMPAT__rt_task_set_periodic() .............. 559
     - 6.92.2.8 pthread_make_periodic_np() ................... 560
     - 6.92.2.9 pthread_wait_np() ................................ 561
     - 6.92.2.10 rt_alarm_wait() ................................ 562

### 7 Data Structure Documentation ............................... 563
   - 7.1 a4l_channel Struct Reference .......................... 563
     - 7.1.1 Detailed Description ................................ 563
     - 7.1.2 Field Documentation .................................. 563
       - 7.1.2.1 flags .............................................. 563
       - 7.1.2.2 nb_bits ........................................... 563
   - 7.2 a4l_channels_desc Struct Reference .................... 564
     - 7.2.1 Detailed Description ................................ 564
     - 7.2.2 Field Documentation .................................. 564
       - 7.2.2.1 chans .............................................. 564
       - 7.2.2.2 length ............................................. 565
       - 7.2.2.3 mode .............................................. 565
   - 7.3 a4l_cmd_desc Struct Reference .......................... 565
7.21.1 Detailed Description ................................................. 581

7.22 RT_EVENT_INFO Struct Reference .............................. 581
  7.22.1 Detailed Description ................................................ 582

7.23 RT_HEAP_INFO Struct Reference .............................. 582
  7.23.1 Detailed Description ................................................ 582
  7.23.2 Field Documentation .................................................
    7.23.2.1 heapsize .......................................................... 582
    7.23.2.2 usablemem ....................................................... 583
    7.23.2.3 usedmem .......................................................... 583

7.24 RT_MUTEX_INFO Struct Reference .............................. 583
  7.24.1 Detailed Description ................................................ 583
  7.24.2 Field Documentation .................................................
    7.24.2.1 owner ............................................................ 584

7.25 RT_QUEUE_INFO Struct Reference .............................. 584
  7.25.1 Detailed Description ................................................ 584

7.26 RT_SEM_INFO Struct Reference .............................. 584
  7.26.1 Detailed Description ................................................ 585

7.27 RT_TASK_INFO Struct Reference .............................. 585
  7.27.1 Detailed Description ................................................ 585

7.28 rt_timer_info Struct Reference .............................. 586
  7.28.1 Detailed Description ................................................ 586
  7.28.2 Field Documentation .................................................
    7.28.2.1 date .............................................................. 586

7.29 rtdm_dev_context Struct Reference .......................... 587
  7.29.1 Detailed Description ................................................ 587
  7.29.2 Field Documentation .................................................
    7.29.2.1 device .......................................................... 588

7.30 rtdm_device Struct Reference ............................... 588
  7.30.1 Detailed Description ................................................ 589
  7.30.2 Field Documentation .................................................
### 7.30.2.1 "@11
- 589

### 7.30.2.2 driver
- 589

### 7.30.2.3 label
- 589

### 7.30.2.4 minor
- 590

#### 7.31 rtdm_device_info Struct Reference
- 590

##### 7.31.1 Detailed Description
- 590

#### 7.32 rtdm_driver Struct Reference
- 591

##### 7.32.1 Detailed Description
- 592

##### 7.32.2 Field Documentation
- 592

- **7.32.2.1 base_minor**
- 592

- **7.32.2.2 context_size**
- 592

- **7.32.2.3 device_count**
- 592

- **7.32.2.4 device_flags**
- 592

- **7.32.2.5 profile_info**
- 593

#### 7.33 rtdm_fd_ops Struct Reference
- 593

##### 7.33.1 Detailed Description
- 594

##### 7.33.2 Field Documentation
- 594

- **7.33.2.1 close**
- 594

- **7.33.2.2 get_unmapped_area**
- 594

- **7.33.2.3 ioctl_nrt**
- 594

- **7.33.2.4 ioctl_rt**
- 594

- **7.33.2.5 mmap**
- 595

- **7.33.2.6 open**
- 595

- **7.33.2.7 read_nrt**
- 595

- **7.33.2.8 read_rt**
- 595

- **7.33.2.9 recvmsg_nrt**
- 595

- **7.33.2.10 recvmsg_rt**
- 595

- **7.33.2.11 select**
- 596

- **7.33.2.12 sendmsg_nrt**
- 596

- **7.33.2.13 sendmsg_rt**
- 596

---

Generated by Doxygen
7.33.2.14 socket .................................................. 596
7.33.2.15 write_nrt ............................................. 596
7.33.2.16 write_rt ............................................. 596
7.34 rtdm_sm_ops Struct Reference .......................... 597
  7.34.1 Detailed Description .................................. 597
7.35 rtdm_spi_config Struct Reference ....................... 597
  7.35.1 Detailed Description .................................. 597
7.36 rtipc_port_label Struct Reference ....................... 597
  7.36.1 Detailed Description .................................. 598
  7.36.2 Field Documentation .................................. 598
    7.36.2.1 label ............................................... 598
7.37 rtser_config Struct Reference ......................... 598
  7.37.1 Detailed Description .................................. 599
7.38 rtser_event Struct Reference .......................... 599
  7.38.1 Detailed Description .................................. 600
7.39 rtser_status Struct Reference .......................... 600
  7.39.1 Detailed Description .................................. 600
7.40 sockaddr_can Struct Reference ......................... 600
  7.40.1 Detailed Description .................................. 601
  7.40.2 Field Documentation .................................. 601
    7.40.2.1 can_ifindex ....................................... 601
7.41 sockaddr_ipc Struct Reference ......................... 601
  7.41.1 Detailed Description .................................. 602
  7.41.2 Field Documentation .................................. 602
    7.41.2.1 sipc_port ......................................... 602
7.42 udd_device Struct Reference .......................... 602
  7.42.1 Detailed Description .................................. 603
  7.42.2 Field Documentation .................................. 603
    7.42.2.1 close ............................................. 603
    7.42.2.2 device_flags ...................................... 604
7.42.2.3 device_subclass ........................................... 604
7.42.2.4 interrupt ..................................................... 604
7.42.2.5 ioctl ............................................................. 605
7.42.2.6 irq ............................................................... 605
7.42.2.7 mem_regions ................................................... 605
7.42.2.8 mmap ............................................................. 605
7.42.2.9 open .............................................................. 606
7.43 udd_memregion Struct Reference ................................. 606
  7.43.1 Detailed Description ......................................... 606
  7.43.2 Field Documentation ......................................... 607
    7.43.2.1 addr ........................................................ 607
    7.43.2.2 len .......................................................... 607
    7.43.2.3 type ........................................................ 607
7.44 udd_device::udd_reserved Struct Reference ..................... 608
  7.44.1 Detailed Description ......................................... 608
7.45 udd_signotify Struct Reference .................................. 608
  7.45.1 Detailed Description ......................................... 609
  7.45.2 Field Documentation ......................................... 609
    7.45.2.1 pid ........................................................ 609
    7.45.2.2 sig ........................................................ 609
7.46 xnheap::xnbucket Struct Reference ................................ 609
  7.46.1 Detailed Description ......................................... 609
7.47 xnsched Struct Reference ......................................... 610
  7.47.1 Detailed Description ......................................... 610
  7.47.2 Field Documentation ......................................... 610
    7.47.2.1 cpu ....................................................... 610
    7.47.2.2 curr ...................................................... 610
    7.47.2.3 htimer .................................................... 610
    7.47.2.4 inesting .................................................. 611
    7.47.2.5 iflags .................................................... 611
## 7.52 Detailed Description

### 7.52.1 Detailed Description

620

## 7.53 xnvfile_snapshot_iterator Struct Reference

### 7.53.1 Detailed Description

620

### 7.53.2 Field Documentation

#### 7.53.2.1 databuf

621

#### 7.53.2.2 endfn

621

#### 7.53.2.3 nrdata

621

#### 7.53.2.4 private

621

#### 7.53.2.5 seq

621

#### 7.53.2.6 vfile

622

## 7.54 xnvfile_snapshot_ops Struct Reference

### 7.54.1 Detailed Description

622

### 7.54.2 Field Documentation

#### 7.54.2.1 begin

622

#### 7.54.2.2 end

623

#### 7.54.2.3 next

623

#### 7.54.2.4 rewind

624

#### 7.54.2.5 show

625

#### 7.54.2.6 store

625

## 8 File Documentation

### 8.1 include/cobalt/kernel/rtdm/analogy/channel_range.h File Reference

629

#### 8.1.1 Detailed Description

631

### 8.2 include/cobalt/kernel/rtdm/analogy/driver.h File Reference

632

#### 8.2.1 Detailed Description

632

### 8.3 include/cobalt/kernel/rtdm/driver.h File Reference

633

#### 8.3.1 Detailed Description

638

#### 8.3.2 Macro Definition Documentation

638

##### 8.3.2.1 RTDM_CLASS_MAGIC

638

##### 8.3.2.2 RTDM_PROFILE_INFO

638

#### 8.3.3 Function Documentation

639

---

Generated by Doxygen
8.3.3.1 rtdm_fd_device() .......................... 639
8.3.3.2 rtdm_fd_is_user() ......................... 640
8.3.3.3 rtdm_fd_to_private() ...................... 640
8.3.3.4 rtdm_private_to_fd() ...................... 640

8.4 include/cobalt/kernel/rtdm/analogy/subdevice.h File Reference ..................... 641
   8.4.1 Detailed Description ......................... 642

8.5 include/rtdm/can.h File Reference ........................................ 642
   8.5.1 Detailed Description .................................. 643

8.6 include/rtdm/uapi/can.h File Reference ...................................... 643
   8.6.1 Detailed Description .................................. 650

8.7 include/cobalt/kernel/rtdm/fd.h File Reference .................................. 650
   8.7.1 Detailed Description .................................. 651
   8.7.2 Function Documentation .................................. 651
      8.7.2.1 rtdm_fd_get() .................................. 652
      8.7.2.2 rtdm_fd_lock() .................................. 652
      8.7.2.3 rtdm_fd_put() .................................. 653
      8.7.2.4 rtdm_fd_select() .................................. 653
      8.7.2.5 rtdm_fd_unlock() .................................. 654

8.8 include/rtdm/uapi/ipc.h File Reference ........................................ 654
   8.8.1 Detailed Description .................................. 657

8.9 include/rtdm/rtdm.h File Reference ........................................... 658
   8.9.1 Detailed Description .................................. 658

8.10 include/rtdm/uapi/rtdm.h File Reference ........................................ 658
   8.10.1 Detailed Description .................................. 660

8.11 include/rtdm/serial.h File Reference ........................................... 660
   8.11.1 Detailed Description .................................. 661

8.12 include/rtdm/uapi/serial.h File Reference ........................................ 661
   8.12.1 Detailed Description .................................. 665
   8.12.2 Macro Definition Documentation .................................. 666
      8.12.2.1 RTSER_RTIOC_BREAK_CTL .......................... 666
## Example Documentation

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.1 bufp-label.c</td>
<td>691</td>
</tr>
<tr>
<td>9.2 bufp-readwrite.c</td>
<td>693</td>
</tr>
<tr>
<td>9.3 can-rtt.c</td>
<td>696</td>
</tr>
<tr>
<td>9.4 cross-link.c</td>
<td>701</td>
</tr>
<tr>
<td>9.5 iddp-label.c</td>
<td>704</td>
</tr>
<tr>
<td>9.6 iddp-sendrecv.c</td>
<td>707</td>
</tr>
<tr>
<td>9.7 rtcanconfig.c</td>
<td>710</td>
</tr>
<tr>
<td>9.8 rtcanrecv.c</td>
<td>713</td>
</tr>
<tr>
<td>9.9 rtcansend.c</td>
<td>716</td>
</tr>
<tr>
<td>9.10 xddp-echo.c</td>
<td>720</td>
</tr>
<tr>
<td>9.11 xddp-label.c</td>
<td>723</td>
</tr>
<tr>
<td>9.12 xddp-stream.c</td>
<td>727</td>
</tr>
</tbody>
</table>

**Index**

731
Chapter 1

API service tags

All services from the Cobalt/POSIX library, or which belong to APIs based on the Copperplate library may be restricted to particular calling contexts, or entail specific side-effects.

In dual kernel mode, the Cobalt API underlies all other application-oriented APIs, providing POSIX real-time services over the Cobalt real-time core. Therefore, the information below applies to all application-oriented APIs available with Xenomai, such as the Cobalt/POSIX library, the Alchemy API, and to all RTOS emulators as well. To describe this information, each service documented by this section bears a set of tags when applicable.

The table below matches the tags used throughout the documentation with the description of their meaning for the caller.

Attention

By Xenomai thread, we mean any thread created by a Xenomai API service, including real-time Cobalt/POSIX threads in dual kernel mode. By regular/plain POSIX thread, we mean any thread directly created by the standard glibc-based POSIX service over Mercury or Cobalt (i.e. NPT←L/linuxthreads __STD(pthread_create())), excluding such threads which have been promoted to the real-time domain afterwards (aka “shadowed”) over Cobalt.

### Context tags

<table>
<thead>
<tr>
<th>Tag</th>
<th>Context on entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>xthread-only</td>
<td>Must be called from a Xenomai thread</td>
</tr>
<tr>
<td>xhandler-only</td>
<td>Must be called from a Xenomai handler. See note.</td>
</tr>
<tr>
<td>xcontext</td>
<td>May be called from any Xenomai context (thread or handler).</td>
</tr>
<tr>
<td>pthread-only</td>
<td>Must be called from a regular POSIX thread</td>
</tr>
<tr>
<td>thread-unrestricted</td>
<td>May be called from a Xenomai or regular POSIX thread indifferently</td>
</tr>
<tr>
<td>xthread-nowait</td>
<td>May be called from a Xenomai thread unrestricted, or from a regular thread as a non-blocking service only. See note.</td>
</tr>
<tr>
<td>unrestricted</td>
<td>May be called from any context previously described</td>
</tr>
</tbody>
</table>
Note

A Xenomai handler is used for callback-based notifications from Copperplate-based APIs, such as timeouts. This context is NOT mapped to a regular Linux signal handler, it is actually underlaid by a special thread context, so that async-unsafe POSIX services may be invoked internally by the API implementation when running on behalf of such handler. Therefore, calling Xenomai API services from asynchronous regular signal handlers is fundamentally unsafe.

Over Cobalt, the main thread is a particular case, which starts as a regular POSIX thread, then is automatically switched to a Cobalt thread as part of the initialization process, before the main() routine is invoked, unless automatic bootstrap was disabled (see http://xenomai.org/2015/05/application-setup-and-init/#Application_entry_CC).

### Possible side-effects when running the application over the Cobalt core (i.e. dual kernel configuration)

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>switch-primary</td>
<td>the caller may switch to primary mode</td>
</tr>
<tr>
<td>switch-secondary</td>
<td>the caller may switch to secondary mode</td>
</tr>
</tbody>
</table>

Note

As a rule of thumb, any service which might block the caller, causes a switch to primary mode if invoked from secondary mode. This rule might not apply in case the service can complete fully from user-space without any syscall entailed, due to a particular optimization (e.g. fast acquisition of semaphore resources directly from user-space in the non-contended case). Therefore, the switch-(primary, secondary) tags denote either services which will always switch the caller to the mode mentioned, or might have to do so, depending on the context. The absence of such tag indicates that such services can complete in either modes and as such will entail no switch.
Chapter 2

Deprecated List

Global COMPAT__rt_alarm_create (RT_ALARM *alarm, const char *name)
This is a compatibility service from the Transition Kit.

Global COMPAT__rt_event_clear (RT_EVENT *event, unsigned long mask, unsigned long *mask_r)
This is a compatibility service from the Transition Kit.

Global COMPAT__rt_event_create (RT_EVENT *event, const char *name, unsigned long ivalue, int mode)
This is a compatibility service from the Transition Kit.

Global COMPAT__rt_event_signal (RT_EVENT *event, unsigned long mask)
This is a compatibility service from the Transition Kit.

Global COMPAT__rt_task_create (RT_TASK *task, const char *name, int stksize, int prio, int mode)
This is a compatibility service from the Transition Kit.

Global COMPAT__rt_task_set_periodic (RT_TASK *task, RTIME idate, RTIME period)
This is a compatibility service from the Transition Kit.

Module IOCTLs
Passing struct ifreq as a request descriptor for CAN IOCTLs is still accepted for backward compatibility, however it is recommended to switch to struct can_ifreq at the first opportunity.

Module IOCTLs
Passing struct ifreq as a request descriptor for CAN IOCTLs is still accepted for backward compatibility, however it is recommended to switch to struct can_ifreq at the first opportunity.

Global pthread_make_periodic_np (pthread_t thread, struct timespec *starttp, struct timespec *periodtp)
This service is a non-portable extension of the Xenomai 2.x POSIX interface, not available with Xenomai 3.x. Instead, Cobalt-based applications should set up a periodic timer using the timer_create(), timer_settime() call pair, then wait for release points via sigwaitinfo(). Overruns can be detected by looking at the siginfo.si_overrun field. Alternatively, applications may obtain a file descriptor referring to a Cobalt timer via the timerfd() call, and read() from it to wait for timeouts.
Global `pthread_wait_np` (unsigned long *overruns_r)

This service is a non-portable extension of the Xenomai 2.x POSIX interface, not available with Xenomai 3.x. Instead, Cobalt-based applications should set up a periodic timer using the `timer_create()`, `timer_settime()` call pair, then wait for release points via `sigwaitinfo()`. Overruns can be detected by looking at the `siginfo.si_overrun` field. Alternatively, applications may obtain a file descriptor referring to a Cobalt timer via the `timerfd()` call, and `read()` from it to wait for timeouts.

Global `rt_alarm_wait` (RT_ALARM *alarm)

This is a compatibility service from the Transition Kit.

Global `RTDM_EXECUTE_ATOMICALLY` (code_block)

This construct will be phased out in Xenomai 3.0. Please use `rtdm_waitqueue` services instead.

Global `rtdm_task_sleep_until` (nanosecs_abs_t wakeup_time)

Use `rtdm_task_sleep_abs` instead!
# Chapter 3

## Module Index

### 3.1 Modules

Here is a list of all modules:

<table>
<thead>
<tr>
<th>Module</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTDM</td>
<td>67</td>
</tr>
<tr>
<td>RTDM User API</td>
<td>70</td>
</tr>
<tr>
<td>Driver programming interface</td>
<td>136</td>
</tr>
<tr>
<td>Driver to driver services</td>
<td>137</td>
</tr>
<tr>
<td>Device Registration Services</td>
<td>158</td>
</tr>
<tr>
<td>Clock Services</td>
<td>169</td>
</tr>
<tr>
<td>Task Services</td>
<td>171</td>
</tr>
<tr>
<td>Timer Services</td>
<td>182</td>
</tr>
<tr>
<td>Synchronisation Services</td>
<td>187</td>
</tr>
<tr>
<td>Big dual kernel lock</td>
<td>24</td>
</tr>
<tr>
<td>Spinlock with preemption deactivation</td>
<td>27</td>
</tr>
<tr>
<td>Event Services</td>
<td>201</td>
</tr>
<tr>
<td>Semaphore Services</td>
<td>207</td>
</tr>
<tr>
<td>Mutex services</td>
<td>211</td>
</tr>
<tr>
<td>Interrupt Management Services</td>
<td>215</td>
</tr>
<tr>
<td>Non-Real-Time Signalling Services</td>
<td>221</td>
</tr>
<tr>
<td>Utility Services</td>
<td>224</td>
</tr>
<tr>
<td>Device Profiles</td>
<td>155</td>
</tr>
<tr>
<td>User-space driver core</td>
<td>31</td>
</tr>
<tr>
<td>CAN Devices</td>
<td>43</td>
</tr>
<tr>
<td>Serial Devices</td>
<td>71</td>
</tr>
<tr>
<td>Testing Devices</td>
<td>73</td>
</tr>
<tr>
<td>Real-time IPC</td>
<td>74</td>
</tr>
<tr>
<td>Cobalt</td>
<td>115</td>
</tr>
<tr>
<td>Cobalt kernel</td>
<td>116</td>
</tr>
<tr>
<td>Asynchronous Procedure Calls</td>
<td>93</td>
</tr>
<tr>
<td>In-kernel arithmetics</td>
<td>96</td>
</tr>
<tr>
<td>Buffer descriptor</td>
<td>97</td>
</tr>
<tr>
<td>Clock services</td>
<td>107</td>
</tr>
<tr>
<td>Debugging services</td>
<td>110</td>
</tr>
<tr>
<td>Dynamic memory allocation services</td>
<td>111</td>
</tr>
<tr>
<td>Interrupt management</td>
<td>119</td>
</tr>
<tr>
<td>Locking services</td>
<td>125</td>
</tr>
<tr>
<td>Lightweight key-to-object mapping service</td>
<td>127</td>
</tr>
<tr>
<td>Module</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------</td>
<td>------</td>
</tr>
<tr>
<td>Registry services</td>
<td>132</td>
</tr>
<tr>
<td>Thread scheduling control</td>
<td>239</td>
</tr>
<tr>
<td>SCHED_QUOTA scheduling policy</td>
<td>238</td>
</tr>
<tr>
<td>Synchronous I/O multiplexing</td>
<td>241</td>
</tr>
<tr>
<td>Thread synchronization services</td>
<td>246</td>
</tr>
<tr>
<td>Thread services</td>
<td>255</td>
</tr>
<tr>
<td>Thread state flags</td>
<td>39</td>
</tr>
<tr>
<td>Thread information flags</td>
<td>42</td>
</tr>
<tr>
<td>Timer services</td>
<td>271</td>
</tr>
<tr>
<td>Virtual file services</td>
<td>280</td>
</tr>
<tr>
<td>Analysis framework</td>
<td>288</td>
</tr>
<tr>
<td>Driver API</td>
<td>289</td>
</tr>
<tr>
<td>Channels and ranges</td>
<td>21</td>
</tr>
<tr>
<td>Driver management services</td>
<td>290</td>
</tr>
<tr>
<td>Subdevice management services</td>
<td>292</td>
</tr>
<tr>
<td>Buffer management services</td>
<td>296</td>
</tr>
<tr>
<td>Interrupt management services</td>
<td>306</td>
</tr>
<tr>
<td>Misc services</td>
<td>309</td>
</tr>
<tr>
<td>Analysis user API</td>
<td>429</td>
</tr>
<tr>
<td>Level 1 API</td>
<td>419</td>
</tr>
<tr>
<td>Asynchronous acquisition API</td>
<td>391</td>
</tr>
<tr>
<td>Descriptor API</td>
<td>406</td>
</tr>
<tr>
<td>Synchronous acquisition API</td>
<td>420</td>
</tr>
<tr>
<td>Level 2 API</td>
<td>424</td>
</tr>
<tr>
<td>Asynchronous acquisition API</td>
<td>399</td>
</tr>
<tr>
<td>Software calibration API</td>
<td>401</td>
</tr>
<tr>
<td>Math API</td>
<td>410</td>
</tr>
<tr>
<td>Range / conversion API</td>
<td>413</td>
</tr>
<tr>
<td>Synchronous acquisition API</td>
<td>425</td>
</tr>
<tr>
<td>Level 0 API</td>
<td>430</td>
</tr>
<tr>
<td>Descriptor Syscall API</td>
<td>404</td>
</tr>
<tr>
<td>Basic Syscall API</td>
<td>431</td>
</tr>
<tr>
<td>Attach / detach Syscall API</td>
<td>434</td>
</tr>
<tr>
<td>POSIX interface</td>
<td>331</td>
</tr>
<tr>
<td>Clocks and timers</td>
<td>310</td>
</tr>
<tr>
<td>Condition variables</td>
<td>320</td>
</tr>
<tr>
<td>Message queues</td>
<td>333</td>
</tr>
<tr>
<td>Mutual exclusion</td>
<td>344</td>
</tr>
<tr>
<td>Process scheduling</td>
<td>357</td>
</tr>
<tr>
<td>Semaphores</td>
<td>368</td>
</tr>
<tr>
<td>Thread management</td>
<td>376</td>
</tr>
<tr>
<td>Scheduling management</td>
<td>382</td>
</tr>
<tr>
<td>Smokey API</td>
<td>387</td>
</tr>
<tr>
<td>Alchemy API</td>
<td>479</td>
</tr>
<tr>
<td>Alarm services</td>
<td>437</td>
</tr>
<tr>
<td>Buffer services</td>
<td>442</td>
</tr>
<tr>
<td>Condition variable services</td>
<td>453</td>
</tr>
<tr>
<td>Event flag group services</td>
<td>461</td>
</tr>
<tr>
<td>Heap management services</td>
<td>471</td>
</tr>
<tr>
<td>Mutex services</td>
<td>481</td>
</tr>
<tr>
<td>Message pipe services</td>
<td>488</td>
</tr>
<tr>
<td>Message queue services</td>
<td>497</td>
</tr>
<tr>
<td>Semaphore services</td>
<td>510</td>
</tr>
<tr>
<td>Task management services</td>
<td>519</td>
</tr>
<tr>
<td>Timer management services</td>
<td>547</td>
</tr>
<tr>
<td>VxWorks® emulator</td>
<td>551</td>
</tr>
<tr>
<td>pSOS® emulator</td>
<td>552</td>
</tr>
<tr>
<td>Transition Kit</td>
<td>553</td>
</tr>
</tbody>
</table>

Generated by Doxygen
Chapter 4

Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

- **a4l_channel**: Structure describing some channel's characteristics
- **a4l_channels_desc**: Structure describing a channels set
- **a4l_cmd_desc**: Structure describing the asynchronous instruction
- **a4l_descriptor**: Structure containing device-information useful to users
- **a4l_driver**: Structure containing driver declaration data
- **a4l_instruction**: Structure describing the synchronous instruction
- **a4l_instruction_list**: Structure describing the list of synchronous instructions
- **a4l_range**: Structure describing a (unique) range
- **a4l_subdevice**: Structure describing the subdevice
- **atomic_t**: Structure containing device-information useful to users
- **can_bittime**: Custom CAN bit-time definition
- **can_bittime_btr**: Hardware-specific BTR bit-times
- **can_bittime_std**: Standard bit-time parameters according to Bosch
- **can_filter**: Filter for reception of CAN messages
- **can_frame**: Raw CAN frame
- **can_ifreq**: CAN interface request descriptor
- **macb_dma_desc**: Hardware DMA descriptor
- **macb_tx_skb**: Data about an skb which is being transmitted
<table>
<thead>
<tr>
<th>Data Structure Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>RT_ALARM_INFO</td>
</tr>
<tr>
<td>Alarm status descriptor</td>
</tr>
<tr>
<td>RT_BUFFER_INFO</td>
</tr>
<tr>
<td>Buffer status descriptor</td>
</tr>
<tr>
<td>RT_COND_INFO</td>
</tr>
<tr>
<td>Condition variable status descriptor</td>
</tr>
<tr>
<td>RT_EVENT_INFO</td>
</tr>
<tr>
<td>Event status descriptor</td>
</tr>
<tr>
<td>RT_HEAP_INFO</td>
</tr>
<tr>
<td>Heap status descriptor</td>
</tr>
<tr>
<td>RT_MUTEX_INFO</td>
</tr>
<tr>
<td>Mutex status descriptor</td>
</tr>
<tr>
<td>RT_QUEUE_INFO</td>
</tr>
<tr>
<td>Queue status descriptor</td>
</tr>
<tr>
<td>RT_SEM_INFO</td>
</tr>
<tr>
<td>Semaphore status descriptor</td>
</tr>
<tr>
<td>RT_TASK_INFO</td>
</tr>
<tr>
<td>Task status descriptor</td>
</tr>
<tr>
<td>rt_timer_info</td>
</tr>
<tr>
<td>Timer status descriptor</td>
</tr>
<tr>
<td>rtdm_dev_context</td>
</tr>
<tr>
<td>Device context</td>
</tr>
<tr>
<td>rtdm_device</td>
</tr>
<tr>
<td>RTDM device</td>
</tr>
<tr>
<td>rtdm_device_info</td>
</tr>
<tr>
<td>Device information</td>
</tr>
<tr>
<td>rtdm_driver</td>
</tr>
<tr>
<td>RTDM driver</td>
</tr>
<tr>
<td>rtdm_fd_ops</td>
</tr>
<tr>
<td>RTDM file operation descriptor</td>
</tr>
<tr>
<td>rtdm_sm_ops</td>
</tr>
<tr>
<td>RTDM state management handlers</td>
</tr>
<tr>
<td>rtpc_port_label</td>
</tr>
<tr>
<td>Port label information structure</td>
</tr>
<tr>
<td>rtser_config</td>
</tr>
<tr>
<td>Serial label configuration</td>
</tr>
<tr>
<td>rtser_event</td>
</tr>
<tr>
<td>Additional information about serial device events</td>
</tr>
<tr>
<td>rtser_status</td>
</tr>
<tr>
<td>Serial device status</td>
</tr>
<tr>
<td>sockaddr_can</td>
</tr>
<tr>
<td>Socket address structure for the CAN address family</td>
</tr>
<tr>
<td>sockaddr_ipc</td>
</tr>
<tr>
<td>Socket address structure for the RTIPC address family</td>
</tr>
<tr>
<td>udd_device</td>
</tr>
<tr>
<td>Socket address structure for the UDD device</td>
</tr>
<tr>
<td>udd_memregion</td>
</tr>
<tr>
<td>Reserved to the UDD core</td>
</tr>
<tr>
<td>udd_device::udd_reserved</td>
</tr>
<tr>
<td>Reserved to the UDD core</td>
</tr>
<tr>
<td>udd_signotify</td>
</tr>
<tr>
<td>UDD event notification descriptor</td>
</tr>
<tr>
<td>xnheap::xnbucket</td>
</tr>
<tr>
<td>Log2 bucket list</td>
</tr>
<tr>
<td>xnsched</td>
</tr>
<tr>
<td>Scheduling information structure</td>
</tr>
<tr>
<td>xnvfile_lock_ops</td>
</tr>
<tr>
<td>Vfile locking operations</td>
</tr>
</tbody>
</table>
### 4.1 Data Structures

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>xnfile_regular_iterator</td>
<td>Regular vfile iterator</td>
<td>613</td>
</tr>
<tr>
<td>xnfile_regular_ops</td>
<td>Regular vfile operation descriptor</td>
<td>614</td>
</tr>
<tr>
<td>xnfile_rev_tag</td>
<td>Snapshot revision tag</td>
<td>619</td>
</tr>
<tr>
<td>xnfile_snapshot</td>
<td>Snapshot vfile descriptor</td>
<td>619</td>
</tr>
<tr>
<td>xnfile_snapshot_iterator</td>
<td>Snapshot-driven vfile iterator</td>
<td>620</td>
</tr>
<tr>
<td>xnfile_snapshot_ops</td>
<td>Snapshot vfile operation descriptor</td>
<td>622</td>
</tr>
</tbody>
</table>
5.1 File List

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

- include/alchemy/alarm.h
- include/alchemy/buffer.h
- include/alchemy/compat.h
- include/alchemy/cond.h
- include/alchemy/event.h
- include/alchemy/heap.h
- include/alchemy/mutex.h
- include/alchemy/pipe.h
- include/alchemy/queue.h
- include/alchemy/sem.h
- include/alchemy/task.h
- include/alchemy/timer.h
- include/boilerplate/ancillaries.h
- include/boilerplate/atomic.h
- include/boilerplate/compiler.h
- include/boilerplate/debug.h
- include/boilerplate/hash.h
- include/boilerplate/libc.h
- include/boilerplate/list.h
- include/boilerplate/lock.h
- include/boilerplate/obstack.h
- include/boilerplate/private-list.h
- include/boilerplate/scope.h
- include/boilerplate/setup.h
- include/boilerplate/shared-list.h
- include/boilerplate/time.h
- include/boilerplate/tunables.h
- include/cobalt/arith.h
- include/cobalt/fcntl.h
- include/cobalt/mqueue.h
- include/cobalt/pthread.h
- include/cobalt/sched.h
- include/cobalt/semaphore.h
- include/cobalt/signal.h
- include/cobalt/stdio.h
### File Index

<table>
<thead>
<tr>
<th>File Path</th>
<th>Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>include/cobalt/stdlib.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/syslog.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/ticks.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/time.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/trace.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/tunables.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/unistd.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/wrappers.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/boilerplate/limits.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/boilerplate/sched.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/boilerplate/signal.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/boilerplate/trace.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/boilerplate/wrappers.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/kernel/ancillaries.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/kernel/apc.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/kernel/arith.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/kernel/assert.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/kernel/bufd.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/kernel/clock.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/kernel/compat.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/kernel/heap.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/kernel/init.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/kernel/intr.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/kernel/list.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/kernel/map.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/kernel/pipe.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/kernel/ppd.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/kernel/registry.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/kernel/sched-idle.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/kernel/sched-quota.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/kernel/sched-rt.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/kernel/sched-sporadic.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/kernel/sched-tp.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/kernel/sched-weak.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/kernel/sched.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/kernel/schedparam.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/kernel/schedqueue.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/kernel/select.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/kernel/stat.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/kernel/synch.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/kernel/thread.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/kernel/timer.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/kernel/trace.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/kernel/tree.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/kernel/vdso.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/kernel/vfile.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/kernel/rtdm/autotune.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/kernel/rtdm/can.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/kernel/rtdm/cobalt.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/kernel/rtdm/compat.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/kernel/rtdm/driver.h</td>
<td>??</td>
</tr>
</tbody>
</table>

Real-Time Driver Model for Xenomai, driver API header  

include/cobalt/kernel/rtdm/fd.h  
include/cobalt/kernel/rtdm/ipc.h  
include/cobalt/kernel/rtdm/rdm.h  
include/cobalt/kernel/rtdm/serial.h  
include/cobalt/kernel/rtdm/testing.h  

633

650
# 5.1 File List

<table>
<thead>
<tr>
<th>Path</th>
<th>Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>include/cobalt/kernel/rtdm/udd.h</td>
<td>672</td>
</tr>
<tr>
<td>Copyright (C) 2014 Philippe Gerum <a href="mailto:rpm@xenomai.org">rpm@xenomai.org</a></td>
<td></td>
</tr>
<tr>
<td>include/cobalt/kernel/rtdm/analogy/channel_range.h</td>
<td>629</td>
</tr>
<tr>
<td>Analogy for Linux, channel, range related features</td>
<td></td>
</tr>
<tr>
<td>include/cobalt/kernel/rtdm/analogy/command.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/kernel/rtdm/analogy/context.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/kernel/rtdm/analogy/device.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/kernel/rtdm/analogy/driver.h</td>
<td></td>
</tr>
<tr>
<td>Analogy for Linux, driver facilities</td>
<td>632</td>
</tr>
<tr>
<td>include/cobalt/kernel/rtdm/analogy/instruction.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/kernel/rtdm/analogy/rtdm_helpers.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/kernel/rtdm/analogy/subdevice.h</td>
<td>641</td>
</tr>
<tr>
<td>Analogy for Linux, subdevice related features</td>
<td></td>
</tr>
<tr>
<td>include/cobalt/kernel/rtdm/analogy/transfer.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/sys/cobalt.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/sys/ioctl.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/sys/mman.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/sys/select.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/sys/socket.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/sys/time.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/sys/timerfd.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/uapi/cond.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/uapi/corectl.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/uapi/event.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/uapi/monitor.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/uapi/mutex.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/uapi/sched.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/uapi/sem.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/uapi/signal.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/uapi/syscall.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/uapi/synch.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/uapi/thread.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/uapi/time.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/uapi/asm-generic/arith.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/uapi/asm-generic/features.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/uapi/asm-generic/syscall.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/uapi/kernel/heap.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/uapi/kernel/limits.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/uapi/kernel/pipe.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/uapi/kernel/synch.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/uapi/kernel/thread.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/uapi/kernel/trace.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/uapi/kernel/types.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/uapi/kernel/urw.h</td>
<td>??</td>
</tr>
<tr>
<td>include/cobalt/uapi/kernel/vdso.h</td>
<td>??</td>
</tr>
<tr>
<td>include/copperplate/clockobj.h</td>
<td>??</td>
</tr>
<tr>
<td>include/copperplate/cluster.h</td>
<td>??</td>
</tr>
<tr>
<td>include/copperplate/debug.h</td>
<td>??</td>
</tr>
<tr>
<td>include/copperplate/eventobj.h</td>
<td>??</td>
</tr>
<tr>
<td>include/copperplate/heapobj.h</td>
<td>??</td>
</tr>
<tr>
<td>include/copperplate/reference.h</td>
<td>??</td>
</tr>
<tr>
<td>include/copperplate/registry-obstack.h</td>
<td>??</td>
</tr>
<tr>
<td>include/copperplate/registry.h</td>
<td>??</td>
</tr>
<tr>
<td>include/copperplate/semobj.h</td>
<td>??</td>
</tr>
<tr>
<td>include/copperplate/syncoobj.h</td>
<td>??</td>
</tr>
<tr>
<td>include/copperplate/threadobj.h</td>
<td>??</td>
</tr>
<tr>
<td>include/copperplate/timerobj.h</td>
<td>??</td>
</tr>
<tr>
<td>include/copperplate/traceobj.h</td>
<td>??</td>
</tr>
</tbody>
</table>

Generated by Doxygen
include/copperplate/tunables.h .................................................. ??
include/copperplate/wrappers.h ................................................ ??
include/mercury/pthread.h ......................................................... ??
include/mercury/boilerplate/limits.h ........................................ ??
include/mercury/boilerplate/sched.h ......................................... ??
include/mercury/boilerplate/signal.h ....................................... ??
include/mercury/boilerplate/trace.h ....................................... ??
include/mercury/boilerplate/wrappers.h ...................................... ??
include/psos/psos.h ............................................................. ??
include/psos/tunables.h .......................................................... ??
include/rtdm/analogy.h
  Analogy for Linux, library facilities ........................................ 675
include/rtdm/autotune.h ......................................................... ??
include/rtdm/can.h .............................................................. 642
include/rtdm/gpio.h .............................................................. ??
include/rtdm/ipc.h .............................................................. ??
include/rtdm/rtdm.h .............................................................. 658
include/rtdm/serial.h
  Real-Time Driver Model for Xenomai, serial device profile header .... 660
include/rtdm/spi.h ............................................................... ??
include/rtdm/testing.h
  Real-Time Driver Model for Xenomai, testing device profile header .... 669
include/rtdm/udd.h ............................................................... ??
include/rtdm/uapi/analogy.h
  Analogy for Linux, UAPI bits .................................................. 676
include/rtdm/uapi/autotune.h .................................................. ??
include/rtdm/uapi/can.h ........................................................ 643
include/rtdm/uapi/gpio.h ........................................................ ??
include/rtdm/uapi/ipc.h ........................................................ ??
This file is part of the Xenomai project ....................................... 654
include/rtdm/uapi/rtdm.h
  Real-Time Driver Model for Xenomai, user API header .................... 658
include/rtdm/uapi/serial.h
  Real-Time Driver Model for Xenomai, serial device profile header .... 661
include/rtdm/uapi/spi.h ........................................................ ??
include/rtdm/uapi/testing.h
  Real-Time Driver Model for Xenomai, testing device profile header .... 670
include/rtdm/uapi/udd.h
This file is part of the Xenomai project ....................................... 674
include/smokey/smokey.h ......................................................... ??
include/trank/rtdk.h .............................................................. ??
include/trank/trank.h .............................................................. ??
include/trank/native/alarm.h ................................................... ??
include/trank/native/buffer.h ................................................... ??
include/trank/native/cond.h ..................................................... ??
include/trank/native/event.h ..................................................... ??
include/trank/native/heap.h ..................................................... ??
include/trank/native/misc.h ..................................................... ??
include/trank/native/mutex.h ..................................................... ??
include/trank/native/pipe.h ..................................................... ??
include/trank/native/queue.h ................................................... ??
include/trank/native/sem.h ...................................................... ??
include/trank/native/task.h ..................................................... ??
include/trank/native/timer.h .................................................... ??
include/trank/native/types.h .................................................... ??
include/trank/posix/pthread.h ................................................... ??
include/trank/rtdm/rtcan.h ...................................................... ??

Generated by Doxygen
include/trank/tdm/tdm.h ................................................. ??
include/trank/tdm/rtipc.h ............................................. ??
include/trank/tdm/rtserial.h ......................................... ??
include/trank/tdm/rttesting.h ....................................... ??
include/vxworks/errnoLib.h .......................................... ??
include/vxworks/intLib.h ............................................ ??
include/vxworks/kernLib.h ........................................... ??
include/vxworks/lstLib.h ............................................ ??
include/vxworks/memPartLib.h ........................................ ??
include/vxworks/msgQLib.h ........................................... ??
include/vxworks/rngLib.h ............................................. ??
include/vxworks/semLib.h ............................................ ??
include/vxworks/sysLib.h ............................................ ??
include/vxworks/taskHookLib.h ..................................... ??
include/vxworks/taskInfo.h .......................................... ??
include/vxworks/taskLib.h ........................................... ??
include/vxworks/tickLib.h ........................................... ??
include/vxworks/types.h ............................................. ??
include/vxworks/wdLib.h ............................................. ??
include/xenomai/init.h .............................................. ??
include/xenomai/tunables.h .......................................... ??
include/xenomai/version.h .......................................... ??
kernel/cobalt/debug.h ................................................ ??
kern...
<table>
<thead>
<tr>
<th>Path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kernel/cobalt/arch/powerpc/include/asm/xenomai/uapi/features.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/cobalt/arch/powerpc/include/asm/xenomai/uapi/fptest.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/cobalt/arch/powerpc/include/asm/xenomai/uapi/syscall.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/cobalt/arch/x86/include/asm/xenomai/c1e.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/cobalt/arch/x86/include/asm/xenomai/calibration.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/cobalt/arch/x86/include/asm/xenomai/features.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/cobalt/arch/x86/include/asm/xenomai/fptest.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/cobalt/arch/x86/include/asm/xenomai/machine.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/cobalt/arch/x86/include/asm/xenomai/sgi.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/cobalt/arch/x86/include/asm/xenomai/syscall32-table.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/cobalt/arch/x86/include/asm/xenomai/syscall32.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/cobalt/arch/x86/include/asm/xenomai/thread.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/cobalt/arch/x86/include/asm/xenomai/.wrappers.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/cobalt/arch/x86/include/asm/xenomai/arith.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/cobalt/arch/x86/include/asm/xenomai/uapi/features.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/cobalt/arch/x86/include/asm/xenomai/uapi/syscall.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/cobalt/include/asm-generic/xenomai/machine.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/cobalt/include/asm-generic/xenomai/mayday.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/cobalt/include/asm-generic/xenomai/pci_ids.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/cobalt/include/asm-generic/xenomai/syscall.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/cobalt/include/asm-generic/xenomai/syscall32.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/cobalt/include/asm-generic/xenomai/thread.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/cobalt/include/asm-generic/xenomai/wrappers.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/cobalt/include/ipipe/thread_info.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/cobalt/posix/clock.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/cobalt/posix/cond.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/cobalt/posix/corecl.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/cobalt/posix/event.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/cobalt/posix/extension.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/cobalt/posix/extension.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/cobalt/posix/internal.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/cobalt/posix/io.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/cobalt/posix/memory.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/cobalt/posix/monitor.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/cobalt/posix/mqueue.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/cobalt/posix/mutex.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/cobalt/posix/process.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/cobalt/posix/sched.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/cobalt/posix/sem.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/cobalt/posix/signal.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/cobalt/posix/syscall.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/cobalt/posix/syscall32.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/cobalt/posix/thread.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/cobalt/posix/timer.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/cobalt/posix/timerfd.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/cobalt/rdm/rdm_internal.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/cobalt/trace/cobalt-core.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/cobalt/trace/cobalt-posix.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/cobalt/trace/cobalt-rdm.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/analogy/pro.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/analogy/intel/6255.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/analogy/national_instruments/mite.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/analogy/national_instruments/ni_mio.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/analogy/national_instruments/ni_stc.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/analogy/national_instruments/ni_tio.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/can/rtcan_dev.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/can/rtcan_internal.h</td>
<td>??</td>
</tr>
</tbody>
</table>
5.1 File List

- kernel/drivers/can/rtcan_list.h
- kernel/drivers/can/rtcan_raw.h
- kernel/drivers/can/rtcan_socket.h
- kernel/drivers/can/rtcan_version.h
- kernel/drivers/can/mscan/rtcan_mscan.h
- kernel/drivers/can/mscan/rtcan_mscan_regs.h
- kernel/drivers/can/sja1000/rtcan_sja1000.h
- kernel/drivers/can/sja1000/rtcan_sja1000_regs.h
- kernel/drivers/gpio/gpio-core.h
- kernel/drivers/ipc/internal.h
- kernel/drivers/net/drivers/rt_at91Ether.h
- kernel/drivers/net/drivers/rt_eth1394.h
- kernel/drivers/net/drivers/rt_macb.h
- kernel/drivers/net/drivers/rt_smc91111.h
- kernel/drivers/net/drivers/e1000/e1000.h
- kernel/drivers/net/drivers/e1000/e1000_osdep.h
- kernel/drivers/net/drivers/e1000/kcompat.h
- kernel/drivers/net/drivers/e1000/defines.h
- kernel/drivers/net/drivers/e1000/e1000.hw.h
- kernel/drivers/net/drivers/e1000/hw.h
- kernel/drivers/net/drivers/experimental/e1000/e1000.h
- kernel/drivers/net/drivers/experimental/e1000/800003es2lan.h
- kernel/drivers/net/drivers/experimental/e1000/82541.h
- kernel/drivers/net/drivers/experimental/e1000/82543.h
- kernel/drivers/net/drivers/experimental/e1000/82571.h
- kernel/drivers/net/drivers/experimental/e1000/82573.h
- kernel/drivers/net/drivers/experimental/e1000_api.h
- kernel/drivers/net/drivers/experimental/e1000/defines.h
- kernel/drivers/net/drivers/experimental/e1000/hw.h
- kernel/drivers/net/drivers/experimental/e1000/ich8lan.h
- kernel/drivers/net/drivers/experimental/e1000/mac.h
- kernel/drivers/net/drivers/experimental/e1000/manage.h
- kernel/drivers/net/drivers/experimental/e1000/nvm.h
- kernel/drivers/net/drivers/experimental/e1000/osdep.h
- kernel/drivers/net/drivers/experimental/e1000/phy.h
- kernel/drivers/net/drivers/experimental/e1000/regs.h
- kernel/drivers/net/drivers/experimental/rtcan.kcompat
- kernel/drivers/net/drivers/experimental/rt2500/rt2500pci.h
- kernel/drivers/net/drivers/experimental/rt2500/rt2x00.h
- kernel/drivers/net/drivers/igb/e1000/82575.h
- kernel/drivers/net/drivers/igb/e1000/defines.h
- kernel/drivers/net/drivers/igb/e1000/hw.h
- kernel/drivers/net/drivers/igb/e1000/i210.h
- kernel/drivers/net/drivers/igb/e1000/mac.h
- kernel/drivers/net/drivers/igb/e1000/mbx.h
- kernel/drivers/net/drivers/igb/e1000/nvm.h
- kernel/drivers/net/drivers/igb/phy.h
- kernel/drivers/net/drivers/igb/regs.h
- kernel/drivers/net/drivers/igb/gb.h
- kernel/drivers/net/drivers/mpc52xx_fec/rt_mpc52xx_fec.h
- kernel/drivers/net/drivers/tulip/tulip.h
- kernel/drivers/net/stack/include/ipv4_chrdev.h
- kernel/drivers/net/stack/include/nomac_chrdev.h
- kernel/drivers/net/stack/include/rtev_chrdev.h
- kernel/drivers/net/stack/include/tdev.h
- kernel/drivers/net/stack/include/tdev_mgr.h
- kernel/drivers/net/stack/include/rmac.h

Generated by Doxygen
<table>
<thead>
<tr>
<th>Directory</th>
<th>File Name</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>kernel/drivers/net/stack/include</td>
<td>rnet.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/net/stack/include</td>
<td>rnet_chrdev.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/net/stack/include</td>
<td>rnet_internal.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/net/stack/include</td>
<td>rnet_ioctl.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/net/stack/include</td>
<td>rnet_port.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/net/stack/include</td>
<td>rnet_rtpc.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/net/stack/include</td>
<td>rnet_socket.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/net/stack/include</td>
<td>rtskb.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/net/stack/include</td>
<td>rtspan.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/net/stack/include</td>
<td>rtwlan.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/net/stack/include</td>
<td>stack_mgr.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/net/stack/include</td>
<td>tdma_chrdev.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/net/stack/include</td>
<td>ethernet/eth.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/net/stack/include</td>
<td>ipv4/af_inet.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/net/stack/include</td>
<td>ipv4/arp.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/net/stack/include</td>
<td>ipv4/icmp.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/net/stack/include</td>
<td>ipv4/ip_fragment.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/net/stack/include</td>
<td>ipv4/ip_input.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/net/stack/include</td>
<td>ipv4/ip_output.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/net/stack/include</td>
<td>ipv4/ip_sock.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/net/stack/include</td>
<td>ipv4/protocol.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/net/stack/include</td>
<td>ipv4/route.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/net/stack/include</td>
<td>ipv4/tcp.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/net/stack/include</td>
<td>ipv4/udp.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/net/stack/include</td>
<td>rtcfg/rtcfg.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/net/stack/include</td>
<td>rtcfg/rtcfg_client_event.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/net/stack/include</td>
<td>rtcfg/rtcfg_conn_event.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/net/stack/include</td>
<td>rtcfg/rtcfg_event.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/net/stack/include</td>
<td>rtcfg/rtcfg_file.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/net/stack/include</td>
<td>rtcfg/rtcfg_frame.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/net/stack/include</td>
<td>rtcfg/rtcfg_ioct1.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/net/stack/include</td>
<td>rtcfg/rtcfg_ioctl.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/net/stack/include</td>
<td>rtmac/rtmac_disc.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/net/stack/include</td>
<td>rtmac/rtmac_proto.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/net/stack/include</td>
<td>rtmac/rtmac_vnic.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/net/stack/include</td>
<td>rtmac/nomac/nomac.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/net/stack/include</td>
<td>rtmac/nomac/nomac_dev.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/net/stack/include</td>
<td>rtmac/nomac/nomac_ioctl.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/net/stack/include</td>
<td>rtmac/nomac/nomac_proto.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/net/stack/include</td>
<td>rtmac/tdma/tdma.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/net/stack/include</td>
<td>rtmac/tdma/tdma_dev.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/net/stack/include</td>
<td>rtmac/tdma/tdma_ioctl.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/net/stack/include</td>
<td>rtmac/tdma/tdma_proto.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/net/stack/include</td>
<td>rtmac/tdma/tdma_worker.h</td>
<td>??</td>
</tr>
<tr>
<td>kernel/drivers/net/stack/include</td>
<td>ipv4/tcp/timerwheel.h</td>
<td>??</td>
</tr>
</tbody>
</table>
lib/alchemy/current.h
lib/alchemy/internal.h
lib/alchemy/ummm.h
lib/alchemy/arch/arm/include/asm/xenomai/features.h
lib/alchemy/arch/arm/include/asm/xenomai/syscall.h
lib/alchemy/arch/blackfin/include/asm/xenomai/features.h
lib/alchemy/arch/blackfin/include/asm/xenomai/syscall.h
lib/alchemy/arch/powerpc/include/asm/xenomai/features.h
lib/alchemy/arch/powerpc/include/asm/xenomai/syscall.h
lib/alchemy/arch/x86/include/asm/xenomai/features.h
lib/alchemy/arch/x86/include/asm/xenomai/syscall.h
lib/copperplate/internal.h
lib/copperplate/regd/sysregfs.h
lib/psos/internal.h
lib/psos/pt.h
lib/psos/queue.h
lib/psos/reference.h
lib/psos/rm.h
lib/psos/sem.h
lib/psos/task.h
lib/psos/tm.h
lib/trank/internal.h
lib/vxworks/memPartLib.h
lib/vxworks/msgQLib.h
lib/vxworks/reference.h
lib/vxworks/rngLib.h
lib/vxworks/semLib.h
lib/vxworks/taskHookLib.h
lib/vxworks/taskLib.h ...................................................... ??
lib/vxworks/tickLib.h ...................................................... ??
lib/vxworks/wdLib.h ...................................................... ??
Chapter 6
Module Documentation

6.1 Channels and ranges

Channels.

Collaboration diagram for Channels and ranges:

Data Structures

- `struct a4l_channel`
  
  Structure describing some channel's characteristics.

- `struct a4l_channels_desc`
  
  Structure describing a channels set.

- `struct a4l_range`
  
  Structure describing a (unique) range.

Macros

- `#define A4L_CHAN_GLOBAL 0x10`
  
  Internal use flag (must not be used by driver developer)

- `#define A4L_RNG_GLOBAL 0x8`
  
  Internal use flag (must not be used by driver developer)

- `#define RANGE(x, y)`
  
  Macro to declare a (unique) range with no unit defined.

- `#define RANGE_V(x, y)`
Macro to declare a (unique) range in Volt.

- `#define RANGE_mA(x, y)`

Macro to declare a (unique) range in milliAmpere.

- `#define RANGE_ext(x, y)`

Macro to declare a (unique) range in some external reference.

- `#define A4L_RNG_GLOBAL_RNGDESC 0`
  Constant to define a ranges descriptor as global (inter-channel)
- `#define A4L_RNG_PERCHAN_RNGDESC 1`
  Constant to define a ranges descriptor as specific for a channel.
- `#define RNG_GLOBAL(x)`
  Macro to declare a ranges global descriptor in one line.

Channel reference

Flags to define the channel's reference

- `#define A4L_CHAN_AREF_GROUND 0x1`
  Ground reference.
- `#define A4L_CHAN_AREF_COMMON 0x2`
  Common reference.
- `#define A4L_CHAN_AREF_DIFF 0x4`
  Differential reference.
- `#define A4L_CHAN_AREF_OTHER 0x8`
  Misc reference.

Channels declaration mode

Constant to define whether the channels in a descriptor are identical

- `#define A4L_CHAN_GLOBAL_CHANDESC 0`
  Global declaration, the set contains channels with similar characteristics.
- `#define A4L_CHAN_PERCHAN_CHANDESC 1`
  Per channel declaration, the descriptor gathers different channels.

6.1.1 Detailed Description

Channels.

According to the Analogy nomenclature, the channel is the elementary acquisition entity. One channel is supposed to acquire one data at a time. A channel can be:

- an analog input or an analog output;
- a digital input or a digital output;

Channels are defined by their type and by some other characteristics like:
6.1 Channels and ranges

- their resolutions for analog channels (which usually ranges from 8 to 32 bits);
- their references;

Such parameters must be declared for each channel composing a subdevice. The structure `a4l_channel` (struct `a4l_channel`) is used to define one channel.

Another structure named `a4l_channels_desc` (struct `a4l_channels_desc`) gathers all channels for a specific subdevice. This latter structure also stores:

- the channels count;
- the channels declaration mode (A4L_CHAN_GLOBAL_CHANDESC or A4L_CHAN_PERCHAN_CHANDESC): if all the channels composing a subdevice are identical, there is no need to declare the parameters for each channel; the global declaration mode eases the structure composition.

Usually the channels descriptor looks like this:

```c
struct a4l_channels_desc example_chan = {
  mode: A4L_CHAN_GLOBAL_CHANDESC, -> Global declaration
  length: 8, -> 8 channels
  chans: { -> Each channel is 16 bits
    {A4L_CHAN_AREF_GROUND, 16}, -> with the ground as reference
  },
};
```

Ranges

So as to perform conversion from logical values acquired by the device to physical units, some range structure(s) must be declared on the driver side.

Such structures contain:

- the physical unit type (Volt, Ampere, none);
- the minimal and maximal values;

These range structures must be associated with the channels at subdevice registration time as a channel can work with many ranges. At configuration time (thanks to an Analogy command), one range will be selected for each enabled channel.

Consequently, for each channel, the developer must declare all the possible ranges in a structure called `struct a4l_rngtab`. Here is an example:

```c
struct a4l_rngtab example_tab = {
  length: 2,
  rngs: { -> Range of values
    RANGE_V(-5,5),
    RANGE_V(-10,10),
  },
};
```

For each subdevice, a specific structure is designed to gather all the ranges tabs of all the channels. In this structure, called `struct a4l_rngdesc`, three fields must be filled:

- the declaration mode (A4L_RNG_GLOBAL_RNGDESC or A4L_RNG_PERCHAN_RNGDESC);
- the number of ranges tab;
- the tab of ranges tabs pointers;

Most of the time, the channels which belong to the same subdevice use the same set of ranges. So, there is no need to declare the same ranges for each channel. A macro is defined to prevent redundant declarations: `RNG_GLOBAL()`.

Here is an example:

```c
struct a4l_rngdesc example_rng = RNG_GLOBAL(example_tab);
```
6.2 Big dual kernel lock

Collaboration diagram for Big dual kernel lock:

![Collaboration diagram for Big dual kernel lock](image)

Macros

- `#define cobalt_atomic_enter(__context)`
  
  Enter atomic section (dual kernel only)

- `#define cobalt_atomic_leave(__context)`
  
  Leave atomic section (dual kernel only)

- `#define RTDM_EXECUTE_ATOMICALY(code_block)`
  
  Execute code block atomically (DEPRECATED)

6.2.1 Detailed Description

6.2.2 Macro Definition Documentation

6.2.2.1 cobalt_atomic_enter

`#define cobalt_atomic_enter(__context)`

**Value:**

```c
do {
    xnlock_get_irqsave(&nklock, (__context));
    xnsched_lock();
} while (0)
```

Enter atomic section (dual kernel only)

This call opens a fully atomic section, serializing execution with respect to all interrupt handlers (including for real-time IRQs) and Xenomai threads running on all CPUs.
Parameters

| __context | name of local variable to store the context in. This variable updated by the real-time core will hold the information required to leave the atomic section properly. |

Note

Atomic sections may be nested. The caller is allowed to sleep on a blocking Xenomai service from primary mode within an atomic section delimited by cobalt_atomic_enter/cobalt_atomic__leave calls. On the contrary, sleeping on a regular Linux kernel service while holding such lock is NOT valid.

Since the strongest lock is acquired by this service, it can be used to synchronize real-time and non-real-time contexts.

Warning

This service is not portable to the Mercury core, and should be restricted to Cobalt-specific use cases, mainly for the purpose of porting existing dual-kernel drivers which still depend on the obsolete RTDM_EXECUTE_ATOMICALLY() construct.

6.2.2.2 cobalt_atomic_leave

#define cobalt_atomic_leave(  
    __context )

Value:

    do {  
        xnschedUnlock();  \  
        xnlockPut_irqrestore(&nklock, (__context)); \  
    } while (0)

Leave atomic section (dual kernel only)

This call closes an atomic section previously opened by a call to cobalt_atomic_enter(), restoring the preemption and interrupt state which prevailed prior to entering the exited section.

Parameters

| __context | name of local variable which stored the context. |

Warning

This service is not portable to the Mercury core, and should be restricted to Cobalt-specific use cases.
6.2.2.3 RTDM_EXECUTE_ATOMICALLY

#define RTDM_EXECUTE_ATOMICALLY(
   code_block )

Value:

{
   <ENTER_ATOMIC_SECTION>
   code_block;
   <LEAVE_ATOMIC_SECTION>
}

Execute code block atomically (DEPRECATED)

Generally, it is illegal to suspend the current task by calling rtdm_task_sleep(), rtdm_event_wait(), etc. while holding a spinlock. In contrast, this macro allows to combine several operations including a potentially rescheduling call to an atomic code block with respect to other RTDM_EXECUTE_ATOMICALLY() blocks. The macro is a light-weight alternative for protecting code blocks via mutexes, and it can even be used to synchronise real-time and non-real-time contexts.

Parameters

| code_block | Commands to be executed atomically |

Note

It is not allowed to leave the code block explicitly by using break, return, goto, etc. This would leave the global lock held during the code block execution in an inconsistent state. Moreover, do not embed complex operations into the code block. Consider that they will be executed under preemption lock with interrupts switched-off. Also note that invocation of rescheduling calls may break the atomicity until the task gains the CPU again.

Tags

unrestricted

Deprecated This construct will be phased out in Xenomai 3.0. Please use rtdm_waitqueue services instead.

See also
cobalt_atomic_enter().
6.3 Spinlock with preemption deactivation

Collaboration diagram for Spinlock with preemption deactivation:

```
Synchronisation Services

Spinlock with preemption deactivation
```

Macros

- `#define RTDM_LOCK_UNLOCKED(__name) IPIPE_SPIN_LOCK_UNLOCKED`
  Static lock initialisation.
- `#define rtdm_lock_get_irqsave(__lock, __context) ((__context) = __rtdm_lock_get_irqsave(__lock))`
  Acquire lock and disable preemption, by stalling the head domain.
- `#define rtdm_lock_irqsave(__context) splhigh(__context)`
  Disable preemption locally.
- `#define rtdm_lock_irqrestore(__context) splexit(__context)`
  Restore preemption state.

Typedefs

- `typedef ipipe_spinlock_t rtdm_lock_t`
  Lock variable.
- `typedef unsigned long rtdm_lockctx_t`
  Variable to save the context while holding a lock.

Functions

- `static void rtdm_lock_init (rtdm_lock_t *lock)`
  Dynamic lock initialisation.
- `static void rtdm_lock_get (rtdm_lock_t *lock)`
  Acquire lock from non-preemptible contexts.
- `static void rtdm_lock_put (rtdm_lock_t *lock)`
  Release lock without preemption restoration.
- `static void rtdm_lock_put_irqrestore (rtdm_lock_t *lock, rtdm_lockctx_t context)`
  Release lock and restore preemption state.

6.3.1 Detailed Description

6.3.2 Macro Definition Documentation
6.3.2.1  rtdm_lock_get_irqsave

#define rtdm_lock_get_irqsave(
  __lock,
  __context ) ((__context) = __rtdm_lock_get_irqsave(__lock))

Acquire lock and disable preemption, by stalling the head domain.

Parameters

<table>
<thead>
<tr>
<th>__lock</th>
<th>Address of lock variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>__context</td>
<td>name of local variable to store the context in</td>
</tr>
</tbody>
</table>

Tags

unrestricted

Referenced by rtdm_ratelimit().

6.3.2.2  rtdm_lock_irqrestore

#define rtdm_lock_irqrestore(
  __context ) splexit(__context)

Restore preemption state.

Parameters

| __context  | name of local variable which stored the context |

Tags

unrestricted

6.3.2.3  rtdm_lock_irqsave

#define rtdm_lock_irqsave(
  __context ) splhigh(__context)

Disable preemption locally.

Parameters

| __context  | name of local variable to store the context in |
6.3 Spinlock with preemption deactivation

6.3.3 Function Documentation

6.3.3.1 rtdm_lock_get()

static void rtdm_lock_get ( rtdm_lock_t *lock ) [inline], [static]

Acquire lock from non-preemptible contexts.

Parameters

lock Address of lock variable

Tags

unrestricted

References spltest.

6.3.3.2 rtdm_lock_init()

static void rtdm_lock_init ( rtdm_lock_t *lock ) [inline], [static]

Dynamic lock initialisation.

Parameters

lock Address of lock variable

Tags

task-unrestricted

6.3.3.3 rtdm_lock_put()

static void rtdm_lock_put ( rtdm_lock_t *lock ) [inline], [static]
Release lock without preemption restoration.

Parameters

| lock | Address of lock variable |

Tags

unrestricted, might-switch

6.3.3.4 rtdm_lock_put_irqrestore()

static void rtdm_lock_put_irqrestore (  
    rtdm_lock_t * lock,  
    rtdm_lockctx_t context ) [inline], [static]

Release lock and restore preemption state.

Parameters

| lock | Address of lock variable |
| context | name of local variable which stored the context |

Tags

unrestricted

Referenced by rtdm_ratelimit().
6.4 User-space driver core

This profile includes all mini-drivers sitting on top of the User-space Device Driver framework (UDD).

Collaboration diagram for User-space driver core:

![Collaboration diagram]

Data Structures

- struct udd_memregion
- struct udd_device
- struct udd_signotify

  *UDD event notification descriptor.*

Functions

- int udd_register_device (struct udd_device *udd)
  
  Register a UDD device.

- int udd_unregister_device (struct udd_device *udd)
  
  Unregister a UDD device.

- struct udd_device * udd_get_device (struct rtdm_fd *fd)
  
  RTDM file descriptor to target UDD device.

- void udd_notify_event (struct udd_device *udd)
  
  Notify an IRQ event for an unmanaged interrupt.

- void udd_enable_irq (struct udd_device *udd, rtdm_event_t *done)
  
  Enable the device IRQ line.

- void udd_disable_irq (struct udd_device *udd, rtdm_event_t *done)
  
  Disable the device IRQ line.

- #define UDD_IRQ_NONE 0
  
  No IRQ managed.

- #define UDD_IRQ_CUSTOM (-1)
  
  IRQ directly managed from the mini-driver on top of the UDD core.
Memory types for mapping

Types of memory for mapping

The UDD core implements a default ->mmap() handler which first attempts to hand over the request to the corresponding handler defined by the mini-driver. If not present, the UDD core establishes the mapping automatically, depending on the memory type defined for the region.

- `#define UDD_MEM_NONE 0
  No memory region.
- `#define UDD_MEM_PHYS 1
  Physical I/O memory region.
- `#define UDD_MEM_LOGICAL 2
  Kernel logical memory region (e.g.
- `#define UDD_MEM_VIRTUAL 3
  Virtual memory region with no direct physical mapping (e.g.

UDD_IOCTL

IOCTL requests

- `#define UDD_RTIOC_IRQEN _IO(RTDM_CLASS_UDD, 0)
  Enable the interrupt line.
- `#define UDD_RTIOC_IRQDIS _IO(RTDM_CLASS_UDD, 1)
  Disable the interrupt line.
- `#define UDD_RTIOCIRQSIG _IOW(RTDM_CLASS_UDD, 2, struct udd_signotify)
  Enable/Disable signal notification upon interrupt event.

6.4.1 Detailed Description

This profile includes all mini-drivers sitting on top of the User-space Device Driver framework (UDD).

The generic UDD core driver enables interrupt control and I/O memory access interfaces to user-space device drivers, as defined by the mini-drivers when registering.

A mini-driver supplements the UDD core with ancillary functions for dealing with memory mappings and interrupt control for a particular I/O card/device.

UDD-compliant mini-drivers only have to provide the basic support for dealing with the interrupt sources present in the device, so that most part of the device requests can be handled from a Xenomai application running in user-space. Typically, a mini-driver would handle the interrupt top-half, and the user-space application would handle the bottom-half.

This profile is reminiscent of the UIO framework available with the Linux kernel, adapted to the dual kernel Cobalt environment.

6.4.2 Macro Definition Documentation
6.4.2.1 UDD_IRQ_CUSTOM

#define UDD_IRQ_CUSTOM (-1)

IRQ directly managed from the mini-driver on top of the UDD core.

The mini-driver is in charge of attaching the handler(s) to the IRQ(s) it manages, notifying the Cobalt threads waiting for IRQ events by calling the `udd_notify_event()` service.

Referenced by `udd_disable_irq()`, `udd_enable_irq()`, `udd_register_device()`, and `udd_unregister_device()`.

6.4.2.2 UDD_IRQ_NONE

#define UDD_IRQ_NONE 0

No IRQ managed.

Special IRQ values for `udd_device.irq` Passing this code implicitly disables all interrupt-related services, including control (disable/enable) and notification.

Referenced by `udd_disable_irq()`, `udd_enable_irq()`, `udd_register_device()`, and `udd_unregister_device()`.

6.4.2.3 UDD_MEM_LOGICAL

#define UDD_MEM_LOGICAL 2

Kernel logical memory region (e.g. `kmalloc()`). By default, the UDD core maps such memory to a virtual user range by calling the `rtdm_mmap_kmem()` service.

6.4.2.4 UDD_MEM_NONE

#define UDD_MEM_NONE 0

No memory region.

Use this type code to disable an entry in the array of memory mappings, i.e. `udd_device.mem_regions[]`.

6.4.2.5 UDD_MEM_PHYS

#define UDD_MEM_PHYS 1

Physical I/O memory region.

By default, the UDD core maps such memory to a virtual user range by calling the `rtdm_mmap_iomem()` service.
6.4.2.6  UDD_MEM_VIRTUAL

#define UDD_MEM_VIRTUAL 3

Virtual memory region with no direct physical mapping (e.g. vmalloc()). By default, the UDD core maps such memory to a virtual user range by calling the rtdm->mmap_vmem() service.

6.4.2.7  UDD_RTIOC_IRQDIS

#define UDD_RTIOC_IRQDIS _IO(RTDM_CLASS_UDD, 1)

Disable the interrupt line.

The UDD-class mini-driver should handle this request when received through its ->ioctl() handler if provided. Otherwise, the UDD core disables the interrupt line in the interrupt controller before returning to the caller.

Note

The mini-driver must handle the UDD_RTIOC_IRQEN request for a custom IRQ from its ->ioctl() handler, otherwise such request receives -EIO from the UDD core.

6.4.2.8  UDD_RTIOC_IRQEN

#define UDD_RTIOC_IRQEN _IO(RTDM_CLASS_UDD, 0)

Enable the interrupt line.

The UDD-class mini-driver should handle this request when received through its ->ioctl() handler if provided. Otherwise, the UDD core enables the interrupt line in the interrupt controller before returning to the caller.

6.4.2.9  UDD_RTIOC_IRQSIG

#define UDD_RTIOC_IRQSIG _IOW(RTDM_CLASS_UDD, 2, struct udd_signotify)

Enable/Disable signal notification upon interrupt event.

A valid notification descriptor must be passed along with this request, which is handled by the UDD core directly.

Note

The mini-driver must handle the UDD_RTIOC_IRQDIS request for a custom IRQ from its ->ioctl() handler, otherwise such request receives -EIO from the UDD core.
6.4.3 Function Documentation

6.4.3.1 udd_disable_irq()

```c
void udd_disable_irq ( 
   struct udd_device * udd, 
   rtdm_event_t * done )
```

Disable the device IRQ line.

This service issues a request to the regular kernel for disabling the IRQ line registered by the driver. If the caller runs in primary mode, the request is scheduled but deferred until the current CPU leaves the real-time domain (see note). Otherwise, the request is immediately handled.

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>udd</td>
<td>The UDD driver handling the IRQ to disable. If no IRQ was registered by the driver at the UDD core, this routine has no effect.</td>
</tr>
<tr>
<td>done</td>
<td>Optional event to signal upon completion. If non-NULL, done will be posted by a call to rtdm_event_signal() after the interrupt line is disabled.</td>
</tr>
</tbody>
</table>

Tags

- unrestricted

Note

The deferral is required as some interrupt management code involved in disabling interrupt lines may not be safely executed from primary mode. By passing a valid done object address, the caller can wait for the request to complete, by sleeping on rtdm_event_wait().

References udd_device::irq, UDD_IRQ_CUSTOM, and UDD_IRQ_NONE.

6.4.3.2 udd_enable_irq()

```c
void udd_enable_irq ( 
   struct udd_device * udd, 
   rtdm_event_t * done )
```

Enable the device IRQ line.

This service issues a request to the regular kernel for enabling the IRQ line registered by the driver. If the caller runs in primary mode, the request is scheduled but deferred until the current CPU leaves the real-time domain (see note). Otherwise, the request is immediately handled.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>udd</em></td>
<td>The UDD driver handling the IRQ to disable. If no IRQ was registered by the driver at the UDD core, this routine has no effect.</td>
</tr>
<tr>
<td><em>done</em></td>
<td>Optional event to signal upon completion. If non-NULL, <em>done</em> will be posted by a call to <em>rtdm_event_signal()</em> after the interrupt line is enabled.</td>
</tr>
</tbody>
</table>

Tags

- unrestricted

Note

The deferral is required as some interrupt management code involved in enabling interrupt lines may not be safely executed from primary mode. By passing a valid *done* object address, the caller can wait for the request to complete, by sleeping on *rtdm_event_wait()*.

References *udd_device::*irq, UDD_IRQ_CUSTOM, and UDD_IRQ_NONE.

6.4.3.3 udd_get_device()

```c
struct udd_device* udd_get_device (  
    struct rtdm_fd * fd )
```

RTDM file descriptor to target UDD device.

Retrieves the UDD device from a RTDM file descriptor.

Parameters

- *fd*  
  File descriptor received by an ancillary I/O handler from a mini-driver based on the UDD core.

Returns

- A pointer to the UDD device to which *fd* refers to.

Note

This service is intended for use by mini-drivers based on the UDD core exclusively. Passing file descriptors referring to other RTDM devices will certainly lead to invalid results.

Tags

- mode-unrestricted

References *rtdm_device::*driver, rtdm_driver::*profile_info, and *rtdm_fd_device()*.
6.4 User-space driver core

6.4.3.4 udd_notify_event()

```c
void udd_notify_event (
    struct udd_device * udd )
```

Notify an IRQ event for an unmanaged interrupt.

When the UDD core shall hand over the interrupt management for a device to the mini-driver (see UDD_IRQ_CUSTOM), the latter should notify the UDD core when IRQ events are received by calling this service.

As a result, the UDD core wakes up any Cobalt thread waiting for interrupts on the device via a read(2) or select(2) call.

**Parameters**

- `udd` | UDD device descriptor receiving the IRQ.

**Tags**

- coreirq-only

**Note**

In case the IRQ handler from the mini-driver requested the UDD core not to re-enable the interrupt line, the application may later request the unmasking by issuing the UDD_RTIOC_IRQEN ioctl(2) command. Writing a non-zero integer to the device via the write(2) system call has the same effect.

**References** rtdm_event_signal(), rtdm_irq_disable(), and rtdm_irq_enable().

6.4.3.5 udd_register_device()

```c
int udd_register_device ( 
    struct udd_device * udd )
```

Register a UDD device.

This routine registers a mini-driver at the UDD core.

**Parameters**

- `udd` | UDD device descriptor which should describe the new device properties.

**Returns**

Zero is returned upon success, otherwise a negative error code is received, from the set of error codes defined by rtdm_dev_register(). In addition, the following error codes can be returned:

---

Generated by Doxygen
• -EINVAL, some of the memory regions declared in the `udd_device.mem_regions[]` array have invalid properties, i.e. bad type, NULL name, zero length or address. Any undeclared region entry from the array must bear the `UDD_MEM_NONE` type.

• -EINVAL, if `udd_device.irq` is different from `UDD_IRQ_CUSTOM` and `UDD_IRQ_NONE` but invalid, causing `rtdm_irq_request()` to fail.

• -EINVAL, if `udd_device.device_flags` contains invalid flags.

• -ENXIO can be received if this service is called while the Cobalt kernel is disabled.

Tags

`secondary-only`

References `udd_device::device_flags`, `udd_device::interrupt`, `udd_device::irq`, `RTDM_PROTOCOL_DVICE`, `UDD_IRQ_CUSTOM`, and `UDD_IRQ_NONE`.

6.4.3.6 udd_unregister_device()

```c
int udd_unregister_device (  
   struct udd_device ∗udd  
)
```

Unregister a UDD device.

This routine unregisters a mini-driver from the UDD core. This routine waits until all connections to `udd` have been closed prior to unregistering.

Parameters

- `udd` UDD device descriptor

Returns

Zero is returned upon success, otherwise -ENXIO is received if this service is called while the Cobalt kernel is disabled.

Tags

`secondary-only`

References `udd_device::irq`, `rtdm_event_destroy()`, `rtdm_irq_free()`, `UDD_IRQ_CUSTOM`, and `UDD_IRQ_NONE`. 
6.5 Thread state flags

Bits reporting permanent or transient states of threads.

Collaboration diagram for Thread state flags:

![Collaboration diagram for Thread state flags](image)

**Macros**

- `#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 XNDELAY 0x00000004`
  Delayed.
- `#define XNREADY 0x00000008`
  Linked to the ready queue.
- `#define XNDORMANT 0x00000010`
  Not started yet.
- `#define XNZOMBIE 0x00000020`
  Zombie thread in deletion process.
- `#define XNMAPPE 0x00000040`
  Thread is mapped to a linux task.
- `#define XNRELAX 0x00000080`
  Relaxed shadow thread (blocking bit)
- `#define XNMIGRATE 0x00000100`
  Thread is currently migrating to another CPU.
- `#define XNHELD 0x00000200`
  Thread is held to process emergency.
- `#define XNBOOST 0x00000300`
  Undergoes a PIP boost.
- `#define XNSSTEP 0x00000400`
  Single-stepped by debugger.
- `#define XNLOCK 0x00000600`
  Scheduler lock control (pseudo-bit, not in -state)
- `#define XNRRB 0x00000700`
  Undergoes a round-robin scheduling.
- `#define XNWARN 0x00000800`
  Issue SIGDEBUG on error detection.
- `#define XNFPU 0x00000800`
  Thread uses FPU.
- `#define XNROOT 0x00001000`

Generated by Doxygen
Root thread (that is, Linux/IDLE)
- #define XNWEAK 0x00020000
  Non real-time shadow (from the WEAK class)
- #define XNUSER 0x00040000
  Shadow thread running in userland.
- #define XNJOINED 0x00080000
  Another thread waits for joining this thread.
- #define XNTRAPLB 0x00100000
  Trap lock break (i.e.
- #define XNDEBUG 0x00200000
  User-level debugging enabled.

6.5.1 Detailed Description

Bits reporting permanent or transient states of threads.

6.5.2 Macro Definition Documentation

6.5.2.1 XNHELD

#define XNHELD 0x00000200

Thread is held to process emergency.

6.5.2.2 XNMIGRATE

#define XNMIGRATE 0x00000100

Thread is currently migrating to another CPU.

6.5.2.3 XNPEND

#define XNPEND 0x00000002

Sleep-wait for a resource.
6.5.2.4  XNREADY

#define XNREADY 0x00000008

Linked to the ready queue.

6.5.2.5  XNSUSP

#define XNSUSP 0x00000001

Suspended.

Referenced by xnthread_init().

6.5.2.6  XNTRAPLB

#define XNTRAPLB 0x00100000

Trap lock break (i.e.
may not sleep with sched lock)
6.6 Thread information flags

Bits reporting events notified to threads.

Collaboration diagram for Thread information flags:

Macros

- `#define XNTIMEO 0x00000001`
  Woken up due to a timeout condition.
- `#define XNRMID 0x00000002`
  Pending on a removed resource.
- `#define XNBREAK 0x00000004`
  Forcibly awakened from a wait state.
- `#define XNKICKED 0x00000008`
  Forced out of primary mode.
- `#define XNWAKEN 0x00000010`
  Thread waken up upon resource availability.
- `#define XNROBBED 0x00000020`
  Robbed from resource ownership.
- `#define XNCANCELD 0x00000040`
  Cancellation request is pending.
- `#define XNPIALERT 0x00000080`
  Priority inversion alert (SIGDEBUG sent)
- `#define XNMOVED 0x00000001`
  CPU migration in primary mode occurred.
- `#define XNLBALERT 0x00000002`
  Scheduler lock break alert (SIGDEBUG sent)
- `#define XNDESCENT 0x00000004`
  Adaptive transitioning to secondary mode.
- `#define XNSYSRST 0x00000008`
  Thread awaiting syscall restart after signal.

6.6.1 Detailed Description

Bits reporting events notified to threads.
6.7 CAN Devices

This is the common interface a RTDM-compliant CAN device has to provide.

Collaboration diagram for CAN Devices:

![Collaboration diagram for CAN Devices](image)

Data Structures

- **struct can_bittime_std**
  
  *Standard bit-time parameters according to Bosch.*

- **struct can_bittime_btr**
  
  *Hardware-specific BTR bit-times.*

- **struct can_bittime**
  
  *Custom CAN bit-time definition.*

- **struct can_filter**
  
  *Filter for reception of CAN messages.*

- **struct sockaddr_can**
  
  *Socket address structure for the CAN address family.*

- **struct can_frame**
  
  *Raw CAN frame.*

- **struct can_ifreq**
  
  *CAN interface request descriptor.*

Macros

- **#define AF_CAN 29**
  
  *CAN address family.*

- **#define PF_CAN AF_CAN**
  
  *CAN protocol family.*

- **#define SOL_CAN_RAW 103**
  
  *CAN socket levels.*
Typedefs

- `typedef uint32_t can_id_t`  
  Type of CAN id (see CAN_xxx_MASK and CAN_xxx_FLAG)
- `typedef can_id_t can_err_mask_t`  
  Type of CAN error mask.
- `typedef uint32_t can_baudrate_t`  
  Baudrate definition in bits per second.
- `typedef enum CAN_BITTIME_TYPE can_bittime_type_t`  
  See CAN_BITTIME_TYPE.
- `typedef enum CAN_MODE can_mode_t`  
  See CAN_MODE.
- `typedef int can_ctrlmode_t`  
  See CAN_CTRLMODE.
- `typedef enum CAN_STATE can_state_t`  
  See CAN_STATE.
- `typedef struct can_filter can_filter_t`  
  Filter for reception of CAN messages.
- `typedef struct can_frame can_frame_t`  
  Raw CAN frame.

Enumerations

- `enum CAN_BITTIME_TYPE { CAN_BITTIME_STD, CAN_BITTIME_BTR }`  
  Supported CAN bit-time types.

CAN ID masks

Bit masks for masking CAN IDs

- `#define CAN_EFF_MASK 0x1FFFFFFF`  
  Bit mask for extended CAN IDs.
- `#define CAN_SFF_MASK 0x000007FF`  
  Bit mask for standard CAN IDs.

CAN ID flags

Flags within a CAN ID indicating special CAN frame attributes

- `#define CAN_EFF_FLAG 0x80000000`  
  Extended frame.
- `#define CAN_RTR_FLAG 0x40000000`  
  Remote transmission frame.
- `#define CAN_ERR_FLAG 0x20000000`  
  Error frame (see Errors), not valid in struct can_filter.
- `#define CAN_INV_FILTER CAN_ERR_FLAG`  
  Invert CAN filter definition, only valid in struct can_filter.
Particular CAN protocols

Possible protocols for the PF_CAN protocol family

Currently only the RAW protocol is supported.

- \#define CAN_RAW 1
  
  Raw protocol of PF_CAN, applicable to socket type SOCK_RAW.

CAN operation modes

Modes into which CAN controllers can be set

- enum CAN_MODE { CAN_MODE_STOP = 0, CAN_MODE_START, CAN_MODE_SLEEP }

CAN controller modes

Special CAN controllers modes, which can be or'ed together.

Note

These modes are hardware-dependent. Please consult the hardware manual of the CAN controller for more detailed information.

- \#define CAN_CTRLMODE_LISTENONLY 0x1
- \#define CAN_CTRLMODE_LOOPBACK 0x2
- \#define CAN_CTRLMODE_3_SAMPLES 0x4

CAN controller states

States a CAN controller can be in.

- enum CAN_STATE { CAN_STATE_ERROR_ACTIVE = 0, CAN_STATE_ACTIVE = 0, CAN_STATE_ERROR_WARNING = 1, CAN_STATE_BUS_WARNING = 1, CAN_STATE_ERROR_PASSIVE = 2, CAN_STATE_BUS_PASSIVE = 2, CAN_STATE_BUS_OFF, CAN_STATE_SCANNING_BAUDRATE, CAN_STATE_STOPPED, CAN_STATE_SLEEPING }

Timestamp switches

Arguments to pass to RTCAN_RTIOC_TAKE_TIMESTAMP

- \#define RTCAN_TAKE_NO_TIMESTAMPS 0
  
  Switch off taking timestamps.
- \#define RTCAN_TAKE_TIMESTAMPS 1
  
  Do take timestamps.
RAW socket options

Setting and getting CAN RAW socket options.

- `#define CAN_RAW_FILTER 0x1`  
  CAN filter definition.
- `#define CAN_RAW_ERR_FILTER 0x2`  
  CAN error mask.
- `#define CAN_RAW_LOOPBACK 0x3`  
  CAN TX loopback.
- `#define CAN_RAW_RECV_OWN_MSGS 0x4`  
  CAN receive own messages.

IOCTLS

CAN device IOCTLS

**Deprecated** Passing `struct ifreq` as a request descriptor for CAN IOCTLS is still accepted for backward compatibility, however it is recommended to switch to `struct can_ifreq` at the first opportunity.

- `#define SIOCGIFINDEX defined_by_kernel_header_file`  
  Get CAN interface index by name.
- `#define SIOCSCANBAUDRATE _IOW(RTIOC_TYPE_CAN, 0x01, struct can_ifreq)`  
  Set baud rate.
- `#define SIOCSCANCUSTOMBITTIME _IOW(RTIOC_TYPE_CAN, 0x03, struct can_ifreq)`  
  Set custom bit time parameter.
- `#define SIOCSCANCTRLMODE _IOW(RTIOC_TYPE_CAN, 0x07, struct can_ifreq)`  
  Set special controller modes.
- `#define RTCAN_RTIOC_TAKE_TIMESTAMP _IOW(RTIOC_TYPE_CAN, 0x09, int)`  
  Enable or disable storing a high precision timestamp upon reception of a CAN frame.
- `#define RTCAN_RTIOC_RCV_TIMEOUT _IOW(RTIOC_TYPE_CAN, 0x0A, nanosecs_rel_t)`  
  Specify a reception timeout for a socket.
- `#define RTCAN_RTIOC_SND_TIMEOUT _IOW(RTIOC_TYPE_CAN, 0x0B, nanosecs_rel_t)`  
  Specify a transmission timeout for a socket.
Error mask

Error class (mask) in `can_id` field of struct `can_frame` to be used with `CAN_RAW_ERR_FILTER`.

**Note:** Error reporting is hardware dependent and most CAN controllers report less detailed error conditions than the SJA1000.

**Note:** In case of a bus-off error condition (`CAN_ERR_BUSOFF`), the CAN controller is **not** restarted automatically. It is the application's responsibility to react appropriately, e.g. calling `CAN_MODE_STANDBY`.

**Note:** Bus error interrupts (`CAN_ERR_BUSERROR`) are enabled when an application is calling a `Recv` function on a socket listening on bus errors (using `CAN_RAW_ERR_FILTER`). After one bus error has occurred, the interrupt will be disabled to allow the application time for error processing and to efficiently avoid bus error interrupt flooding.

- `#define CAN_ERR_TX_TIMEOUT 0x00000001U`  
  TX timeout (netdevice driver)
- `#define CAN_ERR_LOSTARB 0x00000002U`  
  Lost arbitration (see `data[0]`)
- `#define CAN_ERR_CRTL 0x00000004U`  
  Controller problems (see `data[1]`)
- `#define CAN_ERR_PROT 0x00000008U`  
- `#define CAN_ERR_TRX 0x00000010U`  
  Transceiver status (see `data[4]`)
- `#define CAN_ERR_ACK 0x00000020U`  
  Received no ACK on transmission.
- `#define CAN_ERR_BUSOFF 0x00000040U`  
  Bus off.
- `#define CAN_ERR_BUSERROR 0x00000080U`  
  Bus error (may flood!)
- `#define CAN_ERR_RESTARTED 0x00000100U`  
  Controller restarted.
- `#define CAN_ERR_MASK 0x1FFFFFFFU`  
  Omit EFF, RTR, ERR flags.

Arbitration lost error

Error in the `data[0]` field of struct `can_frame`.

- `#define CAN_ERR_LOSTARB_UNSPEC 0x00`  
  unspecified
Controller problems

Error in the data[1] field of struct can_frame.

- #define CAN_ERR_CRTL_UNSPEC 0x00
  unspecified
- #define CAN_ERR_CRTL_RX_OVERFLOW 0x01
  RX buffer overflow.
- #define CAN_ERR_CRTL_TX_OVERFLOW 0x02
  TX buffer overflow.
- #define CAN_ERR_CRTL_RX_WARNING 0x04
  reached warning level for RX errors
- #define CAN_ERR_CRTL_TX_WARNING 0x08
  reached warning level for TX errors
- #define CAN_ERR_CRTL_RX_PASSIVE 0x10
  reached passive level for RX errors
- #define CAN_ERR_CRTL_TX_PASSIVE 0x20
  reached passive level for TX errors

Protocol error type

Error in the data[2] field of struct can_frame.

- #define CAN_ERR_PROT_UNSPEC 0x00
  unspecified
- #define CAN_ERR_PROT_BIT 0x01
  single bit error
- #define CAN_ERR_PROT_FORM 0x02
  frame format error
- #define CAN_ERR_PROT_STUFF 0x04
  bit stuffing error
- #define CAN_ERR_PROT_BIT0 0x08
  unable to send dominant bit
- #define CAN_ERR_PROT_BIT1 0x10
  unable to send recessive bit
- #define CAN_ERR_PROT_OVERLOAD 0x20
  bus overload
- #define CAN_ERR_PROT_ACTIVE 0x40
  active error announcement
- #define CAN_ERR_PROT_TX 0x80
  error occurred on transmission
Protocol error location

Error in the data[4] field of struct `can_frame`.

- `#define CAN_ERR_PROT_LOC_UNSPEC 0x00` unspecified
- `#define CAN_ERR_PROT_LOC_SOF 0x03` start of frame
- `#define CAN_ERR_PROT_LOC_ID28_21 0x02` ID bits 28 - 21 (SFF: 10 - 3)
- `#define CAN_ERR_PROT_LOC_ID20_18 0x06` ID bits 20 - 18 (SFF: 2 - 0)
- `#define CAN_ERR_PROT_LOC_SRTR 0x04` substitute RTR (SFF: RTR)
- `#define CAN_ERR_PROT_LOC_IDE 0x05` identifier extension
- `#define CAN_ERR_PROT_LOC_ID17_13 0x07` ID bits 17-13.
- `#define CAN_ERR_PROT_LOC_ID12_05 0x0F` ID bits 12-5.
- `#define CAN_ERR_PROT_LOC_ID04_00 0x0E` ID bits 4-0.
- `#define CAN_ERR_PROT_LOC_RTR 0x0C` RTR.
- `#define CAN_ERR_PROT_LOC_RES1 0x0D` reserved bit 1
- `#define CAN_ERR_PROT_LOC_RES0 0x09` reserved bit 0
- `#define CAN_ERR_PROT_LOC_DLC 0x0B` data length code
- `#define CAN_ERR_PROT_LOC_DATA 0x0A` data section
- `#define CAN_ERR_PROT_LOC_CRC_SEQ 0x08` CRC sequence.
- `#define CAN_ERR_PROT_LOC_CRC_DEL 0x18` CRC delimiter.
- `#define CAN_ERR_PROT_LOC_ACK 0x19` ACK slot.
- `#define CAN_ERR_PROT_LOC_ACK_DEL 0x1B` ACK delimiter.
- `#define CAN_ERR_PROT_LOC_EOF 0x1A` end of frame
- `#define CAN_ERR_PROT_LOC_INTERM 0x12` intermission
- `#define CAN_ERR_TRX_UNSPEC 0x00` 0000 0000
- `#define CAN_ERR_TRX_CANH_NO_WIRE 0x04` 0000 0100
- `#define CAN_ERR_TRX_CANH_SHORT_TO_BAT 0x05` 0000 0101
6.7.1 Detailed Description

This is the common interface a RTDM-compliant CAN device has to provide.

Feel free to report bugs and comments on this profile to the "Socketcan" mailing list (Socketcan-core@lists.berlios.de) or directly to the authors (wg@grandegger.com or Sebastian.Smolorz@stud.uni-hannover.de).

Profile Revision: 2

Device Characteristics

- **Device Flags**: RTDM_PROTOCOL_DEVICE
- **Protocol Family**: PF_CAN
- **Socket Type**: SOCK_RAW
- **Device Class**: RTDM_CLASS_CAN

Supported Operations

- **Socket**

Tags

- `secondary-only`

Specific return values:

- `-EPROTONOSUPPORT` (Protocol is not supported by the driver. See CAN protocols for possible protocols.)
**Close**
Blocking calls to any of the Send or Receive functions will be unblocked when the socket is closed and return with an error.

**Tags**
- **secondary-only**
  Specific return values: none

**IOCTL**

**Tags**
- **task-unrestricted.** see below Specific return values: see below

**Bind**
Binds a socket to one or all CAN devices (see struct sockaddr_can). If a filter list has been defined with setsockopt (see Sockopts), it will be used upon reception of CAN frames to decide whether the bound socket will receive a frame. If no filter has been defined, the socket will receive all CAN frames on the specified interface(s).

Binding to special interface index 0 will make the socket receive CAN frames from all CAN interfaces.

Binding to an interface index is also relevant for the Send functions because they will transmit a message over the interface the socket is bound to when no socket address is given to them.

**Tags**
- **secondary-only**
  Specific return values:
  - -EFAULT (It was not possible to access user space memory area at the specified address.)
  - -ENOMEM (Not enough memory to fulfill the operation)
  - -EINVAL (Invalid address family, or invalid length of address structure)
  - -ENODEV (Invalid CAN interface index)
  - -ENOSPC (No enough space for filter list)
  - -EBADF (Socket is about to be closed)
  - -EAGAIN (Too many receivers. Old binding (if any) is still active. Close some sockets and try again.)

**Setsockopt, Getsockopt**
These functions allow to set and get various socket options. Currently, only CAN raw sockets are supported.

**Supported Levels and Options:**
Module Documentation

- Level **SOL_CAN_RAW**: CAN RAW protocol (see **CAN_RAW**)
  - Option **CAN_RAW_FILTER**: CAN filter list
  - Option **CAN_RAW_ERR_FILTER**: CAN error mask
  - Option **CAN_RAW_LOOPBACK**: CAN TX loopback to local sockets

Tags

**task-unrestricted** Specific return values: see links to options above.

**Recv, Recvfrom, Recvmsg**
These functions receive CAN messages from a socket. Only one message per call can be received, so only one buffer with the correct length must be passed. For **SOCK_RAW**, this is the size of struct **can_frame**.

Unlike a call to one of the **Send** functions, a **Recv** function will not return with an error if an interface is down (due to bus-off or setting of stop mode) or in sleep mode. Moreover, in such a case there may still be some CAN messages in the socket buffer which could be read out successfully.

It is possible to receive a high precision timestamp with every CAN message. The condition is a former instruction to the socket via **RTCAN_RTIOC_TAKE_TIMESTAMP**. The timestamp will be copied to the **msg_control** buffer of **struct msghdr** if it points to a valid memory location with size of **nanosecs_abs_t**. If this is a NULL pointer the timestamp will be discarded silently.

**Note**: A **msg_controllen** of 0 upon completion of the function call indicates that no timestamp is available for that message.

Supported Flags [in]:
- **MSG_DONTWAIT** (By setting this flag the operation will only succeed if it would not block, i.e. if there is a message in the socket buffer. This flag takes precedence over a timeout specified by **RTCAN_RTIOC_RCV_TIMEOUT**.)
- **MSG_PEEK** (Receive a message but leave it in the socket buffer. The next receive operation will get that message again.)

Supported Flags [out]: none

Tags

**mode-unrestricted**
Specific return values:
- Non-negative value (indicating the successful reception of a CAN message. For **SOCK_RAW**, this is the size of struct **can_frame** regardless of the actual size of the payload.)
- **EFAULT** (It was not possible to access user space memory area at one of the specified addresses.)
- **EINVAL** (Unsupported flag detected, or invalid length of socket address buffer, or invalid length of message control buffer)
- **EMSGSIZE** (Zero or more than one iovec buffer passed, or buffer too small)
- **EAGAIN** (No data available in non-blocking mode)
- **EBADF** (Socket was closed.)
-EINTR (Operation was interrupted explicitly or by signal.)
-ETIMEDOUT (Timeout)

Send, Sendto, Sendmsg
These functions send out CAN messages. Only one message per call can be transmitted, so only one buffer with the correct length must be passed. For SOCK_RAW, this is the size of struct can_frame.

The following only applies to SOCK_RAW: If a socket address of struct sockaddr_can is given, only can_ifindex is used. It is also possible to omit the socket address. Then the interface the socket is bound to will be used for sending messages.

If an interface goes down (due to bus-off or setting of stop mode) all senders that were blocked on this interface will be woken up.

Supported Flags:
- MSG_DONTWAIT (By setting this flag the transmit operation will only succeed if it would not block. This flag takes precedence over a timeout specified by RTCAN_RTIOM_SND_TIME_OUT.)

Tags
mode-unrestricted
Specific return values:
- Non-negative value equal to given buffer size (Indicating the successful completion of the function call. See also note.)
- EOPNOTSUPP (MSG_OOB flag is not supported.)
- EINVAL (Unsupported flag detected or: Invalid length of socket address or: Invalid address family or: Data length code of CAN frame not between 0 and 15 or: CAN standard frame has got an ID not between 0 and 2031)
- EMSGSIZE (Zero or more than one buffer passed or invalid size of buffer)
-EFAULT (It was not possible to access user space memory area at one of the specified addresses.)
- ENXIO (Invalid CAN interface index - 0 is not allowed here - or socket not bound or rather bound to all interfaces.)
- ENETDOWN (Controller is bus-off or in stopped state.)
- ECOMM (Controller is sleeping)
- EAGAIN (Cannot transmit without blocking but a non-blocking call was requested.)
- EINTR (Operation was interrupted explicitly or by signal)
- EBADF (Socket was closed.)
- ETIMEDOUT (Timeout)

Note: A successful completion of the function call does not implicate a successful transmission of the message.
6.7.2.1 CAN_CTRLMODE_3_SAMPLES

#define CAN_CTRLMODE_3_SAMPLES 0x4

Triple sampling mode
In this mode the CAN controller uses Triple sampling.

6.7.2.2 CAN_CTRLMODE_LISTENONLY

#define CAN_CTRLMODE_LISTENONLY 0x1

Listen-Only mode
In this mode the CAN controller would give no acknowledge to the CAN-bus, even if a message is received successfully and messages would not be transmitted. This mode might be useful for bus-monitoring, hot-plugging or throughput analysis.

Examples:
   \texttt{rtcanconfig.c}.

6.7.2.3 CAN_CTRLMODE_LOOPBACK

#define CAN_CTRLMODE_LOOPBACK 0x2

Loopback mode
In this mode the CAN controller does an internal loop-back, a message is transmitted and simultaneously received. That mode can be used for self test operation.

Examples:
   \texttt{rtcanconfig.c}.

6.7.2.4 CAN_ERR_LOSTARB_UNSPEC

#define CAN_ERR_LOSTARB_UNSPEC 0x00

unspecced
else bit number in bitstream

6.7.2.5 CAN_RAW_ERR_FILTER

#define CAN_RAW_ERR_FILTER 0x2

CAN error mask.
A CAN error mask (see \texttt{Errors}) can be set with \texttt{setsockopt}. This mask is then used to decide if error frames are delivered to this socket in case of error conditions. The error frames are marked with the \texttt{CAN_ERR_FLAG} of \texttt{CAN_xxx_FLAG} and must be handled by the application properly. A detailed description of the errors can be found in the \texttt{can_id} and the \texttt{data} fields of struct \texttt{can_frame} (see \texttt{Errors} for further details).
6.7 CAN Devices

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>level</th>
<th>SOL_CAN_RAW</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>optname</td>
<td>CAN_RAW_ERR_FILTER</td>
</tr>
<tr>
<td>in</td>
<td>optval</td>
<td>Pointer to error mask of type can_err_mask_t.</td>
</tr>
<tr>
<td>in</td>
<td>optlen</td>
<td>Size of error mask: sizeof(can_err_mask_t).</td>
</tr>
</tbody>
</table>

Tags

- task-unrestricted

Specific return values:

- -EFAULT (It was not possible to access user space memory area at the specified address.)
- -EINVAL (Invalid length "optlen")

Examples:

- rtcanrecv.c.

6.7.2.6 CAN_RAW_FILTER

#define CAN_RAW_FILTER 0x1

CAN filter definition.

A CAN raw filter list with elements of struct can_filter can be installed with setsockopt. This list is used upon reception of CAN frames to decide whether the bound socket will receive a frame. An empty filter list can also be defined using optlen = 0, which is recommended for write-only sockets.

If the socket was already bound with Bind, the old filter list gets replaced with the new one. Be aware that already received, but not read out CAN frames may stay in the socket buffer.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>level</th>
<th>SOL_CAN_RAW</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>optname</td>
<td>CAN_RAW_FILTER</td>
</tr>
<tr>
<td>in</td>
<td>optval</td>
<td>Pointer to array of struct can_filter.</td>
</tr>
<tr>
<td>in</td>
<td>optlen</td>
<td>Size of filter list: count * sizeof( struct can_filter).</td>
</tr>
</tbody>
</table>

Tags

- task-unrestricted

Specific return values:

- -EFAULT (It was not possible to access user space memory area at the specified address.)
- -ENOMEM (Not enough memory to fulfill the operation)
- -EINVAL (Invalid length "optlen")

Generated by Doxygen
• -ENOSPC (No space to store filter list, check RT-Socket-CAN kernel parameters)

Examples:

    can-rtt.c, rtcanrecv.c, and rtcansend.c.

6.7.2.7 CAN_RAW_LOOPBACK

#define CAN_RAW_LOOPBACK 0x3

CAN TX loopback.

The TX loopback to other local sockets can be selected with this setsockopt.

Note

The TX loopback feature must be enabled in the kernel and then the loopback to other local TX sockets is enabled by default.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>level</th>
<th>SOL_CAN_RAW</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>optname</td>
<td>CAN_RAW_LOOPBACK</td>
</tr>
<tr>
<td>in</td>
<td>optval</td>
<td>Pointer to integer value.</td>
</tr>
<tr>
<td>in</td>
<td>optlen</td>
<td>Size of int: sizeof(int).</td>
</tr>
</tbody>
</table>

Tags

    task-unrestricted

Specific return values:

- -EFAULT (It was not possible to access user space memory area at the specified address.)
- -EINVAL (Invalid length "optlen")
- -EOPNOTSUPP (not supported, check RT-Socket-CAN kernel parameters).

Examples:

    rtcansend.c.

6.7.2.8 CAN_RAW_RECV_OWN_MSGS

#define CAN_RAW_RECV_OWN_MSGS 0x4

CAN receive own messages.

Not supported by RT-Socket-CAN, but defined for compatibility with Socket-CAN.
6.7.2.9 RTCAN_RTIOC_RCV_TIMEOUT

#define RTCAN_RTIOC_RCV_TIMEOUT _IOW(RTIOC_TYPE_CAN, 0x0A, nanosecs_rel_t)

Specify a reception timeout for a socket.

Defines a timeout for all receive operations via a socket which will take effect when one of the receive functions is called without the MSG_DONTWAIT flag set.

The default value for a newly created socket is an infinite timeout.

Note

The setting of the timeout value is not done atomically to avoid locks. Please set the value before receiving messages from the socket.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>arg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pointer to nanosecs_rel_t variable. The value is interpreted as relative timeout in nanoseconds in case of a positive value. See Timeouts for special timeouts.</td>
<td></td>
</tr>
</tbody>
</table>

Returns

0 on success, otherwise:

- EFAULT: It was not possible to access user space memory area at the specified address.

Tags

task-unrestricted

Examples:

rtnanrecv.c.

6.7.2.10 RTCAN_RTIOC_SND_TIMEOUT

#define RTCAN_RTIOC_SND_TIMEOUT _IOW(RTIOC_TYPE_CAN, 0x0B, nanosecs_rel_t)

Specify a transmission timeout for a socket.

Defines a timeout for all send operations via a socket which will take effect when one of the send functions is called without the MSG_DONTWAIT flag set.

The default value for a newly created socket is an infinite timeout.

Note

The setting of the timeout value is not done atomically to avoid locks. Please set the value before sending messages to the socket.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>arg</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Pointer to <code>nanosecs_rel_t</code> variable. The value is interpreted as relative timeout in nanoseconds in case of a positive value. See <strong>Timeouts</strong> for special timeouts.</td>
</tr>
</tbody>
</table>

Returns

- 0 on success, otherwise:
  - -EFAULT: It was not possible to access user space memory area at the specified address.

Tags

- task-unrestricted

Examples:

- `rtcansend.c`

---

### 6.7.2.11 RTCAN_RTIOC_TAKE_TIMESTAMP

```c
#define RTCAN_RTIOC_TAKE_TIMESTAMP _IOW(RTIOC_TYPE_CAN, 0x09, int)
```

Enable or disable storing a high precision timestamp upon reception of a CAN frame.

A newly created socket takes no timestamps by default.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>arg</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>int</code> variable, see <strong>Timestamp switches</strong></td>
</tr>
</tbody>
</table>

Returns

- 0 on success.

Tags

- task-unrestricted

Note

- Activating taking timestamps only has an effect on newly received CAN messages from the bus. Frames that already are in the socket buffer do not have timestamps if it was deactivated before. See **Receive** for more details.

Examples:

- `rtcanrecv.c`
6.7.2.12 SIOCGCANBAUDRATE

#define SIOCGCANBAUDRATE _IOWR(RTIOC_TYPE_CAN, 0x02, struct can_ifreq)

Get baud rate.

Parameters

<table>
<thead>
<tr>
<th>in, out</th>
<th>arg</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Pointer to interface request structure buffer (struct can_ifreq). ifr_name must hold a valid CAN interface name, ifr_ifru will be filled with an instance of can_baudrate_t.</td>
</tr>
</tbody>
</table>

Returns

0 on success, otherwise:
- -EFAULT: It was not possible to access user space memory area at the specified address.
- -ENODEV: No device with specified name exists.
- -EINVAL: No baud rate was set yet.

Tags

  task-unrestricted

6.7.2.13 SIOCGCANCRTLMODE

#define SIOCGCANCRTLMODE _IOWR(RTIOC_TYPE_CAN, 0x08, struct can_ifreq)

Get special controller modes.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>arg</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Pointer to interface request structure buffer (struct can_ifreq). ifr_name must hold a valid CAN interface name, ifr_ifru must be filled with an instance of can_ctrlmode_t.</td>
</tr>
</tbody>
</table>

Returns

0 on success, otherwise:
- -EFAULT: It was not possible to access user space memory area at the specified address.
- -ENODEV: No device with specified name exists.
- -EINVAL: No baud rate was set yet.

Tags

  task-unrestricted, might-switch
6.7.2.14 SIOCGCANCUSTOMBITTIME

#define SIOCGCANCUSTOMBITTIME _IOWR(RTIOC_TYPE_CAN, 0x04, struct can_ifreq)

Get custom bit-time parameters.

Parameters

<table>
<thead>
<tr>
<th>in, out</th>
<th>arg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pointer to interface request structure buffer (struct can_ifreq). ifr_name must hold a valid CAN interface name, ifr_ifru will be filled with an instance of struct can_bittime.</td>
<td></td>
</tr>
</tbody>
</table>

Returns

0 on success, otherwise:

- -EFAULT: It was not possible to access user space memory area at the specified address.
- -ENODEV: No device with specified name exists.
- -EINVAL: No baud rate was set yet.

Tags

task-unrestricted

6.7.2.15 SIOCGCANSTATE

#define SIOCGCANSTATE _IOWR(RTIOC_TYPE_CAN, 0x06, struct can_ifreq)

Get current state of CAN controller.

States are divided into main states and additional error indicators. A CAN controller is always in exactly one main state. CAN bus errors are registered by the CAN hardware and collected by the driver. There is one error indicator (bit) per error type. If this IOCTL is triggered the error types which occurred since the last call of this IOCTL are reported and thereafter the error indicators are cleared. See also CAN controller states.

Parameters

<table>
<thead>
<tr>
<th>in, out</th>
<th>arg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pointer to interface request structure buffer (struct can_ifreq). ifr_name must hold a valid CAN interface name, ifr_ifru will be filled with an instance of struct can_mode_t.</td>
<td></td>
</tr>
</tbody>
</table>

Returns

0 on success, otherwise:

- -EFAULT: It was not possible to access user space memory area at the specified address.
- -ENODEV: No device with specified name exists.
6.7 CAN Devices

Tags

  task-unrestricted, might-switch

6.7.2.16 SIOCGIFINDEX

#define SIOCGIFINDEX defined_by_kernel_header_file

Get CAN interface index by name.

Parameters

| in, out | arg | Pointer to interface request structure buffer (struct can_ifreq). If ifr_name holds a valid CAN interface name ifr_ifindex will be filled with the corresponding interface index. |

Returns

  0 on success, otherwise:
  
  - -EFAULT: It was not possible to access user space memory area at the specified address.
  - -ENODEV: No device with specified name exists.

Tags

  task-unrestricted

Examples:

  can-rtt.c, rtcanconfig.c, rtcanrecv.c, and rt cansend.c.

6.7.2.17 SIOCSCANBAUDRATE

#define SIOCSCANBAUDRATE _IOW(RTIOC_TYPE_CAN, 0x01, struct can_ifreq)

Set baud rate.

The baudrate must be specified in bits per second. The driver will try to calculate resonable CAN bit-timing parameters. You can use SIOCSCANCUSTOMBITTIME to set custom bit-timing.

Parameters

| in | arg | Pointer to interface request structure buffer (struct can_ifreq). ifr_name must hold a valid CAN interface name, ifr_ifru must be filled with an instance of can_baudrate_t. |

Generated by Doxygen
Returns

0 on success, otherwise:
- EFAULT: It was not possible to access user space memory area at the specified address.
- ENODEV: No device with specified name exists.
- EINVAL: No valid baud rate, see `can_baudrate_t`.
- EDOM: Baud rate not possible.
- EAGAIN: Request could not be successfully fulfilled. Try again.

Tags

`task-unrestricted, might-switch`

Note

Setting the baud rate is a configuration task. It should be done deliberately or otherwise CAN messages will likely be lost.

Examples:

`rtcanconfig.c`

6.7.2.18 SIOCSCANCTRLMODE

#define SIOCSCANCTRLMODE _IOW(RTIOC_TYPE_CAN, 0x07, struct can_ifreq)

Set special controller modes.

Various special controller modes could be or'ed together (see `CAN_CTRLMODE` for further information).

Parameters

```
<table>
<thead>
<tr>
<th>in</th>
<th>arg</th>
</tr>
</thead>
<tbody>
<tr>
<td>arg</td>
<td>Pointer to interface request structure buffer (struct <code>can_ifreq</code>). <code>ifr_name</code> must hold a valid CAN interface name, <code>ifr_ifru</code> must be filled with an instance of <code>can_ctrlmode_t</code>.</td>
</tr>
</tbody>
</table>
```

Returns

0 on success, otherwise:
- EFAULT: It was not possible to access user space memory area at the specified address.
- ENODEV: No device with specified name exists.
- EINVAL: No valid baud rate, see `can_baudrate_t`.
- EAGAIN: Request could not be successfully fulfilled. Try again.

Tags

`task-unrestricted, might-switch`
Note
Setting special controller modes is a configuration task. It should be done deliberately or otherwise CAN messages will likely be lost.

Examples:
rtcancanfig.c.

6.7.2.19  SIOCSCANCUSTOMBITTIME

#define SIOCSCANCUSTOMBITTIME _IOW(RTIOC_TYPE_CAN, 0x03, struct can_ifreq)

Set custom bit time parameter.

Custom-bit time could be defined in various formats (see struct can_bittime).

Parameters

| in | arg | Pointer to interface request structure buffer (struct can_ifreq). ifr_name must hold a valid CAN interface name, ifr_ifru must be filled with an instance of struct can_bittime. |

Returns
0 on success, otherwise:
- -EFAULT: It was not possible to access user space memory area at the specified address.
- -ENOMEM: No device with specified name exists.
- -EBADADDR: No valid baud rate, see can_baudrate_t.
- -EAGAIN: Request could not be successfully fulfilled. Try again.

Tags
task-unrestricted, might-switch

Note
Setting the bit-time is a configuration task. It should be done deliberately or otherwise CAN messages will likely be lost.

Examples:
rtcancanfig.c.

6.7.2.20  SIOCSCANMODE

#define SIOCSCANMODE _IOW(RTIOC_TYPE_CAN, 0x05, struct can_ifreq)

Set operation mode of CAN controller.

See CAN controller modes for available modes.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>arg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pointer to interface request structure buffer (struct <code>can_ifreq</code>), <code>ifr_name</code> must hold a valid CAN interface name, <code>ifr_ifru</code> must be filled with an instance of <code>can_mode_t</code>.</td>
<td></td>
</tr>
</tbody>
</table>

Returns

0 on success, otherwise:
- -EFAULT: It was not possible to access user space memory area at the specified address.
- -ENODEV: No device with specified name exists.
- -EAGAIN: (CAN_MODE_START, CAN_MODE_STOP) Could not successfully set mode, hardware is busy. Try again.
- -EINVAL: (CAN_MODE_START) Cannot start controller, set baud rate first.
- -ENETDOWN: (CAN_MODE_SLEEP) Cannot go into sleep mode because controller is stopped or bus off.
- -EOPNOTSUPP: unknown mode

Tags

task-unrestricted, might-switch

Note

Setting a CAN controller into normal operation after a bus-off can take some time (128 occurrences of 11 consecutive recessive bits). In such a case, although this IOCTL will return immediately with success and `SIOCGCANSTATE` will report `CAN_STATE_ACTIVE`, bus-off recovery may still be in progress.

If a controller is bus-off, setting it into stop mode will return no error but the controller remains bus-off.

Examples:

`rtcanconfig.c`

---

6.7.2.21  SOL_CAN_RAW

#define SOL_CAN_RAW 103

CAN socket levels.

Used for `Sockopts` for the particular protocols.

Examples:

`can-rtt.c, rtcanrecv.c, and rtcsend.c`

6.7.3  Typedef Documentation
6.7.3.1  can_filter_t

typedef struct can_filter can_filter_t

Filter for reception of CAN messages.

This filter works as follows: A received CAN ID is AND'ed bitwise with `can_mask` and then compared to `can_id`. This also includes the `CAN_EFF_FLAG` and `CAN_RTR_FLAG` of `CAN_xxx_FLAG`. If this comparison is true, the message will be received by the socket. The logic can be inverted with the `can_id` flag `CAN_INV_FILTER`:

```c
if (can_id & CAN_INV_FILTER) {
    if ((received_can_id & can_mask) != (can_id & ~CAN_INV_FILTER))
        accept-message;
} else {
    if ((received_can_id & can_mask) == can_id)
        accept-message;
}
```

Multiple filters can be arranged in a filter list and set with `Sockopts`. If one of these filters matches a CAN ID upon reception of a CAN frame, this frame is accepted.

6.7.3.2  can_frame_t

typedef struct can_frame can_frame_t

Raw CAN frame.

Central structure for receiving and sending CAN frames.

Examples:

```
rtcanrecv.c.
```

6.7.4  Enumeration Type Documentation

6.7.4.1  CAN_BITTIME_TYPE

typedef enum CAN_BITTIME_TYPE

Supported CAN bit-time types.

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAN_BITTIME_STD</td>
<td>Standard bit-time definition according to Bosch.</td>
</tr>
<tr>
<td>CAN_BITTIME_BTR</td>
<td>Hardware-specific BTR bit-time definition.</td>
</tr>
</tbody>
</table>
6.7.4.2 CAN_MODE

enum CAN_MODE

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAN_MODE_STOP</td>
<td>Set controller in Stop mode (no reception / transmission possible)</td>
</tr>
<tr>
<td>CAN_MODE_START</td>
<td>Set controller into normal operation. Coming from stopped mode or bus off,</td>
</tr>
<tr>
<td></td>
<td>the controller begins with no errors in CAN_STATE_ACTIVE.</td>
</tr>
<tr>
<td>CAN_MODE_SLEEP</td>
<td>Set controller into Sleep mode.</td>
</tr>
<tr>
<td></td>
<td>This is only possible if the controller is not stopped or bus-off.</td>
</tr>
<tr>
<td></td>
<td>Notice that sleep mode will only be entered when there is no bus activity.</td>
</tr>
<tr>
<td></td>
<td>If the controller detects bus activity while &quot;sleeping&quot; it will go into</td>
</tr>
<tr>
<td></td>
<td>operating mode again.</td>
</tr>
<tr>
<td></td>
<td>To actively leave sleep mode again trigger CAN_MODE_START.</td>
</tr>
</tbody>
</table>

6.7.4.3 CAN_STATE

enum CAN_STATE

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAN_STATE_ERROR_ACTIVE</td>
<td>CAN controller is error active.</td>
</tr>
<tr>
<td>CAN_STATE_ACTIVE</td>
<td>CAN controller is active.</td>
</tr>
<tr>
<td>CAN_STATE_ERROR_WARNING</td>
<td>CAN controller is error active, warning level is reached.</td>
</tr>
<tr>
<td>CAN_STATE_ERROR_PASSIVE</td>
<td>CAN controller is error passive, warning level is reached.</td>
</tr>
<tr>
<td>CAN_STATE_BUS_PASSIVE</td>
<td>CAN controller is error passive.</td>
</tr>
<tr>
<td>CAN_STATE_BUS_OFF</td>
<td>CAN controller went into Bus Off.</td>
</tr>
<tr>
<td>CAN_STATE_SCANNING_BAUDRATE</td>
<td>CAN controller is scanning to get the baudrate.</td>
</tr>
<tr>
<td>CAN_STATE_STOPPED</td>
<td>CAN controller is in stopped mode.</td>
</tr>
<tr>
<td>CAN_STATE_SLEPPING</td>
<td>CAN controller is in Sleep mode.</td>
</tr>
</tbody>
</table>
The Real-Time Driver Model (RTDM) provides a unified interface to both users and developers of real-time device drivers.

Collaboration diagram for RTDM:

### Modules

- **RTDM User API**
  
  *Application interface to RTDM services.*

- **Driver programming interface**
  
  *RTDM driver programming interface.*

- **Device Profiles**
  
  *Pre-defined classes of real-time devices.*

### Typedefs

- `typedef uint64_t nanosecs_abs_t`
  
  *RTDM type for representing absolute dates.*

- `typedef int64_t nanosecs_rel_t`
  
  *RTDM type for representing relative intervals.*

### API Versioning

- `#define RTDM_API_VER 9`
  
  *Common user and driver API version.*

- `#define RTDM_API_MIN_COMPAT_VER 9`
  
  *Minimum API revision compatible with the current release.*
RTDM_TIMEOUT.xxx

Special timeout values

- `#define RTDM_TIMEOUT_INFINITE 0
  Block forever.
- `#define RTDM_TIMEOUT_NONE (-1)
  Any negative timeout means non-blocking.

6.8.1 Detailed Description

The Real-Time Driver Model (RTDM) provides a unified interface to both users and developers of real-time device drivers.

Specifically, it addresses the constraints of mixed RT/non-RT systems like Xenomai. RTDM conforms to POSIX semantics (IEEE Std 1003.1) where available and applicable.

API Revision: 8

6.8.2 Macro Definition Documentation

6.8.2.1 RTDM_TIMEOUT_INFINITE

`#define RTDM_TIMEOUT_INFINITE 0

Block forever.

6.8.2.2 RTDM_TIMEOUT_NONE

`#define RTDM_TIMEOUT_NONE (-1)

Any negative timeout means non-blocking.

6.8.3 Typedef Documentation
6.8.3.1 nanosecs_abs_t

typedef uint64_t nanosecs_abs_t

RTDM type for representing absolute dates.

Its base type is a 64 bit unsigned integer. The unit is 1 nanosecond.

Examples:
   rtcanrecv.c.

6.8.3.2 nanosecs_rel_t

typedef int64_t nanosecs_rel_t

RTDM type for representing relative intervals.

Its base type is a 64 bit signed integer. The unit is 1 nanosecond. Relative intervals can also encode the special timeouts "infinite" and "non-blocking", see RTDM_TIMEOUT_xxx.

Examples:
   rtcanrecv.c.
6.9  RTDM User API

Application interface to RTDM services.

Collaboration diagram for RTDM User API:

```
RTDM      RTDM User API
```

Files

- file rtdm.h
  
  *Real-Time Driver Model for Xenomai, user API header.*

6.9.1  Detailed Description

Application interface to RTDM services.

This is the upper interface of RTDM provided to application programs both in kernel and user space. Note that certain functions may not be implemented by every device. Refer to the Device Profiles for precise information.
6.10 Serial Devices

This is the common interface a RTDM-compliant serial device has to provide.

Collaboration diagram for Serial Devices:

![Collaboration diagram](Diagram.png)

Feel free to comment on this profile via the Xenomai mailing list xenomai@xenomai.org or directly to the author jan.kiszka@web.de.

**Profile Revision:** 3

Device Characteristics

- **Device Flags:** RTDM_NAMED_DEVICE, RTDM_EXCLUSIVE
- **Device Class:** RTDM_CLASS_SERIAL
- **Device Name:** "/dev/rtdm/rtser<N>", N \( \geq 0 \)

Supported Operations

- **Open**

Tags

- `secondary-only` Specific return values: none
- **Close**

Tags

- `secondary-only` Specific return values: none
- **IOCTL**
Tags

**task-unrestricted. See below**
Specific return values: see below

**Read**

Tags

**mode-unrestricted** Specific return values:
- -ETIMEDOUT
- -EINTR (interrupted explicitly or by signal)
- -EAGAIN (no data available in non-blocking mode)
- -EBADF (device has been closed while reading)
- -EIO (hardware error or broken bit stream)

**Write**

Tags

**mode-unrestricted** Specific return values:
- -ETIMEDOUT
- -EINTR (interrupted explicitly or by signal)
- -EAGAIN (no data written in non-blocking mode)
- -EBADF (device has been closed while writing)
6.11 Testing Devices

This group of devices is intended to provide in-kernel testing results.

Collaboration diagram for Testing Devices:

```
Device Profiles → Testing Devices
```

This group of devices is intended to provide in-kernel testing results.

Feel free to comment on this profile via the Xenomai mailing list xenomai@xenomai.org or directly to the author jan.kiszka@web.de.

**Profile Revision**: 2

Device Characteristics

- **Device Flags**: RTDM_NAMED_DEVICE
- **Device Class**: RTDM_CLASS_TESTING

Supported Operations

- **Open**

Tags

- **secondary-only** Specific return values: none

- **Close**

Tags

- **secondary-only** Specific return values: none

- **IOCTL**

Tags

- **task-unrestricted**. See TSTIOCTLs below
  Specific return values: see TSTIOCTLs below
6.12 Real-time IPC

Profile Revision: 1

Collaboration diagram for Real-time IPC:

![Collaboration Diagram](image)

Data Structures

- `struct rtipc_port_label`
  - Port label information structure.
- `struct sockaddr_ipc`
  - Socket address structure for the RTIPC address family.

Typedefs

- `typedef int16_t rtipc_port_t`
  - Port number type for the RTIPC address family.

Supported operations

Standard socket operations supported by the RTIPC protocols.

- `int socket__AF_RTIPC (int domain=AF_RTIPC, int type=SOCK_DGRAM, int protocol)`  
  - Create an endpoint for communication in the AF_RTIPC domain.
- `int close__AF_RTIPC (int sockfd)`  
  - Close a RTIPC socket descriptor.
- `int bind__AF_RTIPC (int sockfd, const struct sockaddr_ipc *addr, socklen_t addrlen)`  
  - Bind a RTIPC socket to a port.
- `int connect__AF_RTIPC (int sockfd, const struct sockaddr_ipc *addr, socklen_t addrlen)`  
  - Initiate a connection on a RTIPC socket.
- `int socketopt__AF_RTIPC (int sockfd, int level, int optname, const void *optval, socklen_t optlen)`  
  - Set options on RTIPC sockets.
- `int getsockopt__AF_RTIPC (int sockfd, int level, int optname, void *optval, socklen_t *optlen)`  
  - Get options on RTIPC sockets.
- `ssize_t sendmsg__AF_RTIPC (int sockfd, const struct msghdr *msg, int flags)`  
  - Send a message on a RTIPC socket.
- `ssize_t recvmsg__AF_RTIPC (int sockfd, struct msghdr *msg, int flags)`  
  - Receive a message from a RTIPC socket.
- `int getsockname__AF_RTIPC (int sockfd, struct sockaddr_ipc *addr, socklen_t *addrlen)`  
  - Get socket name.
- `int getpeername__AF_RTIPC (int sockfd, struct sockaddr_ipc *addr, socklen_t *addrlen)`  
  - Get socket peer.
RTIPC protocol list

protocols for the PF_RTIPC protocol family

- enum { IPCPROTO_IPC = 0, IPCPROTO_XDDP = 1, IPCPROTO_IDDP = 2, IPCPROTO_BUFP = 3 }

XDDP socket options

Setting and getting XDDP socket options.

- #define XDDP_LABEL 1
  XDDP label assignment.
- #define XDDP_POOLSZ 2
  XDDP local pool size configuration.
- #define XDDP_BUFSZ 3
  XDDP streaming buffer size configuration.
- #define XDDP_MONITOR 4
  XDDP monitoring callback.

XDDP events

Specific events occurring on XDDP channels, which can be monitored via the XDDP_MONITOR socket option.

- #define XDDP_EVTIN 1
  Monitor writes to the non real-time endpoint.
- #define XDDP_EVTOUT 2
  Monitor reads from the non real-time endpoint.
- #define XDDP_EVTDOWN 3
  Monitor close from the non real-time endpoint.
- #define XDDP_EVTNOBUF 4
  Monitor memory shortage for non real-time datagrams.

IDDP socket options

Setting and getting IDDP socket options.

- #define IDDP_LABEL 1
  IDDP label assignment.
- #define IDDP_POOLSZ 2
  IDDP local pool size configuration.
BUFP socket options

Setting and getting BUFP socket options.

- #define **BUFP_LABEL** 1
  
  BUFP label assignment.

- #define **BUFP_BUFSZ** 2
  
  BUFP buffer size configuration.

Socket level options

Setting and getting supported standard socket level options.

- #define **SO_SNDTIMEO** defined_by_kernel_header_file
  
  IPCPROTO_IDDP and IPCPROTO_BUFP protocols support the standard SO_SNDTIMEO socket option, from the SOL_SOCKET level.

- #define **SO_RCVTIMEO** defined_by_kernel_header_file
  
  All RTIPC protocols support the standard SO_RCVTIMEO socket option, from the SOL_SOCKET level.

6.12.1 Detailed Description

**Profile Revision:** 1

Device Characteristics

- **Device Flags:** RTDM_PROTOCOL_DEVICE
- **Protocol Family:** PF_RTIPC
- **Socket Type:** SOCK_DGRAM
- **Device Class:** RTDM_CLASS_RTIPC

6.12.2 Macro Definition Documentation

6.12.2.1 BUFP_BUFSZ

#define **BUFP_BUFSZ** 2

BUFP buffer size configuration.

All messages written to a BUFP socket are buffered in a single per-socket memory area. Configuring the size of such buffer prior to binding the socket to a destination port is mandatory.

It is not allowed to configure a buffer size after the socket was bound. However, multiple configuration calls are allowed prior to the binding; the last value set will be used.

**Note:** the buffer memory is obtained from the host allocator by the bind call.
### 6.12 Real-time IPC

#### 6.12.1 Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Level</th>
<th>Optname</th>
<th>Optval</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>SOL_BUFP</td>
<td>BUFP_BUFSZ</td>
<td>Pointer to a variable of type size_t, containing the required size of the buffer to reserve at binding time</td>
</tr>
<tr>
<td>in</td>
<td></td>
<td></td>
<td>sizeof(size_t)</td>
</tr>
</tbody>
</table>

#### 6.12.2 Returns

0 is returned upon success. Otherwise:

- EFAULT (Invalid data address given)
- EALREADY (socket already bound)
- EINVAL (optlen is invalid or ∗optval is zero)

**Calling context:**

RT/non-RT

**Examples:**

bufp-label.c, and bufp-readwrite.c.

### 6.12.2.2 BUFP_LABEL

#define BUFP_LABEL 1

**BUFP label assignment.**

ASCII label strings can be attached to BUFP ports, in order to connect sockets to them in a more descriptive way than using plain numeric port values.

When available, this label will be registered when binding, in addition to the port number (see BUFP port binding).

It is not allowed to assign a label after the socket was bound. However, multiple assignment calls are allowed prior to the binding; the last label set will be used.

#### Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Level</th>
<th>Optname</th>
<th>Optval</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>SOL_BUFP</td>
<td>BUFP_LABEL</td>
<td>Pointer to struct rtipc_port_label</td>
</tr>
<tr>
<td>in</td>
<td></td>
<td></td>
<td>sizeof(struct rtipc_port_label)</td>
</tr>
</tbody>
</table>
Returns

0 is returned upon success. Otherwise:

- -EFAULT (Invalid data address given)
- -EALREADY (socket already bound)
- -EINVAL (optlen is invalid)

Calling context:

RT/non-RT

Examples:

bufp-label.c.

6.12.2.3 IDDP_LABEL

#define IDDP_LABEL 1

IDDP label assignment.

ASCII label strings can be attached to IDDP ports, in order to connect sockets to them in a more
descriptive way than using plain numeric port values.

When available, this label will be registered when binding, in addition to the port number (see IDDP port
binding).

It is not allowed to assign a label after the socket was bound. However, multiple assignment calls are
allowed prior to the binding; the last label set will be used.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>level</th>
<th>SOL_IDDP</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>optname</td>
<td>IDDP_LABEL</td>
</tr>
<tr>
<td>in</td>
<td>optval</td>
<td>Pointer to struct rtipc_port_label</td>
</tr>
<tr>
<td>in</td>
<td>optlen</td>
<td>sizeof(struct rtipc_port_label)</td>
</tr>
</tbody>
</table>

Returns

0 is returned upon success. Otherwise:

- -EFAULT (Invalid data address given)
- -EALREADY (socket already bound)
- -EINVAL (optlen is invalid)
6.12 Real-time IPC

Calling context:

RT/non-RT

Examples:

iddp-label.c.

6.12.2.4 IDDP_POOLSZ

#define IDDP_POOLSZ 2

IDDP local pool size configuration.

By default, the memory needed to convey the data is pulled from Xenomai’s system pool. Setting a local pool size overrides this default for the socket.

If a non-zero size was configured, a local pool is allocated at binding time. This pool will provide storage for pending datagrams.

It is not allowed to configure a local pool size after the socket was bound. However, multiple configuration calls are allowed prior to the binding; the last value set will be used.

Note

: the pool memory is obtained from the host allocator by the bind call.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>level</th>
<th>SOL_IDDP</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>optname</td>
<td>IDDP_POOLSZ</td>
</tr>
<tr>
<td>in</td>
<td>optval</td>
<td>Pointer to a variable of type size_t, containing the required size of the local pool to reserve at binding time</td>
</tr>
<tr>
<td>in</td>
<td>optlen</td>
<td>sizeof(size_t)</td>
</tr>
</tbody>
</table>

Returns

0 is returned upon success. Otherwise:

- -EFAULT (Invalid data address given)
- -EALREADY (socket already bound)
- -EINVAL (optlen is invalid or *optval is zero)

Calling context:

RT/non-RT

Examples:

iddp-sendrecv.c.
6.12.2.5 SO_RCVTIMEO
#define SO_RCVTIMEO defined_by_kernel_header_file
All RTIPC protocols support the standard SO_RCVTIMEO socket option, from the SOL_SOCKET level.
See also
Examples:
xxdp-label.c.

6.12.2.6 SO_SNDTIMEO
#define SO_SNDTIMEO defined_by_kernel_header_file
IPCPROTO_IDDP and IPCPROTO_BUFP protocols support the standard SO_SNDTIMEO socket option, from the SOL_SOCKET level.
See also

6.12.2.7 XDDP_BUFSZ
#define XDDP_BUFSZ 3
XDDP streaming buffer size configuration.
In addition to sending datagrams, real-time threads may stream data in a byte-oriented mode through the port as well. This increases the bandwidth and reduces the overhead, when the overall data to send to the Linux domain is collected by bits, and keeping the message boundaries is not required.
This feature is enabled when a non-zero buffer size is set for the socket. In that case, the real-time data accumulates into the streaming buffer when MSG_MORE is passed to any of the send functions, until:
• the receiver from the Linux domain wakes up and consumes it,
• a different source port attempts to send data to the same destination port,
• MSG_MORE is absent from the send flags,
• the buffer is full,
whichever comes first.
Setting *optval to zero disables the streaming buffer, in which case all sendings are conveyed in separate datagrams, regardless of MSG_MORE.
Note
only a single streaming buffer exists per socket. When this buffer is full, the real-time data stops accumulating and sending operations resume in mere datagram mode. Accumulation may happen again after some or all data in the streaming buffer is consumed from the Linux domain endpoint.
The streaming buffer size may be adjusted multiple times during the socket lifetime; the latest configuration change will take effect when the accumulation resumes after the previous buffer was flushed.
6.12 Real-time IPC

Parameters

<table>
<thead>
<tr>
<th>In</th>
<th>Level</th>
<th>SOL_XDDP</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Optname</td>
<td>XDDP_BUFSZ</td>
</tr>
<tr>
<td>In</td>
<td>Optval</td>
<td>Pointer to a variable of type size_t, containing the required size of the streaming buffer</td>
</tr>
<tr>
<td>In</td>
<td>Optlen</td>
<td>sizeof(size_t)</td>
</tr>
</tbody>
</table>

Returns

0 is returned upon success. Otherwise:

- -EFAULT (Invalid data address given)
- -ENOMEM (Not enough memory)
- -EINVAL (optlen is invalid)

Calling context:

RT/non-RT

Examples:

xddp-stream.c.

6.12.2.8 XDDP_EVTDOWN

#define XDDP_EVTDOWN 3

Monitor close from the non real-time endpoint.

XDDP_EVTDOWN is sent when the non real-time endpoint is closed. The argument is always 0.

6.12.2.9 XDDP_EVTIN

#define XDDP_EVTIN 1

Monitor writes to the non real-time endpoint.

XDDP_EVTIN is sent when data is written to the non real-time endpoint the socket is bound to (i.e. via /dev/rtpN), which means that some input is pending for the real-time endpoint. The argument is the size of the incoming message.

Generated by Doxygen
6.12.2.10 XDDP_EVTNOBUF

#define XDDP_EVTNOBUF 4

Monitor memory shortage for non real-time datagrams.

XDDP_EVTNOBUF is sent when no memory is available from the pool to hold the message currently sent from the non real-time endpoint. The argument is the size of the failed allocation. Upon return from the callback, the caller will block and retry until enough space is available from the pool; during that process, the callback might be invoked multiple times, each time a new attempt to get the required memory fails.

6.12.2.11 XDDP_EVTOUT

#define XDDP_EVTOUT 2

Monitor reads from the non real-time endpoint.

XDDP_EVTOUT is sent when the non real-time endpoint successfully reads a complete message (i.e. via /dev/rtpN). The argument is the size of the outgoing message.

6.12.2.12 XDDP_LABEL

#define XDDP_LABEL 1

XDDP label assignment.

ASCII label strings can be attached to XDDP ports, so that opening the non-RT endpoint can be done by specifying this symbolic device name rather than referring to a raw pseudo-device entry (i.e. /dev/rtpN).

When available, this label will be registered when binding, in addition to the port number (see XDDP port binding).

It is not allowed to assign a label after the socket was bound. However, multiple assignment calls are allowed prior to the binding; the last label set will be used.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>level</td>
<td>SOL_XDDP</td>
</tr>
<tr>
<td>opname</td>
<td>XDDP_LABEL</td>
</tr>
<tr>
<td>optval</td>
<td>Pointer to struct rtipc_port_label</td>
</tr>
<tr>
<td>optlen</td>
<td>sizeof(struct rtipc_port_label)</td>
</tr>
</tbody>
</table>

Returns

0 is returned upon success. Otherwise:

- EFAULT (Invalid data address given)
- EALREADY (socket already bound)
6.12 Real-time IPC

- EINVAL (optlen invalid)

Calling context:
RT/non-RT

Examples:
xddp-label.c.

6.12.2.13 XDDP_MONITOR

#define XDDP_MONITOR 4

XDDP monitoring callback.

Other RTDM drivers may install a user-defined callback via the rtdm_setsockopt call from the inter-driver API, in order to collect particular events occurring on the channel.

This notification mechanism is particularly useful to monitor a channel asynchronously while performing other tasks.

The user-provided routine will be passed the RTDM file descriptor of the socket receiving the event, the event code, and an optional argument. Four events are currently defined, see XDDP_EVENTS.

The XDDP_EVTIN and XDDP_EVTOUT events are fired on behalf of a fully atomic context; therefore, care must be taken to keep their overhead low. In those cases, the Xenomai services that may be called from the callback are restricted to the set allowed to a real-time interrupt handler.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>level</th>
<th>SOL_XDDP</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>optname</td>
<td>XDDP_MONITOR</td>
</tr>
<tr>
<td>in</td>
<td>optval</td>
<td>Pointer to a pointer to function of type int (∗)(int fd, int event, long arg), containing the address of the user-defined callback. Passing a NULL callback pointer in optval disables monitoring.</td>
</tr>
<tr>
<td>in</td>
<td>optlen</td>
<td>sizeof(int (∗)(int fd, int event, long arg))</td>
</tr>
</tbody>
</table>

Returns

0 is returned upon success. Otherwise:

- EFAULT (Invalid data address given)
- EPERM (Operation not allowed from user-space)
- EINVAL (optlen is invalid)

Calling context:
RT/non-RT, kernel space only
6.12.2.14  XDDP_POOLSZ

#define XDDP_POOLSZ 2

XDDP local pool size configuration.

By default, the memory needed to convey the data is pulled from Xenomai's system pool. Setting a local pool size overrides this default for the socket.

If a non-zero size was configured, a local pool is allocated at binding time. This pool will provide storage for pending datagrams.

It is not allowed to configure a local pool size after the socket was bound. However, multiple configuration calls are allowed prior to the binding; the last value set will be used.

Note
- the pool memory is obtained from the host allocator by the bind call.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>level</th>
<th>SOL_XDDP</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>optname</td>
<td>XDDP_POOLSZ</td>
</tr>
<tr>
<td>in</td>
<td>optval</td>
<td>Pointer to a variable of type size_t, containing the required size of the local pool to reserve at binding time</td>
</tr>
<tr>
<td>in</td>
<td>optlen</td>
<td>sizeof(size_t)</td>
</tr>
</tbody>
</table>

Returns

0 is returned upon success. Otherwise:

- -EFAULT (Invalid data address given)
- -EALREADY (socket already bound)
- -EINVAL (optlen invalid or ∗optval is zero)

Calling context:
- RT/non-RT

Examples:
- xddp-echo.c.

6.12.3  Enumeration Type Documentation

6.12.3.1  anonymous enum

anonymous enum
**6.12 Real-time IPC**

<table>
<thead>
<tr>
<th>Enumerator</th>
<th>Default protocol (IDDP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPCPROTO_IPC</td>
<td>Cross-domain datagram protocol (RT &lt;&gt; non-RT). Real-time Xenomai threads and regular Linux threads may want to exchange data in a way that does not require the former to leave the real-time domain (i.e. primary mode). The RTDM-based XDDP protocol is available for this purpose. On the Linux domain side, pseudo-device files named /dev/rtp&lt;minor&gt; give regular POSIX threads access to non real-time communication endpoints, via the standard character-based I/O interface. On the Xenomai domain side, sockets may be bound to XDDP ports, which act as proxies to send and receive data to/from the associated pseudo-device files. Ports and pseudo-device minor numbers are paired, meaning that e.g. socket port 7 will proxy the traffic to/from /dev/rtp7. All data sent through a bound/connected XDDP socket via sendto(2) or write(2) will be passed to the peer endpoint in the Linux domain, and made available for reading via the standard read(2) system call. Conversely, all data sent using write(2) through the non real-time endpoint will be conveyed to the real-time socket endpoint, and made available to the recvfrom(2) or read(2) system calls.</td>
</tr>
<tr>
<td>IPCPROTO_XDDP</td>
<td>Intra-domain datagram protocol (RT &lt;&gt; RT). The RTDM-based IDDP protocol enables real-time threads to exchange datagrams within the Xenomai domain, via socket endpoints.</td>
</tr>
<tr>
<td>IPCPROTO_BUFP</td>
<td>Buffer protocol (RT &lt;&gt; RT, byte-oriented). The RTDM-based BUFP protocol implements a lightweight, byte-oriented, one-way Producer-Consumer data path. All messages written are buffered into a single memory area in strict FIFO order, until read by the consumer. This protocol always prevents short writes, and only allows short reads when a potential deadlock situation arises (i.e. readers and writers waiting for each other indefinitely).</td>
</tr>
</tbody>
</table>

### 6.12.4 Function Documentation

#### 6.12.4.1 bind__AF_RTIPC()

```
int bind__AF_RTIPC (  
    int sockfd,  
    const struct sockaddr_ipc * addr,  
    socklen_t addrlen )
```

Bind a RTIPC socket to a port.

Bind the socket to a destination port.

**Parameters**

| in | sockfd | The RTDM file descriptor obtained from the socket creation call. |
| in | addr | The address to bind the socket to (see struct sockaddr_ipc). The meaning of such address depends on the RTIPC protocol in use for the socket: |
• **IPCPROTO_XDDP**
  This action creates an endpoint for channelling traffic between the Xenomai and Linux domains.
  \texttt{sipc\_family} must be AF\_RTIPC, \texttt{sipc\_port} is either -1, or a valid free port number between 0 and \texttt{CONFIG\_XENO\_OPT\_PIPE\_NRDEV-1}.
  If \texttt{sipc\_port} is -1, a free port will be assigned automatically.
  Upon success, the pseudo-device /dev/rtpN will be reserved for this communication channel, where \textit{N} is the assigned port number. The non real-time side shall open this device to exchange data over the bound socket.

If a label was assigned (see \texttt{XDDP\_LABEL}) prior to binding the socket to a port, a registry link referring to the created pseudo-device will be automatically set up as /proc/xenomai/registry/rtipc/xddp/label, where \textit{label} is the label string passed to setsockopt() for the \texttt{XDDP\_LABEL} option.

• **IPCPROTO_IDDP**
  This action creates an endpoint for exchanging datagrams within the Xenomai domain.
  \texttt{sipc\_family} must be AF\_RTIPC, \texttt{sipc\_port} is either -1, or a valid free port number between 0 and \texttt{CONFIG\_XENO\_OPT\_IDDP\_NRPORT-1}.
  If \texttt{sipc\_port} is -1, a free port will be assigned automatically. The real-time peer shall connect to the same port for exchanging data over the bound socket.

If a label was assigned (see \texttt{IDDP\_LABEL}) prior to binding the socket to a port, a registry link referring to the assigned port number will be automatically set up as /proc/xenomai/registry/rtipc/iddp/label, where \textit{label} is the label string passed to setsockopt() for the \texttt{IDDP\_LABEL} option.

• **IPCPROTO_BUFP**
  This action creates an endpoint for a one-way byte stream within the Xenomai domain.
  \texttt{sipc\_family} must be AF\_RTIPC, \texttt{sipc\_port} is either -1, or a valid free port number between 0 and \texttt{CONFIG\_XENO\_OPT\_BUFP\_NRPORT-1}.
  If \texttt{sipc\_port} is -1, an available port will be assigned automatically. The real-time peer shall connect to the same port for exchanging data over the bound socket.

If a label was assigned (see \texttt{BUFP\_LABEL}) prior to binding the socket to a port, a registry link referring to the assigned port number will be automatically set up as /proc/xenomai/registry/rtipc/bufp/label, where \textit{label} is the label string passed to setsockopt() for the \texttt{BUFP\_LABEL} option.

### Parameters

| in: | \texttt{addrlen} | The size in bytes of the structure pointed to by \texttt{addr}. |

### Returns

In addition to the standard error codes for \texttt{bind(2)}, the following specific error code may be returned:

- \texttt{-EFAULT} (Invalid data address given)
- \texttt{-ENOMEM} (Not enough memory)
- \texttt{-EINVAL} (Invalid parameter)
- \texttt{-EADDRINUSE} (Socket already bound to a port, or no port available)
-EAGAIN (no registry slot available, check/raise CONFIG_XENO_OPT_REGISTRY_NRSL--OTS).

Calling context:

non-RT

6.12.4.2 close__AF_RTIPC()

int close__AF_RTIPC (  
    int sockfd )

Close a RTIPC socket descriptor.

Blocking calls to any of the sendmsg or recvmsg functions will be unblocked when the socket is closed and return with an error.

Parameters

| in   | sockfd | The socket descriptor to close. |

Returns

In addition to the standard error codes for close(2), the following specific error code may be returned: none

Calling context:

non-RT

6.12.4.3 connect__AF_RTIPC()

int connect__AF_RTIPC (  
    int sockfd,  
    const struct sockaddr_ipc * addr,  
    socklen_t addrlen )

Initiate a connection on a RTIPC socket.

Parameters

| in   | sockfd | The RTDM file descriptor obtained from the socket creation call. |
| in   | addr   | The address to connect the socket to (see struct sockaddr_ipc). |
• If sipc_port is a valid port for the protocol, it is used verbatim and the connection succeeds imme-
diately, regardless of whether the destination is bound at the time of the call.

• If sipc_port is -1 and a label was assigned to the socket, connect() blocks for the requested amount
  of time (see SO_RCVTIMEO) until a socket is bound to the same label via bind(2) (see XDDP←
  LABEL, IDDP_LABEL, BUFP_LABEL), in which case a connection is established between both
  endpoints.

• If sipc_port is -1 and no label was assigned to the socket, the default destination address is cleared,
  meaning that any subsequent write to the socket will return -EDESTADDRREQ, until a valid des-
  tination address is set via connect(2) or bind(2).

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>addrlen</th>
<th>The size in bytes of the structure pointed to by addr.</th>
</tr>
</thead>
</table>

Returns

In addition to the standard error codes for connect(2), the following specific error code may be
returned: none.

Calling context:

RT/non-RT

6.12.4.4 getpeername__AF_RTIPC()

int getpeername__AF_RTIPC (  
    int sockfd,  
    struct sockaddr_ipc * addr,  
    socklen_t * addrlen )

Get socket peer.

The name of the remote endpoint for the socket is copied back (see struct sockaddr_ipc). This is
the default destination address for messages sent on the socket. It can be set either explicitly via
connect(2), or implicitly via bind(2) if no connect(2) was called prior to binding the socket to a port,
in which case both the local and remote names are equal.

Returns

In addition to the standard error codes for getpeername(2), the following specific error code may be
returned: none.

Calling context:

RT/non-RT
6.12.4.5 getsockname__AF_RTIPC()

```c
int getsockname__AF_RTIPC (  
    int sockfd,  
    struct sockaddr_ipc * addr,  
    socklen_t * addrlen )
```

Get socket name.
The name of the local endpoint for the socket is copied back (see struct `sockaddr_ipc`).

Returns
In addition to the standard error codes for `getsockname(2)`, the following specific error code may be returned: none.

Calling context:
RT/non-RT

6.12.4.6 getsockopt__AF_RTIPC()

```c
int getsockopt__AF_RTIPC (  
    int sockfd,  
    int level,  
    int optname,  
    void * optval,  
    socklen_t * optlen )
```

Get options on RTIPC sockets.
These functions allow to get various socket options. Supported Levels and Options:

- Level SOL_SOCKET
- Level SOL_XDDP
- Level SOL_IDDP
- Level SOL_BUFP

Returns
In addition to the standard error codes for `getsockopt(2)`, the following specific error code may be returned: follow the option links above.

Calling context:
RT/non-RT

6.12.4.7 recvmsg__AF_RTIPC()

```c
ssize_t recvmsg__AF_RTIPC (  
    int sockfd,  
    struct msghdr * msg,  
    int flags )
```

Receive a message from a RTIPC socket.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>sockfd</th>
<th>The RTDM file descriptor obtained from the socket creation call.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>msg</td>
<td>The address the message header will be copied at.</td>
</tr>
<tr>
<td>in</td>
<td>flags</td>
<td>Operation flags:</td>
</tr>
</tbody>
</table>

- MSG_DONTWAIT  Non-blocking I/O operation. The caller will not be blocked whenever no message is immediately available for receipt at the time of the call, but will rather return with -EWOU→ LDBLOCK.

Note

IPCPROTO_BUFP does not allow for short reads and always returns the requested amount of bytes, except in one situation: whenever some writer is waiting for sending data upon a buffer full condition, while the caller would have to wait for receiving a complete message. This is usually the sign of a pathological use of the BUFP socket, like defining an incorrect buffer size via BUFP_B→ UFSZ. In that case, a short read is allowed to prevent a deadlock.

Returns

In addition to the standard error codes for recvmsg(2), the following specific error code may be returned: none.

Calling context:

RT

6.12.4.8 sendmsg__AF_RTIPC()

ssize_t sendmsg__AF_RTIPC (  
  int sockfd,  
  const struct msghdr * msg,  
  int flags )

Send a message on a RTIPC socket.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>sockfd</th>
<th>The RTDM file descriptor obtained from the socket creation call.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>msg</td>
<td>The address of the message header conveying the datagram.</td>
</tr>
<tr>
<td>in</td>
<td>flags</td>
<td>Operation flags:</td>
</tr>
</tbody>
</table>

- MSG_OOB  Send out-of-band message. For all RTIPC protocols except IPCPROTO_BUFP, sending out-of-band data actually means pushing them to the head of the receiving queue, so that the reader will always receive them before normal messages. IPCPROTO_BUFP does not support out-of-band sending.
• MSG_DONTWAIT Non-blocking I/O operation. The caller will not be blocked whenever the message cannot be sent immediately at the time of the call (e.g., memory shortage), but will rather return with -EWOULDBLOCK. Unlike other RTIPC protocols, IPCPROTO_XDDP accepts but never considers MSG_DONTWAIT since writing to a real-time XDDP endpoint is inherently a non-blocking operation.

• MSG_MORE Accumulate data before sending. This flag is accepted by the IPCPROTO_XDDP protocol only, and tells the send service to accumulate the outgoing data into an internal streaming buffer, instead of issuing a datagram immediately for it. See XDDP_BUFSZ for more.

Note
No RTIPC protocol allows for short writes, and only complete messages are sent to the peer.

Returns
In addition to the standard error codes for sendmsg(2), the following specific error code may be returned: none.

Calling context:
RT

6.12.4.9 setsockopt__AF_RTIPC()

int setsockopt__AF_RTIPC (  
    int sockfd,  
    int level,  
    int optname,  
    const void *optval,  
    socklen_t optlen )

Set options on RTIPC sockets.

These functions allow to set various socket options. Supported Levels and Options:

• Level SOL_SOCKET
• Level SOL_XDDP
• Level SOL_IDDP
• Level SOL_BUFP

Returns
In addition to the standard error codes for setsockopt(2), the following specific error code may be returned: follow the option links above.

Calling context:
non-RT
6.12.4.10  socket__AF_RTIPC()

```c
int socket__AF_RTIPC (  
    int domain = AF_RTIPC,  
    int type = SOCK_DGRAM,  
    int protocol  )
```

Create an endpoint for communication in the AF_RTIPC domain.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>domain</th>
<th>The communication domain. Must be AF_RTIPC.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>type</td>
<td>The socket type. Must be SOCK_DGRAM.</td>
</tr>
</tbody>
</table>
| in   | protocol | Any of IPCPROTO_XDDP, IPCPROTO_IDDP, or IPCPROTO_BUFP.  
                  IPCPROTO_IPC is also valid, and refers to the default RTIPC protocol, namely IPCPROTO_IDDP. |

Returns

In addition to the standard error codes for socket(2), the following specific error code may be returned:

- -ENOPROTOOPT (Protocol is known, but not compiled in the RTIPC driver). See RTIPC protocols for available protocols.

Calling context:

- non-RT
6.13 Asynchronous Procedure Calls

Services for scheduling function calls in the Linux domain.

Collaboration diagram for Asynchronous Procedure Calls:

![Collaboration Diagram](image)

Functions

- `int xnapc_alloc (const char *name, void (*)(void *)handler, void *cookie)`
  
  Allocate an APC slot.

- `void xnapc_free (int apc)`
  
  Releases an APC slot.

- `static void xnapc_schedule (int apc)`
  
  Schedule an APC invocation.

6.13.1 Detailed Description

Services for scheduling function calls in the Linux domain.

APC is the acronym for Asynchronous Procedure Call, a mean by which activities from the Xenomai domain can schedule deferred invocations of handlers to be run into the Linux domain, as soon as possible when the Linux kernel gets back in control.

Up to BITS_PER_LONG APC slots can be active at any point in time.

APC support is built upon the interrupt pipeline's virtual interrupt support.

6.13.2 Function Documentation

6.13.2.1 xnapc_alloc()

```c
int xnapc_alloc (  
    const char * name,  
    void (*)(void *)handler,  
    void * cookie  
)
```

Allocate an APC slot.

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

Any Linux kernel service which is callable from a regular Linux interrupt handler is in essence available to APC handlers.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>is a symbolic name identifying the APC which will get reported through the /proc/xenomai/apc interface. Passing NULL to create an anonymous APC is allowed.</td>
</tr>
<tr>
<td>handler</td>
<td>The address of the fault handler to call upon exception condition. The handle will be passed the cookie value unmodified.</td>
</tr>
<tr>
<td>cookie</td>
<td>A user-defined opaque pointer the APC handler receives as its sole argument.</td>
</tr>
</tbody>
</table>

Returns

a valid APC identifier is returned upon success, or a negative error code otherwise:

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

Tags

unrestricted

6.13.2.2 xnapc_free()

```c
void xnapc_free ( int apc )
```

Releases an APC slot.

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>apc</td>
<td>The APC id. to release, as returned by a successful call to the xnapc_alloc() service.</td>
</tr>
</tbody>
</table>

Tags

unrestricted

6.13.2.3 xnapc_schedule()

```c
static inline int xnapc_schedule ( int apc ) [inline], [static]
```

Schedule an APC invocation.
This service marks the APC as pending for the Linux domain, so that its handler will be called as soon as possible, when the Linux domain gets back in control.

When posted from the Linux domain, the APC handler is fired as soon as the interrupt mask is explicitly cleared by some kernel code. When posted from the Xenomai domain, the APC handler is fired as soon as the Linux domain is resumed, i.e. after Xenomai has completed all its pending duties.

Parameters

- `apc` - The APC id. to schedule.

This service can be called from:

- Any domain context, albeit the usual calling place is from the Xenomai domain.
6.14 In-kernel arithmetics

A collection of helpers performing arithmetics not implicitly available from kernel context via GCC helpers.

Collaboration diagram for In-kernel arithmetics:

![Collaboration Diagram](image)

Functions

- `unsigned long long xnarch_generic_full_divmod64 (unsigned long long a, unsigned long long b, unsigned long long *rem)`
  
  *Architecture-independent div64 operation with remainder.*

6.14.1 Detailed Description

A collection of helpers performing arithmetics not implicitly available from kernel context via GCC helpers.

Many of these routines enable 64bit arithmetics on 32bit systems. Xenomai architecture ports normally implement the performance critical ones in hand-crafted assembly code (see `kernel/cobalt/arch/<arch>/include/asm/xenomai/uapi/arith.h`).

6.14.2 Function Documentation

6.14.2.1 `xnarch_generic_full_divmod64()`

```c
unsigned long long xnarch_generic_full_divmod64 (  
  unsigned long long a,  
  unsigned long long b,  
  unsigned long long *rem )
```

*Architecture-independent div64 operation with remainder.*

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>a</code></td>
<td>dividend</td>
</tr>
<tr>
<td><code>b</code></td>
<td>divisor</td>
</tr>
<tr>
<td><code>rem</code></td>
<td>if non-NULL, a pointer to a 64bit variable for collecting the remainder from the division.</td>
</tr>
</tbody>
</table>
6.15 Buffer descriptor

Abstraction for copying data to/from different address spaces.

Collaboration diagram for Buffer descriptor:

![Collaboration diagram for Buffer descriptor](image)

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.
6.15.1 Detailed Description

Abstraction for copying data to/from different address spaces.

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:

```c
[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 = (__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:

```c
[Syscall implementation]
void rt_bulk_write_inner(struct xnbufd *bufd, void *data, size_t size)
{
    void *bulk = get_next_writable_bulk();
    xnbufd_copy_to_kmem(bufd, data, size);
    free_bulk(bulk);
    xnbufd_invalidate(bufd);
    return;
}
[Kernel wrapper for in-kernel calls]
void rt_bulk_write(void *ptr, size_t size)
{
    struct xnbufd bufd;
    void *data;
    size_t len;
    xnbufd_map_kwrite(&bufd, ptr, len);
    data = xnbufd_copy_to_kmem(bufd, NULL, len);
    rt_bulk_write_inner(&bufd, data, len);
    xnbufd_unmap_kwrite(&bufd);
}
[Userland trampoline for user syscalls]
void __rt_bulk_write(struct pt_regs *regs)
{
    struct xnbufd bufd;
    void __user *ptr;
    size_t size;
    ptr = (__user *)__xn_reg_arg1(regs);
    size = __xn_reg_arg2(regs);
    xnbufd_map_uwrite(&bufd, ptr, size);
    rt_bulk_write_inner(&bufd, ptr, size);
    xnbufd_unmap_uwrite(&bufd);
}
```
6.15 Buffer descriptor

```c
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);
}
```

```c
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;
}
```

```c
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;
}
```

6.15.2 Function Documentation

6.15.2.1 xnbufd_copy_from_kmem()

```c
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 immediately 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 `bufd`, 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.

Generated by Doxygen
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{bufd}</td>
<td>The address of the buffer descriptor covering the user memory to copy data to.</td>
</tr>
<tr>
<td>\texttt{from}</td>
<td>The start address of the kernel memory to copy from.</td>
</tr>
<tr>
<td>\texttt{len}</td>
<td>The length of the kernel memory to copy to \texttt{bufd}.</td>
</tr>
</tbody>
</table>

Returns

The number of bytes written so far to the memory area covered by \texttt{ubufd}. Otherwise,

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

Tags

\texttt{unrestricted}

Note

Calling this routine while holding the nklock and/or running with interrupts disabled is invalid, and doing so will trigger a debug assertion.

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

6.15.2.2 \texttt{xnbufd_copy_to_kmem()}

\begin{verbatim}
ssize_t xnbufd_copy_to_kmem (  
        void * to,  
        struct xnbufd * bufd,  
        size_t len )
\end{verbatim}

Copy memory covered by a buffer descriptor to kernel memory.

This routine copies \texttt{len} bytes from the area referred to by the buffer descriptor \texttt{bufd} to the kernel memory area \texttt{to}. \texttt{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 \texttt{bufd} refers to readable kernel area (i.e. see \texttt{xnbufd_map_kread()}), the copy is immediately and fully performed with no restriction.
- if \texttt{bufd} refers to a readable user area (i.e. see \texttt{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 \texttt{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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>to</code></td>
<td>The start address of the kernel memory to copy to.</td>
</tr>
<tr>
<td><code>bufd</code></td>
<td>The address of the buffer descriptor covering the user memory to copy data from.</td>
</tr>
<tr>
<td><code>len</code></td>
<td>The length of the user memory to copy from <code>bufd</code>.</td>
</tr>
</tbody>
</table>

Returns

The number of bytes read so far from the memory area covered by `bufd`. 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.

Tags

- `task-unrestricted`

Note

Calling this routine while holding the nklock and/or running with interrupts disabled is invalid, and doing so will trigger a debug assertion.

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

6.15.2.3  xnbud_invalidate()

```c
void xnbud_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 `xnbud_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, `xnbud_invalidate()` should be invoked on the error path when data may have been transferred to the carry over buffer.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bufd</code></td>
<td>The address of the buffer descriptor to invalidate.</td>
</tr>
</tbody>
</table>

Tags

- `unrestricted`
6.15.2.4  xnbufd_map_kread()

```c
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**

<table>
<thead>
<tr>
<th>bufd</th>
<th>The address of the buffer descriptor which will map a len bytes kernel memory area, starting from ptr.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ptr</td>
<td>The start of the kernel buffer to map.</td>
</tr>
<tr>
<td>len</td>
<td>The length of the kernel buffer starting at ptr.</td>
</tr>
</tbody>
</table>

**Tags**

- unrestricted

6.15.2.5  xnbufd_map_kwrite()

```c
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**

<table>
<thead>
<tr>
<th>bufd</th>
<th>The address of the buffer descriptor which will map a len bytes kernel memory area, starting from ptr.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ptr</td>
<td>The start of the kernel buffer to map.</td>
</tr>
<tr>
<td>len</td>
<td>The length of the kernel buffer starting at ptr.</td>
</tr>
</tbody>
</table>

**Tags**

- unrestricted
6.15.2.6  xnbufd_map_uread()

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. |

Tags

  task-unrestricted

6.15.2.7  xnbufd_map_uwrite()

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. |

Tags

  task-unrestricted
6.15.2.8 xnbufd_reset()

```c
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.

Tags

- unrestricted

6.15.2.9 xnbufd_unmap_kread()

```c
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 `bufd`.

Tags

- task-unrestricted

6.15.2.10 xnbufd_unmap_kwrite()

```c
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.
6.15 Buffer descriptor

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.

Tags

task-unrestricted

6.15.2.11 xnbufd_unmap_uread()

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.

Tags

task-unrestricted

Note

Calling this routine while holding the nklock and/or running with interrupts disabled is invalid, and doing so will trigger a debug assertion.

6.15.2.12 xnbufd_unmap_uwrite()

ssize_t xnbufd_unmap_uwrite ( struct xnbufd *bufd )

Generated by Doxygen
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`.

Tags

`task-unrestricted`

Note

Calling this routine while holding the nklock and/or running with interrupts disabled is invalid, and doing so will trigger a debug assertion.
6.16 Clock services

Collaboration diagram for Clock services:

```
Cobalt kernel ← Clock services
```

Functions

- `int xnclock_register (struct xnclock *clock, const cpumask_t *affinity)`
  
  Register a Xenomai clock.

- `void xnclock_deregister (struct xnclock *clock)`
  
  Deregister a Xenomai clock.

- `void xnclock_tick (struct xnclock *clock)`
  
  Process a clock tick.

- `void xnclock_adjust (struct xnclock *clock, xnsticks_t delta)`
  
  Adjust a clock time.

6.16.1 Detailed Description

6.16.2 Function Documentation

6.16.2.1 xnclock_adjust()

```
void xnclock_adjust ( 
    struct xnclock * clock, 
    xnsticks_t delta )
```

Adjust a clock time.

This service changes the epoch for the given clock by applying the specified tick delta on its wallclock offset.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>clock</code></td>
<td>The clock to adjust.</td>
</tr>
<tr>
<td><code>delta</code></td>
<td>The adjustment value expressed in nanoseconds.</td>
</tr>
</tbody>
</table>
Xenomai tracks the system time in `nclk`, as a monotonously increasing count of ticks since the epoch. The epoch is initially the same as the underlying machine time.

### 6.16.2.2 xnclock_deregister()

```c
void xnclock_deregister (  
    struct xnclock *clock  
)
```

Deregister a Xenomai clock.

This service uninstalls a Xenomai clock previously registered with `xnclock_register()`.

This service may be called once all timers driven by `clock` have been stopped.

**Parameters**

| clock | The clock to deregister. |

### 6.16.2.3 xnclock_register()

```c
int xnclock_register (  
    struct xnclock *clock,  
    const cpumask_t *affinity  
)
```

Register a Xenomai clock.

This service installs a new clock which may be used to drive Xenomai timers.

**Parameters**

| clock | The new clock to register. |
| affinity | The set of CPUs we may expect the backing clock device to tick on. As a special case, passing a NULL affinity mask means that timer IRQs cannot be seen as percpu events, in which case all outstanding timers will be maintained into a single global queue instead of percpu timer queues. |
6.16 Clock services

Tags

    secondary-only

6.16.2.4  xnclock_tick()

void xnclock_tick (  
    struct xnclock * clock )

Process a clock tick.

This routine processes an incoming clock event, firing elapsed timers as appropriate.

Parameters

    clock  The clock for which a new event was received.

Tags

    coreirq-only, atomic-entry

Note

    The current CPU must be part of the real-time affinity set unless the clock device has no percpu semantics, otherwise weird things may happen.
6.17 Debugging services

Collaboration diagram for Debugging services:

![Collaboration diagram]

6.17.1 Detailed Description
6.18 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.

Implementation constraints

- Minimum page size is $2^{\text{XNHEAP_MINLOG2}}$ (must be large enough to hold a pointer).
- Maximum page size is $2^{\text{XNHEAP_MAXLOG2}}$.
- Requested block size is rounded up to XNHEAP_MINLOG2.
- Requested block size larger than 2 times the XNHEAP_PAGESZ is rounded up to the next page boundary and obtained from the free page list. So we need a bucket for each power of two between XNHEAP_MINLOG2 and XNHEAP_MAXLOG2 inclusive, plus one to honor requests ranging from the maximum page size to twice this size.
6.18.2 Function Documentation

6.18.2.1 xnheap_alloc()

```c
void *xnheap_alloc (  
    struct xnheap *heap,  
    u32 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>heap</code></td>
<td>The descriptor address of the heap to get memory from.</td>
</tr>
<tr>
<td><code>size</code></td>
<td>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.</td>
</tr>
</tbody>
</table>

Returns

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

Tags

unrestricted

6.18.2.2 xnheap_destroy()

```c
void xnheap_destroy (  
    struct xnheap *heap )
```

Destroys a memory heap.

Destroys a memory heap.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>heap</code></td>
<td>The heap descriptor.</td>
</tr>
</tbody>
</table>
6.18 Dynamic memory allocation services

Tags

secondary-only

6.18.2.3 xnheap_free()

void xnheap_free (
    struct xnheap ∗ heap,
    void ∗ block )

Release a block to a memory heap.
Releases a memory block to a heap.

Parameters

<table>
<thead>
<tr>
<th>heap</th>
<th>The heap descriptor.</th>
</tr>
</thead>
<tbody>
<tr>
<td>block</td>
<td>The block to be returned to the heap.</td>
</tr>
</tbody>
</table>

Tags

unrestricted

6.18.2.4 xnheap_init()

int xnheap_init ( 
    struct xnheap ∗ heap,
    void ∗ membase,
    u32 size )

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

Parameters

<table>
<thead>
<tr>
<th>heap</th>
<th>The address of a heap descriptor to initialize.</th>
</tr>
</thead>
<tbody>
<tr>
<td>membase</td>
<td>The address of the storage area.</td>
</tr>
<tr>
<td>size</td>
<td>The size in bytes of the storage area. size must be a multiple of PAGE_SIZE and smaller than 2 Gb in the current implementation.</td>
</tr>
</tbody>
</table>

Returns

0 is returned upon success, or:
-EINVAL is returned if size is either:
  - not aligned on PAGE_SIZE
  - smaller than $2 \times \text{PAGE\_SIZE}$
  - greater than 2 Gb (XNHEAP\_MAXHEAPSZ)

-ENOMEM is returned upon failure of allocating the meta-data area used internally to maintain the heap.

Tags

secondary-only

6.18.2.5 xnheap_set_name()

void xnheap_set_name (  
    struct xnheap *heap,  
    const char *name,  
    ... )

Set the heap's name string.

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>heap</td>
<td>The address of a heap descriptor.</td>
</tr>
<tr>
<td>name</td>
<td>Name displayed in statistic outputs. This parameter can be a printk()-like format argument list.</td>
</tr>
</tbody>
</table>

Tags

task-unrestricted
6.19 Cobalt

Cobalt supplements the native Linux kernel in dual kernel configurations.

Collaboration diagram for Cobalt:

```
Analogy framework

Cobalt

POSIX interface

Cobalt kernel
```

Modules

- **Cobalt kernel**
  
  *The Cobalt core is a co-kernel which supplements the Linux kernel for delivering real-time services with very low latency.*

- **Analogy framework**
  
  *A RTDM-based interface for implementing DAQ card drivers.*

- **POSIX interface**
  
  *The Cobalt/POSIX interface is an implementation of a subset of the Single Unix specification over the Cobalt core.*

6.19.1 Detailed Description

Cobalt supplements the native Linux kernel in dual kernel configurations.

It deals with all time-critical activities, such as handling interrupts, and scheduling real-time threads. The Cobalt kernel has higher priority over all the native kernel activities.

Cobalt provides an implementation of the POSIX and RTDM interfaces based on a set of generic RTOS building blocks.
6.20 Cobalt kernel

The Cobalt core is a co-kernel which supplements the Linux kernel for delivering real-time services with very low latency.

Collaboration diagram for Cobalt kernel:

Modules
- Asynchronous Procedure Calls

Generated by Doxygen
6.20 Cobalt kernel

Services for scheduling function calls in the Linux domain.

- **In-kernel arithmetics**
  A collection of helpers performing arithmetics not implicitly available from kernel context via GCC helpers.

- **Buffer descriptor**
  Abstraction for copying data to/from different address spaces.

- **Clock services**
- **Debugging services**
- **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.

- **Interrupt management**
- **Locking services**
  The Xenomai core deals with concurrent activities from two distinct kernels running side-by-side.

- **Lightweight key-to-object mapping service**
  A map is a simple indexing structure which associates unique integer keys with pointers to objects.

- **Registry services**
  The registry provides a mean to index object descriptors on unique alphanumeric keys.

- **Thread scheduling control**
- **Synchronous I/O multiplexing**
  This module implements the services needed for implementing the POSIX select() service, or any other event multiplexing services.

- **Thread synchronization services**
- **Thread services**
- **Timer services**
  The Xenomai timer facility depends on a clock source (xnclock) for scheduling the next activation times.

- **Virtual file services**
  Virtual files provide a mean to export Xenomai object states to user-space, based on common kernel interfaces.

6.20.1 Detailed Description

The Cobalt core is a co-kernel which supplements the Linux kernel for delivering real-time services with very low latency.

It implements a set of generic RTOS building blocks, which the Cobalt/POSIX and Cobalt/RTDM APIs are based on. Cobalt has higher priority over the Linux kernel activities.

6.20.1.1 Dual kernel service tags

The Cobalt kernel services may be restricted to particular calling contexts, or entail specific side-effects.

To describe this information, each service documented by this section bears a set of tags when applicable.

The table below matches the tags used throughout the documentation with the description of their meaning for the caller.

**Context tags**
<table>
<thead>
<tr>
<th>Tag</th>
<th>Context on entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>primary-only</td>
<td>Must be called from a Cobalt task in primary mode</td>
</tr>
<tr>
<td>primary-timed</td>
<td>Requires a Cobalt task in primary mode if timed</td>
</tr>
<tr>
<td>coreirq-only</td>
<td>Must be called from a Cobalt IRQ handler</td>
</tr>
<tr>
<td>secondary-only</td>
<td>Must be called from a Cobalt task in secondary mode or regular Linux task</td>
</tr>
<tr>
<td>rtdm-task</td>
<td>Must be called from a RTDM driver task</td>
</tr>
<tr>
<td>mode-unrestricted</td>
<td>May be called from a Cobalt task in either primary or secondary mode</td>
</tr>
<tr>
<td>task-unrestricted</td>
<td>May be called from a Cobalt or regular Linux task indifferently</td>
</tr>
<tr>
<td>unrestricted</td>
<td>May be called from any context previously described</td>
</tr>
<tr>
<td>atomic-entry</td>
<td>Caller must currently hold the big Cobalt kernel lock (nklock)</td>
</tr>
</tbody>
</table>

### Possible side-effects

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>might-switch</td>
<td>The Cobalt kernel may switch context</td>
</tr>
</tbody>
</table>
6.21 Interrupt management

Collaboration diagram for Interrupt management:

```
Cobalt kernel    Interrupt management
```

Functions

- **void xnintr_destroy (struct xnintr *intr)**
  
  *Destroy an interrupt descriptor.*

- **int xnintr_attach (struct xnintr *intr, void *cookie)**
  
  *Attach an interrupt descriptor.*

- **void xnintr_detach (struct xnintr *intr)**
  
  *Detach an interrupt descriptor.*

- **void xnintr_enable (struct xnintr *intr)**
  
  *Enable an interrupt line.*

- **void xnintr_disable (struct xnintr *intr)**
  
  *Disable an interrupt line.*

- **void xnintr_affinity (struct xnintr *intr, cpumask_t cpumask)**
  
  *Set processor affinity of interrupt.*

- **int xnintr_init (struct xnintr *intr, const char *name, unsigned int irq, xnisr_t isr, xniack_t iack, int flags)**
  
  *Initialize an interrupt descriptor.*

6.21.1 Detailed Description

6.21.2 Function Documentation

6.21.2.1 xnintr_affinity()

```
void xnintr_affinity (  
    struct xnintr * intr,  
    cpumask_t cpumask )
```

*Set processor affinity of interrupt.*

Restricts the IRQ line associated with the interrupt descriptor *intr* to be received only on processors which bits are set in *cpumask.*
Parameters

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>intr</td>
<td>The address of the interrupt descriptor.</td>
</tr>
<tr>
<td>cpumask</td>
<td>The new processor affinity.</td>
</tr>
</tbody>
</table>

Note

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

Tags

secondary-only

6.21.2.2  `xnintr_attach()`

```c
int xnintr_attach (         
    struct xnintr * intr,  
    void * cookie      
)
```

Attach an interrupt descriptor.

Attach an interrupt descriptor previously initialized by `xnintr_init()`. This operation registers the descriptor at the interrupt pipeline, but does not enable the interrupt line yet. A call to `xnintr_enable()` is required to start receiving IRQs from the interrupt line associated to the descriptor.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>intr</td>
<td>The address of the interrupt descriptor to attach.</td>
</tr>
<tr>
<td>cookie</td>
<td>A user-defined opaque value which is stored into the descriptor for further retrieval by the interrupt handler.</td>
</tr>
</tbody>
</table>

Returns

0 is returned on success. Otherwise:

- `-EINVAL` is returned if an error occurred while attaching the descriptor.
- `-EBUSY` is returned if the descriptor was already attached.

Note

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

Tags

secondary-only

Note

Attaching an interrupt descriptor resets the tracked number of IRQ receipts to zero.
6.21.2.3  xnintr_destroy()

```c
void xnintr_destroy(
    struct xnintr *intr )
```

Destroy an interrupt descriptor.

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

Parameters

- `intr` The address of the interrupt descriptor to destroy.

Tags

- secondary-only

References `xnintr_detach()`.

6.21.2.4  xnintr_detach()

```c
void xnintr_detach(
    struct xnintr *intr )
```

Detach an interrupt descriptor.

This call unregisters an interrupt descriptor previously attached by `xnintr_attach()` from the interrupt pipeline. Once detached, the associated interrupt line is disabled, but the descriptor remains valid. The descriptor can be attached anew by a call to `xnintr_attach()`.

Parameters

- `intr` The address of the interrupt descriptor to detach.

Note

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

Tags

- secondary-only

Referenced by `xnintr_destroy()`.
6.21.2.5  xnintr_disable()

void xnintr_disable (  
    struct xnintr * intr  )

Disable an interrupt line.
Disables the interrupt line associated with an interrupt descriptor.

Parameters

| intr | The address of the interrupt descriptor. |

Tags

- secondary-only

6.21.2.6  xnintr_enable()

void xnintr_enable (  
    struct xnintr * intr  )

Enable an interrupt line.
Enables the interrupt line associated with an interrupt descriptor.

Parameters

| intr | The address of the interrupt descriptor. |

Tags

- secondary-only

6.21.2.7  xnintr_init()

int xnintr_init (  
    struct xnintr * intr,  
    const char * name,  
    unsigned int irq,  
    xnisr_t isr,  
    xniack_t iack,  
    int flags )
Initialize an interrupt descriptor.

When an interrupt occurs on the given irq line, the interrupt service routine isr is fired in order to deal with the hardware event. The interrupt handler may call any non-blocking service from the Cobalt core.

Upon receipt of an IRQ, the interrupt handler isr is immediately called on behalf of the interrupted stack context, the rescheduling procedure is locked, and the interrupt line is masked in the system interrupt controller chip. Upon return, the status of the interrupt handler is checked for the following bits:

- XN_IRQ_HANDLED indicates that the interrupt request was successfully handled.
- XN_IRQ_NONE indicates the opposite to XN_IRQ_HANDLED, meaning that no interrupt source could be identified for the ongoing request by the handler.

In addition, one of the following bits may be present in the status:

- XN_IRQ_DISABLE tells the Cobalt core to disable the interrupt line before returning from the interrupt context.
- XN_IRQ_PROPAGATE propagates the IRQ event down the interrupt pipeline to Linux. Using this flag is strongly discouraged, unless you fully understand the implications of such propagation.

Warning

The handler should not use these bits if it shares the interrupt line with other handlers in the real-time domain. When any of these bits is detected, the interrupt line is left masked.

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>intr</td>
<td>The address of a descriptor the Cobalt core will use to store the interrupt-specific data.</td>
</tr>
<tr>
<td>name</td>
<td>An ASCII string standing for the symbolic name of the interrupt or NULL.</td>
</tr>
<tr>
<td>irq</td>
<td>The IRQ line number associated with the interrupt descriptor. This value is architecture-dependent. An interrupt descriptor must be attached to the system by a call to xnintr_attach() before irq events can be received.</td>
</tr>
<tr>
<td>isr</td>
<td>The address of an interrupt handler, which is passed the address of the interrupt descriptor receiving the IRQ.</td>
</tr>
<tr>
<td>iack</td>
<td>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.</td>
</tr>
<tr>
<td>flags</td>
<td>A set of creation flags affecting the operation. The valid flags are:</td>
</tr>
</tbody>
</table>

- XN_IRQTYPE_SHARED enables IRQ-sharing with other interrupt objects.
- XN_IRQTYPE_EDGE is an additional flag need to be set together with XN_IRQTYPE_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.

Tags

secondary-only

Referenced by xntimer_grab_hardware().
6.22 Locking services

The Xenomai core deals with concurrent activities from two distinct kernels running side-by-side. When interrupts are involved, the services from this section control the hard interrupt state exclusively, for protecting against processor-local or SMP concurrency.

Note

In a dual kernel configuration, hard interrupts are gated by the CPU. When enabled, hard interrupts are immediately delivered to the Xenomai core if they belong to a real-time source, or deferred until enabled by a second-stage virtual interrupt mask, if they belong to regular Linux devices/sources.

6.22.2 Macro Definition Documentation

6.22.2.1 splexit

#define splexit(x) ipipe_restore_head(x & 1)

Restore the saved hard interrupt state on the local processor.
6.22.2.2  splhigh

#define splhigh(
    x ) ((x) = ipipe_test_and_stall_head() & 1)

Hard disable interrupts on the local processor, saving previous state.

Parameters

| in  | x | The context variable previously updated by splhigh() |

6.22.2.3  spltest

#define spltest( ) ipipe_test_head()

Test hard interrupt state on the local processor.

Returns

Zero if the local processor currently accepts interrupts, non-zero otherwise.

Referenced by rtdm_fd_select(), and rtdm_lock_get().
6.23 Lightweight key-to-object mapping service

A map is a simple indexing structure which associates unique integer keys with pointers to objects.

Collaboration diagram for Lightweight key-to-object mapping service:

```
    Cobalt kernel -------- Lightweight key-to-object mapping service
```

Functions

- struct xnmap * `xnmap_create` (int nkeys, int reserve, int offset)
  
  Create a map.

- void `xnmap_delete` (struct xnmap *map)
  
  Delete a map.

- int `xnmap_enter` (struct xnmap *map, int key, void *objaddr)
  
  Index an object into a map.

- int `xnmap_remove` (struct xnmap *map, int key)
  
  Remove an object reference from a map.

- static void * `xnmap_fetch_nocheck` (struct xnmap *map, int key)
  
  Search an object into a map - unchecked form.

- static void * `xnmap_fetch` (struct xnmap *map, int key)
  
  Search an object into a map.

6.23.1 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 objects, 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).

6.23.2 Function Documentation
6.23.2.1  xnmap_create()

```c
struct xnmap * 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>nkeys</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>reserve</strong></td>
<td>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 <code>xnmap_enter()</code> 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 <code>reserve</code>, keeping the low key values for the reserved space. For instance, passing <code>reserve = 32</code> would cause the index range <code>[ 0 .. 31 ]</code> to be kept for reserved keys. When non-zero, <code>reserve</code> is rounded to the next multiple of BITS_PER_LONG. If <code>reserve</code> is zero no reservation will be available from the map.</td>
</tr>
<tr>
<td><strong>offset</strong></td>
<td>The lowest key value <code>xnmap_enter()</code> will return to the caller. Key values will be in the range <code>[ 0 + offset .. nkeys + offset - 1 ]</code>. Negative offsets are valid.</td>
</tr>
</tbody>
</table>

Returns

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

Tags

`task-unrestricted`

6.23.2.2  xnmap_delete()

```c
void xnmap_delete (  
    struct xnmap * map )
```

Delete a map.

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>map</strong></td>
<td>The address of the map to delete.</td>
</tr>
</tbody>
</table>
6.23 Lightweight key-to-object mapping service

Tags

- task-unrestricted

6.23.2.3  xnmap_enter()

```c
int xnmap_enter (  
    struct xnmap * map,  
    int key,  
    void * objaddr );
```

Index an object into a map.

Insert a new object into the given map.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>map</td>
<td>The address of the map to insert into.</td>
</tr>
<tr>
<td>key</td>
<td>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.</td>
</tr>
<tr>
<td>objaddr</td>
<td>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.</td>
</tr>
</tbody>
</table>

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.

Tags

- unrestricted

6.23.2.4  xnmap_fetch()

```c
void xnmap_fetch (  
    struct xnmap * 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.

Tags

- unrestricted

---

### 6.23.2.5 `xnmap_fetch_nocheck()`

```c
void xnmap_fetch_nocheck (  
    struct xnmap * 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`.

Tags

- unrestricted

---

### 6.23.2.6 `xnmap_remove()`

```c
int xnmap_remove (  
    struct xnmap * map,  
    int key )
```

Remove an object reference from a map.

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

<table>
<thead>
<tr>
<th>$map$</th>
<th>The address of the map to remove from.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$key$</td>
<td>The key the object reference to be removed is indexed on.</td>
</tr>
</tbody>
</table>

Returns

0 is returned on success. Otherwise:

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

Tags

unrestricted
6.24 Registry services

The registry provides a mean to index object descriptors on unique alphanumeric keys.

Collaboration diagram for Registry services:

```
Cobalt kernel ——— Registry services
```

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.
- static void * `xnregistry_lookup` (xnhandle_t handle, unsigned long *cstamp_r)
  * Find a real-time object into the registry.
- int `xnregistry_unlink` (const char *key)
  * Turn a named object into an anonymous object.

6.24.1 Detailed Description

The registry provides a mean to index object descriptors on unique alphanumeric keys.

When labeled this way, an 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.

6.24.2 Function Documentation

6.24.2.1 `xnregistry_bind()`

```c
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

<table>
<thead>
<tr>
<th><strong>key</strong></th>
<th>A valid NULL-terminated string which identifies the object to bind to.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>timeout</strong></td>
<td>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_REALTIME 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.</td>
</tr>
<tr>
<td><strong>timeout_mode</strong></td>
<td>The mode of the timeout parameter. It can either be set to XN_RELATIVE, XN_ABSOLUTE, or XN_REALTIME (see also xntimer_start()).</td>
</tr>
<tr>
<td><strong>phandle</strong></td>
<td>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.</td>
</tr>
</tbody>
</table>

### Returns

0 is returned upon success. Otherwise:

- EINVAL is returned if key is NULL.
- EINTR is returned if xnthread_unblock() 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.

### Tags

- primary-only, might-switch

### Note

xnregistry_bind() only returns the index portion of a handle, which might include other fixed bits to be complete (e.g. XNSYNCH_PSHARED). The caller is responsible for completing the handle returned with those bits if applicable, depending on the context.

### 6.24.2.2 xnregistry_enter()

```c
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 or NULL string if the object shall only occupy a registry slot for handle-based lookups. The slash character is not accepted in key if pnode is non-NULL. |
|-----------------
| 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 is NULL.
- -EINVAL if pnode is non-NULL, and key points to a valid string containing a '/' 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.

Tags

unrestricted, might-switch, atomic-entry

6.24.2.3 xnregistry_lookup()

void *xnregistry_lookup (  
  xnhandle_t handle,  
  unsigned long *cstamp_r ) [inline], [static]

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. Optionally, it also copies back the object's creation stamp which is unique across object registration calls.

Parameters

| handle | The generic handle of the object to fetch. |
|-----------------
| cstamp_r | If not-NULL, the object's creation stamp will be copied to this memory area. |
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.

Tags

unrestricted

6.24.2.4  xnregistry_remove()

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.

Tags

unrestricted

6.24.2.5  xnregistry_unlink()

int xnregistry_unlink (  
    const char * key )

Turn a named object into an anonymous object.

Tags

unrestricted
Module Documentation

6.25  Driver programming interface

RTDM driver programming interface.

Collaboration diagram for Driver programming interface:

- Interrupt Management Services
- Utility Services
- Timer Services
- Synchronisation Services
- Clock Services
- Task Services
- Non-Real-Time Signalling Services
- Device Registration Services
- Driver to driver services

Modules

- Driver to driver services
  
  *Inter-driver interface.*
- Device Registration Services
- Clock Services
- Task Services
- Timer Services
- Synchronisation Services
- Interrupt Management Services
- Non-Real-Time Signalling Services
  
  *These services provide a mechanism to request the execution of a specified handler in non-real-time context.*
- Utility Services

6.25.1  Detailed Description

RTDM driver programming interface.
6.26 Driver to driver services

Inter-driver interface.

Collaboration diagram for Driver to driver services:

![Collaboration diagram for Driver to driver services](image)

Functions

- int `rdm_open` (const char *path, int oflag,....)
  *Open a device.*
- int `rdm_socket` (int protocol_family, int socket_type, int protocol)
  *Create a socket.*
- int `rdm_close` (int fd)
  *Close a device or socket.*
- int `rdm_ioctl` (int fd, int request,...)
  *Issue an IOCTL.*
- ssize_t `rdm_read` (int fd, void *buf, size_t nbyte)
  *Read from device.*
- ssize_t `rdm_write` (int fd, const void *buf, size_t nbyte)
  *Write to device.*
- ssize_t `rdm_recvmsg` (int fd, struct user_msghdr *msg, int flags)
  *Receive message from socket.*
- ssize_t `rdm_recvfrom` (int fd, void *buf, size_t len, int flags, struct sockaddr *from, socklen_t *fromlen)
  *Receive message from socket.*
- ssize_t `rdm_recv` (int fd, void *buf, size_t len, int flags)
  *Receive message from socket.*
- ssize_t `rdm_sendmsg` (int fd, const struct user_msghdr *msg, int flags)
  *Transmit message to socket.*
- ssize_t `rdm_sendto` (int fd, const void *buf, size_t len, int flags, const struct sockaddr *to, socklen_t *tolen)
  *Transmit message to socket.*
- ssize_t `rdm_send` (int fd, const void *buf, size_t len, int flags)
  *Transmit message to socket.*
- int `rdm_bind` (int fd, const struct sockaddr *my_addr, socklen_t addrlen)
  *Bind to local address.*
- int `rdm_connect` (int fd, const struct sockaddr *serv_addr, socklen_t addrlen)
  *Connect to remote address.*
- int `rdm_listen` (int fd, int backlog)
  *Listen to incoming connection requests.*
• int rtdm_accept (int fd, struct sockaddr *addr, socklen_t *addrlen)
  Accept a connection request.
• int rtdm_shutdown (int fd, int how)
  Shut down parts of a connection.
• int rtdm_getsockopt (int fd, int level, int optname, void *optval, socklen_t *optlen)
  Get socket option.
• int rtdm_setsockopt (int fd, int level, int optname, const void *optval, socklen_t optlen)
  Set socket option.
• int rtdm_getsockname (int fd, struct sockaddr *name, socklen_t *namelen)
  Get local socket address.
• int rtdm_getpeername (int fd, struct sockaddr *name, socklen_t *namelen)
  Get socket destination address.

6.26.1 Detailed Description

Inter-driver interface.

6.26.2 Function Documentation

6.26.2.1 rtdm_accept()

```c
int rtdm_accept (       
    int fd,           
    struct sockaddr * addr,       
    socklen_t * addrlen )
```

Accept a connection request.

Accept connection requests.

Refer to `rtdm_accept()` for parameters and return values. Action depends on driver implementation, see Device Profiles.

Tags

mode-unrestricted, might-switch

Parameters

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>fd</code> File descriptor as returned by <code>rtdm_socket()</code></td>
</tr>
<tr>
<td>out</td>
<td><code>addr</code> Buffer for remote address</td>
</tr>
<tr>
<td>in,out</td>
<td><code>addrlen</code> Address buffer size</td>
</tr>
</tbody>
</table>
Returns

0 on success, otherwise negative error code

Action depends on driver implementation, see Device Profiles.

See also


Tags

mode-unrestricted, might-switch

6.26.2.2  rtdm_bind()

int rtdm_bind (  
    int fd,  
    const struct sockaddr * my_addr,  
    socklen_t addrlen )

Bind to local address.

Refer to rtdm_bind() for parameters and return values. Action depends on driver implementation, see Device Profiles.

Tags

task-unrestricted, might-switch

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>fd</th>
<th>File descriptor as returned by rtdm_socket()</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>my_addr</td>
<td>Address buffer</td>
</tr>
<tr>
<td>in</td>
<td>addrlen</td>
<td>Address buffer size</td>
</tr>
</tbody>
</table>

Returns

0 on success, otherwise negative error code

Action depends on driver implementation, see Device Profiles.

See also


Tags

mode-unrestricted, might-switch
6.26.2.3 rtdm_close()

```c
int rtdm_close ( int fd )
```

Close a device or socket.

Refer to `rtdm_close()` for parameters and return values. Action depends on driver implementation, see Device Profiles.

Tags

`secondary-only, might-switch`

Parameters

| in | fd | File descriptor as returned by rtdm_open() or rtdm_socket() |

Returns

0 on success, otherwise a negative error code.

Note

If the matching `rtdm_open()` or `rtdm_socket()` call took place in non-real-time context, `rtdm_close()` must be issued within non-real-time as well. Otherwise, the call will fail.

Action depends on driver implementation, see Device Profiles.

See also

`close()` in IEEE Std 1003.1, [http://www.opengroup.org/onlinepubs/009695399](http://www.opengroup.org/onlinepubs/009695399)

Tags

`secondary-only, might-switch`

6.26.2.4 rtdm_connect()

```c
int rtdm_connect ( int fd,  
                    const struct sockaddr * serv_addr,  
                    socklen_t addrlen )
```

Connect to remote address.

Refer to `rtdm_connect()` for parameters and return values. Action depends on driver implementation, see Device Profiles.

Tags

`mode-unrestricted, might-switch`
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>fd</th>
<th>File descriptor as returned by rtdm_socket()</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>serv_addr</td>
<td>Address buffer</td>
</tr>
<tr>
<td>in</td>
<td>addrlen</td>
<td>Address buffer size</td>
</tr>
</tbody>
</table>

Returns

0 on success, otherwise negative error code

Action depends on driver implementation, see Device Profiles.

See also

connect() in IEEE Std 1003.1, http://www.opengroup.org/onlinepubs/009695399

Tags

mode-unrestricted, might-switch

6.26.2.5 rtdm_getpeername()

int rtdm_getpeername (  
    int fd,  
    struct sockaddr * name,  
    socklen_t * namelen )

Get socket destination address.

Refer to rtdm_getpeername() for parameters and return values. Action depends on driver implementation, see Device Profiles.

Tags

task-unrestricted, might-switch

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>fd</th>
<th>File descriptor as returned by rtdm_socket()</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>name</td>
<td>Address buffer</td>
</tr>
<tr>
<td>in,out</td>
<td>namelen</td>
<td>Address buffer size</td>
</tr>
</tbody>
</table>

Returns

0 on success, otherwise negative error code

Action depends on driver implementation, see Device Profiles.
See also


Tags

task-unrestricted, might-switch

6.26.2.6 rtdm_getsockname()

int rtdm_getsockname (  
   int fd,  
   struct sockaddr * name,  
   socklen_t * namelen )

Get local socket address.

Refer to rtdm_getsockname() for parameters and return values. Action depends on driver implementation, see Device Profiles.

Tags

task-unrestricted, might-switch

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>fd</th>
<th>File descriptor as returned by rtdm_socket()</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>name</td>
<td>Address buffer</td>
</tr>
<tr>
<td>in,out</td>
<td>namelen</td>
<td>Address buffer size</td>
</tr>
</tbody>
</table>

Returns

0 on success, otherwise negative error code

Action depends on driver implementation, see Device Profiles.

See also


Tags

task-unrestricted, might-switch
6.26.2.7 rtdm_getsockopt()

```c
int rtdm_getsockopt (  
    int fd,  
    int level,  
    int optname,  
    void *optval,  
    socklen_t *optlen )
```

Get socket option.

Refer to `rtdm_getsockopt()` for parameters and return values. Action depends on driver implementation, see Device Profiles.

Tags

`task-unrestricted, might-switch`

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>fd</th>
<th>File descriptor as returned by <code>rtdm_socket()</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>level</td>
<td>Addressed stack level</td>
</tr>
<tr>
<td>in</td>
<td>optname</td>
<td>Option name ID</td>
</tr>
<tr>
<td>out</td>
<td>optval</td>
<td>Value buffer</td>
</tr>
<tr>
<td>in,out</td>
<td>optlen</td>
<td>Value buffer size</td>
</tr>
</tbody>
</table>

Returns

0 on success, otherwise negative error code

Action depends on driver implementation, see Device Profiles.

See also

`getsockopt()` in IEEE Std 1003.1, [http://www.opengroup.org/onlinepubs/009695399](http://www.opengroup.org/onlinepubs/009695399)

Tags

`task-unrestricted, might-switch`

6.26.2.8 rtdm_ioctl()

```c
int rtdm_ioctl (  
    int fd,  
    int request,  
    ... )
```

Issue an IOCTL.

Refer to `rtdm_ioctl()` for parameters and return values. Action depends on driver implementation, see Device Profiles.

Tags

`task-unrestricted, might-switch`
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>fd</th>
<th>File descriptor as returned by <code>rtdm_open()</code> or <code>rtdm_socket()</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>request</td>
<td>IOCTL code</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>Optional third argument, depending on IOCTL function (void * or unsigned long)</td>
</tr>
</tbody>
</table>

Returns

Positive value on success, otherwise negative error code

Action depends on driver implementation, see `Device Profiles`.

See also


Tags

task-unrestricted, might-switch

6.26.2.9  `rtdm_listen()`

```c
int rtdm_listen (  
    int fd,  
    int backlog  
)
```

Listen to incoming connection requests.

Listen for incoming connection requests.

Refer to `rtdm_listen()` for parameters and return values. Action depends on driver implementation, see `Device Profiles`.

Tags

task-unrestricted, might-switch

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>fd</th>
<th>File descriptor as returned by <code>rtdm_socket()</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>backlog</td>
<td>Maximum queue length</td>
</tr>
</tbody>
</table>

Returns

0 on success, otherwise negative error code

Action depends on driver implementation, see `Device Profiles`.
6.26 Driver to driver services

See also


Tags

task-unrestricted, might-switch

6.26.2.10 rtdm_open()

int rtdm_open(
  const char * path,
  int oflag,
  ...
)

Open a device.

Refer to rtdm_open() for parameters and return values

Tags

secondary-only, might-switch

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>path</th>
<th>Device name</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>oflag</td>
<td>Open flags</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>Further parameters will be ignored.</td>
</tr>
</tbody>
</table>

Returns

Positive file descriptor value on success, otherwise a negative error code.

Action depends on driver implementation, see Device Profiles.

See also


Tags

secondary-only, might-switch
6.26.2.11  rtdm_read()

ssize_t rtdm_read (  
    int fd,  
    void *buf,  
    size_t nbyte )

Read from device.

Refer to rtdm_read() for parameters and return values. Action depends on driver implementation, see Device Profiles.

Tags
    mode-unrestricted, might-switch

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>fd</th>
<th>File descriptor as returned by rtdm_open()</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>buf</td>
<td>Input buffer</td>
</tr>
<tr>
<td>in</td>
<td>nbyte</td>
<td>Number of bytes to read</td>
</tr>
</tbody>
</table>

Returns
    Number of bytes read, otherwise negative error code

Action depends on driver implementation, see Device Profiles.

See also

Tags
    mode-unrestricted, might-switch

6.26.2.12  rtdm_recv()

ssize_t rtdm_recv (  
    int fd,  
    void *buf,  
    size_t len,  
    int flags )

Receive message from socket.

Refer to rtdm_recv() for parameters and return values. Action depends on driver implementation, see Device Profiles.

Tags
    mode-unrestricted, might-switch
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>fd</th>
<th>File descriptor as returned by <code>rtdm_socket()</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>buf</td>
<td>Message buffer</td>
</tr>
<tr>
<td>in</td>
<td>len</td>
<td>Message buffer size</td>
</tr>
<tr>
<td>in</td>
<td>flags</td>
<td>Message flags</td>
</tr>
</tbody>
</table>

Returns

Number of bytes received, otherwise negative error code

Action depends on driver implementation, see Device Profiles.

See also


Tags

- mode-unrestricted, might-switch

6.26.2.13 `rtdm_recvfrom()`

```c
ssize_t rtdm_recvfrom (  
    int fd,  
    void *buf,  
    size_t len,  
    int flags,  
    struct sockaddr *from,  
    socklen_t *fromlen )
```

Receive message from socket.

Refer to `rtdm_recvfrom()` for parameters and return values. Action depends on driver implementation, see Device Profiles.

Tags

- mode-unrestricted, might-switch

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>fd</th>
<th>File descriptor as returned by <code>rtdm_socket()</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>buf</td>
<td>Message buffer</td>
</tr>
<tr>
<td>in</td>
<td>len</td>
<td>Message buffer size</td>
</tr>
<tr>
<td>in</td>
<td>flags</td>
<td>Message flags</td>
</tr>
<tr>
<td>out</td>
<td>from</td>
<td>Buffer for message sender address</td>
</tr>
<tr>
<td>in,out</td>
<td>fromlen</td>
<td>Address buffer size</td>
</tr>
</tbody>
</table>
Returns

Number of bytes received, otherwise negative error code

Action depends on driver implementation, see Device Profiles.

See also


Tags

mode-unrestricted, might-switch

6.26.2.14 rtdm_recvmsg()

ssize_t rtdm_recvmsg(
    int fd,
    struct user_msghdr *msg,
    int flags
)

Receive message from socket.

Refer to rtdm_recvmsg() for parameters and return values. Action depends on driver implementation, see Device Profiles.

Tags

mode-unrestricted, might-switch

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>fd</th>
<th>File descriptor as returned by rtdm_socket()</th>
</tr>
</thead>
<tbody>
<tr>
<td>in,out</td>
<td>msg</td>
<td>Message descriptor</td>
</tr>
<tr>
<td>in</td>
<td>flags</td>
<td>Message flags</td>
</tr>
</tbody>
</table>

Returns

Number of bytes received, otherwise negative error code

Action depends on driver implementation, see Device Profiles.

See also


Tags

mode-unrestricted, might-switch
6.26.2.15  rtdm_send()

ssize_t rtdm_send (  
    int fd,  
    const void * buf,  
    size_t len,  
    int flags )

Transmit message to socket.

Refer to rtdm_send() for parameters and return values. Action depends on driver implementation, see Device Profiles.

Tags
mode-unrestricted, might-switch

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in fd</td>
<td>int</td>
<td>File descriptor as returned by rtdm_socket()</td>
</tr>
<tr>
<td>in buf</td>
<td>void</td>
<td>Message buffer</td>
</tr>
<tr>
<td>in len</td>
<td>size_t</td>
<td>Message buffer size</td>
</tr>
<tr>
<td>in flags</td>
<td>int</td>
<td>Message flags</td>
</tr>
</tbody>
</table>

Returns
Number of bytes sent, otherwise negative error code

Action depends on driver implementation, see Device Profiles.

See also

Tags
mode-unrestricted, might-switch

6.26.2.16  rtdm_sendmsg()

ssize_t rtdm_sendmsg (  
    int fd,  
    const struct user_msghdr * msg,  
    int flags )

Transmit message to socket.

Refer to rtdm_sendmsg() for parameters and return values. Action depends on driver implementation, see Device Profiles.

Tags
mode-unrestricted, might-switch
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>fd</th>
<th>File descriptor as returned by <code>rtdm_socket()</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>msg</td>
<td>Message descriptor</td>
</tr>
<tr>
<td>in</td>
<td>flags</td>
<td>Message flags</td>
</tr>
</tbody>
</table>

Returns

Number of bytes sent, otherwise negative error code

Action depends on driver implementation, see Device Profiles.

See also

`sendmsg()` in IEEE Std 1003.1, [http://www.opengroup.org/onlinepubs/009695399](http://www.opengroup.org/onlinepubs/009695399)

Tags

`mode-unrestricted, might-switch`

6.26.2.17  `rtdm_sendto()`

```c
ssize_t rtdm_sendto (  
    int fd,  
    const void * buf,  
    size_t len,  
    int flags,  
    const struct sockaddr * to,  
    socklen_t tolen )
```

Transmit message to socket.

Refer to `rtdm_sendto()` for parameters and return values. Action depends on driver implementation, see Device Profiles.

Tags

`mode-unrestricted, might-switch`

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>fd</th>
<th>File descriptor as returned by <code>rtdm_socket()</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>buf</td>
<td>Message buffer</td>
</tr>
<tr>
<td>in</td>
<td>len</td>
<td>Message buffer size</td>
</tr>
<tr>
<td>in</td>
<td>flags</td>
<td>Message flags</td>
</tr>
<tr>
<td>in</td>
<td>to</td>
<td>Buffer for message destination address</td>
</tr>
<tr>
<td>in</td>
<td>tolen</td>
<td>Address buffer size</td>
</tr>
</tbody>
</table>
6.26 Driver to driver services

Returns

Number of bytes sent, otherwise negative error code

Action depends on driver implementation, see Device Profiles.

See also


Tags

mode-unrestricted, might-switch

6.26.2.18 rtdm_setsockopt()

```
int rtdm_setsockopt( 
    int fd, 
    int level, 
    int optname, 
    const void *optval, 
    socklen_t optlen )
```

Set socket option.

Refer to rtdm_setsockopt() for parameters and return values. Action depends on driver implementation, see Device Profiles.

Tags

task-unrestricted, might-switch

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fd</code></td>
<td>File descriptor as returned by rtdm_socket()</td>
</tr>
<tr>
<td><code>level</code></td>
<td>Addressed stack level</td>
</tr>
<tr>
<td><code>optname</code></td>
<td>Option name ID</td>
</tr>
<tr>
<td><code>optval</code></td>
<td>Value buffer</td>
</tr>
<tr>
<td><code>optlen</code></td>
<td>Value buffer size</td>
</tr>
</tbody>
</table>

Returns

0 on success, otherwise negative error code

Action depends on driver implementation, see Device Profiles.
See also


Tags

task-unrestricted, might-switch

6.26.2.19 rtdm_shutdown()

int rtdm_shutdown ( 
    int fd, 
    int how )

Shut down parts of a connection.

Refer to rtdm_shutdown() for parameters and return values. Action depends on driver implementation, see Device Profiles.

Tags

secondary-only, might-switch

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>fd</th>
<th>File descriptor as returned by rtdm_socket()</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>how</td>
<td>Specifies the part to be shut down (SHUT_xxx)</td>
</tr>
</tbody>
</table>

Returns

0 on success, otherwise negative error code

Action depends on driver implementation, see Device Profiles.

See also


Tags

secondary-only, might-switch
6.26.2.20 rtdm_socket()

```c
int rtdm_socket (  
    int protocol_family,  
    int socket_type,  
    int protocol )
```

Create a socket.

Refer to `rtdm_socket()` for parameters and return values. Action depends on driver implementation, see Device Profiles.

**Tags**

*secondary-only, might-switch*

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>protocol_family</code></td>
<td>Protocol family (PF_xxx)</td>
</tr>
<tr>
<td><code>socket_type</code></td>
<td>Socket type (SOCK_xxx)</td>
</tr>
<tr>
<td><code>protocol</code></td>
<td>Protocol ID, 0 for default</td>
</tr>
</tbody>
</table>

**Returns**

Positive file descriptor value on success, otherwise a negative error code.

Action depends on driver implementation, see Device Profiles.

See also

`socket()` in IEEE Std 1003.1, [http://www.opengroup.org/onlinepubs/009695399](http://www.opengroup.org/onlinepubs/009695399)

**Tags**

*secondary-only, might-switch*

6.26.2.21 rtdm_write()

```c
ssize_t rtdm_write (  
    int fd,  
    const void ∗ buf,  
    size_t nbyte )
```

Write to device.

Refer to `rtdm_write()` for parameters and return values. Action depends on driver implementation, see Device Profiles.

**Tags**

*mode-unrestricted, might-switch*
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>fd</th>
<th>File descriptor as returned by rtdm_open()</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>buf</td>
<td>Output buffer</td>
</tr>
<tr>
<td>in</td>
<td>nbyte</td>
<td>Number of bytes to write</td>
</tr>
</tbody>
</table>

Returns

Number of bytes written, otherwise negative error code

Action depends on driver implementation, see Device Profiles.

See also


Tags

mode-unrestricted, might-switch
6.27 Device Profiles

Pre-defined classes of real-time devices.

Collaboration diagram for Device Profiles:

- CAN Devices
- Real-time IPC
- User-space driver core
- Serial Devices
- Testing Devices

Modules

- **User-space driver core**
  
  *This profile includes all mini-drivers sitting on top of the User-space Device Driver framework (UDD).*

- **CAN Devices**
  
  *This is the common interface a RTDM-compliant CAN device has to provide.*

- **Serial Devices**
  
  *This is the common interface a RTDM-compliant serial device has to provide.*

- **Testing Devices**
  
  *This group of devices is intended to provide in-kernel testing results.*

- **Real-time IPC**
  
  *Profile Revision: 1*

Data Structures

- **struct rtdm_device_info**
  
  *Device information.*

Typedefs

- **typedef struct rtdm_device_info rtdm_device_info_t**
  
  *Device information.*
RTDM_CLASS_xxx

Device classes

- \#define RTDM_CLASS_PARPORT 1
- \#define RTDM_CLASS_SERIAL 2
- \#define RTDM_CLASS_CAN 3
- \#define RTDM_CLASS_NETWORK 4
- \#define RTDM_CLASS_RTMAC 5
- \#define RTDM_CLASS_TESTING 6
- \#define RTDM_CLASS_RTIPC 7
- \#define RTDM_CLASS_COBALT 8
- \#define RTDM_CLASS_UDD 9
- \#define RTDM_CLASS_MEMORY 10
- \#define RTDM_CLASS_GPIO 11
- \#define RTDM_CLASS_SPI 12
- \#define RTDM_CLASS_MISC 223
- \#define RTDM_CLASS_EXPERIMENTAL 224
- \#define RTDM_CLASS_MAX 255

Device Naming

Maximum length of device names (excluding the final null character)

- \#define RTDM_MAX_DEVNAME_LEN 31

RTDM_PURGE_xxx_BUFFER

Flags selecting buffers to be purged

- \#define RTDM_PURGE_RX_BUFFER 0x0001
- \#define RTDM_PURGE_TX_BUFFER 0x0002

Common IOCTLs

The following IOCTLs are common to all device rtdm_profiles.

- \#define RTIOC_DEVICE_INFO _IOR(RTIOC_TYPE_COMMON, 0x00, struct rtdm_device_info)  
  Retrieve information about a device or socket.
- \#define RTIOC_PURGE _IOW(RTIOC_TYPE_COMMON, 0x10, int)  
  Purge internal device or socket buffers.
6.27 Device Profiles

6.27.1 Detailed Description

Pre-defined classes of real-time devices.

Device profiles define which operation handlers a driver of a certain class of devices has to implement, which name or protocol it has to register, which IOCTLs it has to provide, and further details. Sub-classes can be defined in order to extend a device profile with more hardware-specific functions.

6.27.2 Macro Definition Documentation

6.27.2.1 RTIOC_DEVICE_INFO

#define RTIOC_DEVICE_INFO _IOR(RTIOC_TYPE_COMMON, 0x00, struct rtdm_device_info)

Retrieve information about a device or socket.

Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>arg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pointer to information buffer (struct rtdm_device_info)</td>
<td></td>
</tr>
</tbody>
</table>

6.27.2.2 RTIOC_PURGE

#define RTIOC_PURGE _IOW(RTIOC_TYPE_COMMON, 0x10, int)

Purge internal device or socket buffers.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>arg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purge mask, see RTDM_PURGE_***_BUFFER</td>
<td></td>
</tr>
</tbody>
</table>
6.28 Device Registration Services

Collaboration diagram for Device Registration Services:

![Collaboration diagram](image)

Data Structures

- `struct rtdm_fd_ops`
  
  *RTDM file operation descriptor.*

Macros

- `#define RTDM_MAX_MINOR 1024`
  
  *Maximum number of named devices per driver.*

Functions

- `int rtdm_open_handler (struct rtdm_fd *fd, int oflags)`
  
  *Open handler for named devices.*

- `int rtdm_socket_handler (struct rtdmFd *fd, int protocol)`
  
  *Socket creation handler for protocol devices.*

- `void rtdm_close_handler (struct rtdm_fd *fd)`
  
  *Close handler.*

- `int rtdm_ioctl_handler (struct rtdm_fd *fd, unsigned int request, void __user *arg)`
  
  *IOCTL handler.*

- `ssize_t rtdm_read_handler (struct rtdm_fd *fd, void __user *buf, size_t size)`
  
  *Read handler.*

- `ssize_t rtdm_write_handler (struct rtdm_fd *fd, const void __user *buf, size_t size)`
  
  *Write handler.*

- `ssize_t rtdm_recvmsg_handler (struct rtdm_fd *fd, struct user_msghdr *msg, int flags)`
  
  *Receive message handler.*

- `ssize_t rtdm_sendmsg_handler (struct rtdm_fd *fd, const struct user_msghdr *msg, int flags)`
  
  *Transmit message handler.*

- `int rtdm_select_handler (struct rtdm_fd *fd, struct xnselector *selector, unsigned int type, unsigned int index)`
  
  *Select handler.*

- `int rtdm_mmap_handler (struct rtdm_fd *fd, struct vm_area_struct *vma)`
  
  *Memory mapping handler.*
6.28 Device Registration Services

- `unsigned long rtdm_get_unmapped_area_handler` (struct rtdm_fd *fd, unsigned long len, unsigned long pgoff, unsigned long flags)
  
  Allocate mapping region in address space.

- `int rtdm_dev_register` (struct rtdm_device *dev)
  
  Register a RTDM device.

- `void rtdm_dev_unregister` (struct rtdm_device *dev)
  
  Unregister a RTDM device.

- `int rtdm_drv_set_sysclass` (struct rtdm_driver *drv, struct class *cls)
  
  Set the kernel device class of a RTDM driver.

Device Flags

Static flags describing a RTDM device

- `#define RTDM_EXCLUSIVE 0x0001`
  
  If set, only a single instance of the device can be requested by an application.

- `#define RTDM_FIXED_MINOR 0x0002`
  
  Use fixed minor provided in the rtdm_device description for registering.

- `#define RTDM_NAMED_DEVICE 0x0010`
  
  If set, the device is addressed via a clear-text name.

- `#define RTDM_PROTOCOL_DEVICE 0x0020`
  
  If set, the device is addressed via a combination of protocol ID and socket type.

- `#define RTDM_DEVICE_TYPE_MASK 0x00F0`
  
  Mask selecting the device type.

- `#define RTDM_SECURE_DEVICE 0x80000000`
  
  Flag indicating a secure variant of RTDM (not supported here)

6.28.1 Detailed Description

6.28.2 Macro Definition Documentation

6.28.2.1 RTDM_DEVICE_TYPE_MASK

`#define RTDM_DEVICE_TYPE_MASK 0x00F0`

Mask selecting the device type.

6.28.2.2 RTDM_EXCLUSIVE

`#define RTDM_EXCLUSIVE 0x0001`

If set, only a single instance of the device can be requested by an application.
6.28.2.3 RTDM_FIXED_MINOR

#define RTDM_FIXED_MINOR 0x0002

Use fixed minor provided in the rtdm_device description for registering.

If this flag is absent, the RTDM core assigns minor numbers to devices managed by a driver in order of registration.

6.28.2.4 RTDM_MAX_MINOR

#define RTDM_MAX_MINOR 1024

Maximum number of named devices per driver.

6.28.2.5 RTDM_NAMED_DEVICE

#define RTDM_NAMED_DEVICE 0x0010

If set, the device is addressed via a clear-text name.

6.28.2.6 RTDM_PROTOCOL_DEVICE

#define RTDM_PROTOCOL_DEVICE 0x0020

If set, the device is addressed via a combination of protocol ID and socket type.

Referenced by udd_register_device().

6.28.3 Function Documentation

6.28.3.1 rtdm_close_handler()

void rtdm_close_handler (  
    struct rtdm_fd *fd  )

Close handler.
Parameters

| in | fd | File descriptor associated with opened device instance. |

See also

`close()` in IEEE Std 1003.1, [http://www.opengroup.org/onlinepubs/009695399](http://www.opengroup.org/onlinepubs/009695399)

6.28.3.2 rtdm_dev_register()

```c
int rtdm_dev_register (  
    struct rtdm_device * dev )
```

Register a RTDM device.

Registers a device in the RTDM namespace.

Parameters

| in | dev | Device descriptor. |

Returns

0 is returned upon success. Otherwise:

- **-EINVAL** is returned if the descriptor contains invalid entries. `RTDM_PROFILE_INFO()` must appear in the list of initializers for the driver properties.
- **-EEXIST** is returned if the specified device name of protocol ID is already in use.
- **-ENOMEM** is returned if a memory allocation failed in the process of registering the device.
- **-EAGAIN** is returned if no registry slot is available (check/raise `CONFIG_XENO_OPT_REGISTRY_NRSLOTS`).

Tags

`secondary-only`

6.28.3.3 rtdm_dev_unregister()

```c
void rtdm_dev_unregister (  
    struct rtdm_device * dev )
```

Unregister a RTDM device.

Removes the device from the RTDM namespace. This routine waits until all connections to `device` have been closed prior to unregistering.

Generated by Doxygen
6.28.3.4  rtdm_drv_set_sysclass()

int rtdm_drv_set_sysclass (  
    struct rtdm_driver * drv,  
    struct class * cls )

Set the kernel device class of a RTDM driver.

Set the kernel device class assigned to the RTDM driver. By default, RTDM drivers belong to Linux's "rtdm" device class, creating a device node hierarchy rooted at /dev/rtdm, and sysfs nodes under /sys/class/rtdm.

This call assigns a user-defined kernel device class to the RTDM driver, so that its devices are created into a different system hierarchy.

rtdm_drv_set_sysclass() is meaningful only before the first device which is attached to drv is registered by a call to rtdm_dev_register().

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>drv</th>
<th>Address of the RTDM driver descriptor.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>cls</td>
<td>Pointer to the kernel device class. NULL is allowed to clear a previous setting, switching back to the default &quot;rtdm&quot; device class.</td>
</tr>
</tbody>
</table>

Returns

0 on success, otherwise:

- -EBUSY is returned if the kernel device class has already been set for drv, or some device(s) attached to drv are currently registered.

Tags

task-unrestricted

Attention

The kernel device class set by this call is not related to the RTDM class identification as defined by the RTDM profiles in any way. This is strictly related to the Linux kernel device hierarchy.

References rtdm_driver::profile_info.
6.28.3.5  \texttt{rtdm\_get\_unmapped\_area\_handler()}

\begin{verbatim}
unsigned long rtdm_get_unmapped_area_handler ( 
    struct rtdm_fd * fd, 
    unsigned long len, 
    unsigned long pgoff, 
    unsigned long flags )

Allocate mapping region in address space.

When present, this optional handler should return the start address of a free region in the process's 
address space, large enough to cover the ongoing \texttt{mmap()} operation. If unspecified, the default 
architecture-defined handler is invoked.

Most drivers can omit this handler, except on MMU-less platforms (see second note).

\begin{verbatim}
Parameters

| in | \textit{fd} | File descriptor |
| in | \textit{len} | Length of the requested region |
| in | \textit{pgoff} | Page frame number to map to (see second note). |
| in | \textit{flags} | Requested mapping flags |

Returns

The start address of the mapping region on success. On failure, a negative error code should 
be returned, with -ENOSYS meaning that the driver does not want to provide such information, in 
which case the ongoing \texttt{mmap()} operation will fail.

Note

The address hint passed to the \texttt{mmap()} request is deliberately ignored by RTDM, and therefore 
not passed to this handler.
On MMU-less platforms, this handler is required because RTDM issues mapping requests over 
a shareable character device internally. In such context, the RTDM core may pass a null \texttt{pgoff} 
argument to the handler, for probing for the logical start address of the memory region to map to. 
Otherwise, when \texttt{pgoff} is non-zero, pgoff \texttt{\ll PAGE\_SHIFT} is usually returned.
\end{verbatim}

6.28.3.6  \texttt{rtdm\_ioctl\_handler()}

\begin{verbatim}
int rtdm_ioctl_handler ( 
    struct rtdm_fd * fd, 
    unsigned int request, 
    void __user * arg )

IOCTL handler.
\end{verbatim}
Parameters

<table>
<thead>
<tr>
<th></th>
<th>fd</th>
<th>File descriptor</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>request</td>
<td>Request number as passed by the user</td>
</tr>
<tr>
<td>in,out</td>
<td>arg</td>
<td>Request argument as passed by the user</td>
</tr>
</tbody>
</table>

Returns

A positive value or 0 on success. On failure return either -ENOSYS, to request that the function be called again from the opposite realtime/non-realtime context, or another negative error code.

See also


6.28.3.7 rtdm_mmap_handler()

int rtdm_mmap_handler (  
    struct rtdm_fd * fd,  
    struct vm_area_struct * vma )

Memory mapping handler.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>fd</th>
<th>File descriptor</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>vma</td>
<td>Virtual memory area descriptor</td>
</tr>
</tbody>
</table>

Returns

0 on success. On failure, a negative error code is returned.

See also


Note

The address hint passed to the mmap() request is deliberately ignored by RTDM.

6.28.3.8 rtdm_open_handler()

int rtdm_open_handler (  
    struct rtdm_fd * fd,  
    int oflags )

Open handler for named devices.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>fd</th>
<th>File descriptor associated with opened device instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>oflags</td>
<td>Open flags as passed by the user</td>
</tr>
</tbody>
</table>

The file descriptor carries a device minor information which can be retrieved by a call to rtdm_fd->minor(fd). The minor number can be used for distinguishing devices managed by a driver.

Returns

0 on success. On failure, a negative error code is returned.

See also

open() in IEEE Std 1003.1, [http://www.opengroup.org/onlinepubs/009695399](http://www.opengroup.org/onlinepubs/009695399)

### 6.28.3.9 rtdm_read_handler()

ssize_t rtdm_read_handler (
    struct rtdm_fd *fd,
    void __user *buf,
    size_t size )

Read handler.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>fd</th>
<th>File descriptor</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>buf</td>
<td>Input buffer as passed by the user</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>Number of bytes the user requests to read</td>
</tr>
</tbody>
</table>

Returns

On success, the number of bytes read. On failure return either -ENOSYS, to request that this handler be called again from the opposite realtime/non-realtime context, or another negative error code.

See also

read() in IEEE Std 1003.1, [http://www.opengroup.org/onlinepubs/009695399](http://www.opengroup.org/onlinepubs/009695399)
6.28.3.10  rtdm_recvmsg_handler()

ssize_t rtdm_recvmsg_handler (  
    struct rtdm_fd * fd,  
    struct user_msghdr * msg,  
    int flags )

Receive message handler.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th></th>
<th>File descriptor</th>
</tr>
</thead>
<tbody>
<tr>
<td>in,out</td>
<td>msg</td>
<td>Message descriptor as passed by the user, automatically mirrored to safe kernel memory in case of user mode call</td>
</tr>
<tr>
<td>in</td>
<td>flags</td>
<td>Message flags as passed by the user</td>
</tr>
</tbody>
</table>

Returns

On success, the number of bytes received. On failure return either -ENOSYS, to request that this handler be called again from the opposite realtime/non-realtime context, or another negative error code.

See also

6.28.3.11  rtdm_select_handler()

int rtdm_select_handler (  
    struct rtdm_fd * fd,  
    struct xnselector * selector,  
    unsigned int type,  
    unsigned int index )

Select handler.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th></th>
<th>File descriptor</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>selector</td>
<td>Pointer to the selector structure</td>
</tr>
<tr>
<td></td>
<td>type</td>
<td>Type of events (XNSELECT_READ, XNSELECT_WRITE, or XNSELECT_EXCEPT)</td>
</tr>
<tr>
<td></td>
<td>index</td>
<td>Index of the file descriptor</td>
</tr>
</tbody>
</table>

Returns

0 on success. On failure, a negative error code is returned.
See also


6.28.3.12  rtdm_sendmsg_handler()

ssize_t rtdm_sendmsg_handler (  
   struct rtdm_fd * fd,  
   const struct user_msghdr * msg,  
   int flags )

Transmit message handler.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>fd</th>
<th>File descriptor</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>msg</td>
<td>Message descriptor as passed by the user, automatically mirrored to safe kernel memory in case of user mode call</td>
</tr>
<tr>
<td>in</td>
<td>flags</td>
<td>Message flags as passed by the user</td>
</tr>
</tbody>
</table>

Returns

On success, the number of bytes transmitted. On failure return either -ENOSYS, to request that this handler be called again from the opposite realtime/non-realtime context, or another negative error code.

See also


6.28.3.13  rtdm_socket_handler()

int rtdm_socket_handler (  
   struct rtdm_fd * fd,  
   int protocol )

Socket creation handler for protocol devices.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>fd</th>
<th>File descriptor associated with opened device instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>protocol</td>
<td>Protocol number as passed by the user</td>
</tr>
</tbody>
</table>
Returns

0 on success. On failure, a negative error code is returned.

See also


6.28.3.14 rtdm_write_handler()

ssize_t rtdm_write_handler (
    struct rtdm_fd * fd,
    const void __user * buf,
    size_t size )

Write handler.

Parameters

in   fd File descriptor
in   buf Output buffer as passed by the user
in   size Number of bytes the user requests to write

Returns

On success, the number of bytes written. On failure return either -ENOSYS, to request that this
handler be called again from the opposite realtime/non-realtime context, or another negative error
code.

See also

6.29 Clock Services

Collaboration diagram for Clock Services:

![Collaboration Diagram](image.png)

Functions

- `nanosecs_abs_t rtdm_clock_read (void)`
  *Get system time.*

- `nanosecs_abs_t rtdm_clock_read_monotonic (void)`
  *Get monotonic time.*

6.29.1 Detailed Description

6.29.2 Function Documentation

6.29.2.1 `rtdm_clock_read()`

```c
nanosecs_abs_t rtdm_clock_read (    
   void    )
```

*Get system time.*

Returns

- The system time in nanoseconds is returned

Note

The resolution of this service depends on the system timer. In particular, if the system timer is running in periodic mode, the return value will be limited to multiples of the timer tick period. The system timer may have to be started to obtain valid results. Whether this happens automatically (as on Xenomai) or is controlled by the application depends on the RTDM host environment.

Tags

- unrestricted

Referenced by `a4l_get_time()`, and `rtdm_ratelimit()`.

---

Generated by Doxygen
6.29.2.2 `rtdm_clock_read_monotonic()`

```c
nanosecs_abs_t rtdm_clock_read_monotonic ( void )
```

Get monotonic time.

Returns

The monotonic time in nanoseconds is returned.

Note

The resolution of this service depends on the system timer. In particular, if the system timer is running in periodic mode, the return value will be limited to multiples of the timer tick period. The system timer may have to be started to obtain valid results. Whether this happens automatically (as on Xenomai) or is controlled by the application depends on the RTDM host environment.

Tags

unrestricted
### 6.30 Task Services

Collaboration diagram for Task Services:

```
Driver programming interface → Task Services
```

**Typedefs**
- typedef void(* rtdm_task_proc_t) (void *arg)  
  *Real-time task procedure.*

**Functions**
- int rtdm_task_init (rtdm_task_t *task, const char *name, rtdm_task_proc_t task_proc, void *arg, int priority, nanosecs_rel_t period)  
  *Initialise and start a real-time task.*
- void rtdm_task_destroy (rtdm_task_t *task)  
  *Destroy a real-time task.*
- int rtdm_task_should_stop (void)  
  *Check for pending termination request.*
- void rtdm_task_set_priority (rtdm_task_t *task, int priority)  
  *Adjust real-time task priority.*
- int rtdm_task_set_period (rtdm_task_t *task, nanosecs_abs_t start_date, nanosecs_rel_t period)  
  *Adjust real-time task period.*
- int rtdm_task_wait_period (unsigned long *overruns_r)  
  *Wait on next real-time task period.*
- int rtdm_task_unblock (rtdm_task_t *task)  
  *Activate a blocked real-time task.*
- rtdm_task_t * rtdm_task_current (void)  
  *Get current real-time task.*
- int rtdm_task_sleep (nanosecs_rel_t delay)  
  *Sleep a specified amount of time.*
- int rtdm_task_sleep_until (nanosecs_abs_t wakeup_time)  
  *Sleep until a specified absolute time.*
- int rtdm_task_sleep_abs (nanosecs_abs_t wakeup_time, enum rtdm_timer_mode mode)  
  *Sleep until a specified absolute time.*
- int rtdm_task_busy_wait (bool condition, nanosecs_rel_t spin_ns, nanosecs_rel_t sleep_ns)  
  *Safe busy waiting.*
- void rtdm_wait_prepare (struct rtdm_wait_context *wc)  
  *Register wait context.*
- void rtdm_wait_complete (struct rtdm_wait_context *wc)
Mark completion for a wait context.

- `int rtdm_wait_is_completed (struct rtdm_wait_context *wc)`
  Test completion of a wait context.

- `void rtdm_task_join (rtdm_task_t *task)`
  Wait on a real-time task to terminate.

- `void rtdm_task_busy_sleep (nanosecs_rel_t delay)`
  Busy-wait a specified amount of time.

Task Priority Range

Maximum and minimum task priorities

- `#define RTDM_TASK_LOWEST_PRIORITY 0`
- `#define RTDM_TASK_HIGHEST_PRIORITY 99`

Task Priority Modification

Raise or lower task priorities by one level

- `#define RTDM_TASK_RAISE_PRIORITY (+1)`
- `#define RTDM_TASK_LOWER_PRIORITY (-1)`

6.30.1 Detailed Description

6.30.2 Typedef Documentation

6.30.2.1 rtdm_task_proc_t

t typedef void(* rtdm_task_proc_t) (void *arg)

Real-time task procedure.

Parameters

| in, out | arg | argument as passed to rtdm_task_init() |

6.30.3 Function Documentation
6.30.3.1 rtdm_task_busy_sleep()

```c
void rtdm_task_busy_sleep (  
nanosecs_rel_t delay )
```

Busy-wait a specified amount of time.

This service does not schedule out the caller, but rather spins in a tight loop, burning CPU cycles until the timeout elapses.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in</code></td>
<td><code>delay</code></td>
</tr>
<tr>
<td></td>
<td>Delay in nanoseconds. Note that a zero delay does <strong>not</strong> have the meaning of \texttt{RTDM_TIMEOUT_INFINITE} here.</td>
</tr>
</tbody>
</table>

Note

The caller must not be migratable to different CPUs while executing this service. Otherwise, the actual delay will be undefined.

Tags

**unrestricted**

6.30.3.2 rtdm_task_busy_wait()

```c
int rtdm_task_busy_wait (  
bool condition,  
nanosecs_rel_t spin_ns,  
nanosecs_rel_t sleep_ns )
```

Safe busy waiting.

This service alternates active spinning and sleeping within a wait loop, until a condition is satisfied. While sleeping, a task is scheduled out and does not consume any CPU time.

**rtdm_task_busy_wait()** is particularly useful for waiting for a state change reading an I/O register, which usually happens shortly after the wait starts, without incurring the adverse effects of long busy waiting if it doesn't.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in</code></td>
<td><code>condition</code></td>
</tr>
<tr>
<td></td>
<td>The C expression to be tested for detecting completion.</td>
</tr>
<tr>
<td><code>in</code></td>
<td><code>spin_ns</code></td>
</tr>
<tr>
<td></td>
<td>The time to spin on <code>condition</code> before sleeping, expressed as a count of nanoseconds.</td>
</tr>
<tr>
<td><code>in</code></td>
<td><code>sleep_ns</code></td>
</tr>
<tr>
<td></td>
<td>The time to sleep for before spinning again, expressed as a count of nanoseconds.</td>
</tr>
</tbody>
</table>
Returns

0 on success if condition is satisfied, otherwise:

- -EINTR is returned if the calling task has been unblocked by a Linux signal or explicitly via rtdm_task_unblock().
- -EPERM may be returned if an illegal invocation environment is detected.

Tags

primary-only, might-switch

6.30.3.3 rtdm_task_current()

rtdm_task_t* rtdm_task_current(
    void
)

Get current real-time task.

Returns

Pointer to task handle

Tags

mode-unrestricted

6.30.3.4 rtdm_task_destroy()

d Void rtdm_task_destroy(
    rtdm_task_t * task
)

Destroy a real-time task.

This call sends a termination request to task, then waits for it to exit. All RTDM task should check for pending termination requests by calling rtdm_task_should_stop() from their work loop.

If task is current, rtdm_task_destroy() terminates the current context, and does not return to the caller.

Parameters

| in, out | task | Task handle as returned by rtdm_task_init() |
Note

Passing the same task handle to RTDM services after the completion of this function is not allowed.

Tags

secondary-only, might-switch

6.30.3.5 rtdm_task_init()

```c
int rtdm_task_init (
    rtdm_task_t ∗ task,
    const char ∗ name,
    rtdm_task_proc_t task_proc,
    void ∗ arg,
    int priority,
    nanosecs_rel_t period )
```

Initialise and start a real-time task.

After initialising a task, the task handle remains valid and can be passed to RTDM services until either
rtdm_task_destroy() or rtdm_task_join() was invoked.

Parameters

<table>
<thead>
<tr>
<th>in,out</th>
<th>task</th>
<th>Task handle</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>name</td>
<td>Optional task name</td>
</tr>
<tr>
<td>in</td>
<td>task_proc</td>
<td>Procedure to be executed by the task</td>
</tr>
<tr>
<td>in</td>
<td>arg</td>
<td>Custom argument passed to task_proc() on entry</td>
</tr>
<tr>
<td>in</td>
<td>priority</td>
<td>Priority of the task, see also Task Priority Range</td>
</tr>
<tr>
<td>in</td>
<td>period</td>
<td>Period in nanoseconds of a cyclic task, 0 for non-cyclic mode. Waiting for the first and subsequent periodic events is done using rtdm_task_wait_period().</td>
</tr>
</tbody>
</table>

Returns

0 on success, otherwise negative error code

Tags

secondary-only, might-switch

6.30.3.6 rtdm_task_join()

```c
void rtdm_task_join ( 
    rtdm_task_t ∗ task )
```

Wait on a real-time task to terminate.
Parameters

| in,out | task | Task handle as returned by rtdm_task_init() |

Note

Passing the same task handle to RTDM services after the completion of this function is not allowed. This service does not trigger the termination of the targeted task. The user has to take of this, otherwise rtdm_task_join() will never return.

Tags

mode-unrestricted

References xnthread_join().

6.30.3.7 rtdm_task_set_period()

```
int rtdm_task_set_period (   
    rtdm_task_t * task,   
    nanosecs_abs_t start_date,   
    nanosecs_rel_t period )
```

Adjust real-time task period.

Parameters

<table>
<thead>
<tr>
<th>in,out</th>
<th>task</th>
<th>Task handle as returned by rtdm_task_init(), or NULL for referring to the current RTDM task or Cobalt thread.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>in</th>
<th>start_date</th>
<th>The initial (absolute) date of the first release point, expressed in nanoseconds. task will be delayed by the first call to rtdm_task_wait_period() until this point is reached. If start_date is zero, the first release point is set to period nanoseconds after the current date.</th>
</tr>
</thead>
</table>

| in | period | New period in nanoseconds of a cyclic task, zero to disable cyclic mode for task. |

Tags

task-unrestricted

6.30.3.8 rtdm_task_set_priority()

```
void rtdm_task_set_priority (  
    rtdm_task_t * task,  
    int priority )
```

Adjust real-time task priority.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in, out</td>
<td>task</td>
<td>Task handle as returned by rtdm_task_init()</td>
</tr>
<tr>
<td>in</td>
<td>priority</td>
<td>New priority of the task, see also Task Priority Range</td>
</tr>
</tbody>
</table>

Tags

- task-unrestricted, might-switch

6.30.3.9  rtdm_task_should_stop()

```c
int rtdm_task_should_stop (
    void )
```

Check for pending termination request.

Check whether a termination request was received by the current RTDM task. Termination requests are sent by calling rtdm_task_destroy().

Returns

Non-zero indicates that a termination request is pending, in which case the caller should wrap up and exit.

Tags

- rtdm-task, might-switch

6.30.3.10  rtdm_task_sleep()

```c
int rtdm_task_sleep (
    nanosecs_rel_t delay )
```

Sleep a specified amount of time.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>delay</td>
<td>Delay in nanoseconds, see RTDM_TIMEOUT_xxx for special values.</td>
</tr>
</tbody>
</table>

Returns

- 0 on success, otherwise:
  - -EINTR is returned if calling task has been unblock by a signal or explicitly via rtdm_task_unblock().
- EPERM may be returned if an illegal invocation environment is detected.

Tags

primary-only, might-switch

6.30.3.11 rtdm_task_sleep_abs()

```c
int rtdm_task_sleep_abs (  
    nanosecs_abs_t wakeup_time,  
    enum rtdm_timer_mode mode )
```

Sleep until a specified absolute time.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>wakeup_time</th>
<th>Absolute timeout in nanoseconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>mode</td>
<td>Selects the timer mode, see RTDM_TIMERMODE_xxx for details</td>
</tr>
</tbody>
</table>

Returns

0 on success, otherwise:

- EINTR is returned if calling task has been unblock by a signal or explicitly via rtdm_task_unblock().
- EPERM may be returned if an illegal invocation environment is detected.
- EINVAL is returned if an invalid parameter was passed.

Tags

primary-only, might-switch

6.30.3.12 rtdm_task_sleep_until()

```c
int rtdm_task_sleep_until (  
    nanosecs_abs_t wakeup_time )
```

Sleep until a specified absolute time.

**Deprecated** Use rtdm_task_sleep_abs instead!
Parameters

| in | wakeup_time | Absolute timeout in nanoseconds |

Returns

0 on success, otherwise:

- EINTR is returned if calling task has been unblock by a signal or explicitly via rtdm_task_unblock().
- EPERM may be returned if an illegal invocation environment is detected.

Tags

primary-only, might-switch

6.30.3.13 rtdm_task_unblock()

int rtdm_task_unblock (
    rtdm_task_t *task )

Activate a blocked real-time task.

Returns

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

Tags

unrestricted, might-switch

6.30.3.14 rtdm_task_wait_period()

int rtdm_task_wait_period ( 
    unsigned long *overruns_r )

Wait on next real-time task period.

Parameters

| in | overruns_→_r | Address of a long word receiving the count of overruns if -ETIMEDOUT is returned, or NULL if the caller don't need that information. |
Returns

0 on success, otherwise:

- EINVAL is returned if calling task is not in periodic mode.
- ETIMEDOUT is returned if a timer overrun occurred, which indicates that a previous release point has been missed by the calling task.

Tags

primary-only, might-switch

6.30.3.15 rtdm_wait_complete()

void rtdm_wait_complete (
    struct rtdm_wait_context * wc )

Mark completion for a wait context.

rtdm_complete_wait() marks a wait context as completed, so that rtdm_wait_is_completed() returns true for such context.

Parameters

| wc     | Wait context to complete. |

6.30.3.16 rtdm_wait_is_completed()

int rtdm_wait_is_completed ( 
    struct rtdm_wait_context * wc )

Test completion of a wait context.

rtdm_wait_is_completed() returns true if rtdm_complete_wait() was called for wc. The completion mark is reset each time rtdm_wait_prepare() is called for a wait context.

Parameters

| wc     | Wait context to check for completion. |

Returns

non-zero/true if rtdm_wait_complete() was called for wc, zero otherwise.
6.30.3.17  rtdm_wait_prepare()

void rtdm_wait_prepare (  
    struct rtdm_wait_context ∗ wc )

Register wait context.

rtdm_wait_prepare() registers a wait context structure for the caller, which can be later retrieved by a call to rtdm_wait_get_context(). This call is normally issued before the current task blocks on a wait object, waiting for some (producer) code to wake it up. Arbitrary data can be exchanged between both sites via the wait context structure, which is allocated by the waiter (consumer) side.

wc is the address of an anchor object which is commonly embedded into a larger structure with arbitrary contents, which needs to be shared between the consumer (waiter) and the producer for implementing the wait code.

A typical implementation pattern for the wait side is:

```c
struct rtdm_waitqueue wq;
struct some_wait_context {
    int input_value;
    int output_value;
    struct rtdm_wait_context wc;
} wait_context;

wait_context.input_value = 42;
rtdm_wait_prepare(&wait_context);
ret = rtdm_wait_condition(&wq, rtdm_wait_is_completed(&wait_context));
if (ret) goto wait_failed;
handle_event(wait_context.output_value);
```

On the producer side, the implementation would look like:

```c
struct rtdm_waitqueue wq;
struct some_wait_context {
    int input_value;
    int output_value;
    struct rtdm_wait_context wc;
} *wait_context_ptr;
struct rtdm_wait_context *wc;
struct rtdm_task_t *task;
rtdm_for_each_waiter(task, &wq) {
    wc = rtdm_wait_get_context(task);
    wait_context_ptr = container_of(wc, struct some_wait_context, wc);
    wait_context_ptr->output_value = 12;
}
rtdm_waitqueue_broadcast(&wq);
```

Parameters

| wc  | Wait context to register. |

Generated by Doxygen
6.31 Timer Services

Collaboration diagram for Timer Services:

![Collaboration diagram](image)

Typedefs

- typedef void(* rtdm_timer_handler_t) (rtdm_timer_t *timer)  
  Timer handler.

Functions

- int rtdm_timer_init (rtdm_timer_t *timer, rtdm_timer_handler_t handler, const char *name)  
  Initialise a timer.
- void rtdm_timer_destroy (rtdm_timer_t *timer)  
  Destroy a timer.
- int rtdm_timer_start (rtdm_timer_t *timer, nanosecs_abs_t expiry, nanosecs_rel_t interval, enum rtdm_timer_mode mode)  
  Start a timer.
- void rtdm_timer_stop (rtdm_timer_t *timer)  
  Stop a timer.
- int rtdm_timer_start_in_handler (rtdm_timer_t *timer, nanosecs_abs_t expiry, nanosecs_rel_t interval, enum rtdm_timer_mode mode)  
  Start a timer from inside a timer handler.
- void rtdm_timer_stop_in_handler (rtdm_timer_t *timer)  
  Stop a timer from inside a timer handler.

RTDM_TIMERMODE_xxx

Timer operation modes

- enum rtdm_timer_mode { RTDM_TIMERMODE_RELATIVE = XN_RELATIVE, RTDM_TIMERMODE_ABSOLUTE = XN_ABSOLUTE, RTDM_TIMERMODE_REALTIME = XN_REALTIME }  

6.31.1 Detailed Description

6.31.2 Typedef Documentation
6.31 Timer Services

6.31.2.1 rtdm_timer_handler_t

typedef void(* rtdm_timer_handler_t) (rtdm_timer_t *timer)

Timer handler.

Parameters

| in | timer | Timer handle as returned by rtdm_timer_init() |

6.31.3 Enumeration Type Documentation

6.31.3.1 rtdm_timer_mode

enum rtdm_timer_mode

Enumerator

<table>
<thead>
<tr>
<th>RTDM_TIMERMODE_RELATIVE</th>
<th>Monotonic timer with relative timeout.</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTDM_TIMERMODE_ABSOLUTE</td>
<td>Monotonic timer with absolute timeout.</td>
</tr>
<tr>
<td>RTDM_TIMERMODE_REALTIME</td>
<td>Adjustable timer with absolute timeout.</td>
</tr>
</tbody>
</table>

6.31.4 Function Documentation

6.31.4.1 rtdm_timer_destroy()

void rtdm_timer_destroy (  
  rtdm_timer_t *timer  )

Destroy a timer.

Parameters

| in,out | timer | Timer handle as returned by rtdm_timer_init() |

Tags

| task-unrestricted |

Generated by Doxygen
6.31.4.2  rtdm_timer_init()

void rtdm_timer_init ( rtdm_timer_t *timer,
                        rtdm_timer_handler_t handler,
                        const char * name )

Initialise a timer.

Parameters

<table>
<thead>
<tr>
<th>in,out</th>
<th>timer</th>
<th>Timer handle</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>handler</td>
<td>Handler to be called on timer expiry</td>
</tr>
<tr>
<td>in</td>
<td>name</td>
<td>Optional timer name</td>
</tr>
</tbody>
</table>

Returns

0 on success, otherwise negative error code

Tags

task-unrestricted

6.31.4.3  rtdm_timer_start()

int rtdm_timer_start ( rtdm_timer_t *timer,
                        nanosecs_abs_t expiry,
                        nanosecs_rel_t interval,
                        enum rtdm_timer_mode mode )

Start a timer.

Parameters

<table>
<thead>
<tr>
<th>in,out</th>
<th>timer</th>
<th>Timer handle as returned by rtdm_timer_init()</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>expiry</td>
<td>Firing time of the timer, mode defines if relative or absolute</td>
</tr>
<tr>
<td>in</td>
<td>interval</td>
<td>Relative reload value, &gt; 0 if the timer shall work in periodic mode with the specific interval, 0 for one-shot timers</td>
</tr>
<tr>
<td>in</td>
<td>mode</td>
<td>Defines the operation mode, see RTDM_TIMERMODE_xxx for possible values</td>
</tr>
</tbody>
</table>

Returns

0 on success, otherwise:

- -ETIMEDOUT is returned if expiry describes an absolute date in the past. In such an event, the timer is nevertheless armed for the next shot in the timeline if interval is non-zero.
6.31.4.4  rtdm_timer_start_in_handler()

```c
int rtdm_timer_start_in_handler (  
    rtdm_timer_t *timer,  
    nanosecs_abs_t expiry,  
    nanosecs_rel_t interval,  
    enum rtdm_timer_mode mode )
```

Start a timer from inside a timer handler.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in,out</td>
<td><code>timer</code> Timer handle as returned by <code>rtdm_timer_init()</code></td>
</tr>
<tr>
<td>in</td>
<td><code>expiry</code> Firing time of the timer, <code>mode</code> defines if relative or absolute</td>
</tr>
<tr>
<td>in</td>
<td><code>interval</code> Relative reload value, &gt; 0 if the timer shall work in periodic mode with the specific interval, 0 for one-shot timers</td>
</tr>
<tr>
<td>in</td>
<td><code>mode</code> Defines the operation mode, see <code>RTDM_TIMERMODE_xxx</code> for possible values</td>
</tr>
</tbody>
</table>

Returns

0 on success, otherwise:

- `-ETIMEDOUT` is returned if `expiry` describes an absolute date in the past.

Tags

`coreirq-only`

6.31.4.5  rtdm_timer_stop()

```c
void rtdm_timer_stop (  
    rtdm_timer_t *timer )
```

Stop a timer.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in,out</td>
<td><code>timer</code> Timer handle as returned by <code>rtdm_timer_init()</code></td>
</tr>
</tbody>
</table>
6.31.4.6  rtdm_timer_stop_in_handler()

void rtdm_timer_stop_in_handler (  
   rtdm_timer_t * timer )

Stop a timer from inside a timer handler.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in, out</td>
<td><strong>timer</strong></td>
<td>Timer handle as returned by rtdm_timer_init()</td>
</tr>
</tbody>
</table>

Tags

- coreirq-only
6.32 Synchronisation Services

Collaboration diagram for Synchronisation Services:

Modules

- Big dual kernel lock
- Spinlock with preemption deactivation
- Event Services
- Semaphore Services
- Mutex services

Functions

- `void rtdm_waitqueue_init (struct rtdm_waitqueue *wq)`
  
  Initialize a RTDM wait queue.

- `void rtdm_waitqueue_destroy (struct rtdm_waitqueue *wq)`
  
  Deletes a RTDM wait queue.

- `rtdm_timedwait_condition_locked (struct rtdm_wait_queue *wq, C_expr condition, nanosecs_rel_t timeout, rtdm_toseq_t *toseq)`
  
  Timed sleep on a locked waitqueue until a condition gets true.

- `rtdm_wait_condition_locked (struct rtdm_wait_queue *wq, C_expr condition)`
  
  Sleep on a locked waitqueue until a condition gets true.

- `rtdm_timedwait_condition (struct rtdm_wait_queue *wq, C_expr condition, nanosecs_rel_t timeout, rtdm_toseq_t *toseq)`
  
  Timed sleep on a waitqueue until a condition gets true.

- `void rtdm_timedwait (struct rtdm_wait_queue *wq, nanosecs_rel_t timeout, rtdm_toseq_t *toseq)`
  
  Timed sleep on a waitqueue unconditionally.

- `void rtdm_timedwait_locked (struct rtdm_wait_queue *wq, nanosecs_rel_t timeout, rtdm_toseq_t *toseq)`
  
  Timed sleep on a locked waitqueue unconditionally.

- `rtdm_wait_condition (struct rtdm_wait_queue *wq, C_expr condition)`
  
  Sleep on a waitqueue until a condition gets true.

- `void rtdm_wait (struct rtdm_wait_queue *wq)`
Sleep on a waitqueue unconditionally.
• void rtdm_wait_locked (struct rtdm_wait_queue *wq)
  Sleep on a locked waitqueue unconditionally.
• void rtdm_waitqueue_lock (struct rtdm_wait_queue *wq, rtdm_lockctx_t context)
  Lock a waitqueue.
• void rtdm_waitqueue_unlock (struct rtdm_wait_queue *wq, rtdm_lockctx_t context)
  Unlock a waitqueue.
• void rtdm_waitqueue_signal (struct rtdm_wait_queue *wq)
  Signal a waitqueue.
• void rtdm_waitqueue_broadcast (struct rtdm_wait_queue *wq)
  Broadcast a waitqueue.
• void rtdm_waitqueue_flush (struct rtdm_wait_queue *wq)
  Flush a waitqueue.
• void rtdm_waitqueue_wakeup (struct rtdm_wait_queue *wq, rtdm_task_t waiter)
  Signal a particular waiter on a waitqueue.
• rtdm_for_each_waiter (rtdm_task_t pos, struct rtdm_wait_queue *wq)
  Simple iterator for waitqueues.
• rtdm_for_each_waiter_safe (rtdm_task_t pos, rtdm_task_t tmp, struct rtdm_wait_queue *wq)
  Safe iterator for waitqueues.

RTDM_SELECTTYPE_xxx

Event types select can bind to

• enum rtdm_selecttype { RTDM_SELECTTYPE_READ = XNSELECT_READ, RTDM_SELECTTYPE_WRITE = XNSELECT_WRITE, RTDM_SELECTTYPE_EXCEPT = XNSELECT_EXCEPT }

Timeout Sequence Management

• void rtdm_toseq_init (rtdm_toseq_t *timeout_seq, nanosecs_rel_t timeout)
  Initialise a timeout sequence.
6.32 Synchronisation Services

<table>
<thead>
<tr>
<th>Enumerator</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTDM_SELECTTYPE_READ</td>
</tr>
<tr>
<td>RTDM_SELECTTYPE_WRITE</td>
</tr>
<tr>
<td>RTDM_SELECTTYPE_EXCEPT</td>
</tr>
</tbody>
</table>

6.32.3 Function Documentation

6.32.3.1 rtdm_for_each_waiter()

```c
rtdm_for_each_waiter ( 
    rtdm_task_t pos, 
    struct rtdm_wait_queue * wq )
```

Simple iterator for waitqueues.

This construct traverses the wait list of a given waitqueue `wq`, assigning each RTDM task pointer to the cursor variable `pos`, which must be of type `rtdm_task_t`.

`wq` must have been locked by a call to `rtdm_waitqueue_lock()` prior to traversing its wait list.

Parameters

<table>
<thead>
<tr>
<th>pos</th>
<th>cursor variable holding a pointer to the RTDM task being fetched.</th>
</tr>
</thead>
<tbody>
<tr>
<td>wq</td>
<td>waitqueue to scan.</td>
</tr>
</tbody>
</table>

Note

The waitqueue should not be signaled, broadcast or flushed during the traversal, unless the loop is aborted immediately after. Should multiple waiters be readied while iterating, the safe form `rtdm_for_each_waiter_safe()` must be used for traversal instead.

Tags

unrestricted

6.32.3.2 rtdm_for_each_waiter_safe()

```c
rtdm_for_each_waiter_safe ( 
    rtdm_task_t pos, 
    rtdm_task_t tmp, 
    struct rtdm_wait_queue * wq )
```
Safe iterator for waitqueues.

This construct traverses the wait list of a given waitqueue \texttt{wq}, assigning each RTDM task pointer to the cursor variable \texttt{pos}, which must be of type \texttt{rtdm_task_t}.

Unlike with \texttt{rtdm_for_each_waiter()}, the waitqueue may be signaled, broadcast or flushed during the traversal.

\texttt{wq} must have been locked by a call to \texttt{rtdm_waitqueue_lock()} prior to traversing its wait list.

Parameters

\begin{tabular}{|l|l|}
\hline
\texttt{pos} & cursor variable holding a pointer to the RTDM task being fetched. \\
\texttt{tmp} & temporary cursor variable. \\
\texttt{wq} & waitqueue to scan. \\
\hline
\end{tabular}

Tags

\begin{itemize}
  \item \texttt{unrestricted}
\end{itemize}

6.32.3.3 \texttt{rtdm_timedwait()}

\begin{verbatim}
void rtdm_timedwait (  
    struct rtdm_wait_queue *wq,  
    nanosecs_rel_t timeout,  
    rtdm_toseq_t *toseq )
\end{verbatim}

Timed sleep on a waitqueue unconditionally.

The calling task is put to sleep until the waitqueue is signaled by either \texttt{rtdm_waitqueue_signal()} or \texttt{rtdm_waitqueue_broadcast()}, or flushed by a call to \texttt{rtdm_waitqueue_flush()}, or a timeout occurs.

Parameters

\begin{tabular}{|l|l|}
\hline
\texttt{wq} & waitqueue to wait on. \\
\texttt{timeout} & relative timeout in nanoseconds, see \texttt{RTDM_TIMEOUT_xxx} for special values. \\
\texttt{toseq} & handle of a timeout sequence as returned by \texttt{rtdm_toseq_init()} or NULL. \\
\hline
\end{tabular}

Returns

\begin{itemize}
  \item \texttt{-EINTR} is returned if the waitqueue has been flushed, or the calling task has received a Linux signal or has been forcibly unblocked by a call to \texttt{rtdm_task_unblock()}. 
  \item \texttt{-ETIMEDOUT} is returned if the request has not been satisfied within the specified amount of time.
\end{itemize}
6.32 Synchronisation Services

Note

Passing RTDM_TIMEOUT_NONE to timeout makes no sense for such service, and might cause unexpected behavior.

Tags

primary-only, might-switch

6.32.3.4 rtdm_timedwait_condition()

rtdm_timedwait_condition (  
   struct rtdm_wait_queue * wq,  
   C_expr condition,  
   nanosecs_rel_t timeout,  
   rtdm_toseq_t * toseq )

Timed sleep on a waitqueue until a condition gets true.

The calling task is put to sleep until condition evaluates to true or a timeout occurs. The condition is checked each time the waitqueue wq is signaled.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wq</td>
<td>waitqueue to wait on.</td>
</tr>
<tr>
<td>condition</td>
<td>C expression for the event to wait for.</td>
</tr>
<tr>
<td>timeout</td>
<td>relative timeout in nanoseconds, see RTDM_TIMEOUT_xxx for special values.</td>
</tr>
<tr>
<td>in, out</td>
<td>toseq</td>
</tr>
</tbody>
</table>

Returns

0 on success, otherwise:

- -EINTR is returned if calling task has received a Linux signal or has been forcibly unblocked by a call to rtdm_task_unblock().
- -ETIMEDOUT is returned if the if the request has not been satisfied within the specified amount of time.

Note

rtdm_waitqueue_signal() has to be called after changing any variable that could change the result of the wait condition. Passing RTDM_TIMEOUT_NONE to timeout makes no sense for such service, and might cause unexpected behavior.

Tags

primary-only, might-switch
6.32.3.5 rtdm_timedwait_condition_locked()

rtdm_timedwait_condition_locked (  
    struct rtdm_wait_queue *wq,  
    C_expr condition,  
    nanosecs_rel_t timeout,  
    rtdm_toseq_t *toseq )

Timed sleep on a locked waitqueue until a condition gets true.

The calling task is put to sleep until `condition` evaluates to true or a timeout occurs. The condition is checked each time the waitqueue `wq` is signaled.

The waitqueue must have been locked by a call to `rtdm_waitqueue_lock()` prior to calling this service.

Parameters

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wq</td>
<td>locked waitqueue to wait on. The waitqueue lock is dropped when sleeping, then reacquired before this service returns to the caller.</td>
</tr>
<tr>
<td>condition</td>
<td>C expression for the event to wait for.</td>
</tr>
<tr>
<td>timeout</td>
<td>relative timeout in nanoseconds, see <code>RTDM_TIMEOUT_xxx</code> for special values.</td>
</tr>
<tr>
<td>in,out toseq</td>
<td>handle of a timeout sequence as returned by <code>rtdm_toseq_init()</code> or NULL.</td>
</tr>
</tbody>
</table>

Returns

0 on success, otherwise:

- `-EINTR` is returned if calling task has received a Linux signal or has been forcibly unblocked by a call to `rtdm_task_unblock()`.
- `-ETIMEDOUT` is returned if the if the request has not been satisfied within the specified amount of time.

Note

`rtdm_waitqueue_signal()` has to be called after changing any variable that could change the result of the wait condition.
Passing `RTDM_TIMEOUT_NONE` to `timeout` makes no sense for such service, and might cause unexpected behavior.

Tags

`primary-only, might-switch`

6.32.3.6 rtdm_timedwait_locked()

void rtdm_timedwait_locked (  
    struct rtdm_wait_queue *wq,  
    nanosecs_rel_t timeout,  
    rtdm_toseq_t *toseq )

Timed sleep on a locked waitqueue unconditionally.

The calling task is put to sleep until the waitqueue is signaled by either `rtdm_waitqueue_signal()` or `rtdm_waitqueue_broadcast()`, or flushed by a call to `rtdm_waitqueue_flush()`, or a timeout occurs.

The waitqueue must have been locked by a call to `rtdm_waitqueue_lock()` prior to calling this service.
Parameters

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>wq</strong></td>
<td>locked waitqueue to wait on. The waitqueue lock is dropped when sleeping,</td>
</tr>
<tr>
<td></td>
<td>then reacquired before this service returns to the caller.</td>
</tr>
<tr>
<td><strong>timeout</strong></td>
<td>relative timeout in nanoseconds, see RTDM_TIMEOUT_*** for special values.</td>
</tr>
<tr>
<td><strong>in,out</strong></td>
<td>toseq</td>
</tr>
</tbody>
</table>

Returns

0 on success, otherwise:

- -EINTR is returned if the waitqueue has been flushed, or the calling task has received a Linux signal or has been forcibly unblocked by a call to rtdm_task_unblock().
- -ETIMEDOUT is returned if the request has not been satisfied within the specified amount of time.

Note

Passing RTDM_TIMEOUT_NONE to `timeout` makes no sense for such service, and might cause unexpected behavior.

Tags

primary-only, might-switch

6.32.3.7 rtdm_toseq_init()

```c
void rtdm_toseq_init (    
    rtdm_toseq_t * timeout_seq,  
    nanosecs_rel_t timeout )
```

Initialise a timeout sequence.

This service initialises a timeout sequence handle according to the given timeout value. Timeout sequences allow to maintain a continuous `timeout` across multiple calls of blocking synchronisation services. A typical application scenario is given below.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in,out</strong></td>
<td><strong>timeout_seq</strong></td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>timeout</strong></td>
</tr>
</tbody>
</table>

Application Scenario:

```c
int device_service_routine(...)  
```
```c
{ 
    rtdm_toseq_t timeout_seq;
    ...
    rtdm_toseq_init(&timeout_seq, timeout);
    ...
    while (received < requested) {
        ret = rtdm_event_timedwait(&data_available, timeout, &timeout_seq);
        if (ret < 0) // including -ETIMEDOUT
            break;
        // receive some data
    }
    ...
}
```

Using a timeout sequence in such a scenario avoids that the user-provided relative `timeout` is restarted on every call to `rtdm_event_timedwait()`, potentially causing an overall delay that is larger than specified by `timeout`. Moreover, all functions supporting timeout sequences also interpret special timeout values (infinite and non-blocking), disburdening the driver developer from handling them separately.

Tags

```
task-unrestricted
```

### 6.32.3.8 rtdm_wait()

```c
void rtdm_wait ( 
    struct rtdm_wait_queue *wq )
```

Sleep on a waitqueue unconditionally.

The calling task is put to sleep until the waitqueue is signaled by either `rtdm_waitqueue_signal()` or `rtdm_waitqueue_broadcast()`, or flushed by a call to `rtdm_waitqueue_flush()`.

Parameters

```
wq   waitqueue to wait on.
```

Returns

```
0 on success, otherwise:
```

- -EINTR is returned if the waitqueue has been flushed, or the calling task has received a Linux signal or has been forcibly unblocked by a call to `rtdm_task_unblock()`.

Tags

```
primary-only, might-switch
```
6.32 Synchronisation Services

6.32.3.9 rtdm_wait_condition()

rtdm_wait_condition(
    struct rtdm_wait_queue *wq,
    C_expr condition
)

Sleep on a waitqueue until a condition gets true.

The calling task is put to sleep until condition evaluates to true. The condition is checked each time the waitqueue wq is signaled.

Parameters

<table>
<thead>
<tr>
<th>wq</th>
<th>waitqueue to wait on</th>
</tr>
</thead>
<tbody>
<tr>
<td>condition</td>
<td>C expression for the event to wait for</td>
</tr>
</tbody>
</table>

Returns

0 on success, otherwise:

- -EINVAL is returned if calling task has received a Linux signal or has been forcibly unblocked by a call to rtdm_task_unblock().

Note

rtdm_waitqueue_signal() has to be called after changing any variable that could change the result of the wait condition.

Tags

primary-only, might-switch

6.32.3.10 rtdm_wait_condition_locked()

rtdm_wait_condition_locked(
    struct rtdm_wait_queue *wq,
    C_expr condition
)

Sleep on a locked waitqueue until a condition gets true.

The calling task is put to sleep until condition evaluates to true. The condition is checked each time the waitqueue wq is signaled.

The waitqueue must have been locked by a call to rtdm_waitqueue_lock() prior to calling this service.

Parameters

<table>
<thead>
<tr>
<th>wq</th>
<th>locked waitqueue to wait on. The waitqueue lock is dropped when sleeping, then reacquired before this service returns to the caller.</th>
</tr>
</thead>
<tbody>
<tr>
<td>condition</td>
<td>C expression for the event to wait for.</td>
</tr>
</tbody>
</table>
Returns

0 on success, otherwise:

- -EINTR is returned if calling task has received a Linux signal or has been forcibly unblocked by a call to rtdm_task_unblock().

Note

rtdm_waitqueue_signal() has to be called after changing any variable that could change the result of the wait condition.

Tags

primary-only, might-switch

6.32.3.11 rtdm_wait_locked()

void rtdm_wait_locked (
    struct rtdm_wait_queue *wq )

Sleep on a locked waitqueue unconditionally.

The calling task is put to sleep until the waitqueue is signaled by either rtdm_waitqueue_signal() or rtdm_waitqueue_broadcast(), or flushed by a call to rtdm_waitqueue_flush().

The waitqueue must have been locked by a call to rtdm_waitqueue_lock() prior to calling this service.

Parameters

| wq | locked waitqueue to wait on. The waitqueue lock is dropped when sleeping, then reacquired before this service returns to the caller. |

Returns

0 on success, otherwise:

- -EINTR is returned if the waitqueue has been flushed, or the calling task has received a Linux signal or has been forcibly unblocked by a call to rtdm_task_unblock().

Tags

primary-only, might-switch
6.32.3.12 rtdm_waitqueue_broadcast()

void rtdm_waitqueue_broadcast ( 
    struct rtdm_wait_queue * wq )

Broadcast a waitqueue.

Broadcast the waitqueue \texttt{wq}, waking up all waiters. Each readied task may assume to have received the wake up event.

Parameters

\texttt{wq} waitqueue to broadcast.

Returns

non-zero if at least one task has been readied as a result of this call, zero otherwise.

Tags

unrestricted, might-switch

6.32.3.13 rtdm_waitqueue_destroy()

void rtdm_waitqueue_destroy ( 
    struct rtdm_waitqueue * wq )

Deletes a RTDM wait queue.

Dismantles a wait queue structure, releasing all resources attached to it.

Parameters

\texttt{wq} waitqueue to delete.

Tags

task-unrestricted

6.32.3.14 rtdm_waitqueue_flush()

void rtdm_waitqueue_flush ( 
    struct rtdm_wait_queue * wq )

Flush a waitqueue.

Flushes the waitqueue \texttt{wq}, unblocking all waiters with an error status (-EINTR).
Parameters

| wq  | waitqueue to flush. |

Returns

- non-zero if at least one task has been readied as a result of this call, zero otherwise.

Tags

- unrestricted, might-switch

6.32.3.15 rtdm_waitqueue_init()

```c
void rtdm_waitqueue_init(
    struct rtdm_waitqueue *wq)
```

Initialize a RTDM wait queue.

Sets up a wait queue structure for further use.

Parameters

| wq  | waitqueue to initialize. |

Tags

- task-unrestricted

6.32.3.16 rtdm_waitqueue_lock()

```c
void rtdm_waitqueue_lock(
    struct rtdm_wait_queue *wq,
    rtdm_lockctx_t context)
```

Lock a waitqueue.

Acquires the lock on the waitqueue wq.

Parameters

<table>
<thead>
<tr>
<th>wq</th>
<th>waitqueue to lock.</th>
</tr>
</thead>
<tbody>
<tr>
<td>context</td>
<td>name of local variable to store the context in.</td>
</tr>
</tbody>
</table>
Note

Recursive locking might lead to unexpected behavior, including lock up.

Tags

unrestricted

6.32.3.17 rtdm_waitqueue_signal()

```c
void rtdm_waitqueue_signal ( 
    struct rtdm_wait_queue * wq )
```

Signal a waitqueue.

Signals the waitqueue `wq`, waking up a single waiter (if any).

Parameters

| wq | waitqueue to signal. |

Returns

non-zero if a task has been readied as a result of this call, zero otherwise.

Tags

unrestricted, might-switch

6.32.3.18 rtdm_waitqueue_unlock()

```c
void rtdm_waitqueue_unlock ( 
    struct rtdm_wait_queue * wq, 
    rtdm_lockctx_t context )
```

Unlock a waitqueue.

Releases the lock on the waitqueue `wq`.

Parameters

| wq | waitqueue to unlock. |
| context | name of local variable to retrieve the context from. |
6.32.3.19  rtdm_waitqueue_wakeup()

void rtdm_waitqueue_wakeup (    
        struct rtdm_wait_queue *wq,    
        rtdm_task_t waiter )

Signal a particular waiter on a waitqueue.

Signals the waitqueue \texttt{wq}, waking up waiter \texttt{waiter} only, which must be currently sleeping on the waitqueue.

Parameters

| \texttt{wq} | waitqueue to signal. |
| \texttt{waiter} | RTDM task to wake up. |

Tags

\texttt{unrestricted, might-switch}
6.33 Event Services

Collaboration diagram for Event Services:

Functions

- **void rtdm_event_init** (rtdm_event_t *event, unsigned long pending)
  
  *Initialise an event.*

- **void rtdm_event_destroy** (rtdm_event_t *event)
  
  *Destroy an event.*

- **void rtdm_event_pulse** (rtdm_event_t *event)
  
  *Signal an event occurrence to currently listening waiters.*

- **void rtdm_event_signal** (rtdm_event_t *event)
  
  *Signal an event occurrence.*

- **int rtdm_event_wait** (rtdm_event_t *event)
  
  *Wait on event occurrence.*

- **int rtdm_event_timedwait** (rtdm_event_t *event, nanosecs_rel_t timeout, rtdm_toseq_t *timeout_seq)
  
  *Wait on event occurrence with timeout.*

- **void rtdm_event_clear** (rtdm_event_t *event)
  
  *Clear event state.*

- **int rtdm_event_select** (rtdm_event_t *event, rtdm_selector_t *selector, enum rtdm_selecttype type, unsigned int fd_index)
  
  *Bind a selector to an event.*

6.33.1 Detailed Description

6.33.2 Function Documentation

6.33.2.1 rtdm_event_clear()

void rtdm_event_clear ( rtdm_event_t * event )

*Clear event state.*
Parameters

| in, out | event | Event handle as returned by rtdm_event_init() |

Tags

unrestricted

Referenced by a4l_get_time().

6.33.2.2 rtdm_event_destroy()

```c
void rtdm_event_destroy ( 
    rtdm_event_t * event )
```

Destroy an event.

Parameters

| in, out | event | Event handle as returned by rtdm_event_init() |

Tags

task-unrestricted, might-switch

References XNRMID, and xnselect_destroy().

Referenced by a4l_get_time(), and udd_unregister_device().

6.33.2.3 rtdm_event_init()

```c
void rtdm_event_init ( 
    rtdm_event_t * event, 
    unsigned long pending )
```

Initialise an event.

Parameters

| in, out | event | Event handle |
| in | pending | Non-zero if event shall be initialised as set, 0 otherwise |
6.33 Event Services

Tags

- task-unrestricted

Referenced by a4l_get_time().

6.33.2.4 rtdm_event_pulse()

```c
void rtdm_event_pulse ( 
    rtdm_event_t * event )
```

Signal an event occurrence to currently listening waiters.

This function wakes up all current waiters of the given event, but it does not change the event state. Subsequently callers of rtdm_event_wait() or rtdm_event_timedwait() will therefore be blocked first.

Parameters

| in,out | event | Event handle as returned by rtdm_event_init() |

Tags

- unrestricted, might-switch

6.33.2.5 rtdm_event_select()

```c
int rtdm_event_select ( 
    rtdm_event_t * event,
    rtdm_selector_t * selector,
    enum rtdm_selecttype type,
    unsigned int fd_index )
```

Bind a selector to an event.

This function binds the given selector to an event so that the former is notified when the event state changes. Typically the select binding handler will invoke this service.

Parameters

| in,out | event | Event handle as returned by rtdm_event_init() |
| in,out | selector | Selector as passed to the select binding handler |
| in | type | Type of the bound event as passed to the select binding handler |
| in | fd_index | File descriptor index as passed to the select binding handler |
Returns

0 on success, otherwise:

- -ENOMEM is returned if there is insufficient memory to establish the dynamic binding.
- -EINVAL is returned if type or fd_index are invalid.

Tags

task-unrestricted

6.33.2.6 rtdm_event_signal()

```c
void rtdm_event_signal (    
    rtdm_event_t * event )
```

Signal an event occurrence.

This function sets the given event and wakes up all current waiters. If no waiter is presently registered, the next call to rtdm_event_wait() or rtdm_event_timedwait() will return immediately.

Parameters

<table>
<thead>
<tr>
<th>in, out</th>
<th>event</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event handle as returned by rtdm_event_init()</td>
<td></td>
</tr>
</tbody>
</table>

Tags

unrestricted, might-switch

Referenced by a4l_get_time(), and udd_notify_event().

6.33.2.7 rtdm_event_timedwait()

```c
int rtdm_event_timedwait (    
    rtdm_event_t * event,    
    nanosecs_rel_t timeout,    
    rtdm_toseq_t * timeout_seq )
```

Wait on event occurrence with timeout.

This function waits or tests for the occurrence of the given event, taking the provided timeout into account. On successful return, the event is reset.
Parameters

<table>
<thead>
<tr>
<th>in, out</th>
<th>event</th>
<th>Event handle as returned by rtdm_event_init()</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>timeout</td>
<td>Relative timeout in nanoseconds, see RTDM_TIMEOUT_xxx for special values</td>
</tr>
<tr>
<td>in, out</td>
<td>timeout_seq</td>
<td>Handle of a timeout sequence as returned by rtdm_toseq_init() or NULL</td>
</tr>
</tbody>
</table>

Returns

0 on success, otherwise:

- -ETIMEDOUT is returned if the request has not been satisfied within the specified amount of time.
- -EINTR is returned if calling task has been unblock by a signal or explicitly via rtdm_task Unblock().
- -EIDRM is returned if event has been destroyed.
- -EPERM may be returned if an illegal invocation environment is detected.
- -EWOULDBLOCK is returned if a negative timeout (i.e., non-blocking operation) has been specified.

Tags

primary-timed, might-switch

Referenced by a4l_get_time(), and rtdm_event_wait().

6.33.2.8 rtdm_event_wait()

```c
int rtdm_event_wait ( 
    rtdm_event_t * event )
```

Wait on event occurrence.

This is the light-weight version of rtdm_event_timedwait(), implying an infinite timeout.

Parameters

| in, out | event         | Event handle as returned by rtdm_event_init() |

Returns

0 on success, otherwise:

- -EINTR is returned if calling task has been unblock by a signal or explicitly via rtdm_task_unblock().
- -EIDRM is returned if event has been destroyed.
- EPERM may be returned if an illegal invocation environment is detected.

Tags

primary-only, might-switch

References rtdm_event_timedwait().

Referenced by a4l_get_time().
6.34 Semaphore Services

Collaboration diagram for Semaphore Services:

![Collaboration Diagram](image)

Functions

- `void rtdm_sem_init (rtdm_sem_t *sem, unsigned long value)`
  - Initialise a semaphore.
- `void rtdm_sem_destroy (rtdm_sem_t *sem)`
  - Destroy a semaphore.
- `int rtdm_sem_down (rtdm_sem_t *sem)`
  - Decrement a semaphore.
- `int rtdm_sem_timeddown (rtdm_sem_t *sem, nanosecs_rel_t timeout, rtdm_toseq_t *timeout_seq)`
  - Decrement a semaphore with timeout.
- `void rtdm_sem_up (rtdm_sem_t *sem)`
  - Increment a semaphore.
- `int rtdm_sem_select (rtdm_sem_t *sem, rtdm_selector_t *selector, enum rtdm_selecttype type, unsigned int fd_index)`
  - Bind a selector to a semaphore.

6.34.1 Detailed Description

6.34.2 Function Documentation

6.34.2.1 rtdm_sem_destroy()

```c
void rtdm_sem_destroy (rtdm_sem_t *sem )
```

Destroy a semaphore.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in, out</code> sem</td>
<td>Semaphore handle as returned by rtdm_sem_init()</td>
</tr>
</tbody>
</table>
6.34.2.2  rtdm_sem_down()

```c
int rtdm_sem_down (
    rtdm_sem_t * sem )
```

Decrement a semaphore.

This is the light-weight version of `rtdm_sem_timeddown()`, implying an infinite timeout.

Parameters

| in, out | sem | Semaphore handle as returned by `rtdm_sem_init()` |

Returns

0 on success, otherwise:

- `-EINTR` is returned if calling task has been unblock by a signal or explicitly via `rtdm_task_unblock()`.
- `-EIDRM` is returned if `sem` has been destroyed.
- `-EPERM` may be returned if an illegal invocation environment is detected.

Tags

`primary-only, might-switch`

References `rtdm_sem_timeddown()`.

6.34.2.3  rtdm_sem_init()

```c
void rtdm_sem_init ( 
    rtdm_sem_t * sem, 
    unsigned long value )
```

Initialise a semaphore.

Parameters

| in, out | sem | Semaphore handle |
| in      | value | Initial value of the semaphore |
6.34 Semaphore Services

Tags

  task-unrestricted

6.34.2.4 rtdm_sem_select()

```c
int rtdm_sem_select (  
  rtdm_sem_t * sem,  
  rtdm_selector_t * selector,  
  enum rtdm_selecttype type,  
  unsigned int fd_index )
```

Bind a selector to a semaphore.

This functions binds the given selector to the semaphore so that the former is notified when the semaphore state changes. Typically the select binding handler will invoke this service.

Parameters

<table>
<thead>
<tr>
<th>in, out</th>
<th>sem</th>
<th>Semaphore handle as returned by rtdm_sem_init()</th>
</tr>
</thead>
<tbody>
<tr>
<td>in, out</td>
<td>selector</td>
<td>Selector as passed to the select binding handler</td>
</tr>
<tr>
<td>in</td>
<td>type</td>
<td>Type of the bound event as passed to the select binding handler</td>
</tr>
<tr>
<td>in</td>
<td>fd_index</td>
<td>File descriptor index as passed to the select binding handler</td>
</tr>
</tbody>
</table>

Returns

0 on success, otherwise:

- -ENOMEM is returned if there is insufficient memory to establish the dynamic binding.
- -EINVAL is returned if type or fd_index are invalid.

Tags

  task-unrestricted

6.34.2.5 rtdm_sem_timeddown()

```c
int rtdm_sem_timeddown (  
  rtdm_sem_t * sem,  
  nanosecs_rel_t timeout,  
  rtdm_toseq_t * timeout_seq )
```

Decrement a semaphore with timeout.

This function tries to decrement the given semaphore's value if it is positive on entry. If not, the caller is blocked unless non-blocking operation was selected.
### rtdm_sem_down()

This function decrements the given semaphore's value, waking up a potential waiter which was blocked upon \texttt{rtdm\_sem\_down()}.  

#### Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in,out</td>
<td>\texttt{sem}</td>
<td>Semaphore handle as returned by \texttt{rtdm_sem_init()}</td>
</tr>
<tr>
<td>in</td>
<td>\texttt{timeout}</td>
<td>Relative timeout in nanoseconds, see \texttt{RTDM_TIMEOUT_xxx} for special values</td>
</tr>
<tr>
<td>in,out</td>
<td>\texttt{timeout_seq}</td>
<td>Handle of a timeout sequence as returned by \texttt{rtdm_toseq_init()} or NULL</td>
</tr>
</tbody>
</table>

#### Returns

- \texttt{0} on success, otherwise:
  - -\texttt{ETIMEDOUT} is returned if the request has not been satisfied within the specified amount of time.
  - -\texttt{EWOULDBLOCK} is returned if \texttt{timeout} is negative and the semaphore value is currently not positive.
  - -\texttt{EINTR} is returned if calling task has been unblock by a signal or explicitly via \texttt{rtdm\_task\_unblock()}.
  - -\texttt{EIDRM} is returned if \texttt{sem} has been destroyed.
  - -\texttt{EPERM} may be returned if an illegal invocation environment is detected.

#### Tags

- \texttt{primary-timed, might-switch}

Referenced by \texttt{rtdm\_sem\_down()}.  

### 6.34.2.6 rtdm\_sem\_up()

#### Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in,out</td>
<td>\texttt{sem}</td>
<td>Semaphore handle as returned by \texttt{rtdm_sem_init()}</td>
</tr>
</tbody>
</table>

#### Tags

- \texttt{unrestricted, might-switch}
6.35 Mutex services

Collaboration diagram for Mutex services:

![Collaboration diagram for Mutex services](image)

Functions

- `void rtdm_mutex_init (rtdm_mutex_t *mutex)`
  *Initialise a mutex.*
- `void rtdm_mutex_destroy (rtdm_mutex_t *mutex)`
  *Destroy a mutex.*
- `void rtdm_mutex_unlock (rtdm_mutex_t *mutex)`
  *Release a mutex.*
- `int rtdm_mutex_lock (rtdm_mutex_t *mutex)`
  *Request a mutex.*
- `int rtdm_mutex_timedlock (rtdm_mutex_t *mutex, nanosecs_rel_t timeout, rtdm_toseq_t *timeout_seq)`
  *Request a mutex with timeout.*

6.35.1 Detailed Description

6.35.2 Function Documentation

6.35.2.1 rtdm_mutex_destroy()

```c
void rtdm_mutex_destroy (
    rtdm_mutex_t *mutex )
```

*Destroy a mutex.*

Parameters

| in, out | mutex | Mutex handle as returned by rtdm_mutex_init() |

Generated by Doxygen
6.35.2.2 rtdm_mutex_init()

```c
void rtdm_mutex_init (  
  rtdm_mutex_t * mutex )
```

Initialise a mutex.

This function initialises a basic mutex with priority inversion protection. "Basic", as it does not allow a mutex owner to recursively lock the same mutex again.

Parameters

| in, out | mutex | Mutex handle |

Tags

- task-unrestricted

6.35.2.3 rtdm_mutex_lock()

```c
int rtdm_mutex_lock (  
  rtdm_mutex_t * mutex )
```

Request a mutex.

This is the light-weight version of `rtdm_mutex_timedlock()`, implying an infinite timeout.

Parameters

| in, out | mutex | Mutex handle as returned by `rtdm_mutex_init()` |

Returns

0 on success, otherwise:

- -EIDRM is returned if `mutex` has been destroyed.
- -EPERM may be returned if an illegal invocation environment is detected.
6.35 Mutex services

Tags

primary-only, might-switch

References rtdm_mutex_timedlock().

6.35.2.4 rtdm_mutex_timedlock()

```c
int rtdm_mutex_timedlock(
    rtdm_mutex_t *mutex,
    nanosecs_rel_t timeout,
    rtdm_toseq_t *timeout_seq
)
```

Request a mutex with timeout.

This function tries to acquire the given mutex. If it is not available, the caller is blocked unless non-blocking operation was selected.

Parameters

<table>
<thead>
<tr>
<th>in,out</th>
<th>mutex</th>
<th>Mutex handle as returned by rtdm_mutex_init()</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>timeout</td>
<td>Relative timeout in nanoseconds, see RTDM_TIMEOUT_xxx for special values</td>
</tr>
<tr>
<td>in,out</td>
<td>timeout_seq</td>
<td>Handle of a timeout sequence as returned by rtdm_toseq_init() or NULL</td>
</tr>
</tbody>
</table>

Returns

0 on success, otherwise:

- -ETIMEDOUT is returned if the request has not been satisfied within the specified amount of time.
- -EWOULDBLOCK is returned if `timeout` is negative and the semaphore value is currently not positive.
- -EIDRM is returned if `mutex` has been destroyed.
- -EPERM may be returned if an illegal invocation environment is detected.

Tags

primary-only, might-switch

Referenced by rtdm_mutex_lock().

6.35.2.5 rtdm_mutex_unlock()

```c
void rtdm_mutex_unlock(
    rtdm_mutex_t *mutex
)
```

Release a mutex.

This function releases the given mutex, waking up a potential waiter which was blocked upon `rtdm_mutex_lock()`. Referenced by rtdm_mutex_lock().

Generated by Doxygen
Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in, out</td>
<td>mutex</td>
<td>Mutex handle as returned by rtdm_mutex_init()</td>
</tr>
</tbody>
</table>

Tags

- primary-only, might-switch
6.36 Interrupt Management Services

Collaboration diagram for Interrupt Management Services:

Driver programming interface \rightarrow Interrupt Management Services

Macros

- #define rtdm_irq_get_arg(irq_handle, type) ((type *)irq_handle->cookie)
  
  Retrieve IRQ handler argument.

Typedefs

- typedef int(* rtdm_irq_handler_t) (rtdm_irq_t *irq_handle)
  
  Interrupt handler.

Functions

- int rtdm_irq_request (rtdm_irq_t *irq_handle, unsigned int irq_no, rtdm_irq_handler_t handler, unsigned long flags, const char *device_name, void *arg)
  
  Register an interrupt handler.

- int rtdm_irq_free (rtdm_irq_t *irq_handle)
  
  Release an interrupt handler.

- int rtdm_irq_enable (rtdm_irq_t *irq_handle)
  
  Enable interrupt line.

- int rtdm_irq_disable (rtdm_irq_t *irq_handle)
  
  Disable interrupt line.

RTDM_IRQTYPE_xxx

Interrupt registrations flags

- #define RTDM_IRQTYPE_SHARED XN_IRQTYPE_SHARED
  
  Enable IRQ-sharing with other real-time drivers.

- #define RTDM_IRQTYPE_EDGE XN_IRQTYPE_EDGE
  
  Mark IRQ as edge-triggered, relevant for correct handling of shared edge-triggered IRQs.
RTDM_IRQ_xxx

Return flags of interrupt handlers

- #define RTDM_IRQ_NONE XN_IRQ_NONE  
  Unhandled interrupt.
- #define RTDM_IRQ_HANDLED XN_IRQ_HANDLED  
  Denote handled interrupt.
- #define RTDM_IRQ_DISABLE XN_IRQ_DISABLE  
  Request interrupt disabling on exit.

6.36.1 Detailed Description

6.36.2 Macro Definition Documentation

6.36.2.1 rtdm_irq_get_arg

#define rtdm_irq_get_arg(
  irq_handle,
  type 
) ((type ∗)(irq_handle->cookie))

Retrieve IRQ handler argument.

Parameters

<table>
<thead>
<tr>
<th>irq_handle</th>
<th>IRQ handle</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Type of the pointer to return</td>
</tr>
</tbody>
</table>

Returns

The argument pointer registered on rtdm_irq_request() is returned, type-casted to the specified type.

Tags

unrestricted

Referenced by a4l_get_time().

6.36.3 Typedef Documentation
6.36.3.1  rtdm_irq_handler_t

typedef int( rtdm_irq_handler_t) (rtdm_irq_t *irq_handle)

Interrupt handler.
Parameters

| in | irq_handle | IRQ handle as returned by `rtdm_irq_request()` |

Returns

0 or a combination of `RTDM_IRQ_xxx` flags

6.36.4 Function Documentation

6.36.4.1 rtdm_irq_disable()

```c
int rtdm_irq_disable (rtdm_irq_t *irq_handle)
```

Disable interrupt line.

Parameters

| in,out | irq_handle | IRQ handle as returned by `rtdm_irq_request()` |

Returns

0 on success, otherwise negative error code

Note

This service is for exceptional use only. Drivers should always prefer interrupt masking at device level (via corresponding control registers etc.) over masking at line level. Keep in mind that the latter is incompatible with IRQ line sharing and can also be more costly as interrupt controller access requires broader synchronization. Also, such service is solely available from secondary mode. The caller is responsible for excluding such conflicts.

Tags

`secondary-only`

Referenced by `udd_notify_event()`.

6.36.4.2 rtdm_irq_enable()

```c
int rtdm_irq_enable (rtdm_irq_t *irq_handle)
```

Enable interrupt line.
Interrupt Management Services

6.36 Interrupt Management Services

Parameters

| in, out | irq_handle | IRQ handle as returned by rtdm_irq_request() |

Returns
0 on success, otherwise negative error code

Note
This service is for exceptional use only. Drivers should always prefer interrupt masking at device level (via corresponding control registers etc.) over masking at line level. Keep in mind that the latter is incompatible with IRQ line sharing and can also be more costly as interrupt controller access requires broader synchronization. Also, such service is solely available from secondary mode. The caller is responsible for excluding such conflicts.

Tags
secondary-only

Referenced by udd_notify_event().

6.36.4.3 rtdm_irq_free()

int rtdm_irq_free (rtdm_irq_t * irq_handle)

Release an interrupt handler.

Parameters

| in, out | irq_handle | IRQ handle as returned by rtdm_irq_request() |

Returns
0 on success, otherwise negative error code

Note
The caller is responsible for shutting down the IRQ source at device level before invoking this service. In turn, rtdm_irq_free ensures that any pending event on the given IRQ line is fully processed on return from this service.

Tags
secondary-only

Referenced by a4l_get_time(), and udd_unregister_device().

Generated by Doxygen
6.36.4.4 \texttt{rtdm_irq_request()}

```c
int rtdm_irq_request (  
    rtdm_irq_t * irq_handle,  
    unsigned int irq_no,  
    rtdm_irq_handler_t handler,  
    unsigned long flags,  
    const char * device_name,  
    void * arg )
```

Register an interrupt handler.

This function registers the provided handler with an IRQ line and enables the line.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in, out</td>
<td>\textit{irq_handle}</td>
<td>IRQ handle</td>
</tr>
<tr>
<td>in</td>
<td>\textit{irq_no}</td>
<td>Line number of the addressed IRQ</td>
</tr>
<tr>
<td>in</td>
<td>\textit{handler}</td>
<td>Interrupt handler</td>
</tr>
<tr>
<td>in</td>
<td>\textit{flags}</td>
<td>Registration flags, see \texttt{RTDM_IRQTYPE_xxx} for details</td>
</tr>
<tr>
<td>in</td>
<td>\textit{device_name}</td>
<td>Device name to show up in real-time IRQ lists</td>
</tr>
<tr>
<td>in</td>
<td>\textit{arg}</td>
<td>Pointer to be passed to the interrupt handler on invocation</td>
</tr>
</tbody>
</table>

**Returns**

0 on success, otherwise:

- -EINVAL is returned if an invalid parameter was passed.
- -EBUSY is returned if the specified IRQ line is already in use.

**Tags**

\texttt{secondary-only}

Referenced by \texttt{a4l\_get\_time()}. 

Generated by Doxygen
6.37 Non-Real-Time Signalling Services

These services provide a mechanism to request the execution of a specified handler in non-real-time context.

Collaboration diagram for Non-Real-Time Signalling Services:

![Collaboration Diagram](#)

Typedefs

- typedef void(* rtdm_nrtsig_handler_t) (rtdm_nrtsig_t *nrt_sig, void *arg)
  
  Non-real-time signal handler.

Functions

- void rtdm_schedule_nrt_work (struct work_struct *lostage_work)
  
  Put a work task in Linux non real-time global workqueue from primary mode.

- int rtdm_nrtsig_init (rtdm_nrtsig_t *nrt_sig, rtdm_nrtsig_handler_t handler, void *arg)
  
  Register a non-real-time signal handler.

- void rtdm_nrtsig_destroy (rtdm_nrtsig_t *nrt_sig)
  
  Release a non-realtime signal handler.

- void rtdm_nrtsig_pend (rtdm_nrtsig_t *nrt_sig)
  
  Trigger non-real-time signal.

6.37.1 Detailed Description

These services provide a mechanism to request the execution of a specified handler in non-real-time context.

The triggering can safely be performed in real-time context without suffering from unknown delays. The handler execution will be deferred until the next time the real-time subsystem releases the CPU to the non-real-time part.

6.37.2 Typedef Documentation

6.37.2.1 rtdm_nrtsig_handler_t

typedef void(* rtdm_nrtsig_handler_t) (rtdm_nrtsig_t *nrt_sig, void *arg)

Non-real-time signal handler.
Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>nrt_sig</td>
<td>Signal handle pointer as passed to <code>rtdm_nrtsig_init()</code></td>
</tr>
<tr>
<td>in</td>
<td>arg</td>
<td>Argument as passed to <code>rtdm_nrtsig_init()</code></td>
</tr>
</tbody>
</table>

Note

The signal handler will run in soft-IRQ context of the non-real-time subsystem. Note the implications of this context, e.g. no invocation of blocking operations.

6.37.3 Function Documentation

6.37.3.1 `rtdm_nrtsig_destroy()`

```c
void rtdm_nrtsig_destroy ( rtdm_nrtsig_t *nrt_sig )
```

Release a non-realtime signal handler.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in,out</td>
<td>nrt_sig</td>
<td>Signal handle</td>
</tr>
</tbody>
</table>

Tags

- task-unrestricted

Referenced by `a4l_get_time()`.

6.37.3.2 `rtdm_nrtsig_init()`

```c
int rtdm_nrtsig_init ( rtdm_nrtsig_t *nrt_sig, rtdm_nrtsig_handler_t handler, void *arg )
```

Register a non-real-time signal handler.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in,out</td>
<td>nrt_sig</td>
<td>Signal handle</td>
</tr>
<tr>
<td>in</td>
<td>handler</td>
<td>Non-real-time signal handler</td>
</tr>
<tr>
<td>in</td>
<td>arg</td>
<td>Custom argument passed to <code>handler()</code> on each invocation</td>
</tr>
</tbody>
</table>
Returns

0 on success, otherwise:

- EAGAIN is returned if no free signal slot is available.

Tags

task-unrestricted

Referenced by a4l_get_time().

6.37.3.3 rtdm_nrtsig_pend()

void rtdm_nrtsig_pend (  
    rtdm_nrtsig_t *nrt_sig  
)

Trigger non-real-time signal.

Parameters

| in, out | nrt_sig | Signal handle |

Tags

unrestricted

Referenced by a4l_get_time().

6.37.3.4 rtdm_schedule_nrt_work()

void rtdm_schedule_nrt_work (  
    struct work_struct *lostage_work  
)

Put a work task in Linux non-real-time global workqueue from primary mode.

Parameters

| lostage_work |

Generated by Doxygen
6.38 Utility Services

Collaboration diagram for Utility Services:

![Collaboration diagram](image)

Functions

- int `rtdm_mmap_to_user` (struct rtdm_fd *fd, void *src_addr, size_t len, int prot, void **pptr, struct vm_operations_struct *vm_ops, void *vm_private_data)
  
  Map a kernel memory range into the address space of the user.

- int `rtdm_iomap_to_user` (struct rtdm_fd *fd, phys_addr_t src_addr, size_t len, int prot, void **pptr, struct vm_operations_struct *vm_ops, void *vm_private_data)
  
  Map an I/O memory range into the address space of the user.

- int `rtdm_mmap_kmem` (struct vm_area_struct *vma, void *va)
  
  Map a kernel logical memory range to a virtual user area.

- int `rtdm_mmap_vmem` (struct vm_area_struct *vma, void *va)
  
  Map a virtual memory range to a virtual user area.

- int `rtdm_mmap_iomem` (struct vm_area_struct *vma, phys_addr_t pa)
  
  Map an I/O memory range to a virtual user area.

- int `rtdm_munmap` (void *ptr, size_t len)
  
  Unmap a user memory range.

- int `rtdm_ratelimit` (struct rtdm_ratelimit_state *rs, const char *func)
  
  Enforces a rate limit.

- void `rtdm_printk_ratelimited` (const char *format,...)
  
  Real-time safe rate-limited message printing on kernel console.

- void `rtdm_printk` (const char *format,...)
  
  Real-time safe message printing on kernel console.

- void * `rtdm_malloc` (size_t size)
  
  Allocate memory block.

- void `rtdm_free` (void *ptr)
  
  Release real-time memory block.

- int `rtdm_read_user_ok` (struct rtdm_fd *fd, const void __user *ptr, size_t size)
  
  Check if read access to user-space memory block is safe.

- int `rtdm_rwlock_user_ok` (struct rtdm_fd *fd, const void __user *ptr, size_t size)
  
  Check if read/write access to user-space memory block is safe.

- int `rtdm_copy_to_user` (struct rtdm_fd *fd, void __user *dst, const void __user *src, size_t size)
  
  Copy user-space memory block to specified buffer.

- int `rtdm_safe_copy_to_user` (struct rtdm_fd *fd, void *dst, const void __user *src, size_t size)
  
  Check if read access to user-space memory block and copy it to specified buffer.

- int `rtdm_copy_to_user` (struct rtdm_fd *fd, void __user *dst, const void *src, size_t size)
  
  Copy specified buffer to user-space memory block.
6.38 Utility Services

- int rtdm_safe_copy_to_user (struct rtdm_fd *fd, void __user *dst, const void *src, size_t size)
  Check if read/write access to user-space memory block is safe and copy specified buffer to it.
- int rtdm_strncpy_from_user (struct rtdm_fd *fd, char *dst, const char __user *src, size_t count)
  Copy user-space string to specified buffer.
- int rtdm_in_rt_context (void)
  Test if running in a real-time task.
- int rtdm_rt_capable (struct rtdm_fd *fd)
  Test if the caller is capable of running in real-time context.

6.38.1 Detailed Description

6.38.2 Function Documentation

6.38.2.1 rtdm_copy_from_user()

int rtdm_copy_from_user (  
    struct rtdm_fd *fd,  
    void *dst,  
    const void __user *src,  
    size_t size  
)

Copy user-space memory block to specified buffer.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>fd</th>
<th>RTDM file descriptor as passed to the invoked device operation handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>dst</td>
<td>Destination buffer address</td>
</tr>
<tr>
<td>in</td>
<td>src</td>
<td>Address of the user-space memory block</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>Size of the memory block</td>
</tr>
</tbody>
</table>

Returns

0 on success, otherwise:

- -EFAULT is returned if an invalid memory area was accessed.

Note

Before invoking this service, verify via rtdm_read_user_ok() that the provided user-space address can securely be accessed.

Tags

  task-unrestricted

Referenced by rtdm_ratelimit().

Generated by Doxygen
6.38.2.2  rtdm_copy_to_user()

```c
int rtdm_copy_to_user (    
    struct rtdm_fd * fd,     
    void __user * dst,       
    const void * src,        
    size_t size )
```

Copy specified buffer to user-space memory block.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>fd</th>
<th>RTDM file descriptor as passed to the invoked device operation handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>dst</td>
<td>Address of the user-space memory block</td>
</tr>
<tr>
<td>in</td>
<td>src</td>
<td>Source buffer address</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>Size of the memory block</td>
</tr>
</tbody>
</table>

**Returns**

0 on success, otherwise:

- -EFAULT is returned if an invalid memory area was accessed.

**Note**

Before invoking this service, verify via `rtdm_rw_user_ok()` that the provided user-space address can securely be accessed.

**Tags**

`task-unrestricted`

Referenced by `rtdm_ratelimit()`.

6.38.2.3  rtdm_free()

```c
void rtdm_free (    
    void * ptr )
```

Release real-time memory block.

**Parameters**

| in  | ptr     | Pointer to memory block as returned by `rtdm_malloc()`               |

Generated by Doxygen
6.38 Utility Services

Tags

unrestricted

Referenced by rtdm_ratelimit().

6.38.2.4 rtdm_in_rt_context()

int rtdm_in_rt_context (  
    void  )

Test if running in a real-time task.

Returns

Non-zero is returned if the caller resides in real-time context, 0 otherwise.

Tags

unrestricted

Referenced by rtdm_ratelimit().

6.38.2.5 rtdm_iomap_to_user()

int rtdm_iomap_to_user (  
    struct rtdm_fd * fd,  
    phys_addr_t src_addr,  
    size_t len,  
    int prot,  
    void * * pptr,  
    struct vm_operations_struct * vm_ops,  
    void * vm_private_data  )

Map an I/O memory range into the address space of the user.

Parameters

| in   | fd             | RTDM file descriptor as passed to the invoked device operation handler |
| in   | src_addr       | physical I/O address to be mapped                                      |
| in   | len            | Length of the memory range                                             |
| in   | prot           | Protection flags for the user's memory range, typically either PROT_READ or PROT_READ|PROT_WRITE |
| in,out| pptr           | Address of a pointer containing the desired user address or NULL on entry and the finally assigned address on return |
| in   | vm_ops         | vm_operations to be executed on the vm_area of the user memory range or NULL |

Generated by Doxygen
Returns

0 on success, otherwise (most common values):

- -EINVAL is returned if an invalid start address, size, or destination address was passed.
- -ENOMEM is returned if there is insufficient free memory or the limit of memory mapping for the user process was reached.
- -EAGAIN is returned if too much memory has been already locked by the user process.
- -EPERM may be returned if an illegal invocation environment is detected.

Note

RTDM supports two models for unmapping the memory area:

- manual unmapping via `rtdm_munmap()`, which may be issued from a driver in response to an IOCTL call, or by a call to the regular munmap() call from the application.
- automatic unmapping, triggered by the termination of the process which owns the mapping. To track the number of references pending on the resource mapped, the driver can pass the address of a close handler for the vm_area considered, in the `vm_ops` descriptor. See the relevant Linux kernel programming documentation (e.g. Linux Device Drivers book) on virtual memory management for details.

Tags

- secondary-only

### 6.38.2.6 rtdm_malloc()

```c
void* rtdm_malloc (size_t size)
```

Allocate memory block.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>size</th>
<th>Requested size of the memory block</th>
</tr>
</thead>
</table>

Returns

The pointer to the allocated block is returned on success, NULL otherwise.

Tags

- unrestricted

Referenced by `a4l_alloc_subd()`, and `rtdm_ratelimit()`.
6.38.2.7 rtdm_mmap_iomem()

```
int rtdm_mmap_iomem (
    struct vm_area_struct * vma,
    phys_addr_t pa )
```

Map an I/O memory range to a virtual user area.

This routine is commonly used from a \texttt{->mmap()} handler of a RTDM driver, for mapping an I/O memory area over the user address space referred to by \texttt{vma}.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>vma</th>
<th>The VMA descriptor to receive the mapping.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>pa</td>
<td>The physical I/O address to be mapped.</td>
</tr>
</tbody>
</table>

Returns

0 on success, otherwise a negated error code is returned.

Note

To map a chunk of logical space obtained from \texttt{kmalloc()}, or a purely virtual area with no direct physical mapping to a VMA, call \texttt{rtdm_mmap_kmem()} or \texttt{rtdm_mmap_vmem()} respectively instead.

Tags

\texttt{secondary-only}

6.38.2.8 rtdm_mmap_kmem()

```
int rtdm_mmap_kmem (    
    struct vm_area_struct * vma,    
    void * va )
```

Map a kernel logical memory range to a virtual user area.

This routine is commonly used from a \texttt{->mmap()} handler of a RTDM driver, for mapping a virtual memory area with a direct physical mapping over the user address space referred to by \texttt{vma}.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>vma</th>
<th>The VMA descriptor to receive the mapping.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>va</td>
<td>The kernel logical address to be mapped.</td>
</tr>
</tbody>
</table>
Returns

0 on success, otherwise a negated error code is returned.

Note

This service works on memory regions allocated via kmalloc(). To map a chunk of virtual space
with no direct physical mapping, or a physical I/O memory to a VMA, call rtdm_mmap_vmem() or
rtdm_mmap_iomem() respectively instead.

Tags

secondary-only

6.38.2.9  rtdm_mmap_to_user()

int rtdm_mmap_to_user (  
    struct rtdm_fd *fd,
    void *src_addr,
    size_t len,
    int prot,
    void **pptr,
    struct vm_operations_struct *vm_ops,
    void *vm_private_data )

Map a kernel memory range into the address space of the user.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>fd</td>
<td>RTDM file descriptor as passed to the invoked device operation handler</td>
</tr>
<tr>
<td>in</td>
<td>src_addr</td>
<td>Kernel virtual address to be mapped</td>
</tr>
<tr>
<td>in</td>
<td>len</td>
<td>Length of the memory range</td>
</tr>
<tr>
<td>in</td>
<td>prot</td>
<td>Protection flags for the user's memory range, typically either PROT_READ or PROT_READ PROT_WRITE</td>
</tr>
<tr>
<td>in,out</td>
<td>pptr</td>
<td>Address of a pointer containing the desired user address or NULL on entry and the finally assigned address on return</td>
</tr>
<tr>
<td>in</td>
<td>vm_ops</td>
<td>vm_operations to be executed on the vm_area of the user memory range or NULL</td>
</tr>
<tr>
<td>in</td>
<td>vm_private_data</td>
<td>Private data to be stored in the vm_area, primarily useful for vm_operation handlers</td>
</tr>
</tbody>
</table>

Returns

0 on success, otherwise (most common values):

- EINVAL is returned if an invalid start address, size, or destination address was passed.
- ENOMEM is returned if there is insufficient free memory or the limit of memory mapping for the user process was reached.
- EAGAIN is returned if too much memory has been already locked by the user process.
- EPERM may be returned if an illegal invocation environment is detected.

Note
This service only works on memory regions allocated via kmalloc() or vmalloc(). To map physical I/O memory to user-space use rtdm_iomap_to_user() instead.
RTDM supports two models for unmapping the memory area:
- manual unmapping via rtdm_munmap(), which may be issued from a driver in response to an IOCTL call, or by a call to the regular munmap() call from the application.
- automatic unmapping, triggered by the termination of the process which owns the mapping. To track the number of references pending on the resource mapped, the driver can pass the address of a close handler for the vm_area considered, in the vm_ops descriptor. See the relevant Linux kernel programming documentation (e.g. Linux Device Drivers book) on virtual memory management for details.

Tags
- secondary-only

6.38.2.10 rtdm_mmap_vmem()

int rtdm_mmap_vmem ( 
    struct vm_area_struct * vma,
    void * va )

Map a virtual memory range to a virtual user area.

This routine is commonly used from a - > mmap() handler of a RTDM driver, for mapping a purely virtual memory area over the user address space referred to by vma.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>vma</th>
<th>The VMA descriptor to receive the mapping.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>va</td>
<td>The virtual address to be mapped.</td>
</tr>
</tbody>
</table>

Returns
0 on success, otherwise a negated error code is returned.

Note
This service works on memory regions allocated via vmalloc(). To map a chunk of logical space obtained from kmalloc(), or a physical I/O memory to a VMA, call rtdm_mmap_kmem() or rtdm-_mmap_iomem() respectively instead.

Tags
- secondary-only
6.38.2.11  rtdm_munmap()

```c
int rtdm_munmap ( 
    void * ptr, 
    size_t len )
```

Unmap a user memory range.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>ptr</th>
<th>User address or the memory range</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>len</td>
<td>Length of the memory range</td>
</tr>
</tbody>
</table>

Returns

0 on success, otherwise:

- -EINVAL is returned if an invalid address or size was passed.
- -EPERM may be returned if an illegal invocation environment is detected.

Tags

secondary-only

6.38.2.12  rtdm_printk()

```c
void rtdm_printk ( 
    const char * format, 
    ... )
```

Real-time safe message printing on kernel console.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>format</th>
<th>Format string (conforming standard printf())</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>...</td>
<td>Arguments referred by format</td>
</tr>
</tbody>
</table>

Returns

On success, this service returns the number of characters printed. Otherwise, a negative error code is returned.

Tags

unrestricted

Referenced by rtdm_ratelimit().
6.38.2.13 rtdm_printk_ratelimited()

```c
void rtdm_printk_ratelimited ( 
    const char * format, 
    ... )
```

Real-time safe rate-limited message printing on kernel console.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>format</td>
<td>Format string (conforming standard printf())</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>Arguments referred by format</td>
</tr>
</tbody>
</table>

**Returns**

On success, this service returns the number of characters printed. Otherwise, a negative error code is returned.

**Tags**

unrestricted

Referenced by rtdm_ratelimit().

---

6.38.2.14 rtdm_ratelimit()

```c
int rtdm_ratelimit ( 
    struct rtdm_ratelimit_state * rs, 
    const char * func )
```

Enforces a rate limit.

This function enforces a rate limit: not more than `rs->burst` callbacks in every `rs->interval`.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in,out</td>
<td>rs</td>
<td>rtdm_ratelimit_state data</td>
</tr>
<tr>
<td>in</td>
<td>func</td>
<td>name of calling function</td>
</tr>
</tbody>
</table>

**Returns**

0 means callback will be suppressed and 1 means go ahead and do it

**Tags**

unrestricted

Generated by Doxygen
References rtdm_clock_read(), rtdm_copy_from_user(), rtdm_copy_to_user(), rtdm_free(), rtdm_in_rt_context(), rtdm_lock_get_irqsave(), rtdm_lock_put_irqrestore(), rtdm_malloc(), rtdm_printk(), rtdm_printk_ratelimited(), rtdm_read_user_ok(), rtdm_rt_capable(), rtdm_rw_user_ok(), rtdm_safe_copy_from_user(), rtdm_safe_copy_to_user(), and rtdm_strncpy_from_user.

6.38.2.15 rtdm_read_user_ok()

```
int rtdm_read_user_ok (  
    struct rtdm_fd *fd,  
    const void __user *ptr,  
    size_t size )
```

Check if read access to user-space memory block is safe.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>fd</th>
<th>RTDM file descriptor as passed to the invoked device operation handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>ptr</td>
<td>Address of the user-provided memory block</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>Size of the memory block</td>
</tr>
</tbody>
</table>

**Returns**

Non-zero is return when it is safe to read from the specified memory block, 0 otherwise.

**Tags**

task-unrestricted

Referenced by rtdm_ratelimit().

6.38.2.16 rtdm_rt_capable()

```
int rtdm_rt_capable (  
    struct rtdm_fd *fd )
```

Test if the caller is capable of running in real-time context.

**Parameters**

| in | fd   | RTDM file descriptor as passed to the invoked device operation handler |

**Returns**

Non-zero is returned if the caller is able to execute in real-time context (independent of its current execution mode), 0 otherwise.
Note

This function can be used by drivers that provide different implementations for the same service depending on the execution mode of the caller. If a caller requests such a service in non-real-time context but is capable of running in real-time as well, it might be appropriate for the driver to reject the request via -ENOSYS so that RTDM can switch the caller and restart the request in real-time context.

Tags

unrestricted

Referenced by rtdm_ratelimit().

6.38.2.17 rtdm_rw_user_ok()

int rtdm_rw_user_ok (  
    struct rtdm_fd * fd,  
    const void __user * ptr,  
    size_t size )

Check if read/write access to user-space memory block is safe.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>fd</th>
<th>RTDM file descriptor as passed to the invoked device operation handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>ptr</td>
<td>Address of the user-provided memory block</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>Size of the memory block</td>
</tr>
</tbody>
</table>

Returns

Non-zero is return when it is safe to read from or write to the specified memory block, 0 otherwise.

Tags

task-unrestricted

Referenced by rtdm_ratelimit().

6.38.2.18 rtdm_safe_copy_from_user()

int rtdm_safe_copy_from_user (  
    struct rtdm_fd * fd,  
    void * dst,  
    const void __user * src,  
    size_t size )

Check if read access to user-space memory block and copy it to specified buffer.
6.38.2.19  

rtdm_safe_copy_to_user()  

int rtdm_safe_copy_to_user (  
    struct rtdm_fd *fd,  
    void __user *dst,  
    const void *src,  
    size_t size )  

Check if read/write access to user-space memory block is safe and copy specified buffer to it.

Parameters  

<table>
<thead>
<tr>
<th>in</th>
<th>fd</th>
<th>RTDM file descriptor as passed to the invoked device operation handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>dst</td>
<td>Destination buffer address</td>
</tr>
<tr>
<td>in</td>
<td>src</td>
<td>Address of the user-space memory block</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>Size of the memory block</td>
</tr>
</tbody>
</table>

Returns  

0 on success, otherwise:

- -EFAULT is returned if an invalid memory area was accessed.

Note

This service is a combination of rtdm_read_user_ok and rtdm_copy_from_user.

Tags

- task-unrestricted

Referenced by rtdm_ratelimit().
Note

This service is a combination of rtdm_rw_user_ok and rtdm_copy_to_user.

Tags

task-unrestricted

Referenced by rtdm_ratelimit().

6.38.2.20 rtdm_strncpy_from_user()

```c
int rtdm_strncpy_from_user(
    struct rtdm_fd *fd,
    char * dst,
    const char __user * src,
    size_t count )
```

Copy user-space string to specified buffer.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>fd</strong></td>
<td>RTDM file descriptor as passed to the invoked device operation handler</td>
</tr>
<tr>
<td><strong>dst</strong></td>
<td>Destination buffer address</td>
</tr>
<tr>
<td><strong>src</strong></td>
<td>Address of the user-space string</td>
</tr>
<tr>
<td><strong>count</strong></td>
<td>Maximum number of bytes to copy, including the trailing '0'</td>
</tr>
</tbody>
</table>

Returns

Length of the string on success (not including the trailing '0'), otherwise:

- -EFAULT is returned if an invalid memory area was accessed.

Note

This service already includes a check of the source address, calling rtdm_read_user_ok() for src explicitly is not required.

Tags

task-unrestricted

Referenced by rtdm_ratelimit().
6.39  SCHED_QUOTA scheduling policy

The SCHED_QUOTA policy enforces a limitation on the CPU consumption of threads over a globally defined period, known as the quota interval.

Collaboration diagram for SCHED_QUOTA scheduling policy:

6.39.1 Detailed Description

The SCHED_QUOTA policy enforces a limitation on the CPU consumption of threads over a globally defined period, known as the quota interval.

This is done by pooling threads with common requirements in groups, and giving each group a share of the global period (CONFIG_XENO_OPT_SCHED_QUOTA_PERIOD).

When threads have entirely consumed the quota allotted to the group they belong to, the latter is suspended as a whole, until the next quota interval starts. At this point, a new runtime budget is given to each group, in accordance with its share.
6.40 Thread scheduling control

Collaboration diagram for Thread scheduling control:

```
| Cobalt kernel | Thread scheduling control | SCHED_QUOTA scheduling policy |
```

Modules

- **SCHED_QUOTA scheduling policy**
  
  *The SCHED_QUOTA policy enforces a limitation on the CPU consumption of threads over a globally defined period, known as the quota interval.*

Data Structures

- **struct xnsched**
  
  *Scheduling information structure.*

Functions

- **static int xnsched_run (void)**
  
  *The rescheduling procedure.*

- **static void xnsched_rotate (struct xnsched *sched, struct xnsched_class *sched_class, const union xnsched_policy_param *sched_param)**
  
  *Rotate a scheduler runqueue.*

6.40.1 Detailed Description

6.40.2 Function Documentation

6.40.2.1 xnsched_rotate()

```
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sched</td>
<td>The per-CPU scheduler hosting the target scheduling class.</td>
</tr>
<tr>
<td>sched_class</td>
<td>The scheduling class which should rotate its runqueue.</td>
</tr>
<tr>
<td>sched_param</td>
<td>The scheduling parameter providing rotation information to the specified scheduling class.</td>
</tr>
</tbody>
</table>

Tags

unrestricted, atomic-entry

6.40.2.2 xnsched_run()

The rescheduling procedure.

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 performs context switches as needed. xnsched_run() schedules out the current thread if:

- the current thread is about to block.
- 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 (i.e. round-robin).

The Cobalt core 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. When all changes have been applied, xnsched_run() should be called for considering those changes, and possibly switching context.

As a notable exception to the previous principle however, every action which ends up suspending the current thread begets an implicit call to the rescheduling procedure on behalf of the blocking service.

Typically, self-suspension or sleeping on a synchronization object automatically leads to a call to the rescheduling procedure, therefore the caller does not need to explicitly issue xnsched_run() after such operations.

The rescheduling procedure always leads to a null-effect if it is called on behalf of an interrupt service routine. 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.

Returns

Non-zero is returned if a context switch actually happened, otherwise zero if the current thread was left running.

Tags

unrestricted
6.41 Synchronous I/O multiplexing

This module implements the services needed for implementing the POSIX select() service, or any other event multiplexing services.

Collaboration diagram for Synchronous I/O multiplexing:

```
Cobalt kernel  Synchronous I/O multiplexing
```

Functions

- **void xnselect_init (struct xnselect *select_block)**
  
  Initialize a struct xnselect structure.

- **static int xnselect_signal (struct xnselect *select_block, unsigned int state)**
  
  Signal a file descriptor state change.

- **void xnselect_destroy (struct xnselect *select_block)**
  
  Destroy the xnselect structure associated with a file descriptor.

- **void 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.

- **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.

6.41.1 Detailed Description

This module implements the services needed for implementing the POSIX select() service, or any other event 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.
6.41.2 Function Documentation

6.41.2.1 xnselect()

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>selector</td>
<td>structure to check for pending events</td>
</tr>
<tr>
<td>out_fds</td>
<td>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;</td>
</tr>
<tr>
<td>in_fds</td>
<td>the set of descriptors which events should be checked</td>
</tr>
<tr>
<td>nfds</td>
<td>the highest-numbered descriptor in any of the in_fds sets, plus 1;</td>
</tr>
<tr>
<td>timeout</td>
<td>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.</td>
</tr>
<tr>
<td>timeout_mode</td>
<td>the mode of timeout.</td>
</tr>
</tbody>
</table>

**Return values**

<table>
<thead>
<tr>
<th>Return</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-EINVAL</td>
<td>if nfds is negative;</td>
</tr>
<tr>
<td>-ECHRNG</td>
<td>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;</td>
</tr>
<tr>
<td>-EINTR</td>
<td>if xnselect was interrupted while waiting;</td>
</tr>
<tr>
<td>0</td>
<td>in case of timeout.</td>
</tr>
<tr>
<td>the</td>
<td>number of file descriptors having received an event.</td>
</tr>
</tbody>
</table>

**Tags**

primary-only, might-switch

6.41.2.2 xnselect_bind()

```c
int xnselect_bind ( 
    struct xnselect * select_block,
```
6.41 Synchronous I/O multiplexing

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>select_block</code></td>
<td>pointer to the <code>struct xnselect</code> to be bound;</td>
</tr>
<tr>
<td><code>binding</code></td>
<td>pointer to a newly allocated (using <code>xnmalloc</code>) <code>struct xnselect_binding</code>;</td>
</tr>
<tr>
<td><code>selector</code></td>
<td>pointer to the selector structure;</td>
</tr>
<tr>
<td><code>type</code></td>
<td>type of events (<code>XNSELECT_READ</code>, <code>XNSELECT_WRITE</code>, or <code>XNSELECT_EXCEPT</code>);</td>
</tr>
<tr>
<td><code>index</code></td>
<td>index of the file descriptor (represented by <code>select_block</code>) in the bit fields used by the <code>selector</code> structure;</td>
</tr>
<tr>
<td><code>state</code></td>
<td>current state of the file descriptor.</td>
</tr>
</tbody>
</table>

`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**

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-EINVAL</code></td>
<td>if <code>type</code> or <code>index</code> is invalid;</td>
</tr>
<tr>
<td><code>0</code></td>
<td>otherwise.</td>
</tr>
</tbody>
</table>

**Tags**

`task-unrestricted`, `might-switch`, `atomic-entry`

6.41.2.3 `xnselect_destroy()`

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>select_block</code></td>
<td>pointer to the <code>xnselect</code> structure associated with a file descriptor</td>
</tr>
</tbody>
</table>
Referenced by rtdm_event_destroy(), and rtdm_sem_destroy().

6.41.2.4  xnselect_init()

```c
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 |

Tags

- `task-unrestricted`

6.41.2.5  xnselect_signal()

```c
static int xnselect_signal ( 
   struct xnselect * select_block, 
   unsigned int 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.
6.41.2.6 xnselector_destroy()

```c
void xnselector_destroy ( 
    struct xnselector * selector )
```

Destroy a selector block.

All bindings with file descriptor are destroyed.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>selector</td>
<td>the selector block to be destroyed</td>
</tr>
</tbody>
</table>

Tags

- `task-unrestricted`

6.41.2.7 xnselector_init()

```c
int xnselector_init ( 
    struct xnselector * selector )
```

Initialize a selector structure.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>selector</td>
<td>The selector structure to be initialized.</td>
</tr>
</tbody>
</table>

Return values

- 0

Tags

- `task-unrestricted`

References xnsynch_init().
6.42 Thread synchronization services

Collaboration diagram for Thread synchronization services:

```
Cobalt kernel

Thread synchronization services
```

Functions

- void `xnsynch_init` (struct `xnsynch *synch`, int flags, `atomic_t *fastlock`)
  
  *Initialize a synchronization object.*

- int `xnsynch_destroy` (struct `xnsynch *synch`)
  
  *Destroy a synchronization object.*

- int `__must_check 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`)
  
  *Unblock the heading thread from wait.*

- void `xnsynch_wakeup_this_sleeper` (struct `xnsynch *synch`, struct `xnthread *sleeper`)
  
  *Unblock a particular thread from wait.*

- int `__must_check xnsynch_acquire` (struct `xnsynch *synch`, `xnticks_t timeout`, `xntmode_t timeout_mode`)
  
  *Acquire the ownership of a synchronization object.*

- int `__must_check xnsynch_try_acquire` (struct `xnsynch *synch`)
  
  *Try acquiring the ownership of a synchronization object.*

- struct `xnthread * xnsynch_release` (struct `xnsynch *synch`, struct `xnthread *thread`)
  
  *Give the resource ownership to the next waiting thread.*

- struct `xnthread * xnsynch.peek_pendq` (struct `xnsynch *synch`)
  
  *Access the thread leading a synch object wait queue.*

- int `xnsynch_flush` (struct `xnsynch *synch`, int reason)
  
  *Unblock all waiters pending on a resource.*

6.42.1 Detailed Description

6.42.2 Function Documentation
6.42.2.1 xnsynch_acquire()

```c
int 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 another thread, the caller is suspended.

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>synch</td>
<td>The descriptor address of the synchronization object to acquire.</td>
</tr>
<tr>
<td>timeout</td>
<td>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.</td>
</tr>
<tr>
<td>timeout_mode</td>
<td>The mode of the timeout parameter. It can either be set to XN_RELATIVE, XN_ABSOLUTE, or XN_REALTIME (see also xntimer_start()).</td>
</tr>
</tbody>
</table>

Returns

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

Tags

primary-only, might-switch

Note

Unlike xnsynch_try_acquire(), this call does NOT check for invalid recursive locking request, which means that such request will always cause a deadlock for the caller.

References xnthread_current().

6.42.2.2 xnsynch_destroy()

```c
int xnsynch_destroy (  
    struct xnsynch * synch )
```

Destroy a synchronization object.

Destroys the synchronization object synch, unblocking all waiters with the XNRMID status.
Returns

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

Side effects

Same as xnsynch_flush().

Tags

task-unrestricted

References XNRMID, and xnsynch_flush().

6.42.2.3 xnsynch_flush()

int xnsynch_flush (
    struct xnsynch ∗synch,
    int 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 xnthread_resume()).
- XNBREAK should be set to indicate that the wait has been forcibly interrupted (see xnthread←unblock()).

Returns

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

• After this operation has completed, the synchronization object is not owned by any thread.

Tags

unrestricted

Referenced by xnsynch_destroy().

6.42.2.4 xnsynch_init()

```c
void xnsynch_init (
    struct xnsynch * synch,
    int flags,
    atomic_t * fastlock )
```

Initialize a synchronization object.

Initializes a synchronization object. Xenomai threads can wait on and signal such objects for serializing access to resources. This object has built-in support for priority inheritance.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>synch</td>
<td>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.</td>
</tr>
<tr>
<td>flags</td>
<td>A set of creation flags affecting the operation. The valid flags are:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flags</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XNSYNCH_PRIO</td>
<td>causes the threads waiting for the resource to pend in priority order. Otherwise, FIFO ordering is used (XNSYNCH_FIFO).</td>
</tr>
<tr>
<td>XNSYNCH.Owner</td>
<td>indicates that the synchronization object shall track the resource ownership, allowing a single owner at most at any point in time. Note that setting this flag implies the use of xnsynch_acquire() and xnsynch_release() instead of xnsynch_sleep_on() and xnsynch_wakeup_*().</td>
</tr>
<tr>
<td>XNSYNCH_PIP</td>
<td>enables priority inheritance when a priority inversion is detected among threads using this object. XNSYNCH_PIP enables XNSYNCH.Owner and XNSYNCH_PRIO implicitly.</td>
</tr>
<tr>
<td>XNSYNCH_DREORDER</td>
<td>(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 xnthread_set_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).</td>
</tr>
</tbody>
</table>
Parameters

| fastlock | Address of the fast lock word to be associated with a synchronization object with ownership tracking. Therefore, a valid fast-lock address is required if XNSYNCH_OWNER is set in flags. |

Tags

- task-unrestricted

Referenced by xnselector_init().

6.42.2.5  xnsynch.peek_pendq()

```
struct xnthread * xnsynch.peek_pendq (  
    struct xnsynch * synch )
```

Access the thread leading a synch object wait queue.

This service returns the descriptor address of 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.

Tags

- unrestricted

6.42.2.6  xnsynch.release()

```
struct xnthread * xnsynch.release (  
    struct xnsynch * synch,  
    struct xnthread * thread )
```

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

<table>
<thead>
<tr>
<th>synch</th>
<th>The descriptor address of the synchronization object whose ownership is changed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>thread</td>
<td>The descriptor address of the current owner.</td>
</tr>
</tbody>
</table>

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 transferred to the unblocked thread.

Tags

primary-only, might-switch

6.42.2.7  xnsynch_sleep_on()

```c
int 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_NCH_OWNER).

Parameters

<table>
<thead>
<tr>
<th>synch</th>
<th>The descriptor address of the synchronization object to sleep on.</th>
</tr>
</thead>
<tbody>
<tr>
<td>timeout</td>
<td>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.</td>
</tr>
<tr>
<td>timeout_mode</td>
<td>The mode of the timeout parameter. It can either be set to XN_RELATIVE, XN_ABSOLUTE, or XN_REALTIME (see also xntimer_start()).</td>
</tr>
</tbody>
</table>
Returns

A bitmask which may include zero or one information bit among XNRMIID, 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.

Tags

primary-only, might-switch

References xnthread_current().

6.42.2.8 xnsynch_try_acquire()

int xnsynch_try_acquire ( struct xnsynch * synch )

Try acquiring 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 another thread, the call returns with an error code.

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. |

Returns

Zero is returned if synch has been successfully acquired. Otherwise:

- EDEADLK is returned if synch is currently held by the calling thread.
- EBUSY is returned if synch is currently held by another thread.

Tags

primary-only

References xnthread_current().
6.42.2.9  xnsynch_wakeup_one_sleeper()

```c
struct xnthread * xnsynch_wakeup_one_sleeper ( 
    struct xnsynch * synch )
```

Unblock the heading thread from wait.

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.

Tags

- `unrestricted`

6.42.2.10  xnsynch_wakeup_this_sleeper()

```c
void xnsynch_wakeup_this_sleeper ( 
    struct xnsynch * synch, 
    struct xnthread * sleeper )
```

Unblock a particular thread from wait.

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.
- `sleeper`  The thread to unblock which MUST be currently linked to the synchronization object’s pending queue (i.e. `synch->pendq`).
Tags

unrestricted
6.43 Thread services

Collaboration diagram for Thread services:

```
Cobalt kernel

Thread services

| Thread state flags
| Thread information flags
```

Modules

- **Thread state flags**
  
  *Bits reporting permanent or transient states of threads.*

- **Thread information flags**
  
  *Bits reporting events notified to threads.*

Functions

- **static struct xnthread * xnthread_current (void)**
  
  *Retrieve the current Cobalt core TCB.*

- **static struct xnthread * xnthread_from_task (struct task_struct *p)**
  
  *Retrieve the Cobalt core TCB attached to a Linux task.*

- **static void xnthread_test_cancel (void)**
  
  *Introduce a thread cancellation point.*

- **int xnthread_init (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 xnthread_start (struct xnthread *thread, const struct xnthread_start_attr *attr)**
  
  *Start a newly created thread.*

- **int xnthread_set_mode (int clrmask, int setmask)**
  
  *Change control mode of the current thread.*

- **void xnthread_suspend (struct xnthread *thread, int mask, xnticks_t timeout, xntmode_t timeout_mode, struct xnsynch *wchan)**
  
  *Suspend a thread.*

- **void xnthread_resume (struct xnthread *thread, int mask)**
  
  *Resume a thread.*

- **int xnthread Unblock (struct xnthread *thread)**
  
  *Unblock a thread.*

- **int xnthread_set_periodic (struct xnthread *thread, xnticks_t idate, xntmode_t timeout_mode, xnticks_t _period)**
  
  *Make a thread periodic.*

- **int xnthread_wait_period (unsigned long *overruns_r)**
  
  *Wait for the next periodic release point.*
• int \texttt{xnthread\_set\_slice} (struct xnthread \*thread, xnticks\_t quantum)  
  \textit{Set thread time-slicing information.}
• void \texttt{xnthread\_cancel} (struct xnthread \*thread)  
  \textit{Cancel a thread.}
• int \texttt{xnthread\_join} (struct xnthread \*thread, bool uninterruptible)  
  \textit{Join with a terminated thread.}
• int \texttt{xnthread\_harden} (void)  
  \textit{Migrate a Linux task to the Xenomai domain.}
• void \texttt{xnthread\_relax} (int notify, int reason)  
  \textit{Switch a shadow thread back to the Linux domain.}
• int \texttt{xnthread\_map} (struct xnthread \*thread, struct completion \*done)  
  \textit{Create a shadow thread context over a kernel task.}
• int \texttt{xnthread\_migrate} (int cpu)  
  \textit{Migrate the current thread.}
• int \texttt{xnthread\_set\_schedparam} (struct xnthread \*thread, struct xnsched\_class \*sched\_class, const union xnsched\_policy\_param \*sched\_param)  
  \textit{Change the base scheduling parameters of a thread.}

6.43.1 Detailed Description

6.43.2 Function Documentation

6.43.2.1 \texttt{xnthread\_cancel()}

\begin{verbatim}
void xnthread_cancel (  
  struct xnthread \* thread  
)
\end{verbatim}

\textit{Cancel a thread.}

Request cancellation of a thread. This service forces \textit{thread} to exit from any blocking call, then to switch to secondary mode. \textit{thread} will terminate as soon as it reaches a cancellation point. Cancellation points are defined for the following situations:

• \textit{thread} self-cancels by a call to \texttt{xnthread\_cancel()}.  
  
• \textit{thread} invokes a Linux syscall (user-space shadow only).  
  
• \textit{thread} receives a Linux signal (user-space shadow only).  
  
• \textit{thread} unblocks from a Xenomai syscall (user-space shadow only).  
  
• \textit{thread} attempts to block on a Xenomai syscall (user-space shadow only).  
  
• \textit{thread} explicitly calls \texttt{xnthread\_test\_cancel()}.  

\textbf{Parameters}

\begin{tabular}{|l|l|}
\hline
\textit{thread} & The descriptor address of the thread to terminate. \\
\hline
\end{tabular}
6.43 Thread services

Tags

   task-unrestricted, might-switch

Note

In addition to the common actions taken upon cancellation, a thread which belongs to the SCHE→D_WEAK class is sent a regular SIGTERM signal.

6.43.2.2 xnthread_current()

struct xnthread * xnthread_current (  
    void ) [static]

Retrieve the current Cobalt core TCB.

Returns the address of the current Cobalt core thread descriptor, or NULL if running over a regular Linux task. This call is not affected by the current runtime mode of the core thread.

Note

The returned value may differ from xnsched_current_thread() called from the same context, since the latter returns the root thread descriptor for the current CPU if the caller is running in secondary mode.

Tags

   unrestricted

Referenced by xnsynch_acquire(), xnsynch_sleep_on(), xnsynch_try_acquire(), xnthread_harden(), xnthread_join(), xnthread_relax(), xnthread_set_periodic(), xnthread_test_cancel(), and xnthread←wait_period().

6.43.2.3 xnthread_from_task()

struct xnthread * xnthread_from_task (  
    struct task_struct * p ) [static]

Retrieve the Cobalt core TCB attached to a Linux task.

Returns the address of the Cobalt core thread descriptor attached to the Linux task p, or NULL if p is a regular Linux task. This call is not affected by the current runtime mode of the core thread.

Tags

   unrestricted
6.43.2.4 xnthread_harden()

```c
int xnthread_harden(
    void )
```

Migrate a Linux task to the Xenomai domain.

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().

Tags

- secondary-only, might-switch

References xnthread_current().

6.43.2.5 xnthread_init()

```c
int xnthread_init(
    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. The thread is left dormant until it is actually started by xnthread_start().

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>thread</td>
<td>The address of a thread descriptor Cobalt 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.</td>
</tr>
</tbody>
</table>

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 alignment constraints might not always be satisfied.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attr</td>
<td>A pointer to an attribute block describing the initial properties of the new thread. Members of this structure are defined as follows:</td>
</tr>
</tbody>
</table>

- 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 Cobalt 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:
  - XNSUSP creates the thread in a suspended state. In such a case, the thread shall be explicitly resumed using the xnthread_resume() service for its execution to actually begin, additionally to issuing xnthread_start() for it. This flag can also be specified when invoking xnthread_start() as a starting mode.
  - XNUSER shall be set if thread will be mapped over an existing user-space task. Otherwise, a new kernel host task is created, then paired with the new Xenomai thread.
  - XNFPU (enable FPU) tells Cobalt that the new thread may use the floating-point unit. XNFPU is implicitly assumed for user-space threads even if not set in \texttt{flags}.
  - **affinity**: The processor affinity of this thread. Passing CPU_MASK_ALL means “any cpu” from the allowed core affinity mask \texttt{(cobalt_cpu_affinity)}. Passing an empty set is invalid.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sched_class</td>
<td>The initial scheduling class the new thread should be assigned to.</td>
</tr>
<tr>
<td>sched_param</td>
<td>The initial scheduling parameters to set for the new thread; sched_param must be valid within the context of sched_class.</td>
</tr>
</tbody>
</table>

Returns

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

- -EINVAL is returned if attr->flags has invalid bits set, or attr->affinity is invalid (e.g. empty).

Tags

secondary-only

References XNFPU, XNSUSP, and XNUSER.

6.43.2.6 xnthread_join()

```c
int xnthread_join (  
    struct xnthread *thread,  
    bool uninterruptible )
```

Join with a terminated thread.

This service waits for thread to terminate after a call to xnthread_cancel(). If that thread has already terminated or is dormant at the time of the call, then xnthread_join() returns immediately.

xnthread_join() adapts to the calling context (primary or secondary), switching to secondary mode if needed for the duration of the wait. Upon return, the original runtime mode is restored, unless a Linux signal is pending.
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>thread</strong></td>
<td>The descriptor address of the thread to join with.</td>
</tr>
<tr>
<td><strong>uninterruptible</strong></td>
<td>Boolean telling whether the service should wait for completion uninterruptible.</td>
</tr>
</tbody>
</table>

### Returns

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

- -EDEADLK is returned if the current thread attempts to join itself.
- -EINTR is returned if the current thread was unblocked while waiting for `thread` to terminate.
- -EBUSY indicates that another thread is already waiting for `thread` to terminate.

### Tags

- task-unrestricted, might-switch

### References

xnthread_current().

Referenced by rtdm_task_join().

### 6.43.2.7 xnthread_map()

```c
int xnthread_map (        
    struct xnthread ∗ thread,  
    struct completion ∗ done)  
```

Create a shadow thread context over a kernel task.

This call maps a Cobalt core thread to the "current" Linux task running in kernel space. The priority and scheduling class of the underlying Linux task are not affected; it is assumed that the caller did set them appropriately before issuing the shadow mapping request.

This call immediately moves the calling kernel thread to the Xenomai domain.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>thread</strong></td>
<td>The descriptor address of the new shadow thread to be mapped to &quot;current&quot;. This descriptor must have been previously initialized by a call to xnthread_init().</td>
</tr>
<tr>
<td><strong>done</strong></td>
<td>A completion object to be signaled when <code>thread</code> is fully mapped over the current Linux context, waiting for xnthread_start().</td>
</tr>
</tbody>
</table>
6.43 Thread services

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

Tags

secondary-only, might-switch

6.43.2.8 `xnthread_migrate()`

```c
int xnthread_migrate (
    int cpu )
```

Migrate the current thread.

This call makes the current thread migrate to another (real-time) CPU if its affinity allows it. This call is available from primary mode only.

Parameters

| `cpu` | The destination CPU. |

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>if the thread could migrate;</td>
</tr>
<tr>
<td>-EPERM</td>
<td>if the calling context is invalid, or the scheduler is locked.</td>
</tr>
<tr>
<td>-EINVAL</td>
<td>if the current thread affinity forbids this migration.</td>
</tr>
</tbody>
</table>

Tags

primary-only, might-switch
6.43.2.9  xnthread_relax()

void xnthread_relax (  
    int notify,  
    int reason );

Switch a shadow thread back to the Linux domain.

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 xnthread_suspend() 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. |

Tags

primary-only, might-switch

Note

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

References splmax, and xnthread_current().

6.43.2.10  xnthread_resume()

void xnthread_resume (  
    struct xnthread * thread,  
    int mask );

Resume a thread.

Resumes the execution of a thread previously suspended by one or more calls to xnthread_suspend(). 
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 xnthread_resume(), the caller of xnthread_suspend() 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 xnthread_unblock().

Tags
unrestricted, might-switch

6.43.2.11 xnthread_set_mode()

int xnthread_set_mode (  
  int clrmask,  
  int setmask )

Change control mode of the current thread.

Change the control mode of the current thread. The control mode affects several behaviours of the Cobalt core regarding this thread.

Parameters

<table>
<thead>
<tr>
<th>clrmask</th>
<th>Clear the corresponding bits from the control mode 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.</th>
</tr>
</thead>
<tbody>
<tr>
<td>setmask</td>
<td>The new thread mode. The following flags may be set in this bitmask:</td>
</tr>
</tbody>
</table>

| • XNLOCK makes the current thread non-preemptible by other threads. Unless XNTRAPLB is also set for the thread, the latter may still block, dropping the lock temporarily, in which case, the lock will be reacquired automatically when the thread resumes execution. |
| • XNWARN enables debugging notifications for the current thread. A SIGDEBUG (Linux-originated) signal is sent when the following atypical or abnormal behavior is detected: |
the current thread switches to secondary mode. Such notification comes in handy for detecting spurious relaxes.

- CONFIG_XENO_OPT_DEBUG_MUTEX_RELAXED is enabled in the kernel configuration, and the current thread is sleeping on a Cobalt mutex currently owned by a thread running in secondary mode, which reveals a priority inversion.

- the current thread is about to sleep while holding a Cobalt mutex, and CONFIG_XENO_OPT_DEBUG_MUTEX_SLEEP is enabled in the kernel configuration. Blocking for acquiring a mutex does not trigger such a signal though.

- the current thread has both XNTRAPLB and XNLOCK set, and attempts to block on a Cobalt service, which would cause a lock break.

- XNTRAPLB disallows breaking the scheduler lock. In the default case, a thread which holds the scheduler lock is allowed to drop it temporarily for sleeping. If this mode bit is set, such thread would return immediately with XNBREAK set from xnthread_suspend(). If XNWARN is set for the current thread, SIGDEBUG is sent in addition to raising the break condition.

### Tags

- primary-only, might-switch

### Note

Setting `clrmask` and `setmask` to zero leads to a nop, in which case `xnthread_set_mode()` returns the current mode.

### 6.43.2.12 xnthread_set_periodic()

```c
int xnthread_set_periodic (  
    struct xnthread * 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 `xnthread_wait_period()` will delay the thread until the next periodic release point in the processor timeline is reached.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>thread</code></td>
<td>The core thread to make periodic. If NULL, the current thread is assumed.</td>
</tr>
<tr>
<td><code>idate</code></td>
<td>The initial (absolute) date of the first release point, expressed in nanoseconds. The affected thread will be delayed by the first call to <code>xnthread_wait_period()</code> until this point is reached. If <code>idate</code> is equal to XN_INFINITE, the first release point is set to <code>period</code> nanoseconds after the current date. In the latter case, <code>timeout_mode</code> is not considered and can have any valid value.</td>
</tr>
<tr>
<td><code>timeout_mode</code></td>
<td>The mode of the <code>idate</code> parameter. It can either be set to XN_ABSOLUTE or XN_REALTIME with <code>idate</code> different from XN_INFINITE (see also <code>xntimer_start()</code>).</td>
</tr>
<tr>
<td><code>period</code></td>
<td>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 successfully, regardless of the previous state of this timer.</td>
</tr>
</tbody>
</table>
6.43 Thread services

Returns

0 is returned upon success. Otherwise:

- -ETIMEDOUT is returned if \texttt{idate} is different from XN\_INFINITE and represents a date in the past.
- -EINVAL is returned if \texttt{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 \texttt{timeout\_mode} is not compatible with \texttt{idate}, such as XN\_RELATIVE with \texttt{idate} different from XN\_←\_INFINITE.
- -EPERM is returned if \texttt{thread} is NULL, but the caller is not a Xenomai thread.

Tags

\texttt{task-unrestricted}

References xnsched::cpu, and \texttt{xnthread\_current()}.

6.43.2.13 \texttt{xnthread\_set\_schedparam()}

\begin{verbatim}
int xnthread_set_schedparam ( 
    struct xnthread * thread, 
    struct xnsched_class * sched_class, 
    const union xnsched_policy_param * sched_param )
\end{verbatim}

Change the base scheduling parameters of a thread.

Changes the base scheduling policy and parameters of a thread. If the thread is currently blocked, waiting in priority-pending mode (XNSYNCH\_PRIO) for a synchronization object to be signaled, Cobalt 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

\begin{center}
\begin{tabular}{|l|l|}
\hline
\textbf{thread} & The descriptor address of the affected thread. See note. \\
\hline
\textbf{sched\_class} & The new scheduling class the thread should be assigned to. \\
\hline
\textbf{sched\_param} & The scheduling parameters to set for the thread; \texttt{sched\_param} must be valid within the context of \texttt{sched\_class}. \\
\hline
\end{tabular}
\end{center}

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 \texttt{sched\_class}. Invalid parameters passed into \texttt{sched\_param} are common causes of error.

Side effects

Generated by Doxygen
• 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.

Tags

task-unregistered

Note

The changes only apply to the Xenomai scheduling parameters for thread. There is no propagation/translation of such changes to the Linux scheduler for the task mated to the Xenomai target thread.

6.43.2.14 xnthread_set_slice()

```c
int xnthread_set_slice (  
    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 xnthread_init().

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>thread</code></td>
<td>The descriptor address of the affected thread.</td>
</tr>
<tr>
<td><code>quantum</code></td>
<td>The time quantum assigned to the thread expressed in nanoseconds. If <code>quantum</code> 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 <code>quantum</code> equals XN_INFINITE, time-slicing is stopped for that thread.</td>
</tr>
</tbody>
</table>

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,
• `quantum` is smaller than the master clock gravity for a user thread, which denotes a spurious value.

Tags

task-unrestricted
6.43.2.15 xnthread_start()

```c
int xnthread_start ( 
    struct xnthread *thread, 
    const struct xnthread_start_attr *attr )
```

Start 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 for the new thread.

**Parameters**

| **thread** | The descriptor address of the started thread which must have been previously initialized by a call to `xnthread_init()`.
| **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:
  - XNLOCK causes the thread to lock the scheduler when it starts. The target thread will have to call the `xnsched_unlock()` 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.
  - XNSUSP makes the thread start in a suspended state. In such a case, the thread will have to be explicitly resumed using the `xnthread_resume()` service for its execution to actually begin.

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

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

**Return values**

- `0` if `thread` could be started;
- `-EBUSY` if `thread` was not dormant or stopped;

**Tags**

- `task-unrestricted`, `might-switch`

6.43.2.16 xnthread_suspend()

```c
void xnthread_suspend ( 
    struct xnthread *thread, 
    int mask, 
    xnticks_t timeout, 
    xntmode_t timeout_mode, 
    struct xnsynch *wchan )
```

Generated by Doxygen
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 `xnthread_resume()`.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>thread</code></td>
<td>The descriptor address of the suspended thread.</td>
</tr>
<tr>
<td><code>mask</code></td>
<td>The suspension mask specifying the suspensive condition to add to the thread's wait mask. Possible values usable by the caller are:</td>
</tr>
</tbody>
</table>

- XNSUSP. This flag forcibly suspends a thread, regardless of any resource to wait for. A reverse call to `xnthread_resume()` 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 flag 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>timeout</code></td>
<td>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 <code>timeout_mode</code>.</td>
</tr>
</tbody>
</table>

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 `xnthread_suspend()` has completed, then the target thread will not be suspended, and this routine leads to a null effect.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>timeout_mode</code></td>
<td>The mode of the <code>timeout</code> parameter. It can either be set to XN_RELATIVE, XN_ABSOLUTE, or XN_REALTIME (see also <code>xntimer_start()</code>).</td>
</tr>
<tr>
<td><code>wchan</code></td>
<td>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).</td>
</tr>
</tbody>
</table>

Note

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

Tags

unrestricted, might-switch
6.43.2.17  xnthread_test_cancel()

void xnthread_test_cancel (  
    void  ) [inline], [static]

Introduce a thread cancellation point.

Terminates the current thread if a cancellation request is pending for it, i.e. if xnthread_cancel() was called.

Tags
  mode-unrestricted

References xnthread_current().

6.43.2.18  xnthread_unblock()

int xnthread_unblock (  
    struct xnthread * 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 xnthread_suspend() 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.

Tags
  unrestricted, might-switch
6.43.2.19  xnthread_wait_period()

```c
int xnthread_wait_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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>overruns_r</td>
<td>If non-NULL, <code>overruns_r</code> must be a pointer to a memory location which will be written with the count of pending overruns. This value is copied only when <code>xnthread_wait_period()</code> returns -ETIMEDOUT or success; the memory location remains unmodified otherwise. If NULL, this count will never be copied back.</td>
</tr>
</tbody>
</table>

Returns

- 0 is returned upon success; if `overruns_r` is valid, zero is copied to the pointed memory location.
- Otherwise:
  - -EWOULDBLOCK is returned if `xnthread_set_periodic()` has not previously been called for the calling thread.
  - -EINTR is returned if `xnthread_unblock()` 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.

Tags

- primary-only, might-switch

References `xnthread_current()`.
6.44 Timer services

The Xenomai timer facility depends on a clock source (xnclock) for scheduling the next activation times.

Collaboration diagram for Timer services:

Functions

- void xntimer_destroy (struct xntimer *timer)
  
  Release a timer object.

- static xnticks_t xntimer_interval (struct xntimer *timer)
  
  Return the timer interval value.

- int xntimer_start (struct xntimer *timer, xnticks_t value, xnticks_t interval, xntmode_t mode)
  
  Arm a timer.

- xnticks_t xntimer_get_date (struct xntimer *timer)
  
  Return the absolute expiration date.

- static void xntimer_stop (struct xntimer *timer)
  
  Disarm a timer.

- static xnticks_t xntimer_get_timeout (struct xntimer *timer)
  
  Return the relative expiration date.

- void xntimer_init (struct xntimer *timer, struct xnclock *clock, void(*handler)(struct xntimer *timer), struct xnsched *sched, int flags)
  
  Initialize a timer object.

- void __xntimer_migrate (struct xntimer *timer, struct xnsched *sched)
  
  Migrate a timer.

- unsigned long long xntimer_get_overruns (struct xntimer *timer, xnticks_t now)
  
  Get the count of overruns for the last tick.

- static int program_htick_shot (unsigned long delay, struct clock_event_device *cdev)
  
  Program next host tick as a Xenomai timer event.

- static void switch_htick_mode (enum clock_event_mode mode, struct clock_event_device *cdev)
  
  Tick mode switch emulation callback.

- int xntimer_grab_hardware (void)
  
  Grab the hardware timer on all real-time CPUs.

- void xntimer_release_hardware (void)
  
  Release hardware timers.

6.44.1 Detailed Description

The Xenomai timer facility depends on a clock source (xnclock) for scheduling the next activation times.

The core provides and depends on a monotonic clock source (nkclock) with nanosecond resolution, driving the platform timer hardware exposed by the interrupt pipeline.
6.44.2 Function Documentation

6.44.2.1 __xntimer_migrate()

```c
void __xntimer_migrate ( 
    struct xntimer * timer, 
    struct xnsched * sched )
```

Migrate a timer.

This call migrates a timer to another cpu. In order to avoid pathological cases, it must be called from the CPU to which `timer` is currently attached.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>timer</code></td>
<td>The address of the timer object to be migrated.</td>
</tr>
<tr>
<td><code>sched</code></td>
<td>The address of the destination per-CPU scheduler slot.</td>
</tr>
</tbody>
</table>

Tags

- unrestricted, atomic-entry

6.44.2.2 program_htick_shot()

```c
static int program_htick_shot ( 
    unsigned long delay, 
    struct clock_event_device * cdev ) [static]
```

Program next host tick as a Xenomai timer event.

Program the next shot for the host tick on the current CPU. Emulation is done using a nucleus timer attached to the master timebase.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>delay</code></td>
<td>The time delta from the current date to the next tick, expressed as a count of nanoseconds.</td>
</tr>
<tr>
<td><code>cdev</code></td>
<td>An pointer to the clock device which notifies us.</td>
</tr>
</tbody>
</table>

Tags

- unrestricted
6.44.2.3 switch_htick_mode()

void switch_htick_mode (  
    enum clock_event_mode mode,  
    struct clock_event_device *cdev ) [static]

Tick mode switch emulation callback.

Changes the host tick mode for the tick device of the current CPU.

Parameters

<table>
<thead>
<tr>
<th>mode</th>
<th>The new mode to switch to. The possible values are:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• CLOCK_EVT_MODE_ONESHOT, for a switch to oneshot mode.</td>
</tr>
<tr>
<td></td>
<td>• CLOCK_EVT_MODE_PERIODIC, for a switch to periodic mode. The current implementation for</td>
</tr>
<tr>
<td></td>
<td>the generic clockevent layer Linux exhibits should never downgrade from a oneshot to a periodic</td>
</tr>
<tr>
<td></td>
<td>tick mode, so this mode should not be encountered. This said, the associated code is provided,</td>
</tr>
<tr>
<td></td>
<td>basically for illustration purposes.</td>
</tr>
<tr>
<td></td>
<td>• CLOCK_EVT_MODE_SHUTDOWN, indicates the removal of the current tick device. Normally, the</td>
</tr>
<tr>
<td></td>
<td>nucleus only interposes on tick devices which should never be shut down, so this mode should not</td>
</tr>
<tr>
<td></td>
<td>be encountered.</td>
</tr>
</tbody>
</table>

Parameters

| cdev | An opaque pointer to the clock device which notifies us. |

Tags

unrestricted

Note

GENERIC_CLOCKEVENTS is required from the host kernel.

6.44.2.4 xntimer_destroy()

void xntimer_destroy (  
    struct xntimer *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. |

Tags

unrestricted

6.44.2.5  xntimer_get_date()

xnticks_t xntimer_get_date (struct xntimer * 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_INFINITY is returned if timer is currently disabled.

Tags

unrestricted, atomic-entry

6.44.2.6  xntimer_get_overruns()

unsigned long long xntimer_get_overruns ( struct xntimer * 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 (as xnclock_read_raw(xntimer_clock(timer))) |
Returns
the number of overruns of timer at date now

Tags
unrestricted, atomic-entry

6.44.2.7 xntimer_get_timeout()

xnticks_t xntimer_get_timeout ( 
   struct xntimer * timer ) [inline], [static]

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.

Tags
unrestricted, atomic-entry

6.44.2.8 xntimer_grab_hardware()

int xntimer_grab_hardware ( 
   void )

Grab the hardware timer on all real-time CPUs.

xntimer_grab_hardware() grabs and tunes the hardware timer for all real-time CPUs.

Host tick emulation is performed for sharing the clock chip between Linux and Xenomai.

Generated by Doxygen
Returns

a positive value is returned on success, representing the duration of a Linux periodic tick expressed
as a count of nanoseconds; zero should be returned when the Linux kernel does not undergo
periodic timing on the given CPU (e.g. oneshot mode). Otherwise:

- -EBUSY is returned if the hardware timer has already been grabbed. xntimer_release_hardware()
must be issued before xntimer_grab_hardware() is called again.
- -ENODEV is returned if the hardware timer cannot be used. This situation may occur after the ker-
nel disabled the timer due to invalid calibration results; in such a case, such hardware is unusable
for any timing duties.

Tags

secondary-only

References xnsched::cpu, and xnintr_init().

6.44.2.9  xntimer_init()

```c
void xntimer_init (  
    struct xntimer *timer,  
    struct xnclock *clock,  
    void(*)(struct xntimer *timer) handler,  
    struct xnsched *sched,  
    int flags  )
```

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timer</td>
<td>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.</td>
</tr>
<tr>
<td>clock</td>
<td>The clock the timer relates to. Xenomai defines a monotonic system clock, with nanosecond resolution, named nkclock. In addition, external clocks driven by other tick sources may be created dynamically if CONFIG_XENO_OPT_EXTCLOCK is defined.</td>
</tr>
<tr>
<td>handler</td>
<td>The routine to call upon expiration of the timer.</td>
</tr>
<tr>
<td>sched</td>
<td>An optional pointer to the per-CPU scheduler slot the new timer is affine to. If non-NULL, the timer will fire on the CPU sched is bound to, otherwise it will fire either on the current CPU if real-time, or on the first real-time CPU.</td>
</tr>
<tr>
<td>flags</td>
<td>A set of flags describing the timer. The valid flags are:</td>
</tr>
</tbody>
</table>

- XNTIMER_NOBLCK, the timer won’t be frozen while GDB takes over control of the application.

A set of clock gravity hints can be passed via the `flags` argument, used for optimizing the built-in heuris-
tics aimed at latency reduction:
• XNTIMER_IGRAVITY, the timer activates a leaf timer handler.
• XNTIMER_KGRAVITY, the timer activates a kernel thread.
• XNTIMER_UGRAVITY, the timer activates a user-space thread.

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

Tags

unrestricted

6.44.2.10  xntimer_interval()

xnticks_t xntimer_interval ( 
    struct xntimer * timer ) [inline], [static]

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.

Tags

unrestricted, atomic-entry

6.44.2.11  xntimer_release_hardware()

void xntimer_release_hardware ( 
    void )

Release hardware timers.

Releases hardware timers previously grabbed by a call to xntimer_grab_hardware().

Tags

secondary-only

References xnsched::cpu.
6.44.2.12  xntimer_start()

int xntimer_start (  
    struct xntimer * 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.

A timer is attached to the clock specified in xntimer_init().

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timer</td>
<td>The address of a valid timer descriptor.</td>
</tr>
<tr>
<td>value</td>
<td>The date of the initial timer shot, expressed in nanoseconds.</td>
</tr>
<tr>
<td>interval</td>
<td>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 \textit{interval} is equal to XN_INFINITE, the timer will not be reloaded after it has expired.</td>
</tr>
<tr>
<td>mode</td>
<td>The timer mode. It can be XN_RELATIVE if \textit{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 by xnclock_read_monotonic()), or XN_REALTIME if the absolute date is based on the adjustable real-time date for the relevant clock (obtained from xnclock_read_realtime()).</td>
</tr>
</tbody>
</table>

Returns

0 is returned upon success, or -ETIMEDOUT if an absolute date in the past has been given. In such an event, the timer is nevertheless armed for the next shot in the timeline if \textit{interval} is different from XN_INFINITE.

Tags

unrestricted, atomic-entry

6.44.2.13  xntimer_stop()

int xntimer_stop (  
    struct xntimer * 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. |

Tags

unrestricted, atomic-entry
6.45 Virtual file services

Virtual files provide a mean to export Xenomai object states to user-space, based on common kernel interfaces.

Collaboration diagram for 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.

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 cobalt_vfroot
  
  Xenomai vfile root directory.

- struct xnvfile_directory cobalt_vfroot
  
  Xenomai vfile root directory.

6.45.1 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.

6.45.2 Function Documentation

6.45.2.1 xnvfile_destroy()

void xnvfile_destroy ( 
                   struct xnvfile *vfile )

Removes a virtual file entry.
Parameters

| vfile | A pointer to the virtual file descriptor to remove. |

Tags

secondary-only

6.45.2.2 xnvfile_get_blob()

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

<table>
<thead>
<tr>
<th>input</th>
<th>A pointer to the input descriptor passed to the store() handler.</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>The address of the destination buffer to copy the input data to.</td>
</tr>
<tr>
<td>size</td>
<td>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.</td>
</tr>
</tbody>
</table>

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.

Tags

secondary-only

Referenced by xnvfile_get_integer(), and xnvfile_get_string().
6.45 Virtual file services

6.45.2.3 `xnvfile_get_integer()`

```c
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**

<table>
<thead>
<tr>
<th>input</th>
<th>A pointer to the input descriptor passed to the store() handler.</th>
</tr>
</thead>
<tbody>
<tr>
<td>valp</td>
<td>The address of a long integer variable to receive the value.</td>
</tr>
</tbody>
</table>

**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.

**Tags**

`secondary-only`

**References** `xnvfile_get_blob()`.

6.45.2.4 `xnvfile_get_string()`

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>input</code></td>
<td>A pointer to the input descriptor passed to the store() handler.</td>
</tr>
<tr>
<td><code>s</code></td>
<td>The address of the destination string buffer to copy the input data to.</td>
</tr>
<tr>
<td><code>maxlen</code></td>
<td>The maximum number of bytes to copy to the destination buffer, including the ending null character. If <code>maxlen</code> is larger than the actual string length, the input is truncated to <code>maxlen</code>.</td>
</tr>
</tbody>
</table>

Returns

The number of characters read upon success. Otherwise, a negative error code is returned:

- `-EFAULT` indicates an invalid source buffer address.

Tags

`secondary-only`

References `xnvfile_get_blob()`.

6.45.2.5  `xnvfile_init_dir()`

```c
int xnvfile_init_dir (  
    const char ∗ name,  
    struct xnvfile_directory ∗ vdir,  
    struct xnvfile_directory ∗ parent )
```

Initialize a virtual directory entry.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>name</code></td>
<td>The name which should appear in the pseudo filesystem, identifying the vdir entry.</td>
</tr>
<tr>
<td><code>vdir</code></td>
<td>A pointer to the virtual directory descriptor to initialize.</td>
</tr>
<tr>
<td><code>parent</code></td>
<td>A pointer to a virtual directory descriptor standing for the parent directory of the new vdir. If NULL, the <code>/proc</code> root directory will be used. <code>/proc/xenomai</code> is mapped on the globally available <code>cobalt vfroot</code> vdir.</td>
</tr>
</tbody>
</table>

Returns

0 is returned on success. Otherwise:

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

Tags

`secondary-only`
6.45.2.6 \texttt{xnvfile\_init\_link()}

\begin{verbatim}
int xnvfile_init_link (  
    const char * from,  
    const char * to,  
    struct xnvfile_link * vlink,  
    struct xnvfile_directory * parent)
\end{verbatim}

Initialize a virtual link entry.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{from}</td>
<td>The name which should appear in the pseudo-filesystem, identifying the vlink entry.</td>
</tr>
<tr>
<td>\texttt{to}</td>
<td>The target file name which should be referred to symbolically by \texttt{name}.</td>
</tr>
<tr>
<td>\texttt{vlink}</td>
<td>A pointer to the virtual link descriptor to initialize.</td>
</tr>
<tr>
<td>\texttt{parent}</td>
<td>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 \texttt{cobalt_vfroot} vdir.</td>
</tr>
</tbody>
</table>

**Returns**

0 is returned on success. Otherwise:

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

**Tags**

\texttt{secondary-only}

6.45.2.7 \texttt{xnvfile\_init\_regular()}

\begin{verbatim}
int xnvfile_init_regular (  
    const char * name,  
    struct xnvfile_regular * vfile,  
    struct xnvfile_directory * parent)
\end{verbatim}

Initialize a regular vfile.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{name}</td>
<td>The name which should appear in the pseudo-filesystem, identifying the vfile entry.</td>
</tr>
<tr>
<td>\texttt{vfile}</td>
<td>A pointer to a vfile descriptor to initialize from. The following fields in this structure should be filled in prior to call this routine:</td>
</tr>
</tbody>
</table>

- \texttt{.privsz} is the size (in bytes) of the private data area to be reserved in the \texttt{vfile iterator}. A NULL value indicates that no private area should be reserved.
entry.lockops is a pointer to a lockingdescriptor*, 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parent</td>
<td>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 cobalt_vfroot vdir.</td>
</tr>
</tbody>
</table>

Returns

0 is returned on success. Otherwise:

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

Tags

secondary-only

6.45.2.8 xnvfile_init_snapshot()

Initialize a snapshot-driven vfile.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name which should appear in the pseudo-filesystem, identifying the vfile entry.</td>
</tr>
<tr>
<td>vfile</td>
<td>A pointer to a vfile descriptor to initialize from. The following fields in this structure should be filled in prior to call this routine:</td>
</tr>
</tbody>
</table>

- .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 lock 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 cobalt_vfroot vdir. |

Returns

0 is returned on success. Otherwise:

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

Tags

secondary-only

6.45.3 Variable Documentation

6.45.3.1 cobalt_vfroot [1/2]

struct xnvfile_directory cobalt_vfroot

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.

6.45.3.2 cobalt_vfroot [2/2]

struct xnvfile_directory cobalt_vfroot

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.
6.46 Analogy framework

A RTDM-based interface for implementing DAQ card drivers.

Collaboration diagram for Analogy framework:

![Collaboration diagram for Analogy framework]

Modules

- **Driver API**
  
  *Programming interface provided to DAQ card drivers.*

- **Analogy user API**

6.46.1 Detailed Description

A RTDM-based interface for implementing DAQ card drivers.
6.47 Driver API

Programming interface provided to DAQ card drivers.

Collaboration diagram for Driver API:

Modules

- Channels and ranges
  - Channels.
- Driver management services
  - Analogy driver registration / unregistration.
- Subdevice management services
  - Subdevice declaration in a driver.
- Buffer management services
  - Buffer management services.
- Interrupt management services
- Misc services

6.47.1 Detailed Description

Programming interface provided to DAQ card drivers.
6.48 Driver management services

Analogy driver registration / unregistration.

Collaboration diagram for Driver management services:

```
Driver API ----> Driver management services
```

Functions

- int `a4l_register_drv (struct a4l_driver *drv)`
  Register an Analogy driver.

- int `a4l_unregister_drv (struct a4l_driver *drv)`
  Unregister an Analogy driver.

6.48.1 Detailed Description

Analogy driver registration / unregistration.

In a common Linux char driver, the developer has to register a fops structure filled with callbacks for read / write / mmap / ioctl operations.

Analogy drivers do not have to implement read / write / mmap / ioctl functions, these procedures are implemented in the Analogy generic layer. Then, the transfers between user-space and kernel-space are already managed. Analogy drivers work with commands and instructions which are some kind of more dedicated read / write operations. And, instead of registering a fops structure, a Analogy driver must register some `a4l_driver` structure.

6.48.2 Function Documentation

6.48.2.1 a4l_register_drv()

```
int a4l_register_drv (struct a4l_driver *drv)
```

Register an Analogy driver.

After initialising a driver structure, the driver must be made available so as to be attached.
### 6.48 Driver management services

Parameters

| in | drv | Driver descriptor structure |

Returns

0 on success, otherwise negative error code.

References a4l_driver::board_name.

#### 6.48.2.2 a4l_unregister_drv()

```c
int a4l_unregister_drv (struct a4l_driver * drv )
```

Unregister an Analogy driver.

This function removes the driver descriptor from the Analogy driver list. The driver cannot be attached anymore.

Parameters

| in | drv | Driver descriptor structure |

Returns

0 on success, otherwise negative error code.

References a4l_driver::board_name.
6.49 Subdevice management services

Subdevice declaration in a driver.

Collaboration diagram for Subdevice management services:

![Collaboration diagram](image)

Functions

- `struct a4l_subdevice * a4l_alloc_subd (int sizeof_priv, void (*)(struct a4l_subdevice *))`
  Allocate a subdevice descriptor.
- `int a4l_add_subd (struct a4l_device *dev, struct a4l_subdevice *subd)`
  Add a subdevice to the driver descriptor.
- `struct a4l_subdevice * a4l_get_subd (struct a4l_device *dev, int idx)`
  Get a pointer to the subdevice descriptor referenced by its registration index.

Subdevices types

Flags to define the subdevice type

- `#define A4L_SUBD_UNUSED (A4L_SUBD_MASK_SPECIAL | 0x1)`
  Unused subdevice.
- `#define A4L_SUBD_AI (A4L_SUBD_MASK_READ | 0x2)`
  Analog input subdevice.
- `#define A4L_SUBD_AO (A4L_SUBD_MASK_WRITE | 0x4)`
  Analog output subdevice.
- `#define A4L_SUBD_DI (A4L_SUBD_MASK_READ | 0x8)`
  Digital input subdevice.
- `#define A4L_SUBD_DO (A4L_SUBD_MASK_WRITE | 0x10)`
  Digital output subdevice.
- `#define A4L_SUBD_DIO (A4L_SUBD_MASK_SPECIAL | 0x20)`
  Digital input/output subdevice.
- `#define A4L_SUBD_COUNTER (A4L_SUBD_MASK_SPECIAL | 0x40)`
  Counter subdevice.
- `#define A4L_SUBD_TIMER (A4L_SUBD_MASK_SPECIAL | 0x80)`
  Timer subdevice.
- `#define A4L_SUBD_MEMORY (A4L_SUBD_MASK_SPECIAL | 0x100)`
  Memory, EEPROM, DPRAM.
- `#define A4L_SUBD_CALIB (A4L_SUBD_MASK_SPECIAL | 0x200)`
  Calibration subdevice.
6.49 Subdevice management services

Calibration subdevice DACs.

- \#define A4L_SUBD_PROC (A4L_SUBD_MASK_SPECIAL|0x400)
  Processor, DSP.
- \#define A4L_SUBD_SERIAL (A4L_SUBD_MASK_SPECIAL|0x800)
  Serial IO subdevice.
- \#define A4L_SUBD_TYPES
  Mask which gathers all the types.

Subdevice features

Flags to define the subdevice's capabilities

- \#define A4L_SUBD_CMD 0x1000
  The subdevice can handle command (i.e it can perform asynchronous acquisition)
- \#define A4L_SUBD_MMAP 0x8000
  The subdevice support mmap operations (technically, any driver can do it; however, the developer might want that his driver must be accessed through read / write.

Subdevice status

Flags to define the subdevice's status

- \#define A4L_SUBD_BUSY_NR 0
  The subdevice is busy, a synchronous or an asynchronous acquisition is occurring.
- \#define A4L_SUBD_BUSY \(1 \ll A4L_SUBD_BUSY_NR\)
  The subdevice is busy, a synchronous or an asynchronous acquisition is occurring.
- \#define A4L_SUBD_CLEAN_NR 1
  The subdevice is about to be cleaned in the middle of the detach procedure.
- \#define A4L_SUBD_CLEAN \(1 \ll A4L_SUBD_CLEAN_NR\)
  The subdevice is busy, a synchronous or an asynchronous acquisition is occurring.

6.49.1 Detailed Description

Subdevice declaration in a driver.

The subdevice structure is the most complex one in the Analogy driver layer. It contains some description fields to fill and some callbacks to declare.

The description fields are:

- flags: to define the subdevice type and its capabilities;
- chan_desc: to describe the channels which compose the subdevice;
- rng_desc: to declare the usable ranges;

The functions callbacks are:

Generated by Doxygen
• do_cmd() and do_cmdtest(): to performe asynchronous acquisitions thanks to commands;
• cancel(): to abort a working asynchronous acquisition;
• munge(): to apply modifications on the data freshly acquired during an asynchronous transfer. Warning: using this feature with can significantly reduce the performances (if the munge operation is complex, it will trigger high CPU charge and if the acquisition device is DMA capable, many cache-misses and cache-replaces will occur (the benefits of the DMA controller will vanish);
• trigger(): optionnaly to launch an asynchronous acquisition;
• insn_read(), insn_write(), insn_bits(), insn_config(): to perform synchronous acquisition operations.

Once the subdevice is filled, it must be inserted into the driver structure thanks to a4l_add_subd().

6.49.2 Function Documentation

6.49.2.1 a4l_add_subd()

```c
int a4l_add_subd (  
    struct a4l_device * dev,  
    struct a4l_subdevice * subd)
```

Add a subdevice to the driver descriptor.

Once the driver descriptor structure is initialized, the function a4l_add_subd() must be used so to add some subdevices to the driver.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>dev</th>
<th>Device descriptor structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>subd</td>
<td>Subdevice descriptor structure</td>
</tr>
</tbody>
</table>

Returns the index with which the subdevice has been registered, in case of error a negative error code is returned.

References a4l_subdevice::dev, a4l_subdevice::idx, and a4l_subdevice::list.

6.49.2.2 a4l_alloc_subd()

```c
struct a4l_subdevice* a4l_alloc_subd (  
    int sizeof_priv,  
    void(*)(struct a4l_subdevice *) setup )
```

Allocate a subdevice descriptor.

This is a helper function so as to get a suitable subdevice descriptor
### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>sizeof_priv</th>
<th>Size of the subdevice's private data</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>setup</td>
<td>Setup function to be called after the allocation</td>
</tr>
</tbody>
</table>

### Returns

- the index with which the subdevice has been registered, in case of error a negative error code is returned.

### References rtdm_malloc().

#### 6.49.2.3 a4l_get_subd()

```c
struct a4l_subdevice* a4l_get_subd (  
    struct a4l_device *dev,  
    int idx 
)
```

Get a pointer to the subdevice descriptor referenced by its registration index.

This function is scarcely useful as all the drivers callbacks get the related subdevice descriptor as first argument. This function is not optimized, it goes through a linked list to get the proper pointer. So it must not be used in real-time context but at initialization / cleanup time (attach / detach).

### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>dev</th>
<th>Device descriptor structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>idx</td>
<td>Subdevice index</td>
</tr>
</tbody>
</table>

### Returns

- 0 on success, otherwise negative error code.
6.50 Buffer management services

Buffer management services.

Collaboration diagram for Buffer management services:

![Driver API to Buffer management services diagram]

Functions

- `int a4l_buf_prepare_absput (struct a4l_subdevice *subd, unsigned long count)`
  
  Update the absolute count of data sent from the device to the buffer since the start of the acquisition and after the next DMA shot.

- `int a4l_buf_commit_absput (struct a4l_subdevice *subd, unsigned long count)`
  
  Set the absolute count of data which was sent from the device to the buffer since the start of the acquisition and until the last DMA shot.

- `int a4l_buf_prepare_put (struct a4l_subdevice *subd, unsigned long count)`
  
  Set the count of data which is to be sent to the buffer at the next DMA shot.

- `int a4l_buf_commit_put (struct a4l_subdevice *subd, unsigned long count)`
  
  Set the count of data sent to the buffer during the last completed DMA shots.

- `int a4l_buf_put (struct a4l_subdevice *subd, void *bufdata, unsigned long count)`
  
  Copy some data from the device driver to the buffer.

- `int a4l_buf_prepare_absget (struct a4l_subdevice *subd, unsigned long count)`
  
  Update the absolute count of data sent from the buffer to the device since the start of the acquisition and after the next DMA shot.

- `int a4l_buf_commit_absget (struct a4l_subdevice *subd, unsigned long count)`
  
  Set the absolute count of data which was sent from the buffer to the device since the start of the acquisition and until the last DMA shot.

- `int a4l_buf_prepare_get (struct a4l_subdevice *subd, unsigned long count)`
  
  Set the count of data which is to be sent from the buffer to the device at the next DMA shot.

- `int a4l_buf_commit_get (struct a4l_subdevice *subd, unsigned long count)`
  
  Set the count of data sent from the buffer to the device during the last completed DMA shots.

- `int a4l_buf_get (struct a4l_subdevice *subd, void *bufdata, unsigned long count)`
  
  Copy some data from the buffer to the device driver.

- `int a4l_buf_evt (struct a4l_subdevice *subd, unsigned long evts)`
  
  Signal some event(s) to a user-space program involved in some read / write operation.

- `unsigned long a4l_buf_count (struct a4l_subdevice *subd)`
  
  Get the data amount available in the Analogy buffer.

- `struct a4l_cmd_desc *a4l_get_cmd (struct a4l_subdevice *subd)`
  
  Get the current Analogy command descriptor.

- `int a4l_get_chan (struct a4l_subdevice *subd)`
  
  Get the channel index according to its type.
6.50 Buffer management services

6.50.1 Detailed Description

Buffer management services.
The buffer is the key component of the Analogy infrastructure. It manages transfers between the user-space and the Analogy drivers thanks to generic functions which are described hereafter. Thanks to the buffer subsystem, the driver developer does not have to care about the way the user program retrieves or sends data.

To write a classical char driver, the developer has to fill a fops structure so as to provide transfer operations to the user program (read, write, ioctl and mmap if need be).

The Analogy infrastructure manages the whole interface with the userspace; the common read, write, mmap, etc. callbacks are generic Analogy functions. These functions manage (and perform, if need be) transfers between the user-space and an asynchronous buffer thanks to lockless mechanisms.

Consequently, the developer has to use the proper buffer functions in order to write / read acquired data into / from the asynchronous buffer.

Here are listed the functions:
- \texttt{a4l\_buf\_prepare\_\_abs\_put()} and \texttt{a4l\_buf\_commit\_\_abs\_put()}
- \texttt{a4l\_buf\_prepare\_\_abs\_get()} and \texttt{a4l\_buf\_commit\_\_abs\_get()}
- \texttt{a4l\_buf\_put()}
- \texttt{a4l\_buf\_get()}
- \texttt{a4l\_buf\_evt()}

The functions count might seem high; however, the developer needs a few of them to write a driver. Having so many functions enables to manage any transfer cases:
- If some DMA controller is available, there is no need to make the driver copy the acquired data into the asynchronous buffer, the DMA controller must directly trigger DMA shots into / from the buffer. In that case, a function \texttt{a4l\_buf\_prepare\_\_abs\_put()} must be used so as to set up the DMA transfer and a function \texttt{a4l\_buf\_commit\_\_abs\_put()} has to be called to complete the transfer().
- For DMA controllers which need to work with global counter (the transferred data count since the beginning of the acquisition), the functions \texttt{a4l\_buf\_\_abs\_\_put()} have been made available.
- If no DMA controller is available, the driver has to perform the copy between the hardware component and the asynchronous buffer. In such cases, the functions \texttt{a4l\_buf\_get()} and \texttt{a4l\_buf\_put()} are useful.

6.50.2 Function Documentation

6.50.2.1 \texttt{a4l\_buf\_commit\_abs\_get()}

\begin{verbatim}
int a4l_buf_commit_absget (  
    struct a4l_subdevice * subd,  
    unsigned long count )
\end{verbatim}

Set the absolute count of data which was sent from the buffer to the device since the start of the acquisition and until the last DMA shot.

The functions \texttt{a4l\_buf\_prepare\_\_abs\_put()}, \texttt{a4l\_buf\_commit\_\_abs\_put()}, \texttt{a4l\_buf\_prepare\_\_abs\_get()} and \texttt{a4l\_buf\_commit\_\_abs\_get()} have been made available for DMA transfers. In such situations, no data copy is needed between the Analogy buffer and the device as some DMA controller is in charge of performing data shots from / to the Analogy buffer. However, some pointers still have to be updated so as to monitor the transfers.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>subd</th>
<th>Subdevice descriptor structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>count</td>
<td>The data count transferred to the device during the last DMA shot plus the data count which have been sent since the beginning of the acquisition</td>
</tr>
</tbody>
</table>

Returns

0 on success, otherwise negative error code.

References A4L_SUBD_BUSY_NR, a4l_subdevice::buf, and a4l_subdevice::status.

6.50.2.2 a4l_buf_commit_absput()

```c
int a4l_buf_commit_absput ( 
    struct a4l_subdevice * subd, 
    unsigned long count )
```

Set the absolute count of data which was sent from the device to the buffer since the start of the acquisition and until the last DMA shot.

The functions a4l_buf_prepare_(abs)put(), a4l_buf_commit_(abs)put(), a4l_buf_prepare_(abs)get() and a4l_buf_commit_(abs)get() have been made available for DMA transfers. In such situations, no data copy is needed between the Analogy buffer and the device as some DMA controller is in charge of performing data shots from / to the Analogy buffer. However, some pointers still have to be updated so as to monitor the transfers.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>subd</th>
<th>Subdevice descriptor structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>count</td>
<td>The data count transferred to the buffer during the last DMA shot plus the data count which have been sent / retrieved since the beginning of the acquisition</td>
</tr>
</tbody>
</table>

Returns

0 on success, otherwise negative error code.

References A4L_SUBD_BUSY_NR, a4l_subdevice::buf, and a4l_subdevice::status.

6.50.2.3 a4l_buf_commit_get()

```c
int a4l_buf_commit_get ( 
    struct a4l_subdevice * subd, 
    unsigned long count )
```
Set the count of data sent from the buffer to the device during the last completed DMA shots.

The functions `a4l_buf_prepare_(abs)put()`, `a4l_buf_commit_(abs)put()`, `a4l_buf_prepare_(abs)get()` and `a4l_buf_commit_(abs)get()` have been made available for DMA transfers. In such situations, no data copy is needed between the Analogy buffer and the device as some DMA controller is in charge of performing data shots from / to the Analogy buffer. However, some pointers still have to be updated so as to monitor the transfers.

Parameters

| in | `subd` | Subdevice descriptor structure |
| in | `count` | The amount of data transferred |

Returns

0 on success, otherwise negative error code.

References `A4L_SUBD_BUSY_NR`, `a4l_subdevice::buf`, and `a4l_subdevice::status`.

---

6.50.2.4 `a4l_buf_commit_put()`

```c
int a4l_buf_commit_put (  
    struct a4l_subdevice * subd,  
    unsigned long count  
)
```

Set the count of data sent to the buffer during the last completed DMA shots.

The functions `a4l_buf_prepare_(abs)put()`, `a4l_buf_commit_(abs)put()`, `a4l_buf_prepare_(abs)get()` and `a4l_buf_commit_(abs)get()` have been made available for DMA transfers. In such situations, no data copy is needed between the Analogy buffer and the device as some DMA controller is in charge of performing data shots from / to the Analogy buffer. However, some pointers still have to be updated so as to monitor the transfers.

Parameters

| in | `subd` | Subdevice descriptor structure |
| in | `count` | The amount of data transferred |

Returns

0 on success, otherwise negative error code.

References `A4L_SUBD_BUSY_NR`, `a4l_subdevice::buf`, and `a4l_subdevice::status`. 

---

Generated by Doxygen
6.50.2.5  a4l_buf_count()

unsigned long a4l_buf_count (  
    struct a4l_subdevice * subd )

Get the data amount available in the Analogy buffer.
6.50 Buffer management services

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>subd</th>
<th>Subdevice descriptor structure</th>
</tr>
</thead>
</table>

Returns

the amount of data available in the Analogy buffer.

References A4L_SUBD_BUSY_NR, a4l_subdevice::buf, and a4l_subdevice::status.

6.50.2.6 a4l_buf_evt()

int a4l_buf_evt (
    struct a4l_subdevice * subd,
    unsigned long evts
)

Signal some event(s) to a user-space program involved in some read / write operation.

The function a4l_buf_evt() is useful in many cases:

- To wake-up a process waiting for some data to read.
- To wake-up a process waiting for some data to write.
- To notify the user-process an error has occurred during the acquisition.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>subd</th>
<th>Subdevice descriptor structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>evts</td>
<td>Some specific event to notify:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• A4L_BUF_ERROR to indicate some error has occurred during the transfer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• A4L_BUF_EOA to indicate the acquisition is complete (this event is automatically set, it should not be used).</td>
</tr>
</tbody>
</table>

Returns

0 on success, otherwise negative error code.

References A4L_SUBD_BUSY_NR, a4l_subdevice::buf, and a4l_subdevice::status.

6.50.2.7 a4l_buf_get()

int a4l_buf_get (
    struct a4l_subdevice * subd,

Generated by Doxygen
void *bufdata,
unsigned long count)

Copy some data from the buffer to the device driver.

The function `a4l_buf_get()` must copy data coming from the Analogy buffer to some acquisition device. This ring-buffer is an intermediate area between the device driver and the user-space program, which is supposed to provide the data to send to the device.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>subd</th>
<th>Subdevice descriptor structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>bufdata</td>
<td>The data buffer to copy into the Analogy buffer</td>
</tr>
<tr>
<td>in</td>
<td>count</td>
<td>The amount of data to copy</td>
</tr>
</tbody>
</table>

Returns

0 on success, otherwise negative error code.

References `A4L_SUBD_BUSY_NR`, `a4l_subdevice::buf`, and `a4l_subdevice::status`.

6.50.2.8 `a4l_buf_prepare_absget()`

```c
int a4l_buf_prepare_absget (  
    struct a4l_subdevice *subd,  
    unsigned long count  
)
```

Update the absolute count of data sent from the buffer to the device since the start of the acquisition and after the next DMA shot.

The functions `a4l_buf_prepare_(abs)put()`, `a4l_buf_commit_(abs)put()`, `a4l_buf_prepare_(abs)get()` and `a4l_buf_commit_(absg)et()` have been made available for DMA transfers. In such situations, no data copy is needed between the Analogy buffer and the device as some DMA controller is in charge of performing data shots from / to the Analogy buffer. However, some pointers still have to be updated so as to monitor the transfers.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>subd</th>
<th>Subdevice descriptor structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>count</td>
<td>The data count to be transferred during the next DMA shot plus the data count which have been copied since the start of the acquisition</td>
</tr>
</tbody>
</table>

Returns

0 on success, otherwise negative error code.

References `A4L_SUBD_BUSY_NR`, `a4l_subdevice::buf`, and `a4l_subdevice::status`. 
6.50 Buffer management services

6.50.2.9 a4l_buf_prepare_absput()

```c
int a4l_buf_prepare_absput (  
    struct a4l_subdevice * subd,  
    unsigned long count )
```

Update the absolute count of data sent from the device to the buffer since the start of the acquisition and after the next DMA shot.

The functions a4l_buf_prepare_(abs)put(), a4l_buf_commit_(abs)put(), a4l_buf_prepare_(abs)get() and a4l_buf_commit_(abs)get() have been made available for DMA transfers. In such situations, no data copy is needed between the Analogy buffer and the device as some DMA controller is in charge of performing data shots from / to the Analogy buffer. However, some pointers still have to be updated so as to monitor the transfers.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>subd</th>
<th>Subdevice descriptor structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>count</td>
<td>The data count to be transferred during the next DMA shot plus the data count which have been copied since the start of the acquisition</td>
</tr>
</tbody>
</table>

**Returns**

0 on success, otherwise negative error code.

**References** A4L_SUBD_BUSY_NR, a4l_subdevice::buf, and a4l_subdevice::status.

6.50.2.10 a4l_buf_prepare_get()

```c
int a4l_buf_prepare_get (  
    struct a4l_subdevice * subd,  
    unsigned long count )
```

Set the count of data which is to be sent from the buffer to the device at the next DMA shot.

The functions a4l_buf_prepare_(abs)put(), a4l_buf_commit_(abs)put(), a4l_buf_prepare_(abs)get() and a4l_buf_commit_(abs)get() have been made available for DMA transfers. In such situations, no data copy is needed between the Analogy buffer and the device as some DMA controller is in charge of performing data shots from / to the Analogy buffer. However, some pointers still have to be updated so as to monitor the transfers.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>subd</th>
<th>Subdevice descriptor structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>count</td>
<td>The data count to be transferred</td>
</tr>
</tbody>
</table>

**Returns**

0 on success, otherwise negative error code.
6.50.2.11  a4l_buf_prepare_put()

```c
int a4l_buf_prepare_put (  
    struct a4l_subdevice * subd,  
    unsigned long count)  
```

Set the count of data which is to be sent to the buffer at the next DMA shot.

The functions `a4l_buf_prepare_(abs)put()` and `a4l_buf_commit_(abs)put()` have been made available for DMA transfers. In such situations, no data copy is needed between the Analogy buffer and the device as some DMA controller is in charge of performing data shots from / to the Analogy buffer. However, some pointers still have to be updated so as to monitor the transfers.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>subd</th>
<th>Subdevice descriptor structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>count</td>
<td>The data count to be transferred</td>
</tr>
</tbody>
</table>

**Returns**

0 on success, otherwise negative error code.

References A4L_SUBD_BUSY_NR, a4l_subdevice::buf, and a4l_subdevice::status.

6.50.2.12  a4l_buf_put()

```c
int a4l_buf_put (  
    struct a4l_subdevice * subd,  
    void * bufdata,  
    unsigned long count)  
```

Copy some data from the device driver to the buffer.

The function `a4l_buf_put()` must copy data coming from some acquisition device to the Analogy buffer. This ring-buffer is an intermediate area between the device driver and the user-space program, which is supposed to recover the acquired data.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>subd</th>
<th>Subdevice descriptor structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>bufdata</td>
<td>The data buffer to copy into the Analogy buffer</td>
</tr>
<tr>
<td>in</td>
<td>count</td>
<td>The amount of data to copy</td>
</tr>
</tbody>
</table>

References A4L_SUBD_BUSY_NR, a4l_subdevice::buf, and a4l_subdevice::status.
Returns

0 on success, otherwise negative error code.

References A4L_SUBD_BUSY_NR, a4l_subdevice::buf, and a4l_subdevice::status.

6.50.2.13 a4l_get_chan()

```c
int a4l_get_chan (  
    struct a4l_subdevice * subd )
```

Get the channel index according to its type.

Parameters

| in | subd | Subdevice descriptor structure |

Returns

the channel index.

References A4L_CHAN_GLOBAL_CHANDESC, a4l_get_cmd(), a4l_subdevice::buf, a4l_subdevice::chan_desc, a4l_cmd_desc::chan_descs, a4l_channels_desc::chans, a4l_channels_desc::mode, a4l_channel::nb_bits, and a4l_cmd_desc::nb-chan.

6.50.2.14 a4l_get_cmd()

```c
struct a4l_cmd_desc* a4l_get_cmd (  
    struct a4l_subdevice * subd )
```

Get the current Analogy command descriptor.

Parameters

| in | subd | Subdevice descriptor structure |

Returns

the command descriptor.

Referenced by a4l_get_chan().

Generated by Doxygen
6.51 Interrupt management services

Collaboration diagram for Interrupt management services:

![Driver API -> Interrupt management services](diagram.png)

Functions

- `unsigned int a4l_get_irq (struct a4l_device *dev)`
  
  Get the interrupt number in use for a specific device.

- `int a4l_request_irq (struct a4l_device *dev, unsigned int irq, a4l_irq_hdlr_t handler, unsigned long flags, void *cookie)`
  
  Register an interrupt handler for a specific device.

- `int a4l_free_irq (struct a4l_device *dev, unsigned int irq)`
  
  Release an interrupt handler for a specific device.

6.51.1 Detailed Description

6.51.2 Function Documentation

6.51.2.1 a4l_free_irq()

```c
int a4l_free_irq (  
    struct a4l_device * dev,  
    unsigned int irq  
)
```

Release an interrupt handler for a specific device.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>dev</th>
<th>Device descriptor structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>irq</td>
<td>Line number of the addressed IRQ</td>
</tr>
</tbody>
</table>

Returns

0 on success, otherwise negative error code.
6.51.2.2 a4l_get_irq()

```c
unsigned int a4l_get_irq (const struct a4l_device *dev)
```

Get the interrupt number in use for a specific device.

**Parameters**

- **dev**: Device descriptor structure

**Returns**

The line number used or A4L_IRQ_UNUSED if no interrupt is registered.


6.51.2.3 a4l_request_irq()

```c
int a4l_request_irq (const struct a4l_device *dev, unsigned int irq, a4l_irq_hdlr_t handler, unsigned long flags, void *cookie)
```

Register an interrupt handler for a specific device.

**Parameters**

- **dev**: Device descriptor structure
- **irq**: Line number of the addressed IRQ
- **handler**: Interrupt handler
- **flags**: Registration flags:
  - RTDM_IRQTYPE_SHARED: enable IRQ-sharing with other drivers (Warning: real-time drivers and non-real-time drivers cannot share an interrupt line).
  - RTDM_IRQTYPE_EDGE: mark IRQ as edge-triggered (Warning: this flag is meaningless in RTDM-less context).
  - A4L_IRQ_DISABLED: keep IRQ disabled when calling the action handler (Warning: this flag is ignored in RTDM-enabled configuration).
- **cookie**: Pointer to be passed to the interrupt handler on invocation
Returns

0 on success, otherwise negative error code.
6.52 Misc services

Collaboration diagram for Misc services:

```
Driver API ➔ Misc services
```

Functions

- `unsigned long long a4l_get_time(void)`
  
  *Get the absolute time in nanoseconds.*

6.52.1 Detailed Description

6.52.2 Function Documentation

6.52.2.1 a4l_get_time()

```c
unsigned long long a4l_get_time ( 
    void 
)
```

Get the absolute time in nanoseconds.

Returns

the absolute time expressed in nanoseconds

References `rtdm_clock_read()`, `rtdm_event_clear()`, `rtdm_event_destroy()`, `rtdm_event_init()`, `rtdm_event_signal()`, `rtdm_event_timedwait()`, `rtdm_event_wait()`, `rtdm_irq_free()`, `rtdm_irq_get_arg()`, `RTD_M_IRQ_HANDLED`, `RTDM_IRQ_NONE`, `rtdm_irq_request()`, `rtndm_nrtsig_destroy()`, `rtndm_nrtsig_init()`, and `rtdm_nrtsig_pend()`. 

Generated by Doxygen
6.53 Clocks and timers

Cobalt/POSIX clock and timer services.

Collaboration diagram for Clocks and timers:

![Collaboration Diagram](image)

Functions

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

6.53.1 Detailed Description

Cobalt/POSIX clock and timer services.

Cobalt supports three built-in clocks:

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

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

In addition, external clocks can be dynamically registered using the cobalt_clock_register() service. These clocks are fully managed by Cobalt extension code, which should advertise each incoming tick by calling xnclock_tick() for the relevant clock, from an interrupt context.

Timer objects may be created with the timer_create() service using any of the built-in or external clocks. The resolution of these timers is clock-specific. However, built-in clocks all have nanosecond resolution, as specified for clock_nanosleep().

See also Specification.

6.53.2 Function Documentation

6.53.2.1 clock_getres()

int clock_getres ( clockid_t clock_id, struct timespec * tp )

Get the resolution of the specified clock.

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

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

Parameters

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

Return values

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

Specification.

Tags

unrestricted

6.53.2.2  clock_gettime()

int clock_gettime (  
clockid_t clock_id,  
struct timespec * tp )

Read the specified clock.

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

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

Parameters

| clock_id | clock identifier, either CLOCK_REALTIME, CLOCK_MONOTONIC, or CLOCK_HOST_REALTIME; |
| tp       | the address where the value of the specified clock will be stored. |

Return values

| 0 | on success; |
| 1 | with errno set if: |
|   | ● EINVAL, clock_id is invalid. |

See also

Specification.

Tags

unrestricted
6.53 Clocks and timers

6.53.2.3 clock_nanosleep()

```c
int clock_nanosleep (clockid_t clock_id, int flags, const struct timespec *rqtp, struct timespec *rmtp)
```

Sleep some amount of time.

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

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

The resolution of this service is one system clock tick.

Parameters

<table>
<thead>
<tr>
<th><code>clockid_t clock_id</code></th>
<th>clock identifier, either <code>CLOCK_REALTIME</code> or <code>CLOCK_MONOTONIC</code>.</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int flags</code></td>
<td>one of:</td>
</tr>
<tr>
<td></td>
<td>• 0 meaning that the wakeup time <code>rqtp</code> is a time interval;</td>
</tr>
<tr>
<td></td>
<td>• TIMER_ABSTIME, meaning that the wakeup time is an absolute value of the clock <code>clock_id</code>.</td>
</tr>
<tr>
<td><code>const struct timespec *rqtp</code></td>
<td>address of the wakeup time.</td>
</tr>
<tr>
<td><code>struct timespec *rmtp</code></td>
<td>address where the remaining time before wakeup will be stored if the service is interrupted by a signal.</td>
</tr>
</tbody>
</table>

Returns

0 on success;
an error number if:

- EPERM, the caller context is invalid;
- ENOTSUP, the specified clock is unsupported;
- EINVAL, the specified wakeup time is invalid;
- EINTR, this service was interrupted by a signal.

See also

`Specification`

Tags

- xthread-only, switch-primary

Referenced by `clock_settime()`, and `nanosleep()`.
6.53.2.4 clock_settime()

```c
int clock_settime (
    clockid_t clock_id,
    const struct timespec * tp )
```

Set the specified clock.

This allow setting the CLOCK_REALTIME clock.

Parameters

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

Return values

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

See also

- Specification.

Tags

- unrestricted

References clock_nanosleep().

6.53.2.5 nanosleep()

```c
int nanosleep (  
    const struct timespec * rqtp,  
    struct timespec * rmtp )
```

Sleep some amount of time.

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

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

The resolution of this service is one system clock tick.
Parameters

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

Return values

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

See also

`Specification`. 

Tags

`xthread-only, switch-primary`

References `clock_nanosleep()`.

6.53.2.6 `timer_create()`

```c
int timer_create (    
    clockid_t clockid,    
    const struct sigevent *__restrict__ evp,    
    timer_t *__restrict__ timerid )
```

Create a timer.

This service creates a timer based on the clock `clockid`.

If `evp` is not `NULL`, it describes the notification mechanism used on timer expiration. Only thread-directed notification is supported (`evp->sigev_notify` set to `SIGEV_THREAD_ID`).

If `evp` is `NULL`, the current Cobalt thread will receive the notifications with signal `SIGALRM`.

The recipient thread is delivered notifications when it calls any of the `sigwait()`, `sigtimedwait()` or `sigwaitinfo()` services.

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

- **clockid** clock used as a timing base;
- **evp** description of the asynchronous notification to occur when the timer expires;
- **timerid** address where the identifier of the created timer will be stored on success.

Return values

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

See also

- Specification.

Tags

- thread-unrestricted

Referenced by pthread_make_periodic_np().

6.53.2.7 timer_delete()

```c
int timer_delete ( 
    timer_t timerid )
```

Delete a timer object.

This service deletes the timer **timerid**.

Parameters

- **timerid** identifier of the timer to be removed;

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>on success;</td>
</tr>
</tbody>
</table>
Return values

<table>
<thead>
<tr>
<th>-1</th>
<th>with errno set if:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• EINVAL, timerid is invalid;</td>
</tr>
<tr>
<td></td>
<td>• EPERM, the timer timerid does not belong to the current process.</td>
</tr>
</tbody>
</table>

See also

Specification.

See also

Specification.

Tags

thread-unrestricted

References timer_settime().

6.53.2.8 timer_getoverrun()

int timer_getoverrun (timer_t timerid)

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

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

Parameters

| timerid | Timer identifier. |

Returns

the overruns count on success;
-1 with errno set if:
  • EINVAL, timerid is invalid;
  • EPERM, the timer timerid does not belong to the current process.

See also

Specification.

Tags

unrestricted
6.53.2.9  timer_gettime()

```c
int timer_gettime (
    timer_t timerid,
    struct itimerspec * value )
```

Get timer next expiration date and reload value.

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

Parameters

<table>
<thead>
<tr>
<th>timerid</th>
<th>timer identifier;</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>address where the timer expiration date and reload value are stored on success.</td>
</tr>
</tbody>
</table>

Return values

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

- EINVAL, timerid is invalid. For timerid to be valid, it must belong to the current process.

See also

Specification.

Tags

- unrestricted

6.53.2.10  timer_settime()

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

Start or stop a timer.

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

If the member it_value of the itimerspec structure at value is zero, the timer is stopped, otherwise the timer is started. If the member it_interval is not zero, the timer is periodic. The current thread must be a Cobalt thread (created with pthread_create()) and will be notified via signal of timer expirations.

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

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timerid</td>
<td>identifier of the timer to be started or stopped;</td>
</tr>
<tr>
<td>flags</td>
<td>one of 0 or TIMER_ABSTIME;</td>
</tr>
<tr>
<td>value</td>
<td>address where the specified timer expiration date and reload value are read;</td>
</tr>
<tr>
<td>ovalue</td>
<td>address where the specified timer previous expiration date and reload value are stored if not NULL.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>on success;</td>
</tr>
<tr>
<td>-1</td>
<td>with errno set if:</td>
</tr>
<tr>
<td></td>
<td>• EINVAL, the specified timer identifier, expiration date or reload value is invalid. For timerid to be valid, it must belong to the current process.</td>
</tr>
</tbody>
</table>

See also

Specification.

Tags

xcontext, switch-primary

Referenced by pthread_make_periodic_np(), and timer_delete().

Generated by Doxygen
6.54 Condition variables

Cobalt/POSIX condition variable services.

Collaboration diagram for Condition variables:

![Collaboration diagram](image.png)

### Functions

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

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

- **int pthread_cond_wait (pthread_cond_t *cond, pthread_mutex_t *mutex)**
  
  *Wait on a condition variable.*

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

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

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

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

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

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

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

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

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

6.54.1 Detailed Description

Cobalt/POSIX condition variable services.

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

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

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

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

6.54.2 Function Documentation

6.54.2.1 `pthread_cond_broadcast()`

```c
int pthread_cond_broadcast (  
    pthread_cond_t * cond )
```

Broadcast a condition variable.

This service unblocks all threads blocked on the condition variable `cond`.

Parameters

| cond | the condition variable to be signalled. |

Returns

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

Generated by Doxygen
6.54.2.2  pthread_cond_destroy()

```c
int pthread_cond_destroy ( pthread_cond_t * cond )
```

Destroy a condition variable.

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

Parameters

- `cond` the condition variable to be destroyed.

Returns

- 0 on success,
- an error number if:
  - EINVAL, the condition variable `cond` is invalid;
  - EPERM, the condition variable is not process-shared and does not belong to the current process;
  - EBUSY, some thread is currently using the condition variable.

See also

- Specification.

Tags

- xthread-only

6.54.2.3  pthread_cond_init()

```c
int pthread_cond_init ( pthread_cond_t * cond,
                       const pthread_condattr_t * attr )
```

Initialize a condition variable.

This service initializes the condition variable `cond`, using the condition variable attributes object `attr`. If `attr` is NULL, default attributes are used (see `pthread_condattr_init()`).
6.54 Condition variables

Parameters

| cond | the condition variable to be initialized; |
| attr | the condition variable attributes object. |

Returns

0 on success, an error number if:

- EINVAL, the condition variable attributes object attr is invalid or uninitialized;
- EBUSY, the condition variable cond was already initialized;
- ENOMEM, insufficient memory available from the system heap to initialize the condition variable, increase CONFIG_XENO_OPT_SYS_HEAPSZ.
- EAGAIN, no registry slot available, check/raise CONFIG_XENO_OPT_REGISTRY_NRSL→OTS.

See also

Specification.

Tags

thread-unrestricted

6.54.2.4  pthread_cond_signal()

int pthread_cond_signal ( 
    pthread_cond_t * cond )

Signal a condition variable.

This service unblocks one thread blocked on the condition variable cond.

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

Parameters

| cond | the condition variable to be signalled. |

Returns

0 on success, an error number if:

- EINVAL, the condition variable is invalid;
- EPERM, the condition variable is not process-shared and does not belong to the current process.
6.54.2.5  

`pthread_cond_timedwait()`

```
int pthread_cond_timedwait ( 
    pthread_cond_t *cond, 
    pthread_mutex_t *mutex, 
    const struct timespec *abstime )
```

Wait a bounded time on a condition variable.

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

The timeout `abstime` is expressed as an absolute value of the `clock` attribute passed to `pthread_cond_init()`. By default, `CLOCK_REALTIME` is used.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cond</code></td>
<td>the condition variable to wait for;</td>
</tr>
<tr>
<td><code>mutex</code></td>
<td>the mutex associated with <code>cnd</code>;</td>
</tr>
</tbody>
</table>
| `abstime` | the timeout, expressed as an absolute value of the clock attribute passed to `pthread_cond_init()`.

**Returns**

0 on success,

an error number if:

- EPERM, the caller context is invalid;
- EPERM, the specified condition variable is not process-shared and does not belong to the current process;
- EINVAL, the specified condition variable, mutex or timeout is invalid;
- EINVAL, another thread is currently blocked on `cnd` using another mutex than `mx`;
- EPERM, the specified mutex is not owned by the caller;
- ETIMEDOUT, the specified timeout expired.

**See also**

*Sspecification.*

**Tags**

`xthread-only, switch-primary`
6.54.2.6  pthread_cond_wait()

```c
int pthread_cond_wait (  
    pthread_cond_t * cond,  
    pthread_mutex_t * mutex )
```

Wait on a condition variable.

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

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

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

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

This service is a cancellation point for Cobalt threads (created with the `pthread_create()` service). When such a thread is cancelled while blocked in a call to this service, the mutex `mx` is re-acquired before the cancellation cleanup handlers are called.

**Parameters**

<table>
<thead>
<tr>
<th><code>cond</code></th>
<th>the condition variable to wait for;</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>mutex</code></td>
<td>the mutex associated with <code>cnd</code>.</td>
</tr>
</tbody>
</table>

**Returns**

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

**See also**

- Specification.

**Tags**

- xthread-only, switch-primary
6.54.2.7  pthread_condattr_destroy()

int pthread_condattr_destroy (   
    pthread_condattr_t * attr )

Destroy a condition variable attributes object.

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

Parameters

| attr | the initialized mutex attributes object to be destroyed. |

Returns

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

See also

Specification.

Tags

thread-unrestricted

6.54.2.8  pthread_condattr_getclock()

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

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

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

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

Parameters

| attr | an initialized condition variable attributes object, |
| clk_id | address where the clock attribute value will be stored on success. |
6.54 Condition variables

Returns

0 on success,
an error number if:

- EINVAL, the attribute object attr is invalid.

See also

Specification.

Tags

thread-unrestricted

6.54.2.9 pthread_condattr_getpshared()

int pthread_condattr_getpshared (
   const pthread_condattr_t * attr,
   int * pshared )

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

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

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

Parameters

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

Returns

0 on success,
an error number if:

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

See also

Specification.

Tags

thread-unrestricted
6.54.2.10  pthread_condattr_init()

```c
int pthread_condattr_init (
    pthread_condattr_t * attr )
```

Initialize a condition variable attributes object.

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

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

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

Parameters

- `attr` the condition variable attributes object to be initialized.

Returns

- 0 on success;
- an error number if:
  - ENOMEM, the condition variable attribute object pointer `attr` is `NULL`.

See also

- `Specification`.

Tags

- `thread-unrestricted`

6.54.2.11  pthread_condattr_setclock()

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

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

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

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

<table>
<thead>
<tr>
<th>attr</th>
<th>an initialized condition variable attributes object,</th>
</tr>
</thead>
<tbody>
<tr>
<td>clk_id</td>
<td>value of the clock attribute, may be CLOCK_REALTIME or CLOCK_MONOTONIC.</td>
</tr>
</tbody>
</table>

Returns

0 on success,
an error number if:
- EINVAL, the condition variable attributes object attr is invalid;
- EINVAL, the value of clk_id is invalid for the clock attribute.

See also

Specification.

Tags

thread-unrestricted

6.54.2.12  pthread_condattr_setpshared()

int pthread_condattr_setpshared ( pthread_condattr_t * attr, int pshared )

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

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

Parameters

<table>
<thead>
<tr>
<th>attr</th>
<th>an initialized condition variable attributes object.</th>
</tr>
</thead>
<tbody>
<tr>
<td>pshared</td>
<td>value of the pshared attribute, may be one of:</td>
</tr>
<tr>
<td></td>
<td>-PTHREAD_PROCESS_PRIVATE, meaning that a condition variable created with the attributes object attr will only be accessible by threads within the same process as the thread that initialized the condition variable;</td>
</tr>
<tr>
<td></td>
<td>-PTHREAD_PROCESS_SHARED, meaning that a condition variable created with the attributes object attr will be accessible by any thread that has access to the memory where the condition variable is allocated.</td>
</tr>
</tbody>
</table>

Returns

0 on success,
an error status if:
• EINVAL, the condition variable attributes object attr is invalid;
• EINVAL, the value of pshared is invalid.

See also

Specification.

Tags

thread-unrestricted
6.55 POSIX interface

The Cobalt/POSIX interface is an implementation of a subset of the Single Unix specification over the Cobalt core.

Collaboration diagram for POSIX interface:

Modules

- Clocks and timers
  Cobalt/POSIX clock and timer services.
- Condition variables
  Cobalt/POSIX condition variable services.
- Message queues
  Cobalt/POSIX message queue services.
- Mutual exclusion
  Cobalt/POSIX mutual exclusion services.
- Process scheduling
  Cobalt/POSIX process scheduling.
- Semaphores
  Cobalt/POSIX semaphore services.
- Thread management
  Cobalt (POSIX) thread management services.
- Scheduling management
  Cobalt scheduling management services.
6.55.1 Detailed Description

The Cobalt/POSIX interface is an implementation of a subset of the Single Unix specification over the Cobalt core.
6.56 Message queues

Cobalt/POSIX message queue services.

Collaboration diagram for Message queues:

```
+---------------------+---------------------+
| POSIX interface     | Message queues      |
+---------------------+---------------------+
```

Functions

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

- `int mq_close (mqd_t mqd)`
  
  Close a message queue.

- `int mq_unlink (const char *name)`
  
  Unlink a message queue.

- `int mq_getattr (mqd_t mqd, struct mq_attr *attr)`
  
  Get message queue attributes.

- `int mq_setattr (mqd_t mqd, const struct mq_attr *__restrict__ attr, struct mq_attr *__restrict__ oattr)`
  
  Set message queue attributes.

- `int mq_send (mqd_t q, const char *buffer, size_t len, unsigned prio)`
  
  Send a message to a message queue.

- `int mq_timedsend (mqd_t q, const char *buffer, size_t len, unsigned prio, const struct timespec *timeout)`
  
  Attempt, during a bounded time, to send a message to a message queue.

- `ssize_t mq_receive (mqd_t q, char *buffer, size_t len, unsigned *prio)`
  
  Receive a message from a message queue.

- `ssize_t mq_timedreceive (mqd_t q, char *__restrict__ buffer, size_t len, unsigned *__restrict__ prio, const struct timespec *__restrict__ timeout)`
  
  Attempt, during a bounded time, to receive a message from a message queue.

- `int mq_notify (mqd_t mqd, const struct sigevent *__restrict__ evp)`
  
  Enable notification on message arrival.

6.56.1 Detailed Description

Cobalt/POSIX message queue services.

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

Generated by Doxygen
6.56.2 Function Documentation

6.56.2.1 mq_close()

```c
int mq_close (  
    mqd_t mqd )  
```

Close a message queue.

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

Parameters

- `mqd` message queue descriptor.

Return values

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

See also

- Specification.

Tags

- thread-unrestricted, switch-secondary

6.56.2.2 mq_getattr()

```c
int mq_getattr (  
    mqd_t mqd,  
    struct mq_attr *attr )  
```

Get message queue attributes.

This service stores, at the address `attr`, the attributes of the messages queue descriptor `mqd`. The following attributes are set:

- `mq_flags`, flags of the message queue descriptor `mqd`;
- `mq_maxmsg`, maximum number of messages in the message queue;
- `mq_msgsize`, maximum message size;
- `mq_curmsgs`, number of messages currently in the queue.
6.56 Message queues

Parameters

| mqd | message queue descriptor; |
| attr | address where the message queue attributes will be stored on success. |

Return values

| 0 | on success; |
| -1 | with errno set if: |
| | • EBADF, mqd is not a valid descriptor. |

See also

Specification.

Tags

thread-unrestricted

References mq_setattr().

6.56.2.3 mq_notify()

```c
int mq_notify ( 
    mqd_t mqd, 
    const struct sigevent * evp )
```

Enable notification on message arrival.

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

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

Only one thread may be registered at a time.

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

Parameters

| mqd | message queue descriptor; |
| evp | pointer to an event notification structure. |
Return values

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

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

See also
- Specification.

Tags
- xthread-only, switch-primary

6.56.2.4  mq_open()

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

Open a message queue.

This service opens the message queue named name.

One of the following values should be set in oflags:

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

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

- a mode argument, of type mode_t, currently ignored;
- an attr argument, pointer to an mq_attr structure, specifying the attributes of the new message queue.
If `oflags` has the two bits `O_CREAT` and `O_EXCL` set and the message queue already exists, this service fails.

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

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

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

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

**Parameters**

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

**Returns**

- A message queue descriptor on success;
- -1 with `errno` set if:
  - ENAMETOOLONG, the length of the `name` argument exceeds 64 characters;
  - EEXIST, the bits `O_CREAT` and `O_EXCL` were set in `oflags` and the message queue already exists;
  - ENOENT, the bit `O_CREAT` is not set in `oflags` and the message queue does not exist;
  - ENOSPC, allocation of system memory failed, or insufficient memory available from the system heap to create the queue, try increasing `CONFIG_XENO_OPT_SYS_HEAP_SZ`;
  - EPERM, attempting to create a message queue from an invalid context;
  - EINVAL, the `attr` argument is invalid;
  - EMFILE, too many descriptors are currently open.
  - EAGAIN, no registry slot available, check/raise `CONFIG_XENO_OPT_REGISTRY_NRSL`.

See also

- Specification.

**Tags**

- thread-unrestricted, switch-secondary
6.5.6.2.5  mq_receive()

ssize_t mq_receive (  
    mqd_t  q,  
    char * buffer,  
    size_t len,  
    unsigned * prio )

Receive a message from a message queue.

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

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{q}</td>
<td>the queue descriptor;</td>
</tr>
<tr>
<td>\textit{buffer}</td>
<td>the address where the received message will be stored on success;</td>
</tr>
<tr>
<td>\textit{len}</td>
<td>\textit{buffer} length;</td>
</tr>
<tr>
<td>\textit{prio}</td>
<td>address where the priority of the received message will be stored on success.</td>
</tr>
</tbody>
</table>

Returns

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

- EBADF, \textit{fd} is not a valid descriptor open for reading;
- EMSGSIZE, the length \textit{len} is lesser than the message queue \textit{mq_msgsize} attribute;
- EAGAIN, the queue is empty, and the flag \texttt{O\_NONBLOCK} is set for the descriptor \textit{fd};
- EPERM, the caller context is invalid;
- EINTR, the service was interrupted by a signal.

See also

Specification.

Tags

\texttt{xthread-only, switch-primary}

References \texttt{mq\_timedreceive()}. 

Generated by Doxygen
6.56.2.6  mq_send()

```c
int mq_send (mqd_t q, const char *buffer, size_t len, unsigned prio)
```

Send a message to a message queue.

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

If the message queue is full and the flag `O_NONBLOCK` is not set, the calling thread is suspended until the queue is not full. If the message queue is full and the flag `O_NONBLOCK` is set, the message is not sent and the service returns immediately a value of -1 with `errno` set to EAGAIN.

**Parameters**

<table>
<thead>
<tr>
<th>q</th>
<th>message queue descriptor;</th>
</tr>
</thead>
<tbody>
<tr>
<td>buffer</td>
<td>pointer to the message to be sent;</td>
</tr>
<tr>
<td>len</td>
<td>length of the message;</td>
</tr>
<tr>
<td>prio</td>
<td>priority of the message.</td>
</tr>
</tbody>
</table>

**Returns**

- 0 and send a message on success;
- -1 with no message sent and `errno` set if:
  - EBADF, `fd` is not a valid message queue descriptor open for writing;
  - EMSGSIZE, the message length `len` exceeds the `mq_msgsize` attribute of the message queue;
  - EAGAIN, the flag `O_NONBLOCK` is set for the descriptor `fd` and the message queue is full;
  - EPERM, the caller context is invalid;
  - EINTR, the service was interrupted by a signal.

**See also**

`Specification`.

**Tags**

- xthread-only, switch-primary

**References**

mq_timedsend().
6.56.2.7  mq_setattr()

```
int mq_setattr (  
    mqd_t mqd,  
    const struct mq_attr *__restrict__ attr,  
    struct mq_attr *__restrict__ oattr )
```

Set message queue attributes.

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

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

Only setting or clearing the O_NONBLOCK flag has an effect.

Parameters

- `mqd` message queue descriptor;
- `attr` pointer to new attributes (only `mq_flags` is used);
- `oattr` if not `NULL`, address where previous message queue attributes will be stored on success.

Return values

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

See also

- Specification.

Tags

- thread-unrestricted

Referenced by `mq_getattr()`.

6.56.2.8  mq_timedreceive()

```
ssize_t mq_timedreceive (  
    mqd_t q,  
    char *__restrict__ buffer,  
    size_t len,  
    unsigned *__restrict__ prio,  
    const struct timespec *__restrict__ timeout )
```

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

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

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

Returns

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

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

See also

Specification.

Tags

xthread-only, switch-primary

Referenced by mq_receive().

6.56.2.9  mq_timedsend()

int mq_timedsend (  
    mqd_t q,  
    const char * buffer,  
    size_t len,  
    unsigned prio,  
    const struct timespec * timeout )

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

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>q</td>
<td>message queue descriptor;</td>
</tr>
<tr>
<td>buffer</td>
<td>pointer to the message to be sent;</td>
</tr>
<tr>
<td>len</td>
<td>length of the message;</td>
</tr>
<tr>
<td>prio</td>
<td>priority of the message;</td>
</tr>
<tr>
<td>timeout</td>
<td>the timeout, expressed as an absolute value of the CLOCK_REALTIME clock.</td>
</tr>
</tbody>
</table>
Returns
0 and send a message on success;
-1 with no message sent and errno set if:
  - EBADF, fd is not a valid message queue descriptor open for writing;
  - EMSGSIZE, the message length exceeds the mq_msgsize attribute of the message queue;
  - EAGAIN, the flag O_NONBLOCK is set for the descriptor fd and the message queue is full;
  - EPERM, the caller context is invalid;
  - ETIMEDOUT, the specified timeout expired;
  - EINTR, the service was interrupted by a signal.

See also
Specification.

Tags
xthread-only, switch-primary

Referenced by mq_send().

6.56.2.10 mq_unlink()

int mq_unlink (const char * name)

Unlink a message queue.

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

Parameters

| name | name of the message queue to be unlinked. |

Return values

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

 Specification.

Tags

 thread-unrestricted, switch-secondary
6.57 Mutual exclusion

Cobalt/POSIX mutual exclusion services.

Collaboration diagram for Mutual exclusion:

```
+----------------+      +----------------+
|        |      |        |
| POSIX interface |      | Mutual exclusion |
+----------------+      +----------------+
```

Functions

- `int pthread_mutex_init (pthread_mutex_t *mutex, const pthread_mutexattr_t *attr)`
  
  Initialize a mutex.

- `int pthread_mutex_destroy (pthread_mutex_t *mutex)`
  
  Destroy a mutex.

- `int pthread_mutex_lock (pthread_mutex_t *mutex)`
  
  Lock a mutex.

- `int pthread_mutex_timedlock (pthread_mutex_t *mutex, const struct timespec *to)`
  
  Attempt, during a bounded time, to lock a mutex.

- `int pthread_mutex_trylock (pthread_mutex_t *mutex)`
  
  Attempt to lock a mutex.

- `int pthread_mutex_unlock (pthread_mutex_t *mutex)`
  
  Unlock a mutex.

- `int pthread_mutexattr_init (pthread_mutexattr_t *attr)`
  
  Initialize a mutex attributes object.

- `int pthread_mutexattr_destroy (pthread_mutexattr_t *attr)`
  
  Destroy a mutex attributes object.

- `int pthread_mutexattr_gettype (const pthread_mutexattr_t *attr, int *type)`
  
  Get the mutex type attribute from a mutex attributes object.

- `int pthread_mutexattr_settype (pthread_mutexattr_t *attr, int type)`
  
  Set the mutex type attribute of a mutex attributes object.

- `int pthread_mutexattr_getprotocol (const pthread_mutexattr_t *attr, int *proto)`
  
  Get the protocol attribute from a mutex attributes object.

- `int pthread_mutexattr_setprotocol (pthread_mutexattr_t *attr, int proto)`
  
  Set the protocol attribute of a mutex attributes object.

- `int pthread_mutexattr_getpshared (const pthread_mutexattr_t *attr, int *pshared)`
  
  Get the process-shared attribute of a mutex attributes object.

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

6.57.1 Detailed Description

Cobalt/POSIX mutual exclusion services.

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

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

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

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

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

6.57.2 Function Documentation

6.57.2.1 `pthread_mutex_destroy()`

```c
int pthread_mutex_destroy ( pthread_mutex_t *mutex )
```

Destroy a mutex.

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

Parameters

`mutex` the mutex to be destroyed.

Returns

0 on success,

an error number if:

- EINVAL, the mutex `mx` is invalid;
- EPERM, the mutex is not process-shared and does not belong to the current process;
- EBUSY, the mutex is locked, or used by a condition variable.
6.57.2.2  pthread_mutex_init()

Initialize a mutex.

This service initializes the mutex \textit{mx}, using the mutex attributes object \textit{attr}. If \textit{attr} is \textit{NULL}, default attributes are used (see \texttt{pthread_mutexattr_init()}).

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{mutex}</td>
<td>the mutex to be initialized;</td>
</tr>
<tr>
<td>\textit{attr}</td>
<td>the mutex attributes object.</td>
</tr>
</tbody>
</table>

**Returns**

0 on success, an error number if:

- EINVAL, the mutex attributes object \textit{attr} is invalid or uninitialized;
- EBUSY, the mutex \textit{mx} was already initialized;
- ENOMEM, insufficient memory available from the system heap to initialize the mutex, increase \texttt{CONFIG_XENO_OPT_SYS_HEAPSZ}.
- EAGAIN, insufficient memory available to initialize the mutex, increase \texttt{CONFIG_XENO_OPT_PT_SHARED_HEAPSZ} for a process-shared mutex, or \texttt{CONFIG_XENO_OPT_PRIVATE_HEAPSZ} for a process-private mutex.
- EAGAIN, no registry slot available, check/raise \texttt{CONFIG_XENO_OPT_REGISTRY_NRSLOTS}.

**See also**

*Specification.*

**Tags**

- thread-unrestricted
6.57.2.3  pthread_mutex_lock()

```c
int pthread_mutex_lock (
    pthread_mutex_t * mutex )
```

Lock a mutex.

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

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

Parameters

- `mutex`  the mutex to be locked.

Returns

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

See also

- Specification.

Tags

- xthread-only, switch-primary

References pthread_mutex_timedlock(), XNDEBUG, XNRELAX, and XNWEAK.

6.57.2.4  pthread_mutex_timedlock()

```c
int pthread_mutex_timedlock (
    pthread_mutex_t * mutex,
    const struct timespec * to )
```

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

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

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

Returns

0 on success;
an error number if:

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

See also

Specification.

Tags

xthread-only, switch-primary

References XNDEBUG, XNRELAX, and XNWEAK.

Referenced by pthread_mutex_lock().

6.57.2.5  pthread_mutex_trylock()

int pthread_mutex_trylock ( 
    pthread_mutex_t * mutex )

Attempt to lock a mutex.

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

Parameters

| mutex | the mutex to be locked. |
Returns

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

See also

Specification.

Tags

xthread-only, switch-primary

References XNDEBUG, XNRELAX, and XNWEAK.

6.57.2.6  pthread_mutex_unlock()

int pthread_mutex_unlock ( pthread_mutex_t *mutex )

Unlock a mutex.

This service unlocks the mutex mx. If the mutex is of the PTHREAD_MUTEX_RECURSIVE type and the locking recursion count is greater than one, the lock recursion count is decremented and the mutex remains locked.

Attempting to unlock a mutex which is not locked or which is locked by another thread than the current one yields the EPERM error, whatever the mutex type attribute.

Parameters

mutex  the mutex to be released.

Returns

0 on success;
an error number if:
- EPERM, the caller context is invalid;
- EINVAL, the mutex mx is invalid;
- EPERM, the mutex was not locked by the current thread.
See also

* Specification.

Tags

* xthread-only, switch-primary

References * pthread_mutexattr_destroy(), pthread_mutexattr_getprotocol(), pthread_mutexattr_getpshared(), pthread_mutexattr_gettype(), pthread_mutexattr_init(), pthread_mutexattr_setprotocol(), pthread_mutexattr_setpshared(), pthread_mutexattr_settype(), XNDEBUG, and XNWEAK.

### 6.57.2.7 pthread_mutexattr_destroy()

```
int pthread_mutexattr_destroy (  
    pthread_mutexattr_t * attr )
```

Destroy a mutex attributes object.

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

**Parameters**

| attr | the initialized mutex attributes object to be destroyed. |

**Returns**

* 0 on success;
* an error number if:
  * EINVAL, the mutex attributes object `attr` is invalid.

See also

* Specification.

Tags

* thread-unrestricted

Referenced by pthread_mutex_unlock().
6.57.2.8  pthread_mutexattr_getprotocol()

```c
int pthread_mutexattr_getprotocol (
    const pthread_mutexattr_t *attr,
    int *proto )
```

Get the protocol attribute from a mutex attributes object.

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

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

**Parameters**

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

**Returns**

- 0 on success,
- an error number if:
  - `EINVAL`, the `proto` address is invalid;
  - `EINVAL`, the mutex attributes object `attr` is invalid.

**See also**

- `Specification`.

**Tags**

- thread-unrestricted

Referenced by `pthread_mutex_unlock()`.

6.57.2.9  pthread_mutexattr_getpshared()

```c
int pthread_mutexattr_getpshared (
    const pthread_mutexattr_t *attr,
    int *pshared )
```

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

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

The `pshared` attribute may only be one of `PTHREAD_PROCESS_PRIVATE` or `PTHREAD_PROCESS_SHARED`. See `pthread_mutexattr_setpshared()` for the meaning of these two constants.
Parameters

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

Returns

0 on success; an error number if:
- EINVAL, the pshared address is invalid;
- EINVAL, the mutex attributes object attr is invalid.

See also

Specification.

Tags

thread-unrestricted

Referenced by pthread_mutex_unlock().

6.57.2.10 pthread_mutexattr_gettype()

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

Get the mutex type attribute from a mutex attributes object.

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

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

Parameters

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

Returns

0 on success; an error number if:
- EINVAL, the type address is invalid;
- EINVAL, the mutex attributes object attr is invalid.
6.57 Mutual exclusion

See also

*Specification.*

Tags

*thread-unrestricted*

Referenced by pthread_mutex_unlock().

6.57.2.11 pthread_mutexattr_init()

```c
int pthread_mutexattr_init (  
    pthread_mutexattr_t * attr  )
```

Initialize a mutex attributes object.

This service initializes the mutex attributes object `attr` with default values for all attributes. Default values are:

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

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

Parameters

- `attr` the mutex attributes object to be initialized.

Returns

- 0 on success;
- an error number if:
  - `ENOMEM`, the mutex attributes object pointer `attr` is `NULL`.

See also

*Specification.*

Tags

*thread-unrestricted*

Referenced by pthread_mutex_unlock().
6.57.2.12  pthread_mutexattr_setprotocol()

```c
int pthread_mutexattr_setprotocol (
    pthread_mutexattr_t * attr,
    int proto )
```

Set the protocol attribute of a mutex attributes object.

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

Parameters

| attr  | an initialized mutex attributes object, |
| proto | value of the `protocol` attribute, may be one of: |
|       | • PTHREAD_PRIO_NONE, meaning that a mutex created with the attributes object `attr` will not follow any priority protocol; |
|       | • PTHREAD_PRIO_INHERIT, meaning that a mutex created with the attributes object `attr`, will follow the priority inheritance protocol. |

The value PTHREAD_PRIO_PROTECT (priority ceiling protocol) is unsupported.

Returns

0 on success,

an error number if:

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

See also

`Specification`.

Tags

`thread-unrestricted`

Referenced by pthread_mutex_unlock().

6.57.2.13  pthread_mutexattr_setpshared()

```c
int pthread_mutexattr_setpshared ( 
    pthread_mutexattr_t * attr,
    int pshared )
```

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

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

<table>
<thead>
<tr>
<th>attr</th>
<th>an initialized mutex attributes object.</th>
</tr>
</thead>
<tbody>
<tr>
<td>pshared</td>
<td>value of the pshared attribute, may be one of:</td>
</tr>
<tr>
<td></td>
<td>• PTHREAD_PROCESS_PRIVATE, meaning that a mutex created with the attributes object attr will only be accessible by threads within the same process as the thread that initialized the mutex;</td>
</tr>
<tr>
<td></td>
<td>• PTHREAD_PROCESS_SHARED, meaning that a mutex created with the attributes object attr will be accessible by any thread that has access to the memory where the mutex is allocated.</td>
</tr>
</tbody>
</table>

Returns

0 on success,
an error status if:
• EINVAL, the mutex attributes object attr is invalid;
• EINVAL, the value of pshared is invalid.

See also

Specification.

Tags
thread-unrestricted

Referenced by pthread_mutex_unlock().

6.57.2.14 pthread_mutexattr_settype()

int pthread_mutexattr_settype (
       pthread_mutexattr_t *attr,
       int type)

Set the mutex type attribute of a mutex attributes object.

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

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

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

Parameters

<table>
<thead>
<tr>
<th>attr</th>
<th>an initialized mutex attributes object,</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>value of the type attribute.</td>
</tr>
</tbody>
</table>
Returns

0 on success,
an error number if:

- EINVAL, the mutex attributes object \textit{attr} is invalid;
- EINVAL, the value of \textit{type} is invalid for the \textit{type} attribute.

See also

\textit{Specification}.

Tags

\textit{thread-unrestricted}

Referenced by \textit{pthread_mutex_unlock}().
6.58 Process scheduling

Cobalt/POSIX process scheduling.

Collaboration diagram for Process scheduling:

![Collaboration diagram](image)

Functions

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

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

- `int sched_get_priority_min_ex (int policy)`
  
  Get extended minimum priority of the specified scheduling policy.

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

- `int sched_setscheduler (pid_t pid, int policy, const struct sched_param *param)`
  
  Set the scheduling policy and parameters of the specified process.

- `int sched_setscheduler_ex (pid_t pid, int policy, const struct sched_param_ex *param_ex)`
  
  Set extended scheduling policy of a process.

- `int sched_getscheduler (pid_t pid)`
  
  Get the scheduling policy of the specified process.

- `int sched_getscheduler_ex (pid_t pid, int *policy_r, struct sched_param_ex *param_ex)`
  
  Get extended scheduling policy of a process.

- `int sched_get_priority_max_ex (int policy)`
  
  Get extended maximum priority of the specified scheduling policy.

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

- `ssize_t sched_getconfig_np (int cpu, int policy, union sched_config *config, size_t *len_r)`
  
  Retrieve CPU-specific scheduler settings for a policy.

6.58.1 Detailed Description

Cobalt/POSIX process scheduling.

See also

- Specification.
6.58.2 Function Documentation

6.58.2.1 sched_get_priority_max()

```c
int sched_get_priority_max (int policy )
```

Get maximum priority of the specified scheduling policy.

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

Parameters

- `policy` scheduling policy.

Return values

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

See also

- Specification.

Tags

- thread-unrestricted

References `sched_setscheduler()`.

Referenced by `sched_get_priority_max_ex()`.

6.58.2.2 sched_get_priority_max_ex()

```c
int sched_get_priority_max_ex (int policy )
```

Get extended maximum priority of the specified scheduling policy.

This service returns the maximum priority of the scheduling policy `policy`, reflecting any Cobalt extension to standard classes.
6.58 Process scheduling

Parameters

\begin{itemize}
\item \textit{policy} - scheduling policy.
\end{itemize}

Return values

\begin{itemize}
\item \textbf{0} on success;
\item \textbf{-1} with \textit{errno} set if:
  \begin{itemize}
  \item \texttt{EINVAL}, \textit{policy} is invalid.
  \end{itemize}
\end{itemize}

See also

Specification.

Tags

thread-unrestricted

References sched_get_priority_max().

6.58.2.3 sched_get_priority_min()

\begin{verbatim}
int sched_get_priority_min ( 
    int policy );
\end{verbatim}

Get minimum priority of the specified scheduling policy.

This service returns the minimum priority of the scheduling policy \textit{policy}.

Parameters

\begin{itemize}
\item \textit{policy} - scheduling policy.
\end{itemize}

Return values

\begin{itemize}
\item \textbf{0} on success;
\item \textbf{-1} with \textit{errno} set if:
  \begin{itemize}
  \item \texttt{EINVAL}, \textit{policy} is invalid.
  \end{itemize}
\end{itemize}

See also

Specification.

Tags

thread-unrestricted

Generated by Doxygen
Referenced by sched_get_priority_min_ex().

6.58.2.4 sched_get_priority_min_ex()

```c
int sched_get_priority_min_ex ( int policy )
```

Get extended minimum priority of the specified scheduling policy.

This service returns the minimum priority of the scheduling policy `policy`, reflecting any Cobalt extension to the standard classes.

**Parameters**

- `policy`: scheduling policy.

**Return values**

- 0: on success;
- -1: with `errno` set if:
  - EINVAL: `policy` is invalid.

**See also**

- [Specification](#).

**Tags**

- thread-unrestricted

References sched_get_priority_min().

6.58.2.5 sched_getconfig_np()

```c
ssize_t sched_getconfig_np ( int cpu,
    int policy,
    union sched_config * config,
    size_t * len_r )
```

Retrieve CPU-specific scheduler settings for a policy.

A configuration is strictly local to the target `cpu`, and may differ from other processors.
Parameters

<table>
<thead>
<tr>
<th>cpu</th>
<th>processor to retrieve the configuration of.</th>
</tr>
</thead>
<tbody>
<tr>
<td>policy</td>
<td>scheduling policy to which the configuration data applies. Currently, only SCHED_TP and SCHED_QUOTA are valid input.</td>
</tr>
<tr>
<td>config</td>
<td>a pointer to a memory area which receives the configuration settings upon success of this call.</td>
</tr>
</tbody>
</table>

SCHED_TP specifics

On successful return, config->quota.tp contains the TP schedule active on cpu.

SCHED_QUOTA specifics

On entry, config->quota.get.tgid must contain the thread group identifier to inquire about.

On successful exit, config->quota.info contains the information related to the thread group referenced to by config->quota.get.tgid.

Parameters

| in, out len←_r | a pointer to a variable for collecting the overall length of the configuration data returned (in bytes). This variable must contain the amount of space available in config when the request is issued. |

Returns

the number of bytes copied to config on success;

a negative error number if:

-EINVAL, cpu is invalid, or policy is unsupported by the current kernel configuration, or len cannot hold the retrieved configuration data.

-ESRCH, with policy equal to SCHED_QUOTA, if the group identifier required to perform the operation is not valid (i.e. config->quota.get.tgid is invalid).

-ENOMEM, lack of memory to perform the operation.

-ENOSPC, len is too short.

Tags

thread-unrestricted, switch-primary
6.58.2.6 sched_getscheduler()

```c
int sched_getscheduler ( 
    pid_t pid )
```

Get the scheduling policy of the specified process.

This service retrieves the scheduling policy of the Cobalt process identified by `pid`.

If `pid` does not identify an existing Cobalt thread/process, this service falls back to the regular `sched_getscheduler()` service.

**Parameters**

| pid | target process/thread; |

**Returns**

- 0 on success;
- an error number if:
  - ESRCH, `pid` is not found;
  - EINVAL, `pid` is negative
  -EFAULT, `param_ex` is an invalid address;

**See also**

- Specification.

**Tags**

- thread-unrestricted

6.58.2.7 sched_getscheduler_ex()

```c
int sched_getscheduler_ex ( 
    pid_t pid, 
    int *policy_r, 
    struct sched_param_ex *param_ex )
```

Get extended scheduling policy of a process.

This service is an extended version of the `sched_getscheduler()` service, which supports Cobalt-specific and/or additional scheduling policies, not available with the host Linux environment. It retrieves the scheduling policy of the Cobalt process/thread identified by `pid`, and the associated scheduling parameters (e.g. the priority).

**Parameters**

<table>
<thead>
<tr>
<th>pid</th>
<th>queried process/thread. If zero, the current thread is assumed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>policy_r</td>
<td>a pointer to a variable receiving the current scheduling policy of <code>pid</code>.</td>
</tr>
<tr>
<td>param_ex</td>
<td>a pointer to a structure receiving the current scheduling parameters of <code>pid</code>.</td>
</tr>
</tbody>
</table>
6.58 Process scheduling

Returns

0 on success;
an error number if:

- ESRCH, *pid* is not a Cobalt thread;
- EINVAL, *pid* is negative or *param_ex* is NULL;
- EFAULT, *param_ex* is an invalid address;

Tags

thread-unrestricted

6.58.2.8  sched_setconfig_np()

```c
int sched_setconfig_np ( 
    int cpu, 
    int policy, 
    const union sched_config *config, 
    size_t len )
```

Set CPU-specific scheduler settings for a policy.

A configuration is strictly local to the target *cpu*, and may differ from other processors.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>cpu</em></td>
<td>processor to load the configuration of.</td>
</tr>
<tr>
<td><em>policy</em></td>
<td>scheduling policy to which the configuration data applies. Currently, SCHED_TP and SCHED_QUOTA are valid.</td>
</tr>
<tr>
<td><em>config</em></td>
<td>a pointer to the configuration data to load on <em>cpu</em>, applicable to <em>policy</em>.</td>
</tr>
</tbody>
</table>

Settings applicable to SCHED_TP

This call controls the temporal partitions for *cpu*, depending on the operation requested.

- config.tp.op specifies the operation to perform:
  - sched_tp_install installs a new TP schedule on *cpu*, defined by config.tp.windows[]. The global time frame is not activated upon return from this request yet; sched_tp_start must be issued to activate the temporal scheduling on CPU.
  - sched_tp_uninstall removes the current TP schedule from *cpu*, releasing all the attached resources. If no TP schedule exists on CPU, this request has no effect.
  - sched_tp_start enables the temporal scheduling on *cpu*, starting the global time frame. If no TP schedule exists on cpu, this action has no effect.
  - sched_tp_stop disables the temporal scheduling on *cpu*. The current TP schedule is not uninstalled though, and may be re-started later by a sched_tp_start request.
Attention

As a consequence of this request, threads assigned to the un-scheduled partitions may be starved from CPU time.

- for a `sched_tp_install` operation, `config.tp.nr_windows` indicates the number of elements present in the `config.tp.windows[]` array. If `config.tp.nr_windows` is zero, the action taken is identical to `sched_tp_uninstall`.

- if `config.tp.nr_windows` is non-zero, `config.tp.windows[]` is a set scheduling time slots for threads assigned to `cpu`. Each window is specified by its offset from the start of the global time frame (`windows[].offset`), its duration (`windows[].duration`), and the partition id it should activate during such period of time (`windows[].ptid`). This field is not considered for other requests than `sched← TP_install`.

Time slots must be strictly contiguous, i.e. `windows[n].offset + windows[n].duration` shall equal `windows[n + 1].offset`. If `windows[].ptid` is in the range `[0..CONFIG_XENO_OPT_SCHED_TP_NRP← ART-1]`, SCHED_TP threads which belong to the partition being referred to may be given CPU time on `cpu`, from time `windows[].offset` to `windows[].offset + windows[].duration`, provided those threads are in a runnable state.

Time holes between valid time slots may be defined using windows activating the pseudo partition -1. When such window is active in the global time frame, no CPU time is available to SCHED_TP threads on `cpu`.

Note

The `sched tp_confsize(nr_windows)` macro returns the length of `config.tp` depending on the number of time slots to be defined in `config.tp.windows[]`, as specified by `config.tp.nr_windows`.

Settings applicable to SCHED_QUOTA

This call manages thread groups running on `cpu`, defining per-group quota for limiting their CPU consumption.

- `config.quota.op` should define the operation to be carried out. Valid operations are:
  - `sched_quota_add` for creating a new thread group on `cpu`. The new group identifier will be written back to `info.tgid` upon success. A new group is given no initial runtime budget when created. `sched_quota_set` should be issued to enable it.
  - `sched_quota_remove` for deleting a thread group on `cpu`. The group identifier should be passed in `config.quota.remove.tgid`.
  - `sched_quota_set` for updating the scheduling parameters of a thread group defined on `cpu`. The group identifier should be passed in `config.quota.set.tgid`, along with the allotted percentage of the quota interval (`config.quota.set.quota`), and the peak percentage allowed (`config← quota.set.quota_peak`).

All three operations fill in the `config.info` structure with the information reflecting the state of the scheduler on `cpu` with respect to `policy`, after the requested changes have been applied.

Parameters

| len | overall length of the configuration data (in bytes). |
Returns

0 on success;
an error number if:

- EINVAL, cpu is invalid, or policy is unsupported by the current kernel configuration, len is invalid, or config contains invalid parameters.
- ENOMEM, lack of memory to perform the operation.
- EBUSY, with policy equal to SCHED_QUOTA, if an attempt is made to remove a thread group which still manages threads.
- ESRCH, with policy equal to SCHED_QUOTA, if the group identifier required to perform the operation is not valid.

Tags

thread-unrestricted, switch-primary

6.58.2.9 sched_setscheduler()

int sched_setscheduler (  
    pid_t pid,  
    int policy,  
    const struct sched_param *param )

Set the scheduling policy and parameters of the specified process.

This service set the scheduling policy of the Cobalt process identified by pid to the value policy, and its scheduling parameters (i.e. its priority) to the value pointed to by param.

If the current Linux thread ID is passed (see gettid(2)), this service turns the current regular POSIX thread into a Cobalt thread. If pid is neither the identifier of the current thread nor the identifier of an existing Cobalt thread, this service falls back to the regular sched_setscheduler() service.

Parameters

<table>
<thead>
<tr>
<th>pid</th>
<th>target process/thread;</th>
</tr>
</thead>
<tbody>
<tr>
<td>policy</td>
<td>scheduling policy, one of SCHED_FIFO, SCHED_RR, or SCHED_OTHER;</td>
</tr>
<tr>
<td>param</td>
<td>scheduling parameters address.</td>
</tr>
</tbody>
</table>

Returns

0 on success;
an error number if:

- ESRCH, pid is invalid;
- EINVAL, policy or param->sched_priority is invalid;
- EAGAIN, insufficient memory available from the system heap, increase CONFIG_XENO_O→PT_SYS_HEAP_SZ;
-EFAULT, param is an invalid address;
See also

    Specification.

Note

See sched_setscheduler_ex().

Tags

    thread-unrestricted, switch-secondary, switch-primary

References sched_setscheduler_ex().

Referenced by sched_get_priority_max().

6.58.2.10 sched_setscheduler_ex()

int sched_setscheduler_ex (  
    pid_t pid,  
    int policy,  
    const struct sched_param_ex * param_ex )

Set extended scheduling policy of a process.

This service is an extended version of the sched_setscheduler() service, which supports Cobalt-specific and/or additional scheduling policies, not available with the host Linux environment. It sets the scheduling policy of the Cobalt process/thread identified by pid to the value policy, and the scheduling parameters (e.g. its priority) to the value pointed to by par.

If the current Linux thread ID or zero is passed (see gettid(2)), this service may turn the current regular POSIX thread into a Cobalt thread.

Parameters

<table>
<thead>
<tr>
<th>pid</th>
<th>target process/thread. If zero, the current thread is assumed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>policy</td>
<td>scheduling policy, one of SCHED_WEAK, SCHED_FIFO, SCHED_COBALT, SCHED_RR, SCHED_SPORADIC, SCHED_TP, SCHED_QUOTA or SCHED_NORMAL;</td>
</tr>
<tr>
<td>param_ex</td>
<td>address of scheduling parameters. As a special exception, a negative sched_priority value is interpreted as if SCHED_WEAK was given in policy, using the absolute value of this parameter as the weak priority level.</td>
</tr>
</tbody>
</table>

When CONFIG_XENO_OPT_SCHED_WEAK is enabled, SCHED_WEAK exhibits priority levels in the [0..99] range (inclusive). Otherwise, sched_priority must be zero for the SCHED_WEAK policy.
6.58 Process scheduling

Returns

- 0 on success;
- an error number if:
  - ESRCH, \( \text{pid} \) is not found;
  - EINVAL, \( \text{pid} \) is negative, \( \text{param}_\text{ex} \) is NULL, any of \( \text{policy} \) or \( \text{param}_\text{ex} \rightarrow \text{sched_priority} \) is invalid;
  - EAGAIN, insufficient memory available from the system heap, increase CONFIG\_XENO\_O→PT\_SYS\_HEAPSZ;
  - EFAULT, \( \text{param}_\text{ex} \) is an invalid address;

Note

See sched_setscheduler().

Tags

- thread-unrestricted, switch-secondary, switch-primary

Referenced by sched_setscheduler().

6.58.2.11 sched_yield()

```c
int sched_yield ( )
```

Yield the processor.

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

Return values

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

See also

 Specification.

Tags

- thread-unrestricted, switch-primary

References XNRELAX, and XNWEAK.

Referenced by pthread_yield().
6.59 Semaphores

Cobalt/POSIX semaphore services.

Collaboration diagram for Semaphores:

```
POSIX interface ← Semaphores
```

Functions

- `int sem_init (sem_t *sem, int pshared, unsigned int value)`
  Initialize an unnamed semaphore.
- `int sem_destroy (sem_t *sem)`
  Destroy an unnamed semaphore.
- `int sem_post (sem_t *sem)`
  Post a semaphore.
- `int sem_trywait (sem_t *sem)`
  Attempt to decrement a semaphore.
- `int sem_wait (sem_t *sem)`
  Decrement a semaphore.
- `int sem_timedwait (sem_t *sem, const struct timespec *abs_timeout)`
  Attempt to decrement a semaphore with a time limit.
- `int sem_close (sem_t *sem)`
  Close a named semaphore.
- `int sem_unlink (const char *name)`
  Unlink a named semaphore.

6.59.1 Detailed Description

Cobalt/POSIX semaphore services.

Semaphores are counters for resources shared between threads. The basic operations on semaphores are: increment the counter atomically, and wait until the counter is non-null and decrement it atomically.

Semaphores have a maximum value past which they cannot be incremented. The macro `SEM_VALUE←E_MAX` is defined to be this maximum value.

6.59.2 Function Documentation
6.59.2.1 sem_close()

int sem_close ( 
    sem_t * sem )

Close a named semaphore.

This service closes the semaphore sem. The semaphore is destroyed only when unlinked with a call to the sem_unlink() service and when each call to sem_open() matches a call to this service.

When a semaphore is destroyed, the memory it used is returned to the system heap, so that further references to this semaphore are not guaranteed to fail, as is the case for unnamed semaphores.

This service fails if sem is an unnamed semaphore.

Parameters

- **sem** the semaphore to be closed.

Return values

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

See also

- Specification.

Tags

- thread-unrestricted, switch-secondary

6.59.2.2 sem_destroy()

int sem_destroy ( 
    sem_t * sem )

Destroy an unnamed semaphore.

This service destroys the semaphore sem. Threads currently blocked on sem are unblocked and the service they called return -1 with errno set to EINVAL. The semaphore is then considered invalid by all semaphore services (they all fail with errno set to EINVAL) except sem_init().

This service fails if sem is a named semaphore.
Parameters

<table>
<thead>
<tr>
<th>Param</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sem</code></td>
<td>the semaphore to be destroyed.</td>
</tr>
</tbody>
</table>

Return values

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>always</td>
<td>0 on success. If SEM_WARNDEL was mentioned in sem_init_np(), the semaphore is deleted as requested and a strictly positive value is returned to warn the caller if threads were pending on it, otherwise zero is returned. If SEM_NOBUSYDEL was mentioned in sem_init_np(), <code>sem_destroy()</code> may succeed only if no thread is waiting on the semaphore to delete, otherwise -EBUSY is returned.</td>
</tr>
<tr>
<td>-1</td>
<td>with <code>errno</code> set if:</td>
</tr>
<tr>
<td></td>
<td>• EINVAL, the semaphore <code>sem</code> is invalid or a named semaphore;</td>
</tr>
<tr>
<td></td>
<td>• EPERM, the semaphore <code>sem</code> is not process-shared and does not belong to the current process.</td>
</tr>
<tr>
<td></td>
<td>• EBUSY, a thread is currently waiting on the semaphore <code>sem</code> with SEM_NOBUSYDEL set.</td>
</tr>
</tbody>
</table>

See also

- [Specification](#).

Tags

- `thread-unrestricted`
### 6.59 Semaphores

#### 6.59.2.4 sem_post()

```c
int sem_post (    
    sem_t * sem )
```

Post a semaphore.

This service posts the semaphore `sem`.

If no thread is currently blocked on this semaphore, its count is incremented unless "pulse" mode is enabled for it (see `sem_init_np()`, `SEM_PULSE`). If a thread is blocked on the semaphore, the thread heading the wait queue is unblocked.

#### Parameters

| `sem` | the semaphore to be signaled. |

#### Return values

| 0 | on success; |

See also

- [Specification](#)

**Tags**

- thread-unrestricted

---

Return values

| 0 | on success, |

-1 with `errno` set if:

- **EBUSY**, the semaphore `sm` was already initialized;

- **EAGAIN**, insufficient memory available to initialize the semaphore, increase `CONFIG_XENO_OPT_SHARED_HEAPSZ` for a process-shared semaphore, or `CONFIG_XENO_OPT_PRIVATE_HEAPSZ` for a process-private semaphore.

- **EAGAIN**, no registry slot available, check/raise `CONFIG_XENO_OPT_REGISTRY_NRSLOTS`.

- **EINVAL**, the `value` argument exceeds `SEM_VALUE_MAX`.  

---

Generated by Doxygen
Return values

-1  with errno set if:
  - EINVAL, the specified semaphore is invalid or uninitialized;
  - EPERM, the semaphore \texttt{sm} is not process-shared and does not belong to the current process;
  - EAGAIN, the semaphore count is \texttt{SEM\_VALUE\_MAX}.

See also

\textit{Specification}.

Tags

\textit{unrestricted}

6.59.2.5  \texttt{sem\_timedwait()}

\texttt{int sem\_timedwait (}
\texttt{   \texttt{sem\_t} \ast \texttt{sem},}
\texttt{   \texttt{const} \texttt{struct} \texttt{timespec} \ast \texttt{abs\_timeout} )}

Attempt to decrement a semaphore with a time limit.

This service is equivalent to \texttt{sem\_wait()}, except that the caller is only blocked until the timeout \texttt{abs\_timeout} expires.

Parameters

<table>
<thead>
<tr>
<th>\textit{sem}</th>
<th>the semaphore to be decremented;</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{abs_timeout}</td>
<td>the timeout, expressed as an absolute value of the relevant clock for the semaphore, either \texttt{CLOCK_MONOTONIC} if \texttt{SEM_RAWCLOCK} was mentioned via \texttt{sem_init_np()}, or \texttt{CLOCK_REALTIME} otherwise.</td>
</tr>
</tbody>
</table>

Return values

| \texttt{0} | on success; |
Return values

-1 with `errno` set if:

- EPERM, the caller context is invalid;
- EINVAL, the semaphore is invalid or uninitialized;
- EINVAL, the specified timeout is invalid;
- EPERM, the semaphore `sm` is not process-shared and does not belong to the current process;
- EINTR, the caller was interrupted by a signal while blocked in this service;
- ETIMEDOUT, the semaphore could not be decremented and the specified timeout expired.

See also

`Specification`.

Tags

`xthread-only, switch-primary`

References `sem_trywait()`.

6.59.2.6 `sem_trywait()`

```c
int sem_trywait (sem_t *sem)
```

Attempt to decrement a semaphore.

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

Parameters

- `sem`: the semaphore to be decremented.

Return values

0 on success;
Return values

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

See also

*Specification.*

Tags

`xthread-only`

Referenced by sem_timedwait(), and sem_wait().

6.59.2.7 sem_unlink()

```c
int sem_unlink (const char *name)
```

Unlink a named semaphore.

This service unlinks the semaphore named `name`. This semaphore is not destroyed until all references obtained with `sem_open()` are closed by calling `sem_close()`. However, the unlinked semaphore may no longer be reached with the `sem_open()` service.

When a semaphore is destroyed, the memory it used is returned to the system heap, so that further references to this semaphore are not guaranteed to fail, as is the case for unnamed semaphores.

Parameters

- `name` the name of the semaphore to be unlinked.

Return values

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

See also

*Specification.*

Tags

thread-unrestricted, switch-secondary

6.59.2.8 sem_wait()

```c
int sem_wait (sem_t *sem)
```

Decrement a semaphore.

This service decrements the semaphore `sem` if it is currently if its value is greater than 0. If the semaphore's value is currently zero, the calling thread is suspended until the semaphore is posted, or a signal is delivered to the calling thread.

This service is a cancellation point for Cobalt threads (created with the `pthread_create()` service). When such a thread is cancelled while blocked in a call to this service, the semaphore state is left unchanged before the cancellation cleanup handlers are called.

Parameters

- `sem`: the semaphore to be decremented.

Return values

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

- EPERM, the caller context is invalid;
- EINVAL, the semaphore is invalid or uninitialized;
- EPERM, the semaphore `sem` is not process-shared and does not belong to the current process;
- EINTR, the caller was interrupted by a signal while blocked in this service.

See also

*Specification.*

Tags

xthread-only, switch-primary

References `sem_trywait()`.

Generated by Doxygen
6.60 Thread management

Cobalt (POSIX) thread management services.

Collaboration diagram for Thread management:

```
POSIX interface -- Thread management
```

Functions

- `int pthread_create (pthread_t *ptid_r, const pthread_attr_t *attr, void *(*start)(void *), void *arg)`
  
  Create a new thread.

- `int pthread_setmode_np (int clrmask, int setmask, int *mode_r)`
  
  Set the mode of the current thread.

- `int pthread_setname_np (pthread_t thread, const char *name)`
  
  Set a thread name.

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

- `int pthread_join (pthread_t thread, void **retval)`
  
  Wait for termination of a specified thread.

6.60.1 Detailed Description

Cobalt (POSIX) thread management services.

See also

- Specification.

6.60.2 Function Documentation

6.60.2.1 pthread_create()

```
int pthread_create (
    pthread_t *ptid_r,
    const pthread_attr_t *attr,
    void *(*start)(void *),
    void *arg )
```

Create a new thread.

This service creates a thread managed by the Cobalt core in a dual kernel configuration.

Attributes of the new thread depend on the `attr` argument. If `attr` is NULL, default values for these attributes are used.

Returning from the `start` routine has the same effect as calling `pthread_exit()` with the return value.
Parameters

| ptid→_r  | address where the identifier of the new thread will be stored on success; |
| attr     | thread attributes; |
| start    | thread start routine; |
| arg      | opaque user-supplied argument passed to start; |

Returns

0 on success;
an error number if:

- EINVAL, attr is invalid;
- EAGAIN, insufficient memory available from the system heap to create a new thread, increase CONFIG_XENO_OPT_SYS_HEAPSZ;
- EINVAL, thread attribute inheritsched is set to PTHREAD_INHERIT_SCHED and the calling thread does not belong to the Cobalt interface;

See also

Specification.

Note

When creating a Cobalt thread for the first time, libcobalt installs an internal handler for the SIGSHADOW signal. If you had previously installed a handler for such signal before that point, such handler will be exclusively called for any SIGSHADOW occurrence Xenomai did not send.

If, however, an application-defined handler for SIGSHADOW is installed afterwards, overriding the libcobalt handler, the new handler is required to call cobalt_sigshadow_handler() on entry. This routine returns a non-zero value for every occurrence of SIGSHADOW issued by the Cobalt core. If zero instead, the application-defined handler should process the signal.

```c
int cobalt_sigshadow_handler(int sig, siginfo_t *si, void *ctxt);
```

You should register your handler with sigaction(2), setting the SA_SIGINFO flag.

Tags

thread-unrestricted, switch-secondary
6.60.2.2  `pthread_join()`

```c
int pthread_join (  
    pthread_t thread,  
    void **retval)
```

Wait for termination of a specified thread.

If `thread` is running and joinable, this service blocks the caller until `thread` terminates or detaches. When `thread` terminates, the caller is unblocked and its return value is stored at the address `value_ptr`.

On the other hand, if `thread` has already finished execution, its return value collected earlier is stored at the address `value_ptr` and this service returns immediately.

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

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

Parameters

<table>
<thead>
<tr>
<th>thread</th>
<th>identifier of the thread to wait for;</th>
</tr>
</thead>
<tbody>
<tr>
<td>retval</td>
<td>address where the target thread return value will be stored on success.</td>
</tr>
</tbody>
</table>

Returns

0 on success;  
an error number if:  
- ESRCH, `thread` is invalid;  
- EDEADLK, attempting to join the calling thread;  
- EINVAL, `thread` is detached;  
- EPERM, the caller context is invalid.

See also  
- Specification.

Tags  
- xthread-only, switch-secondary, switch-primary

References `pthread_setschedparam()`.

6.60.2.3  `pthread_kill()`

```c
int pthread_kill (  
    pthread_t thread,  
    int sig)
```

Send a signal to a thread.

This service send the signal `sig` to the Cobalt thread `thread` (created with `pthread_create()`). If `sig` is zero, this service check for existence of the thread `thread`, but no signal is sent.
### 6.60 Thread management

**Parameters**

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

**Returns**

0 on success; an error number if:

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

**See also**

*Specification.*

**Tags**

- thread-unrestricted, switch-primary

---

#### 6.60.2.4 pthread_setmode_np()

```c
int pthread_setmode_np (  
    int clrmask,  
    int setmask,  
    int *mode_r  
)
```

Set the mode of the current thread.

This service sets the mode of the calling thread, which affects its behavior under particular circumstances. `clrmask` and `setmask` are two masks of mode bits which are respectively cleared and set by `pthread_setmode_np()`:

- PTHREAD_LOCK_SCHED, when set, locks the scheduler, which prevents the current thread from being switched out until the scheduler is unlocked. Unless PTHREAD_DISABLE_LOCKBREAK is also set, the thread may still block, dropping the lock temporarily, in which case, the lock will be reacquired automatically when the thread resumes execution. When PTHREAD_LOCK_SCHED is cleared, the current thread drops the scheduler lock, and the rescheduling procedure is initiated.

- When set, PTHREAD_WARNNSW enables debugging notifications for the current thread. A `SIGDEBUG` (Linux-originated) signal is sent when the following atypical or abnormal behavior is detected:
  - the current thread switches to secondary mode. Such notification comes in handy for detecting spurious relaxes, with one of the following reason codes:
    - SIGDEBUG_MIGRATE_SYSCALL, if the thread issued a regular Linux system call.
    - SIGDEBUG_MIGRATE_SIGNAL, if the thread had to leave real-time mode for handling a Linux signal.
* SIGDEBUG_MIGRATE_FAULT, if the thread had to leave real-time mode for handling a processor fault/exception.

– the current thread is sleeping on a Cobalt mutex currently owned by a thread running in secondary mode, which reveals a priority inversion. In such an event, the reason code passed to the signal handler will be SIGDEBUG_MIGRATE_PRIOINV.

– the current thread is about to sleep while holding a Cobalt mutex, and CONFIG_XENO__OPT_DEBUG_MUTEX_SLEEP is enabled in the kernel configuration. In such an event, the reason code passed to the signal handler will be SIGDEBUG_MUTEX_SLEEP. Blocking for acquiring a mutex does not trigger such signal though.

– the current thread has enabled PTHREAD_DISABLE_LOCKBREAK and PTHREAD_LOC→ K_SCHED, then attempts to block on a Cobalt service, which would cause a lock break. In such an event, the reason code passed to the signal handler will be SIGDEBUG_LOCK_BREAK.

• PTHREAD_DISABLE_LOCKBREAK disallows breaking the scheduler lock. Normally, the scheduler lock is dropped implicitly when the current owner blocks, then reacquired automatically when the owner resumes execution. If PTHREAD_DISABLE_LOCKBREAK is set, the scheduler lock owner would return with EINTR immediately from any blocking call instead (see PTHREAD_WA→ RNSW notifications).

• PTHREAD_CONFORMING can be passed in setmask to switch the current Cobalt thread to its preferred runtime mode. The only meaningful use of this switch is to force a real-time thread back to primary mode eagerly. Other usages have no effect.

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

Parameters

<table>
<thead>
<tr>
<th>clrmask</th>
<th>set of bits to be cleared.</th>
</tr>
</thead>
<tbody>
<tr>
<td>setmask</td>
<td>set of bits to be set.</td>
</tr>
<tr>
<td>mode→_r</td>
<td>If non-NULL, mode_r must be a pointer to a memory location which will be written upon success with the previous set of active mode bits. If NULL, the previous set of active mode bits will not be returned.</td>
</tr>
</tbody>
</table>

Returns

0 on success, otherwise:

• EINVAL, some bit in clrmask or setmask is invalid.

Note

Setting clrmask and setmask to zero leads to a nop, only returning the previous mode if mode_r is a valid address.

Attention

Issuing PTHREAD_CONFORMING is most likely useless or even introduces pure overhead in regular applications, since the Cobalt core performs the necessary mode switches, only when required.

Tags

xthread-only, switch-primary
6.60.2.5  pthread_setname_np()

int pthread_setname_np (  
    pthread_t thread,  
    const char * name  
)

Set a thread name.

This service set to name, the name of thread. This name is used for displaying information in /proc/xenomai/sched.

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

Parameters

| thread | target thread; |
| name   | name of the thread. |

Returns

0 on success;
an error number if:
  • ESRCH, thread is invalid.

Tags

xthread-only
6.61 Scheduling management

Cobalt scheduling management services.

Collaboration diagram for Scheduling management:

| POSIX interface | Scheduling management |

Functions

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

- `int pthread_setschedparam_ex (pthread_t thread, int policy, const struct sched_param_ex *param_ex)`
  Set extended scheduling policy of thread.

- `int pthread_getschedparam (pthread_t thread, int *__restrict__ policy, struct sched_param *__restrict__ param)`
  Get the scheduling policy and parameters of the specified thread.

- `int pthread_getschedparam_ex (pthread_t thread, int *__restrict__ policy_r, struct sched_param__ex __restrict__ param_ex)`
  Get extended scheduling policy of thread.

- `int pthread_yield (void)`
  Yield the processor.

6.61.1 Detailed Description

Cobalt scheduling management services.

6.61.2 Function Documentation

6.61.2.1 pthread_getschedparam()

```c
int pthread_getschedparam (  
    pthread_t thread,  
    int *__restrict__ policy,  
    struct sched_param *__restrict__ param )
```

Get the scheduling policy and parameters of the specified thread.

This service returns, at the addresses policy and par, the current scheduling policy and scheduling parameters (i.e. priority) of the Cobalt thread tid. If thread is not the identifier of a Cobalt thread, this service fallback to the regular POSIX pthread_getschedparam() service.
6.61 Scheduling management

Parameters

| thread | target thread; |
| policy | address where the scheduling policy of tid is stored on success; |
| param | address where the scheduling parameters of tid is stored on success. |

Returns

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

See also

Specification.

Tags

thread-unrestricted

References pthread_getschedparam_ex().

Referenced by pthread_getschedparam_ex().

6.61.2.2 pthread_getschedparam_ex()

int pthread_getschedparam_ex (  
    pthread_t thread,  
    int *__restrict__ policy_r,  
    struct sched_param_ex *__restrict__ param_ex )

Get extended scheduling policy of thread.

This service is an extended version of the regular pthread_getschedparam() service, which also supports Cobalt-specific policies, not available with the host Linux environment.

Parameters

| thread | target thread; |
| policy_r | address where the scheduling policy of thread is stored on success; |
| param_ex | address where the scheduling parameters of thread are stored on success. |

Returns

0 on success;
an error number if:
- ESRCH, thread is invalid.
See also

* Specification.

Tags

thread-unrestricted

References pthread_getschedparam().

Referenced by pthread_getschedparam().

6.61.2.3 pthread_setschedparam()

```c
int pthread_setschedparam (  
    pthread_t thread,  
    int policy,  
    const struct sched_param *param )
```

Set the scheduling policy and parameters of the specified thread.

This service set the scheduling policy of the Cobalt thread identified by `pid` to the value `policy`, and its scheduling parameters (i.e. its priority) to the value pointed to by `param`.

If pthread_self() is passed, this service turns the current thread into a Cobalt thread. If `thread` is not the identifier of a Cobalt thread, this service falls back to the regular pthread_setschedparam() service.

Parameters

<table>
<thead>
<tr>
<th>Param</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>thread</td>
<td>target Cobalt thread;</td>
</tr>
<tr>
<td>policy</td>
<td>scheduling policy, one of SCHED_FIFO, SCHED_RR, or SCHED_OTHER;</td>
</tr>
<tr>
<td>param</td>
<td>address of scheduling parameters.</td>
</tr>
</tbody>
</table>

Returns

0 on success;
an error number if:

- ESRCH, `pid` is invalid;
-EINVAL, `policy` or `param`->`sched_priority` is invalid;
- EAGAIN, insufficient memory available from the system heap, increase CONFIG_XENO_O←PT_SYS_HEAPSZ;
-EFAULT, `param` is an invalid address;

See also

* Specification.
6.61 Scheduling management

Note

See `pthread_create()`, `pthread_setschedparam_ex()`.

Tags

thread-unrestricted, switch-secondary, switch-primary

References `pthread_setschedparam_ex()`.

Referenced by `pthread_join()`.

6.61.2.4 `pthread_setschedparam_ex()`

```c
int pthread_setschedparam_ex (
    pthread_t thread,
    int policy,
    const struct sched_param_ex * param_ex)
```

Set extended scheduling policy of thread.

This service is an extended version of the regular `pthread_setschedparam()` service, which supports Cobalt-specific scheduling policies, not available with the host Linux environment.

This service set the scheduling policy of the Cobalt thread `thread` to the value `policy`, and its scheduling parameters (e.g. its priority) to the value pointed to by `param_ex`.

If `thread` does not match the identifier of a Cobalt thread, this action falls back to the regular `pthread_setschedparam()` service.

Parameters

<table>
<thead>
<tr>
<th><code>thread</code></th>
<th>target Cobalt thread:</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>policy</code></td>
<td>scheduling policy, one of SCHED_WEAK, SCHED_FIFO, SCHED_COBALT, SCHED_RR, SCHED_SPORADIC, SCHED_TP, SCHED_QUOTA or SCHED_NORMAL;</td>
</tr>
<tr>
<td><code>param_ex</code></td>
<td>scheduling parameters address. As a special exception, a negative <code>sched_priority</code> value is interpreted as if SCHED_WEAK was given in <code>policy</code>, using the absolute value of this parameter as the weak priority level.</td>
</tr>
</tbody>
</table>

When `CONFIG_XENO_OPT_SCHED_WEAK` is enabled, SCHED_WEAK exhibits priority levels in the [0..99] range (inclusive). Otherwise, sched_priority must be zero for the SCHED_WEAK policy.

Returns

0 on success;
an error number if:

- ESRCH, `thread` is invalid;
• EINVAL, policy or param_ex->sched_priority is invalid;
• EAGAIN, insufficient memory available from the system heap, increase CONFIG_XENO_O->
  PT_SYS_HEAPSZ;
•EFAULT, param_ex is an invalid address;
• EPERM, the calling process does not have superuser permissions.

See also

  Specification.

Note

See pthread_create(), pthread_setschedparam().

Tags

  thread-unrestricted, switch-secondary, switch-primary

Referenced by pthread_setschedparam().

6.61.2.5  pthread_yield()

int pthread_yield ( 
  void )

Yield the processor.

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

Return values

  0

See also

  Specification.

Tags

  thread-unrestricted, switch-primary

References sched_yield().
6.62 Smokey API

A simple infrastructure for writing and running smoke tests.

Smokey is based on the Copperplate API, therefore is available over the single and dual kernel Xenomai configurations indifferently.

The API provides a set of services for declaring any number of test plugins, embodied into a test program. Each plugin usually implements a single smoke test, checking a particular feature of interest. Each plugin present in the running executable is automatically detected by the Smokey init routine. In addition, the Smokey API parses all arguments and options passed on the command line to the executable, running pre-defined actions which are therefore automatically recognized by all programs linked against the Smokey library.

Writing smoke tests with Smokey

A smoke test is composed of a routine which implements the test code, and a set of runtime settings/attributes for running such code. The routine prototype shall be:

```c
int run_<test_name>(struct smokey_test *t, int argc, char *const argv[])
```

The test routine should return a zero value for success, or any negated POSIX error code for indicating the failure to the test driver (e.g. -EINVAL if some value is found to be wrong).

With \( t \) referring to the Smokey test descriptor, and \( argc, argv \) the argument count and vector expunged from all the inner options which may have been previously interpreted by the Smokey API and inner layers (such as Copperplate).

The Smokey API provides the services to declare a complete test (named \( foo \) in this example) as follows:

```c
#include <smokey/smokey.h>

smokey_test_plugin(foo, // test name
    SMOKEY_ARGLIST( // argument list
        SMOKEY_INT(some_integer),
        SMOKEY_STRING(some_string),
        SMOKEY_BOOL(some_boolean),
    ),
    // description
    "A dummy Smokey-based test plugin\n    \n    \tAccepting three optional arguments:\n    \t\tsome_integer=<value>\n    \t\tsome_string=<string>\n    \t\tsome_bool[=0/1]\n"
);  

static int run_foo(struct smokey_test *t, int argc, char *const argv[])
{
    int i_arg = 0, nargs;
    char *s_arg = NULL;
    bool b_arg = false;
    
    nargs = smokey_parse_args(t, argc, argv);
    
    if (SMOKEY_ARG_ISSET(foo, some_integer))
        i_arg = SMOKEY_ARG_INT(foo, some_integer);
    if (SMOKEY_ARG_ISSET(foo, some_string))
        s_arg = SMOKEY_ARG_STRING(foo, some_string);
    if (SMOKEY_ARG_ISSET(foo, some_boolean))
        b_arg = SMOKEY_ARG_INT(foo, some_boolean);
    
    return run_some_hypothetical_smoke_test_code(i_arg, s_arg, b_arg);
}
```

Generated by Doxygen
As illustrated, a smoke test is at least composed of a test plugin descriptor (i.e. smokey_test_plugin()), and a run handler named after the test.

Test arguments

Smokey recognizes three argument declarators, namely: SMOKEY_INT(name) for a C (signed) integer, SMOKEY_BOOL(name) for a boolean value and SMOKEY_STRING(name) for a character string.

Each argument can be passed to the test code as a name=value pair, where name should match one of the declarators. Before the test-specific arguments can be accessed, a call to smokey_parse_args() must be issued by the test code, passing the parameters received in the run handler. This routine returns the number of arguments found on the command line matching the an entry in SMOKEY_ARGLIST().

Once smokey_parse_args() has returned with a non-zero value, each argument can be checked individually for presence. If a valid argument was matched on the command line, SMOKEY_ARG_ISS_TRUE(test_name, arg_name) returns non-zero. In the latter case, its value can be retrieved by a similar call to SMOKEY_ARG_INT(test_name, arg_name), SMOKEY_ARG_STRING(test_name, arg_name) or SMOKEY_ARG_BOOL(test_name, arg_name).

In the above example, passing "some_integer=3" on the command line of any program implementing such Smokey-based test would cause the variable i_arg to receive "3" as a value.

Pre-defined Smokey options

Any program linked against the Smokey API implicitly recognizes the following options:

- –list dumps the list of tests implemented in the program to stdout. The information given includes the description strings provided in the plugin declarators (smokey_test_plugin()). The position and symbolic name of each test is also issued, which may be used in id specifications with the –run option (see below).

Note

Test positions may vary depending on changes to the host program like adding or removing other tests, the symbolic name however is stable and identifies each test uniquely.

- –run[=<id[id...]>] selects the tests to be run, determining the active test list among the overall set of tests detected in the host program. The test driver code (e.g. implementing a test harness program on top of Smokey) may then iterate over the smokey_test_list for accessing each active test individually, in the enumeration order specified by the user (Use for_each_smokey_test() for that).

If no argument is passed to –run, Smokey assumes that all tests detected in the current program should be picked, filling smokey_test_list with tests by increasing position order.

Otherwise, id may be a test position, a symbolic name, or a range thereof delimited by a dash character. A symbolic name may be matched using a glob(3) type regular expression.

id specification may be:
- 0-9, picks tests #0 to #9
- -3, picks tests #0 to #3
– 5-, picks tests #5 to the highest possible test position
– 2-0, picks tests #2 to #0, in decreasing order
– foo, picks test foo only
– 0,1,foo- picks tests #0, #1, and any test from foo up to the last test defined
– fo* picks any test with a name starting by "fo"

• –exclude=<id[,id...]>) excludes the given tests from the test list. The format of the argument is identical to the one accepted by the –run option.

• –keep-going sets the boolean flag smokey_keep_going to a non-zero value, indicating to the test driver that receiving a failure code from a smoke test should not abort the test loop. This flag is not otherwise interpreted by the Smokey API.

• –verbose[level] sets the integer smokey_verbose_mode to a non-zero value, which should be interpreted by all parties as the desired verbosity level (defaults to 1).

• –vm gives a hint to the test code, about running in a virtual environment, such as KVM. When passed, the boolean smokey_on_vm is set. Each test may act upon this setting, such as skipping time-dependent checks that may fail due to any slowdown induced by the virtualization.

Writing a test driver based on the Smokey API

A test driver provides the main() entry point, which should iterate over the test list (smokey_test_list) prepared by the Smokey API, for running each test individually. The for_each_smokey_test() helper is available for iterating over the active test list.

When this entry point is called, all the initialization chores, including the test detection and the active test selection have been performed by the Smokey API already.

Issuing information notices

The printf-like smokey_note() routine is available for issuing notices to the output device (currently std-out), unless –silent was detected on the command line. smokey_note() outputs a terminating newline character. Notes are enabled for any verbosity level greater than zero.

Issuing trace messages

The printf-like smokey_trace() routine is available for issuing progress messages to the output device (currently std-out), unless –silent was detected on the command line. smokey_trace() outputs a terminating newline character. Traces are enabled for any verbosity level greater than one.

Therefore, a possible implementation of a test driver could be as basic as:
#include <stdio.h>
#include <error.h>
#include <smokey/smokey.h>

int main(int argc, char *const argv[]) {
    struct smokey_test *t;
    int ret;

    if (pvlist_empty(&smokey_test_list))
        return 0;

    for_each_smokey_test(t) {
        ret = t->run(t, argc, argv);
        if (ret) {
            if (smokey_keep_going)
                continue;
            error(1, -ret, "test %s failed", t->name);
        }
        smokey_note("%s OK", t->name);
    }

    return 0;
}
6.63 Asynchronous acquisition API

Collaboration diagram for Asynchronous acquisition API:

![Collaboration diagram]

Data Structures

- struct `a4l_cmd_desc`
  
  *Structure describing the asynchronous instruction.*

Functions

- `int a4l_snd_command (a4l_desc_t *dsc, a4l_cmd_t *cmd)`
  *Send a command to an Analoy device.*
- `int a4l_snd_cancel (a4l_desc_t *dsc, unsigned int idx_subd)`
  *Cancel an asynchronous acquisition.*
- `int a4l_set_bufsize (a4l_desc_t *dsc, unsigned int idx_subd, unsigned long size)`
  *Change the size of the asynchronous buffer.*
- `int a4l_get_bufsize (a4l_desc_t *dsc, unsigned int idx_subd, unsigned long *size)`
  *Get the size of the asynchronous buffer.*
- `int a4l_mark_bufrw (a4l_desc_t *dsc, unsigned int idx_subd, unsigned long cur, unsigned long *new)`
  *Update the asynchronous buffer state.*
- `int a4l_poll (a4l_desc_t *dsc, unsigned int idx_subd, unsigned long ms_timeout)`
  *Get the available data count.*
- `int a4l_mmap (a4l_desc_t *dsc, unsigned int idx_subd, unsigned long size, void **ptr)`
  *Map the asynchronous ring-buffer into a user-space.*

ANALOGY_CMD_xxx

Common command flags definitions

- `#define A4L_CMD_SIMUL 0x1`
  *Do not execute the command, just check it.*
- `#define A4L_CMD_BULK 0x2`
  *Perform data recovery / transmission in bulk mode.*
- `#define A4L_CMD_WRITE 0x4`
  *Perform a command which will write data to the device.*
TRIG_xxx

Command triggers flags definitions

- `#define TRIG_NONE 0x00000001`  
  Never trigger.
- `#define TRIG_NOW 0x00000002`  
  Trigger now + N ns.
- `#define TRIG_FOLLOW 0x00000004`  
  Trigger on next lower level trig.
- `#define TRIG_TIME 0x00000008`  
  Trigger at time N ns.
- `#define TRIG_TIMER 0x00000010`  
  Trigger at rate N ns.
- `#define TRIG_COUNT 0x00000020`  
  Trigger when count reaches N.
- `#define TRIG_EXT 0x00000040`  
  Trigger on external signal N.
- `#define TRIG_INT 0x00000080`  
  Trigger on analogy-internal signal N.
- `#define TRIG_OTHER 0x00000100`  
  Driver defined trigger.
- `#define TRIG_WAKE_EOS 0x0020`  
  Wake up on end-of-scan.
- `#define TRIG_ROUND_MASK 0x00030000`  
  Trigger not implemented yet.
- `#define TRIG_ROUND_NEAREST 0x00000000`  
  Trigger not implemented yet.
- `#define TRIG_ROUND_DOWN 0x00010000`  
  Trigger not implemented yet.
- `#define TRIG_ROUND_UP 0x00020000`  
  Trigger not implemented yet.
- `#define TRIG_ROUND_UP_NEXT 0x00030000`  
  Trigger not implemented yet.

Channel macros

Specific precompilation macros and constants useful for the channels descriptors tab located in the command structure

- `#define CHAN(a) ((a) & 0xffff)`  
  Channel indication macro.
- `#define RNG(a) (((a) & 0xff) << 16)`  
  Range definition macro.
- `#define AREF(a) (((a) & 0x03) << 24)`  
  Reference definition macro.
- `#define FLAGS(a) ((a) & CR_FLAGS_MASK)`  
  Flags definition macro.
6.63 Asynchronous acquisition API

- `#define PACK(a, b, c) (a | RNG(b) | AREF(c))`
  Channel + range + reference definition macro.
- `#define PACK_FLAGS(a, b, c, d) (PACK(a, b, c) | FLAGS(d))`
  Channel + range + reference + flags definition macro.
- `#define AREF_GROUND 0x00`
  Analog reference is analog ground.
- `#define AREF_COMMON 0x01`
  Analog reference is analog common.
- `#define AREF_DIFF 0x02`
  Analog reference is differential.
- `#define AREF_OTHER 0x03`
  Analog reference is undefined.

6.63.1 Detailed Description

6.63.2 Function Documentation

6.63.2.1 a4l_get_bufsize()

```c
int a4l_get_bufsize (  
a4l_desc_t * dsc,  
unsigned int idx_subd,  
unsigned long * size )
```

Get the size of the asynchronous buffer.

During asynchronous acquisition, a ring-buffer enables the transfers from / to user-space. Functions like `a4l_read()` or `a4l_write()` recovers / sends data through this intermediate buffer. Please note, there is one ring-buffer per subdevice capable of asynchronous acquisition. By default, each buffer size is set to 64 KB.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th><code>dsc</code></th>
<th>Device descriptor filled by <code>a4l_open()</code> (and optionally <code>a4l_fill_desc()</code>)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>idx_subd</code></td>
<td>Index of the concerned subdevice</td>
</tr>
<tr>
<td>out</td>
<td><code>size</code></td>
<td>Buffer size</td>
</tr>
</tbody>
</table>

Returns

- 0 on success. Otherwise:
  - `-EINVAL` is returned if some argument is missing or wrong (Please, type "dmesg" for more info)
  - `-EFAULT` is returned if a user <-> kernel transfer went wrong

References `a4l_descriptor.fd`.
6.63.2.2 a4l_mark_bufrw()

```c
int a4l_mark_bufrw (  
    a4l_desc_t * dsc,  
    unsigned int idx_subd,  
    unsigned long cur,  
    unsigned long * new  )
```

Update the asynchronous buffer state.

When the mapping of the asynchronous ring-buffer (thanks to a4l_mmap()) is disabled, common read / write syscalls have to be used. In input case, a4l_read() must be used for:

- the retrieval of the acquired data.
- the notification to the Analogy layer that the acquired data have been consumed, then the area in the ring-buffer which was containing becomes available. In output case, a4l_write() must be called to:
  - send some data to the Analogy layer.
  - signal the Analogy layer that a chunk of data in the ring-buffer must be used by the driver.

In mmap configuration, these features are provided by unique function named a4l_mark_bufrw(). In input case, a4l_mark_bufrw() can:

- recover the count of data newly available in the ring-buffer.
- notify the Analogy layer how many bytes have been consumed. In output case, a4l_mark_bufrw() can:
  - recover the count of data available for writing.
  - notify Analogy that some bytes have been written.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>dsc</th>
<th>Device descriptor filled by a4l_open() (and optionally a4l_fill_desc())</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>idx_subd</td>
<td>Index of the concerned subdevice</td>
</tr>
<tr>
<td>in</td>
<td>cur</td>
<td>Amount of consumed data</td>
</tr>
<tr>
<td>out</td>
<td>new</td>
<td>Amount of available data</td>
</tr>
</tbody>
</table>

**Returns**

0 on success. Otherwise:

- EINVAL is returned if some argument is missing or wrong; the descriptor and the new pointer should be checked; check also the kernel log ("dmesg")
-EFAULT is returned if a user <-> kernel transfer went wrong

References a4l_descriptor::fd.
6.63.2.3  a4l_mmap()

```c
int a4l_mmap (  
a4l_desc_t * dsc,  
    unsigned int idx_subd,  
    unsigned long size,  
    void ** ptr )
```

Map the asynchronous ring-buffer into a user-space.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>dsc</th>
<th>Device descriptor filled by <code>a4l_open()</code> (and optionally <code>a4l_fill_desc()</code>)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>idx_subd</td>
<td>Index of the concerned subdevice</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>Size of the buffer to map</td>
</tr>
<tr>
<td>out</td>
<td>ptr</td>
<td>Address of the pointer containing the assigned address on return</td>
</tr>
</tbody>
</table>

Returns

0 on success. Otherwise:

- -EINVAL is returned if some argument is missing or wrong, the descriptor and the pointer should be checked; check also the kernel log
- -EPERM is returned if the function is called in an RT context or if the buffer to resize is mapped in user-space (Please, type "dmesg" for more info)
- -EFAULT is returned if a user <-> kernel transfer went wrong
- -EBUSY is returned if the buffer is already mapped in user-space

References `a4l_descriptor::fd`.

6.63.2.4  a4l_poll()

```c
int a4l_poll (  
a4l_desc_t * dsc,  
    unsigned int idx_subd,  
    unsigned long ms_timeout )
```

Get the available data count.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>dsc</th>
<th>Device descriptor filled by <code>a4l_open()</code> (and optionally <code>a4l_fill_desc()</code>)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>idx_subd</td>
<td>Index of the concerned subdevice</td>
</tr>
<tr>
<td>in</td>
<td>ms_timeout</td>
<td>The number of miliseconds to wait for some data to be available. Passing A4L_INFINITE causes the caller to block indefinitely until some data is available. Passing A4L_NONBLOCK causes the function to return immediately without waiting for any available data</td>
</tr>
</tbody>
</table>

Generated by Doxygen
Returns
the available data count. Otherwise:

- -EINVAL is returned if some argument is missing or wrong (Please, type "dmesg" for more info)
- -EFAULT is returned if a user <-> kernel transfer went wrong
- -EINTR is returned if calling task has been unblocked by a signal

References a4l_descriptor::fd.

6.63.2.5  a4l_set_bufsize()

int a4l_set_bufsize (  
a4l_desc_t * dsc,  
unsigned int idx_subd,  
unsigned long size )

Change the size of the asynchronous buffer.

During asynchronous acquisition, a ring-buffer enables the transfers from / to user-space. Functions like a4l_read() or a4l_write() recovers / sends data through this intermediate buffer. The function a4l_set_bufsize() can change the size of the ring-buffer. Please note, there is one ring-buffer per subdevice capable of asynchronous acquisition. By default, each buffer size is set to 64 KB.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>dsc</th>
<th>Device descriptor filled by a4l_open() (and optionally a4l_fill_desc())</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>idx_subd</td>
<td>Index of the concerned subdevice</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>New buffer size, the maximal tolerated value is 16MB (A4L_BUF_MAXSIZE)</td>
</tr>
</tbody>
</table>

Returns
0 on success. Otherwise:

- -EINVAL is returned if the analogy descriptor is not correct or if some argument is missing or wrong (Please, type "dmesg" for more info)
- -EPERM is returned if the function is called in an RT context or if the buffer to resize is mapped in user-space (Please, type "dmesg" for more info)
- -EFAULT is returned if a user <-> kernel transfer went wrong
- -EBUSY is returned if the selected subdevice is already processing an asynchronous operation
- -ENOMEM is returned if the system is out of memory

References a4l_sys_bufcfg(), and a4l_descriptor::fd.
6.63.2.6  a4l_snd_cancel()

```c
int a4l_snd_cancel (a4l_desc_t * dsc,
                    unsigned int idx_subd )
```

Cancel an asynchronous acquisition.

The function `a4l_snd_cancel()` is devoted to stop an asynchronous acquisition configured thanks to an Analogy command.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>dsc</th>
<th>Device descriptor filled by <code>a4l_open()</code> (and optionally <code>a4l_fill_desc()</code>)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>idx_subd</td>
<td>Subdevice index</td>
</tr>
</tbody>
</table>

**Returns**

- 0 on success. Otherwise:

  - -EINVAL is returned if some argument is missing or wrong (Please, type “dmesg” for more info)
  - -EIO is returned if the selected subdevice does not support asynchronous operation

References `a4l_descriptor::fd`.

6.63.2.7  a4l_snd_command()

```c
int a4l_snd_command (a4l_desc_t * dsc, a4l_cmd_t * cmd )
```

Send a command to an Analogy device.

The function `a4l_snd_command()` triggers asynchronous acquisition.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>dsc</th>
<th>Device descriptor filled by <code>a4l_open()</code> (and optionally <code>a4l_fill_desc()</code>)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>cmd</td>
<td>Command structure</td>
</tr>
</tbody>
</table>

**Returns**

- 0 on success. Otherwise:

  - -EINVAL is returned if some argument is missing or wrong (Please, type “dmesg” for more info)
  - -ENOMEM is returned if the system is out of memory
-EFAULT is returned if a user <-> kernel transfer went wrong
- EIO is returned if the selected subdevice cannot handle command
- EBUSY is returned if the selected subdevice is already processing an asynchronous operation

References a4l_descriptor::fd.
6.64 Asynchronous acquisition API

Collaboration diagram for Asynchronous acquisition API:

![Collaboration diagram for Asynchronous acquisition API](image)

Functions

- int `a4l_async_read (a4l_desc_t *dsc, void *buf, size_t nbyte, unsigned long ms_timeout)`
  
  Perform asynchronous read operation on the analog input subdevice.

- int `a4l_async_write (a4l_desc_t *dsc, void *buf, size_t nbyte, unsigned long ms_timeout)`
  
  Perform asynchronous write operation on the analog input subdevice.

6.64.1 Detailed Description

6.64.2 Function Documentation

6.64.2.1 `a4l_async_read()`

```c
int a4l_async_read (  
        a4l_desc_t * dsc,  
        void * buf,  
        size_t nbyte,  
        unsigned long ms_timeout )
```

Perform asynchronous read operation on the analog input subdevice.

The function `a4l_async_read()` is only useful for acquisition configured through an Analogy command.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in</code></td>
<td><code>dsc</code></td>
</tr>
<tr>
<td><code>out</code></td>
<td><code>buf</code></td>
</tr>
<tr>
<td><code>in</code></td>
<td><code>nbyte</code></td>
</tr>
<tr>
<td><code>in</code></td>
<td><code>ms_timeout</code></td>
</tr>
</tbody>
</table>
Returns

Number of bytes read, otherwise negative error code:

- **EINVAL** is returned if some argument is missing or wrong, the descriptor should be checked; check also the kernel log
- **ENOENT** is returned if the device's reading subdevice is idle (no command was sent)
- **EFAULT** is returned if a user <-> kernel transfer went wrong
- **EINTR** is returned if calling task has been unblocked by a signal

6.64.2.2  a4l_async_write()

```c
int a4l_async_write (  
    a4l_desc_t * dsc,  
    void * buf,  
    size_t nbyte,  
    unsigned long ms_timeout )
```

Perform asynchronous write operation on the analog input subdevice.

The function `a4l_async_write()` is only useful for acquisition configured through an Analogy command.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>dsc</th>
<th>Device descriptor filled by <code>a4l_open()</code> (and optionally <code>a4l_fill_desc()</code>)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>buf</td>
<td>Output buffer</td>
</tr>
<tr>
<td>in</td>
<td>nbyte</td>
<td>Number of bytes to write</td>
</tr>
<tr>
<td>in</td>
<td>ms_timeout</td>
<td>The number of miliseconds to wait for some free area to be available. Passing A4L_INFINITE causes the caller to block indefinitely until some data is available. Passing A4L_NONBLOCK causes the function to return immediately without waiting any available space to write data.</td>
</tr>
</tbody>
</table>

Returns

Number of bytes written, otherwise negative error code:

- **EINVAL** is returned if some argument is missing or wrong, the descriptor should be checked; check also the kernel log
- **ENOENT** is returned if the device's reading subdevice is idle (no command was sent)
- **EFAULT** is returned if a user <-> kernel transfer went wrong
- **EINTR** is returned if calling task has been unblocked by a signal
6.65 Software calibration API

Collaboration diagram for Software calibration API:

![Level 2 API](Software calibration API)

Functions

- `int a4l_read_calibration_file(char *name, struct a4l_calibration_data *data)`
  *Read the analogy generated calibration file.*

- `int a4l_get_softcal_converter(struct a4l_polynomial *converter, int subd, int chan, int range, struct a4l_calibration_data *data)`
  *Get the polynomial that will be use for the software calibration.*

- `int a4l_rawtodcal(a4l_chinfo_t *chan, double *dst, void *src, int cnt, struct a4l_polynomial *converter)`
  *Convert raw data (from the driver) to calibrated double units.*

- `int a4l_dcaltoraw(a4l_chinfo_t *chan, void *dst, double *src, int cnt, struct a4l_polynomial *converter)`
  *Convert double values to raw calibrated data using polynomials.*

6.65.1 Detailed Description

6.65.2 Function Documentation

6.65.2.1 a4l dcaltoraw()

```c
int a4l_dfaltoraw(
    a4l_chinfo_t * chan,
    void * dst,
    double * src,
    int cnt,
    struct a4l_polynomial * converter )
```

Convert double values to raw calibrated data using polynomials.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>chan</th>
<th>Channel descriptor</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>dst</td>
<td>Output buffer</td>
</tr>
<tr>
<td>in</td>
<td>src</td>
<td>Input buffer</td>
</tr>
<tr>
<td>in</td>
<td>cnt</td>
<td>Count of conversion to perform</td>
</tr>
<tr>
<td>in</td>
<td>converter</td>
<td>Conversion polynomial</td>
</tr>
</tbody>
</table>
Returns

the count of conversion performed, otherwise a negative error code:

- **EINVAL** is returned if some argument is missing or wrong; chan, rng and the pointers should be checked; check also the kernel log (“dmesg”); WARNING: **a4l_fill_desc()** should be called before using **a4l_dcalltoraw()**

6.65.2.2  a4l_get_softcal_converter()

```c
int a4l_get_softcal_converter (  
    struct a4l_polynomial * converter,  
    int subd,  
    int chan,  
    int range,  
    struct a4l_calibration_data * data )
```

Get the polynomial that will be use for the software calibration.

Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>converter</th>
<th>Polynomial to be used on the software calibration</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>subd</td>
<td>Subdevice index</td>
</tr>
<tr>
<td>in</td>
<td>chan</td>
<td>Channel</td>
</tr>
<tr>
<td>in</td>
<td>range</td>
<td>Range</td>
</tr>
<tr>
<td>in</td>
<td>data</td>
<td>Calibration data read from the calibration file</td>
</tr>
</tbody>
</table>

Returns

-1 on error

6.65.2.3  a4l_rawtodcal()

```c
int a4l_rawtodcal (  
    a4l_chinfo_t * chan,  
    double * dst,  
    void * src,  
    int cnt,  
    struct a4l_polynomial * converter )
```

Convert raw data (from the driver) to calibrated double units.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>chan</th>
<th>Channel descriptor</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>dst</td>
<td>Output buffer</td>
</tr>
<tr>
<td>in</td>
<td>src</td>
<td>Input buffer</td>
</tr>
<tr>
<td>in</td>
<td>cnt</td>
<td>Count of conversion to perform</td>
</tr>
<tr>
<td>in</td>
<td>converter</td>
<td>Conversion polynomial</td>
</tr>
</tbody>
</table>
Returns

the count of conversion performed, otherwise a negative error code:

- EINVAL is returned if some argument is missing or wrong; chan, rng and the pointers should be checked; check also the kernel log ("dmesg"); WARNING: a4l_fill_desc() should be called before using a4l_rawtodcal()

6.65.2.4 a4l_read_calibration_file()

```c
int a4l_read_calibration_file (  
    char * name,  
    struct a4l_calibration_data * data)
```

Read the analogy generated calibration file.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>name</th>
<th>Name of the calibration file</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>data</td>
<td>Pointer to the calibration file contents</td>
</tr>
</tbody>
</table>
6.66 Descriptor Syscall API

Collaboration diagram for Descriptor Syscall API:

![Collaboration diagram](image)

Data Structures

- struct `a4l_descriptor`
  
  *Structure containing device-information useful to users.*

Functions

- int `a4l_sys_desc (int fd, a4l_desc_t *dsc, int pass)`
  
  *Get a descriptor on an attached device.*

ANALOGY_xxx_DESC

Constants used as argument so as to define the description depth to recover

- #define `A4L_BSC_DESC 0x0`
  
  *BSC stands for basic descriptor (device data)*

- #define `A4L_CPLX_DESC 0x1`
  
  *CPLX stands for complex descriptor (subdevice + channel + range data)*

6.66.1 Detailed Description

6.66.2 Function Documentation
6.66 Descriptor Syscall API

6.66.2.1 a4l_sys_desc()

```c
int a4l_sys_desc (  
    int fd,  
    a4l_desc_t * dsc,  
    int pass  )
```

Get a descriptor on an attached device.

Once the device has been attached, the function `a4l_get_desc()` retrieves various information on the device (subdevices, channels, ranges, etc.). The function `a4l_get_desc()` can be called twice:

- The first time, almost all the fields, except `sbdata`, are set (board_name, nb_subd, idx_read_subd, idx_write_subd, magic, sbsize); the last field, `sbdata`, is supposed to be a pointer on a buffer, which size is defined by the field `sbsize`.
- The second time, the buffer pointed by `sbdata` is filled with data about the subdevices, the channels and the ranges.

Between the two calls, an allocation must be performed in order to recover a buffer large enough to contain all the data. These data are set up according a root-leaf organization (device -> subdevice -> channel -> range). They cannot be accessed directly; specific functions are available so as to retrieve them:

- `a4l_get_subdinfo()` to get some subdevice's characteristics.
- `a4l_get_chaninfo()` to get some channel's characteristics.
- `a4l_get_rnginfo()` to get some range's characteristics.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>fd</th>
<th>Driver file descriptor</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>dsc</td>
<td>Device descriptor</td>
</tr>
<tr>
<td>in</td>
<td>pass</td>
<td>Description level to retrieve:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- A4L_BSC_DESC to get the basic descriptor (notably the size of the data buffer to allocate).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- A4L_CPLX_DESC to get the complex descriptor, the data buffer is filled with characteristics about the subdevices, the channels and the ranges.</td>
</tr>
</tbody>
</table>

**Returns**

0 on success. Otherwise:

- `EINVAL` is returned if some argument is missing or wrong; the pass argument should be checked; check also the kernel log ("dmesg")
- `EFAULT` is returned if a user <> kernel transfer went wrong
- `ENODEV` is returned if the descriptor is incoherent (the device may be unattached)

**References**

A4L_BSC_DESC, and `a4l_descriptor::magic`. Referenced by `a4l_open()`.
6.67 Descriptor API

Collaboration diagram for Descriptor API:

![Collaboration Diagram](image_url)

Functions

- `int a4l_open (a4l_desc_t *dsc, const char *fname)`
  
  *Open an Analogy device and basically fill the descriptor.*

- `int a4l_close (a4l_desc_t *dsc)`

  *Close the Analogy device related with the descriptor.*

- `int a4l_fill_desc (a4l_desc_t *dsc)`

  *Fill the descriptor with subdevices, channels and ranges data.*

- `int a4l_get_subdinfo (a4l_desc_t *dsc, unsigned int subd, a4l_sbinfo_t **info)`

  *Get an information structure on a specified subdevice.*

- `int a4l_get_chinfo (a4l_desc_t *dsc, unsigned int subd, unsigned int chan, a4l_chinfo_t **info)`

  *Get an information structure on a specified channel.*

- `int a4l_get_rnginfo (a4l_desc_t *dsc, unsigned int subd, unsigned int chan, unsigned int rng, a4l_rnginfo_t **info)`

  *Get an information structure on a specified range.*

6.67.1 Detailed Description

This is the API interface used to fill and use Analogy device descriptor structure

6.67.2 Function Documentation

6.67.2.1 a4l_close()

```c
int a4l_close (a4l_desc_t * dsc)
```

*Close the Analogy device related with the descriptor.*

The file descriptor is associated with a context. The context is one of the enabler of asynchronous transfers. So, by closing the file descriptor, the programe must keep in mind that the currently occuring asynchronous transfer will cancelled.
Parameters

| in | dsc | Device descriptor |

Returns

0 on success. Otherwise:

- EINVAL is returned if some argument is missing or wrong; the dsc pointer should be checked; check also the kernel log ("dmesg")

References a4l_sys_close(), and a4l_descriptor::fd.

6.67.2.2 a4l_fill_desc()

int a4l_fill_desc (a4l_desc_t * dsc)

Fill the descriptor with subdevices, channels and ranges data.

Parameters

| in | dsc | Device descriptor partly filled by a4l_open(). |

Returns

0 on success. Otherwise:

- EINVAL is returned if some argument is missing or wrong; the dsc pointer should be checked; check also the kernel log ("dmesg")
-EFAULT is returned if a user <-> kernel transfer went wrong
-ENODEV is returned if the descriptor is incoherent (the device may be unattached)

References a4l_descriptor::fd, and a4l_descriptor::magic.

6.67.2.3 a4l_get_chinfo()

int a4l_get_chinfo (a4l_desc_t * dsc,
                    unsigned int subd,
                    unsigned int chan,
                    a4l_chinfo_t * info)

Get an information structure on a specified channel.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>dsc</th>
<th>Device descriptor filled by a4l_open() and a4l_fill_desc()</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>subd</td>
<td>Subdevice index</td>
</tr>
<tr>
<td>in</td>
<td>chan</td>
<td>Channel index</td>
</tr>
<tr>
<td>out</td>
<td>info</td>
<td>Channel information structure</td>
</tr>
</tbody>
</table>

Returns

0 on success. Otherwise:

- EINVAL is returned if some argument is missing or wrong; subd, chan and the dsc pointer should be checked; check also the kernel log ("dmesg"); WARNING: a4l_fill_desc() should be called before using a4l_get_chinfo()

References a4l_descriptor::magic.

6.67.2.4 a4l_get_rnginfo()

int a4l_get_rnginfo (  
    a4l_desc_t * dsc,  
    unsigned int subd,  
    unsigned int chan,  
    unsigned int rng,  
    a4l_rnginfo_t ** info )

Get an information structure on a specified range.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>dsc</th>
<th>Device descriptor filled by a4l_open() and a4l_fill_desc()</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>subd</td>
<td>Subdevice index</td>
</tr>
<tr>
<td>in</td>
<td>chan</td>
<td>Channel index</td>
</tr>
<tr>
<td>in</td>
<td>rng</td>
<td>Range index</td>
</tr>
<tr>
<td>out</td>
<td>info</td>
<td>Range information structure</td>
</tr>
</tbody>
</table>

Returns

0 on success. Otherwise:

- EINVAL is returned if some argument is missing or wrong; subd, chan, rng and the dsc pointer should be checked; check also the kernel log ("dmesg"); WARNING: a4l_fill_desc() should be called before using a4l_get_rnginfo()

References a4l_descriptor::magic.
6.67.2.5 a4l_get_subdinfo()

```c
int a4l_get_subdinfo (  
a4l_desc_t * dsc,  
unsigned int subd,  
a4l_sbinfo_t ** info )
```

Get an information structure on a specified subdevice.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>dsc</th>
<th>Device descriptor filled by a4l_open() and a4l_fill_desc()</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>subd</td>
<td>Subdevice index</td>
</tr>
<tr>
<td>out</td>
<td>info</td>
<td>Subdevice information structure</td>
</tr>
</tbody>
</table>

Returns

0 on success. Otherwise:

- -EINVAL is returned if some argument is missing or wrong; subd and the dsc pointer should be checked; check also the kernel log ("dmesg"); WARNING: a4l_fill_desc() should be called before using a4l_get_subdinfo().

References a4l_descriptor::magic.

Referenced by a4l_sync_dio().

6.67.2.6 a4l_open()

```c
int a4l_open (  
a4l_desc_t * dsc,  
const char * fname )
```

Open an Analogy device and basically fill the descriptor.

Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>dsc</th>
<th>Device descriptor</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>fname</td>
<td>Device name</td>
</tr>
</tbody>
</table>

Returns

0 on success. Otherwise:

- -EINVAL is returned if some argument is missing or wrong; the fname and the dsc pointer should be checked; check also the kernel log ("dmesg")
- -EFAULT is returned if a user <-> kernel transfer went wrong

References A4L_BSC_DESC, a4l_sys_close(), a4l_sys_desc(), a4l_sys_open(), and a4l_descriptor::fd.
6.68 Math API

Collaboration diagram for Math API:

![Collaboration diagram](Level 2 API → Math API)

Functions

- int `a4l_math_polyfit` (unsigned `r_dim`, double `*r`, double `orig`, const unsigned `dim`, double `*x`, double `*y`)
  
  *Calculate the polynomial fit.*

- void `a4l_math_mean` (double `*pmean`, double `*val`, unsigned `nr`)
  
  *Calculate the arithmetic mean of an array of values.*

- void `a4l_math_stddev` (double `*pstddev`, double `mean`, double `*val`, unsigned `nr`)
  
  *Calculate the standard deviation of an array of values.*

- void `a4l_math_stddev_of_mean` (double `*pstddevm`, double `mean`, double `*val`, unsigned `nr`)
  
  *Calculate the standard deviation of the mean.*

6.68.1 Detailed Description

6.68.2 Function Documentation

6.68.2.1 `a4l_math_mean()`

```c
void a4l_math_mean (  
  double *pmean,  
  double *val,  
  unsigned nr )
```

*Calculate the arithmetic mean of an array of values.*

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>out</code></td>
<td><code>pmean</code></td>
</tr>
<tr>
<td><code>in</code></td>
<td><code>val</code></td>
</tr>
<tr>
<td><code>in</code></td>
<td><code>nr</code></td>
</tr>
</tbody>
</table>
6.68.2.2   a4l_math_polyfit()

```c
int a4l_math_polyfit (  
    unsigned r_dim,  
    double * r,  
    double orig,  
    const unsigned dim,  
    double * x,  
    double * y )
```

Calculate the polynomial fit.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>r_dim</code></td>
</tr>
<tr>
<td>out</td>
<td><code>r</code></td>
</tr>
<tr>
<td>in</td>
<td><code>orig</code></td>
</tr>
<tr>
<td>in</td>
<td><code>dim</code></td>
</tr>
<tr>
<td>in</td>
<td><code>x</code></td>
</tr>
<tr>
<td>in</td>
<td><code>y</code></td>
</tr>
</tbody>
</table>

Operation:

We are looking for Res such that A.Res = Y, with A the Vandermonde matrix made from the X vector.

Using the least square method, this means finding Res such that: \(A^\top A\cdot Res = A^\top Y\)

If we write \(A = Q.R\) with \(Q^\top Q = I\), and R non singular, this can be reduced to: \(R\cdot Res = Q^\top Y\)

`mat_qr()` gives us R and \(Q^\top Y\) from A and Y

We can then obtain Res by back substitution using `mat_upper_triangular_backsub()` with R upper triangular.

6.68.2.3   a4l_math_stddev()

```c
void a4l_math_stddev (  
    double * pstddev,  
    double mean,  
    double * val,  
    unsigned nr )
```

Calculate the standard deviation of an array of values.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td><code>pstddev</code></td>
</tr>
<tr>
<td>in</td>
<td><code>mean</code></td>
</tr>
<tr>
<td>in</td>
<td><code>val</code></td>
</tr>
<tr>
<td>in</td>
<td><code>nr</code></td>
</tr>
</tbody>
</table>
Referenced by `a4l_math_stddev_of_mean()`.

6.68.2.4  `a4l_math_stddev_of_mean()`

```c
void a4l_math_stddev_of_mean (
    double  * pstddevm,
    double  mean,
    double  * val,
    unsigned nr )
```

Calculate the standard deviation of the mean.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pstddevm</code></td>
<td>Pointer to the resulting value</td>
</tr>
<tr>
<td><code>mean</code></td>
<td>Mean value</td>
</tr>
<tr>
<td><code>val</code></td>
<td>Array of input values</td>
</tr>
<tr>
<td><code>nr</code></td>
<td>Number of array elements</td>
</tr>
</tbody>
</table>

References `a4l_math_stddev()`.
6.69  Range / conversion API

Collaboration diagram for Range / conversion API:

![Collaboration diagram](image)

Functions

- `int a4l_sizeof_chan (a4l_chinfo_t *chan)`
  
  Get the size in memory of an acquired element.

- `int a4l_sizeof_subd (a4l_sbinfo_t *subd)`
  
  Get the size in memory of a digital acquired element.

- `int a4l_find_range (a4l_desc_t *dsc, unsigned int idx_subd, unsigned int idx_chan, unsigned long unit, double min, double max, a4l_rnginfo_t **rng)`
  
  Find the most suitable range.

- `int a4l_rawtoul (a4l_chinfo_t *chan, unsigned long *dst, void *src, int cnt)`
  
  Unpack raw data (from the driver) into unsigned long values.

- `int a4l_rawtof (a4l_chinfo_t *chan, a4l_rnginfo_t *rng, float *dst, void *src, int cnt)`
  
  Convert raw data (from the driver) to float-typed samples.

- `int a4l_ultoraw (a4l_chinfo_t *chan, void *dst, unsigned long *src, int cnt)`
  
  Pack unsigned long values into raw data (for the driver).

- `int a4l_ftoraw (a4l_chinfo_t *chan, a4l_rnginfo_t *rng, void *dst, float *src, int cnt)`
  
  Convert float-typed samples to raw data (for the driver).

- `int a4l_dtoraw (a4l_chinfo_t *chan, a4l_rnginfo_t *rng, void *dst, double *src, int cnt)`
  
  Convert double-typed samples to raw data (for the driver).

6.69.1  Detailed Description

6.69.2  Function Documentation

6.69.2.1  a4l_dtoraw()

```c
int a4l_dtoraw (a4l_chinfo_t *chan, a4l_rnginfo_t *rng, void *dst, double *src, int cnt)
```

Convert double-typed samples to raw data (for the driver)
Module Documentation

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>chan</th>
<th>Channel descriptor</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>rng</td>
<td>Range descriptor</td>
</tr>
<tr>
<td>out</td>
<td>dst</td>
<td>Output buffer</td>
</tr>
<tr>
<td>in</td>
<td>src</td>
<td>Input buffer</td>
</tr>
<tr>
<td>in</td>
<td>cnt</td>
<td>Count of conversion to perform</td>
</tr>
</tbody>
</table>

Returns

the count of conversion performed, otherwise a negative error code:

- -EINVAL is returned if some argument is missing or wrong; chan, rng and the pointers should be checked; check also the kernel log ("dmesg"); WARNING: a4l_dtoraw() should be called before using a4l_dtoraw()

6.69.2.2  a4l_find_range()

int a4l_find_range (  
  a4l_desc_t ∗ dsc,  
  unsigned int idx_subd,  
  unsigned int idx_chan,  
  unsigned long unit,  
  double min,  
  double max,  
  a4l_rnginfo_t ∗∗ rng )

Find the most suitable range.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>dsc</th>
<th>Device descriptor filled by a4l_open() and a4l Fill_desc()</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>idx_subd</td>
<td>Index of the concerned subdevice</td>
</tr>
<tr>
<td>in</td>
<td>idx_chan</td>
<td>Index of the concerned channel</td>
</tr>
<tr>
<td>in</td>
<td>unit</td>
<td>Unit type used in the range</td>
</tr>
<tr>
<td>in</td>
<td>min</td>
<td>Minimal limit value</td>
</tr>
<tr>
<td>in</td>
<td>max</td>
<td>Maximal limit value</td>
</tr>
<tr>
<td>out</td>
<td>rng</td>
<td>Found range</td>
</tr>
</tbody>
</table>

Returns

The index of the most suitable range on success. Otherwise:

- -ENOENT is returned if a suitable range is not found.
- -EINVAL is returned if some argument is missing or wrong; idx_subd, idx_chan and the dsc pointer should be checked; check also the kernel log ("dmesg"); WARNING: a4l Fill_desc() should be called before using a4l_fill_desc()
References a4l_descriptor::magic.

### 6.69.2.3 a4l_ftoraw()

```c
int a4l_ftoraw(  
    a4l_chinfo_t *chan,  
    a4l_rnginfo_t *rng,  
    void *dst,  
    float *src,  
    int cnt )
```

Convert float-typed samples to raw data (for the driver)

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>chan</th>
<th>Channel descriptor</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>rng</td>
<td>Range descriptor</td>
</tr>
<tr>
<td>out</td>
<td>dst</td>
<td>Output buffer</td>
</tr>
<tr>
<td>in</td>
<td>src</td>
<td>Input buffer</td>
</tr>
<tr>
<td>in</td>
<td>cnt</td>
<td>Count of conversion to perform</td>
</tr>
</tbody>
</table>

**Returns**

the count of conversion performed, otherwise a negative error code:

- **-EINVAL** is returned if some argument is missing or wrong; chan, rng and the pointers should be checked; check also the kernel log ("dmesg"); WARNING: **a4l_fill_desc()** should be called before using **a4l_ftoraw()**

### 6.69.2.4 a4l_rawtod()

```c
int a4l_rawtod(  
    a4l_chinfo_t *chan,  
    a4l_rnginfo_t *rng,  
    double *dst,  
    void *src,  
    int cnt )
```

Convert raw data (from the driver) to double-typed samples.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>chan</th>
<th>Channel descriptor</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>rng</td>
<td>Range descriptor</td>
</tr>
<tr>
<td>out</td>
<td>dst</td>
<td>Output buffer</td>
</tr>
<tr>
<td>in</td>
<td>src</td>
<td>Input buffer</td>
</tr>
<tr>
<td>in</td>
<td>cnt</td>
<td>Count of conversion to perform</td>
</tr>
</tbody>
</table>
Returns the count of conversion performed, otherwise a negative error code:

-EINVAL is returned if some argument is missing or wrong; chan, rng and the pointers should be checked; check also the kernel log ("dmesg"); WARNING: a4l_fill_desc() should be called before using a4l_rawtod()

6.69.2.5 a4l_rawtof()

int a4l_rawtof (  
a4l_chinfo_t * chan,  
a4l_rnginfo_t * rng,  
float * dst,  
void * src,  
int cnt )

Convert raw data (from the driver) to float-typed samples.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>chan Channel descriptor</td>
</tr>
<tr>
<td>in</td>
<td>rng Range descriptor</td>
</tr>
<tr>
<td>out</td>
<td>dst Output buffer</td>
</tr>
<tr>
<td>in</td>
<td>src Input buffer</td>
</tr>
<tr>
<td>in</td>
<td>cnt Count of conversion to perform</td>
</tr>
</tbody>
</table>

Returns the count of conversion performed, otherwise a negative error code:

-EINVAL is returned if some argument is missing or wrong; chan, rng and the pointers should be checked; check also the kernel log ("dmesg"); WARNING: a4l_fill_desc() should be called before using a4l_rawtod()

6.69.2.6 a4l_rawtoul()

int a4l_rawtoul (  
a4l_chinfo_t * chan,  
unsigned long * dst,  
void * src,  
int cnt )

Unpack raw data (from the driver) into unsigned long values.

This function takes as input driver-specific data and scatters each element into an entry of an unsigned long table. It is a convenience routine which performs no conversion, just copy.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>chan</th>
<th>Channel descriptor</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>dst</td>
<td>Output buffer</td>
</tr>
<tr>
<td>in</td>
<td>src</td>
<td>Input buffer</td>
</tr>
<tr>
<td>in</td>
<td>cnt</td>
<td>Count of transfer to copy</td>
</tr>
</tbody>
</table>

Returns

the count of copy performed, otherwise a negative error code:

- -EINVAL is returned if some argument is missing or wrong; chan, dst and src pointers should be checked; check also the kernel log ("dmesg"); WARNING: \texttt{a4l\_fill\_desc()} should be called before using \texttt{a4l\_ultoraw()}

6.69.2.7 \texttt{a4l\_sizeof\_chan()}

\begin{verbatim}
int a4l\_sizeof\_chan ( 
    a4l\_chinfo\_t \* chan )
\end{verbatim}

Get the size in memory of an acquired element.

According to the board, the channels have various acquisition widths. With values like 8, 16 or 32, there is no problem finding out the size in memory (1, 2, 4); however with widths like 12 or 24, this function might be helpful to guess the size needed in RAM for a single acquired element.

Parameters

| in | chan | Channel descriptor |

Returns

the size in memory of an acquired element, otherwise a negative error code:

- -EINVAL is returned if the argument chan is NULL

6.69.2.8 \texttt{a4l\_sizeof\_subd()}

\begin{verbatim}
int a4l\_sizeof\_subd ( 
    a4l\_shinfo\_t \* subd )
\end{verbatim}

Get the size in memory of a digital acquired element.
This function is only useful for DIO subdevices. Digital subdevices are a specific kind of subdevice on which channels are regarded as bits composing the subdevice's bitfield. During a DIO acquisition, all bits are sampled. Therefore, `a4l_sizeof_chan()` is useless in this case and we have to use `a4l_sizeof_subd()`. With bitfields which sizes are 8, 16 or 32, there is no problem finding out the size in memory (1, 2, 4); however with widths like 12 or 24, this function might be helpful to guess the size needed in RAM for a single acquired element.

**Parameters**

| in  | subd | Subdevice descriptor |

**Returns**

the size in memory of an acquired element, otherwise a negative error code:

- -EINVAL is returned if the argument chan is NULL or if the subdevice is not a digital subdevice

**References**

A4L_SUBD_DI, A4L_SUBD_DIO, A4L_SUBD_DO, and A4L_SUBD_TYPES.

Referenced by `a4l_sync_dio()`.

---

### 6.69.2.9 a4l_ultoraw()

```c
int a4l_ultoraw (  
a4l_chinfo_t * chan,  
void * dst,  
unsigned long * src,  
int cnt )
```

Pack unsigned long values into raw data (for the driver)

This function takes as input a table of unsigned long values and gather them according to the channel width. It is a convenience routine which performs no conversion, just formatting.

**Parameters**

| in  | chan | Channel descriptor |
| out | dst  | Ouput buffer |
| in  | src  | Input buffer |
| in  | cnt  | Count of transfer to copy |

**Returns**

the count of copy performed, otherwise a negative error code:

- -EINVAL is returned if some argument is missing or wrong; chan, dst and src pointers should be checked; check also the kernel log ("dmesg"); WARNING: `a4l_fill_desc()` should be called before using `a4l_ultoraw()`
6.70 Level 1 API

Collaboration diagram for Level 1 API:

![Collaboration diagram](image)

Modules

- Asynchronous acquisition API
- Descriptor API
- Synchronous acquisition API

6.70.1 Detailed Description
6.71 Synchronous acquisition API

Collaboration diagram for Synchronous acquisition API:

![Collaboration diagram](image)

Data Structures

- **struct a4l_instruction**
  
  *Structure describing the synchronous instruction.*

- **struct a4l_instruction_list**
  
  *Structure describing the list of synchronous instructions.*

Macros

- `#define A4L_INSN_WAIT_MAX 100000`
  
  *Maximal wait duration.*

Functions

- `int a4l_snd_insnlist (a4l_desc_t *dsc, a4l_insnlst_t *arg)`
  
  *Perform a list of synchronous acquisition misc operations.*

- `int a4l_snd_insn (a4l_desc_t *dsc, a4l_insn_t *arg)`
  
  *Perform a synchronous acquisition misc operation.*

Instruction type

Flags to define the type of instruction

- `#define A4L_INSN_READ (0 | A4L_INSN_MASK_READ)`
  
  *Read instruction.*

- `#define A4L_INSN_WRITE (1 | A4L_INSN_MASK_WRITE)`
  
  *Write instruction.*

- `#define A4L_INSN_BITS`
  
  *"Bits" instruction*

- `#define A4L_INSN_CONFIG`
  
  *Configuration instruction.*

- `#define A4L_INSN_GTOD`
  
  *Get time instruction.*

- `#define A4L_INSN_WAIT`
  
  *Wait instruction.*

- `#define A4L_INSN_INTTRIG`
  
  *Trigger instruction (to start asynchronous acquisition)*
Configuration instruction type

Values to define the type of configuration instruction

- #define A4L_INSN_CONFIG_DIO_INPUT 0
- #define A4L_INSN_CONFIG_DIO_OUTPUT 1
- #define A4L_INSN_CONFIG_DIO_INPUT_PIN 2
- #define A4L_INSN_CONFIG_DIO_OUTPUT_PIN 3
- #define A4L_INSN_CONFIG_Analog_TRIG 16
- #define A4L_INSN_CONFIG_ALT_SOURCE 20
- #define A4L_INSN_CONFIG_DIGITAL_TRIG 21
- #define A4L_INSN_CONFIG_BLOCK_SIZE 22
- #define A4L_INSN_CONFIG_TIMER_1 23
- #define A4L_INSN_CONFIG_FILTER 24
- #define A4L_INSN_CONFIG_CHANGE_NOTIFY 25
- #define A4L_INSN_CONFIG_SERIAL_CLOCK 26
- #define A4L_INSN_CONFIG_BIDIRECTIONAL_DATA 27
- #define A4L_INSN_CONFIG_QUERY 28
- #define A4L_INSN_CONFIG_PWM_OUTPUT 29
- #define A4L_INSN_CONFIG_GET_PWM_OUTPUT 30
- #define A4L_INSN_CONFIG_ARM 31
- #define A4L_INSN_CONFIG_DISARM 32
- #define A4L_INSN_CONFIG_GET_COUNTER_STATUS 33
- #define A4L_INSN_CONFIG_RESET 34
- #define A4L_INSN_CONFIG_GPCT_SINGLE_PULSE_GENERATOR 1001 /* Use CTR as single pulse generator */
- #define A4L_INSN_CONFIG_GPCT_PULSE_TRAIN_GENERATOR 1002 /* Use CTR as pulse train generator */
- #define A4L_INSN_CONFIG_GPCT_QUADRATURE_ENCODER 1003 /* Use the counter as encoder */
- #define A4L_INSN_CONFIG_SET_GATE_SRC 2001 /* Set gate source */
- #define A4L_INSN_CONFIG_GET_GATE_SRC 2002 /* Get gate source */
- #define A4L_INSN_CONFIG_SET_CLOCK_SRC 2003 /* Set master clock source */
- #define A4L_INSN_CONFIG_GET_CLOCK_SRC 2004 /* Get master clock source */
- #define A4L_INSN_CONFIG_SET_OTHER_SRC 2005 /* Set other source */
- #define A4L_INSN_CONFIG_SET_COUNTER_MODE 4097
- #define A4L_INSN_CONFIG_SET_ROUTING 4099
- #define A4L_INSN_CONFIG_GET_ROUTING 4109

Counter status bits

Status bits for INSN_CONFIG_GET_COUNTER_STATUS

- #define A4L_COUNTER_ARMED 0x1
- #define A4L_COUNTER_COUNTING 0x2
- #define A4L_COUNTER_TERMINAL_COUNT 0x4

IO direction

Values to define the IO polarity

- #define A4L_INPUT 0
- #define A4L_OUTPUT 1
- #define A4L_OPENDRAIN 2
Events types

Values to define the Analogy events. They might used to send some specific events through the instruction interface.

- `#define A4L_EV_START 0x00040000`
- `#define A4L_EV_SCAN_BEGIN 0x00080000`
- `#define A4L_EV_CONVERT 0x00100000`
- `#define A4L_EV_SCAN_END 0x00200000`
- `#define A4L_EV_STOP 0x00400000`

6.71.1 Detailed Description

6.71.2 Function Documentation

6.71.2.1 `a4l_snd_insn()`

```c
int a4l_snd_insn (  
    a4l_desc_t * dsc,  
    a4l_insn_t * arg )
```

Perform a synchronous acquisition misc operation.

The function `a4l_snd_insn()` triggers a synchronous acquisition.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>dsc</th>
<th>Device descriptor filled by <code>a4l_open()</code> (and optionally <code>a4l_fill_desc()</code>)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>arg</td>
<td>Instruction structure</td>
</tr>
</tbody>
</table>

Returns

0 on success. Otherwise:

- `-EINVAL` is returned if some argument is missing or wrong (Please, type "dmesg" for more info)
- `-EFAULT` is returned if a user <-> kernel transfer went wrong
- `-ENOMEM` is returned if the system is out of memory

References `a4l_descriptor::fd`.

Referenced by `a4l_sync_dio()`.
6.71 Synchronous acquisition API

6.71.2.2 a4l_snd_insnlist()

```c
int a4l_snd_insnlist (  
  a4l_desc_t * dsc,  
  a4l_insnlst_t * arg )
```

Perform a list of synchronous acquisition misc operations.

The function `a4l_snd_insnlist()` is able to send many synchronous instructions on a various set of sub-devices, channels, etc.

Parameters

| in | dsc | Device descriptor filled by `a4l_open()` (and optionally `a4l_fill_desc()`) |
| in | arg | Instructions list structure |

Returns

0 on success. Otherwise:

- `-EINVAL` is returned if some argument is missing or wrong (Please, type "dmesg" for more info)
- `-EFAULT` is returned if a user `<->` kernel transfer went wrong
- `-ENOMEM` is returned if the system is out of memory

References `a4l_descriptor::fd`. 

Generated by Doxygen
6.72 Level 2 API

Collaboration diagram for Level 2 API:

![Collaboration diagram for Level 2 API](image)

Modules

- Asynchronous acquisition API
- Software calibration API
- Math API
- Range / conversion API
- Synchronous acquisition API

6.72.1 Detailed Description
6.73 Synchronous acquisition API

Functions

- `int a4l_sync_write (a4l_desc_t *dsc, unsigned int idx_subd, unsigned int chan_desc, unsigned int ns_delay, void *buf, size_t nbyte)`
  - Perform a synchronous acquisition write operation.
- `int a4l_sync_read (a4l_desc_t *dsc, unsigned int idx_subd, unsigned int chan_desc, unsigned int ns_delay, void *buf, size_t nbyte)`
  - Perform a synchronous acquisition read operation.
- `int a4l_sync_dio (a4l_desc_t *dsc, unsigned int idx_subd, void *mask, void *buf)`
  - Perform a synchronous acquisition digital acquisition.
- `int a4l_config_subd (a4l_desc_t *dsc, unsigned int idx_subd, unsigned int type,...)`
  - Configure a subdevice.

6.73.1 Detailed Description

6.73.2 Function Documentation

6.73.2.1 a4l_config_subd()

```
int a4l_config_subd (  
  a4l_desc_t * dsc,  
  unsigned int idx_subd,  
  unsigned int type,  
  ...  )
```

Configure a subdevice.

`a4l_config_subd()` takes a variable count of arguments. According to the configuration type, some additional argument is necessary:

- A4L_INSN_CONFIG_DIO_INPUT: the channel index (unsigned int)
- A4L_INSN_CONFIG_DIO_OUTPUT: the channel index (unsigned int)
- A4L_INSN_CONFIG_DIO_QUERY: the returned DIO polarity (unsigned int *)
### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>dsc</th>
<th>Device descriptor filled by <strong>a4l_open()</strong> (and optionally <strong>a4l_fill_desc()</strong>)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>idx_subd</td>
<td>Index of the concerned subdevice</td>
</tr>
<tr>
<td>in</td>
<td>type</td>
<td>Configuration parameter</td>
</tr>
</tbody>
</table>

### Returns

0 on success. Otherwise:

- -EINVAL is returned if some argument is missing or wrong (Please, type "dmesg" for more info)
- -ENOSYS is returned if the configuration parameter is not supported

References A4L_INSN_CONFIG, and a4l_instruction::type.

#### 6.73.2.2 a4l_sync_dio()

```c
int a4l_sync_dio (  
    a4l_desc_t * dsc,  
    unsigned int idx_subd,  
    void * mask,  
    void * buf )
```

Perform a synchronous acquisition digital acquisition.

### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>dsc</th>
<th>Device descriptor filled by <strong>a4l_open()</strong> (and optionally <strong>a4l_fill_desc()</strong>)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>idx_subd</td>
<td>Index of the concerned subdevice</td>
</tr>
<tr>
<td>in</td>
<td>mask</td>
<td>Write mask which indicates which bit(s) must be modified</td>
</tr>
<tr>
<td>in, out</td>
<td>buf</td>
<td>Input / output buffer</td>
</tr>
</tbody>
</table>

### Returns

Number of bytes read, otherwise negative error code:

- -EINVAL is returned if some argument is missing or wrong (Please, type "dmesg" for more info)
- -EFAULT is returned if a user <-> kernel transfer went wrong
- -ENOMEM is returned if the system is out of memory
- -ENOSYS is returned if the driver does not provide any handler "instruction bits"

References a4l_get_subdinfo(), A4L_INSN_BITS, a4l_sizeof_subd(), a4l_snd_insn(), a4l_instruction::data_size, and a4l_instruction::type.
### 6.73.2.3 a4l_sync_read()

```c
int a4l_sync_read(
    a4l_desc_t * dsc,
    unsigned int idx_subd,
    unsigned int chan_desc,
    unsigned int ns_delay,
    void * buf,
    size_t nbyte )
```

Perform a synchronous acquisition read operation.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dsc</code></td>
<td>Device descriptor filled by <code>a4l_open()</code> (and optionally <code>a4l_fill_desc()</code>)</td>
</tr>
<tr>
<td><code>idx_subd</code></td>
<td>Index of the concerned subdevice</td>
</tr>
<tr>
<td><code>chan_desc</code></td>
<td>Channel descriptor (channel, range and reference)</td>
</tr>
<tr>
<td><code>ns_delay</code></td>
<td>Optional delay (in nanoseconds) to wait between the setting of the input channel and sample(s) acquisition(s).</td>
</tr>
<tr>
<td><code>buf</code></td>
<td>Input buffer</td>
</tr>
<tr>
<td><code>nbyte</code></td>
<td>Number of bytes to read</td>
</tr>
</tbody>
</table>

**Returns**

Number of bytes read, otherwise negative error code:

- `-EINVAL` is returned if some argument is missing or wrong (Please, type "dmesg" for more info)
- `-EFAULT` is returned if a user <-> kernel transfer went wrong
- `-ENOMEM` is returned if the system is out of memory

**References** `A4L_INSN_READ`, `A4L_INSN_WAIT`, and `a4l_instruction::type`.

### 6.73.2.4 a4l_sync_write()

```c
int a4l_sync_write(
    a4l_desc_t * dsc,
    unsigned int idx_subd,
    unsigned int chan_desc,
    unsigned int ns_delay,
    void * buf,
    size_t nbyte )
```

Perform a synchronous acquisition write operation.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dsc</code></td>
<td>Device descriptor filled by <code>a4l_open()</code> (and optionally <code>a4l_fill_desc()</code>)</td>
</tr>
<tr>
<td><code>idx_subd</code></td>
<td>Index of the concerned subdevice</td>
</tr>
<tr>
<td><code>chan_desc</code></td>
<td>Channel descriptor (channel, range and reference)</td>
</tr>
<tr>
<td><code>ns_delay</code></td>
<td>Optional delay (in nanoseconds) to wait between the setting of the input channel and sample(s) acquisition(s).</td>
</tr>
<tr>
<td><code>buf</code></td>
<td>Output buffer</td>
</tr>
<tr>
<td><code>nbyte</code></td>
<td>Number of bytes to write</td>
</tr>
</tbody>
</table>
Returns

Number of bytes written, otherwise negative error code:

- -EINVAL is returned if some argument is missing or wrong (Please, type "dmesg" for more info)
- -EFAULT is returned if a user <--> kernel transfer went wrong
- -ENOMEM is returned if the system is out of memory

References A4L_INSN_WAIT, A4L_INSN_WRITE, and a4l_instruction::type.
6.74 Analogy user API

Collaboration diagram for Analogy user API:

```
Modules
- Level 1 API
- Level 2 API
- Level 0 API
```

6.74.1 Detailed Description

This is the API interface of Analogy library
6.75  Level 0 API

Collaboration diagram for Level 0 API:

![Collaboration diagram for Level 0 API]

Modules

- Descriptor Syscall API
- Basic Syscall API
- Attach / detach Syscall API

6.75.1  Detailed Description

System call interface to core Analogy services

This interface should not be used directly by applications.
6.76 Basic Syscall API

Collaboration diagram for Basic Syscall API:

```
Level 0 API ----> Basic Syscall API
```

Functions

- `int a4l_sys_open` (const char *fname)
  
  Open an Analogy device.

- `int a4l_sys_close` (int fd)
  
  Close an Analogy device.

- `int a4l_sys_read` (int fd, void *buf, size_t nbyte)
  
  Read from an Analogy device.

- `int a4l_sys_write` (int fd, void *buf, size_t nbyte)
  
  Write to an Analogy device.

6.76.1 Detailed Description

6.76.2 Function Documentation

6.76.2.1 `a4l_sys_close()`

```
int a4l_sys_close (  
    int fd  )
```

Close an Analogy device.

Parameters

| in | fd | File descriptor as returned by `a4l_sys_open()` |

Returns

- 0 on success, otherwise a negative error code.

Referenced by `a4l_close()`, and `a4l_open()`.
6.76.2.2  a4l_sys_open()

int a4l_sys_open ( 
    const char * fname )

Open an Analogy device.

Parameters

| in  | fname | Device name |

Returns

Positive file descriptor value on success, otherwise a negative error code.

Referenced by a4l_open().

6.76.2.3  a4l_sys_read()

int a4l_sys_read ( 
    int fd, 
    void * buf, 
    size_t nbyte )

Read from an Analogy device.

The function a4l_read() is only useful for acquisition configured through an Analogy command.

Parameters

| in  | fd      | File descriptor as returned by a4l_sys_open() |
| out | buf     | Input buffer                     |
| in  | nbyte   | Number of bytes to read          |

Returns

Number of bytes read. Otherwise:

- EINVAL is returned if some argument is missing or wrong (Please, type "dmesg" for more info)
- ENOENT is returned if the device's reading subdevice is idle (no command was sent)
- EFAULT is returned if a user <-> kernel transfer went wrong
- EINTR is returned if calling task has been unblocked by a signal
6.76.2.4 a4l_sys_write()

```c
int a4l_sys_write (  
    int fd,  
    void *buf,  
    size_t nbyte )
```

Write to an Analogy device.

The function `a4l_write()` is only useful for acquisition configured through an Analogy command.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>fd</th>
<th>File descriptor as returned by <code>a4l_sys_open()</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>buf</td>
<td>Output buffer</td>
</tr>
<tr>
<td>in</td>
<td>nbyte</td>
<td>Number of bytes to write</td>
</tr>
</tbody>
</table>

Returns

- Number of bytes written. Otherwise:
  - -EINVAL is returned if some argument is missing or wrong (Please, type "dmesg" for more info)
  - -ENOENT is returned if the device's writing subdevice is idle (no command was sent)
  - -EFAULT is returned if a user <-> kernel transfer went wrong
  - -EINTR is returned if calling task has been unblocked by a signal
6.77  Attach / detach Syscall API

Collaboration diagram for Attach / detach Syscall API:

![Collaboration diagram](image)

Functions

- `int a4l_sys_attach (int fd, a4l_lnkdesc_t *arg)`
  
  Attach an Analogy device to a driver.

- `int a4l_sys_detach (int fd)`
  
  Detach an Analogy device from a driver.

- `int a4l_sys_bufcfg (int fd, unsigned int idx_subd, unsigned long size)`
  
  Configure the buffer size.

6.77.1 Detailed Description

6.77.2 Function Documentation

6.77.2.1 a4l_sys_attach()

```c
int a4l_sys_attach (  
    int fd,  
    a4l_lnkdesc_t * arg )
```

Attach an Analogy device to a driver.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>fd</th>
<th>File descriptor as returned by <code>a4l_sys_open()</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>arg</td>
<td>Link descriptor argument</td>
</tr>
</tbody>
</table>

Returns

0 on success. Otherwise:

- `-ENOMEM` is returned if the system is out of memory
- EINVAL is returned if some argument is missing or wrong (Please, type "dmesg" for more info)
- EFAULT is returned if a user <-> kernel transfer went wrong
- ENODEV is returned in case of internal error (Please, type "dmesg" for more info)
- ENXIO is returned in case of internal error (Please, type "dmesg" for more info)

6.77.2.2  a4l_sys_bufcfg()

```c
int a4l_sys_bufcfg (
    int fd,
    unsigned int idx_subd,
    unsigned long size )
```

Configure the buffer size.

This function can configure the buffer size of the file descriptor currently in use. If the subdevice index is set to A4L_BUF_DEFMAGIC, it can also define the default buffer size at open time.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>fd</th>
<th>File descriptor as returned by a4l_sys_open()</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>idx_subd</td>
<td>Index of the concerned subdevice</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>Buffer size to be set</td>
</tr>
</tbody>
</table>

Returns

0 on success. Otherwise:

- EINVAL is returned if some argument is missing or wrong (Please, type "dmesg" for more info)
- EPERM is returned if the function is called in an RT context or if the buffer to resize is mapped in user-space (Please, type "dmesg" for more info)
- EFAULT is returned if a user <-> kernel transfer went wrong
- EBUSY is returned if the selected subdevice is already processing an asynchronous operation
- ENOMEM is returned if the system is out of memory

Referenced by a4l_set_bufsize().

6.77.2.3  a4l_sys_detach()

```c
int a4l_sys_detach ( int fd )
```

Detach an Analogy device from a driver.
Parameters

| in | fd | File descriptor as returned by `a4l_sys_open()` |

Returns

0 on success. Otherwise:

- **-EINVAL** is returned if some argument is missing or wrong (Please, type "dmesg" for more info)
- **-EBUSY** is returned if the device to be detached is in use
- **-EPERM** is returned if the device to be detached still has some buffer mapped in user-space
- **-ENODEV** is returned in case of internal error (Please, type "dmesg" for more info)
- **-ENXIO** is returned in case of internal error (Please, type "dmesg" for more info)
6.78 Alarm services

General-purpose watchdog timers.

Collaboration diagram for Alarm services:

![Collaboration diagram]

Data Structures

- struct RT_ALARM_INFO
  
  Alarm status descriptor.

Functions

- int rt_alarm_start (RT_ALARM *alarm, RTIME value, RTIME interval)
  
  Start an alarm.
- int rt_alarm_stop (RT_ALARM *alarm)
  
  Stop an alarm.
- int rt_alarm_inquire (RT_ALARM *alarm, RT_ALARM_INFO *info)
  
  Query alarm status.
- int rt_alarm_create (RT_ALARM *alarm, const char *name, void(*handler)(void *arg), void *arg)
  
  Create an alarm object.
- int rt_alarm_delete (RT_ALARM *alarm)
  
  Delete an alarm.

6.78.1 Detailed Description

General-purpose watchdog timers.

Alarms are general-purpose watchdog timers. Alchemy tasks may create any number of alarms and use them to run a user-defined handler, after a specified initial delay has elapsed. Alarms can be either one shot or periodic; in the latter case, the real-time system automatically reprograms the alarm for the next shot according to a user-defined interval value.

6.78.2 Function Documentation
6.78.2.1 rt_alarm_create()

```c
int rt_alarm_create ( 
    RT_ALARM ∗ alarm, 
    const char ∗ name, 
    void (∗)(void ∗arg) handler, 
    void ∗ arg )
```

Create an alarm object.

This routine creates an object triggering an alarm routine at a specified time in the future. Alarms can be periodic or oneshot, depending on the reload interval value passed to `rt_alarm_start()`.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>alarm</code></td>
<td>The address of an alarm descriptor which can be later used to identify uniquely the created object, upon success of this call.</td>
</tr>
<tr>
<td><code>name</code></td>
<td>An ASCII string standing for the symbolic name of the alarm. When non-NULL and non-empty, a copy of this string is used for indexing the created alarm into the object registry.</td>
</tr>
<tr>
<td><code>handler</code></td>
<td>The address of the routine to call when the alarm expires. This routine is passed the <code>arg</code> value.</td>
</tr>
<tr>
<td><code>arg</code></td>
<td>A user-defined opaque argument passed to the <code>handler</code>.</td>
</tr>
</tbody>
</table>

**Returns**

Zero is returned upon success. Otherwise:

- -ENOMEM is returned if the system fails to get memory from the local pool in order to create the alarm.
- -EEXIST is returned if the `name` is conflicting with an already registered alarm.
- -EPERM is returned if this service was called from an asynchronous context.

**Tags**

mode-unrestricted, switch-secondary

**Note**

Alarms are process-private objects and thus cannot be shared by multiple processes, even if they belong to the same Xenomai session.

6.78.2.2 rt_alarm_delete()

```c
int rt_alarm_delete ( 
    RT_ALARM ∗ alarm )
```

Delete an alarm.

This routine deletes an alarm object previously created by a call to `rt_alarm_create()`.
Parameters

<table>
<thead>
<tr>
<th>alarm</th>
<th>The alarm descriptor.</th>
</tr>
</thead>
</table>

Returns

Zero is returned upon success. Otherwise:

- EINVAL is returned if alarm is not a valid alarm descriptor.
- EPERM is returned if this service was called from an asynchronous context.

Tags

mode-unrestricted, switch-secondary

6.78.2.3  rt_alarm_inquire()

int rt_alarm_inquire (
    RT_ALARM ∗ alarm,
    RT_ALARM_INFO ∗ info )

Query alarm status.

This routine returns the status information about the specified alarm.

Parameters

<table>
<thead>
<tr>
<th>alarm</th>
<th>The alarm descriptor.</th>
</tr>
</thead>
<tbody>
<tr>
<td>info</td>
<td>A pointer to the returnbuffer&quot; to copy the information to.</td>
</tr>
</tbody>
</table>

Returns

Zero is returned and status information is written to the structure pointed at by info upon success. Otherwise:

- EINVAL is returned if alarm is not a valid alarm descriptor.

Tags

unrestricted, switch-primary
6.78.2.4  rt_alarm_start()

int rt_alarm_start(
    RT_ALARM *alarm,
    RTIME value,
    RTIME interval)

Start an alarm.

This routine programs the trigger date of an alarm object. An alarm can be either periodic or oneshot, depending on the interval value.

Alarm handlers are always called on behalf of Xenomai’s internal timer event routine. Therefore, Xenomai routines which can be called from such handlers are restricted to the set of services available on behalf of an asynchronous context.

This service overrides any previous setup of the expiry date and reload interval for the alarm.

Parameters

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alarm</td>
<td>The alarm descriptor.</td>
</tr>
<tr>
<td>value</td>
<td>The relative date of the first expiry, expressed in clock ticks (see note).</td>
</tr>
<tr>
<td>interval</td>
<td>The reload value of the alarm. It is a periodic interval value to be used for reprogramming the next alarm shot, expressed in clock ticks (see note). If interval is equal to TM_INFINITE, the alarm will not be reloaded after it has expired.</td>
</tr>
</tbody>
</table>

Returns

Zero is returned upon success. Otherwise:

- -EINVAL is returned if alarm is not a valid alarm descriptor.
- -EPERM is returned if this service was called from an invalid context.

Tags

xthread-only, switch-primary

Note

Each of the initial value and interval is interpreted as a multiple of the Alchemy clock resolution (see –alchemy-clock-resolution option, defaults to 1 nanosecond).

6.78.2.5  rt_alarm_stop()

int rt_alarm_stop (
    RT_ALARM *alarm)

Stop an alarm.

This routine disables an alarm object, preventing any further expiry until it is re-enabled via rt_alarm_start().
Parameters

| alarm  | The alarm descriptor. |

Returns

Zero is returned upon success. Otherwise:

- EINVAL is returned if `alarm` is not a valid alarm descriptor.

Tags

unrestricted, switch-primary
6.79  Buffer services

Lightweight FIFO IPC mechanism.

Collaboration diagram for Buffer services:

![Collaboration diagram](image)

Data Structures

- struct RT_BUFFER_INFO
  
  Buffer status descriptor.

Macros

- #define B_PRIO 0x1 /* Pend by task priority order. */
  
  Creation flags.

Functions

- int rt_buffer_create (RT_BUFFER *bf, const char *name, size_t bufsz, int mode)
  
  Create an IPC buffer.

- int rt_buffer_delete (RT_BUFFER *bf)
  
  Delete an IPC buffer.

- ssize_t rt_buffer_write_timed (RT_BUFFER *bf, const void *ptr, size_t size, const struct timespec *abs_timeout)
  
  Write to an IPC buffer.

- static ssize_t rt_buffer_write_until (RT_BUFFER *bf, const void *ptr, size_t size, RTIME timeout)
  
  Write to an IPC buffer (with absolute scalar timeout).

- static ssize_t rt_buffer_write (RT_BUFFER *bf, const void *ptr, size_t size, RTIME timeout)
  
  Write to an IPC buffer (with relative scalar timeout).

- ssize_t rt_buffer_read_timed (RT_BUFFER *bf, void *ptr, size_t size, const struct timespec *abs_timeout)
  
  Read from an IPC buffer.

- static ssize_t rt_buffer_read_until (RT_BUFFER *bf, void *ptr, size_t size, RTIME timeout)
  
  Read from an IPC buffer (with absolute scalar timeout).

- static ssize_t rt_buffer_read (RT_BUFFER *bf, void *ptr, size_t size, RTIME timeout)
  
  Read from an IPC buffer (with relative scalar timeout).

- int rt_buffer_clear (RT_BUFFER *bf)
  
  Clear an IPC buffer.

- int rt_buffer_inquire (RT_BUFFER *bf, RT_BUFFER_INFO *info)
  
  Query buffer status.

- int rt_buffer_bind (RT_BUFFER *bf, const char *name, RTIME timeout)
  
  Bind to an IPC buffer.

- int rt_buffer_unbind (RT_BUFFER *bf)
  
  Unbind from an IPC buffer.
6.79 Buffer services

6.79.1 Detailed Description

Lightweight FIFO IPC mechanism.

A buffer is a lightweight IPC mechanism, implementing a fast, one-way producer-consumer data path. All messages written are buffered in a single memory area in strict FIFO order, until read either in blocking or non-blocking mode.

Message are always atomically handled on the write side (i.e. no interleave, no short writes), whilst only complete messages are normally returned to the read side. However, short reads may happen under a well-defined situation (see note in rt_buffer_read()), albeit they can be fully avoided by proper use of the buffer.

6.79.2 Macro Definition Documentation

6.79.2.1 B_PRIO

#define B_PRIO 0x1 /* Pend by task priority order. */

Creation flags.

6.79.3 Function Documentation

6.79.3.1 rt_buffer_bind()

int rt_buffer_bind (RT_BUFFER * bf, const char * name, RTIME timeout)

Bind to an IPC buffer.

This routine creates a new descriptor to refer to an existing IPC buffer identified by its symbolic name. If the object does not exist on entry, the caller may block until a buffer of the given name is created.

Parameters

<table>
<thead>
<tr>
<th>bf</th>
<th>The address of a buffer descriptor filled in by the operation. Contents of this memory is undefined upon failure.</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>A valid NULL-terminated name which identifies the buffer to bind to. This string should match the object name argument passed to rt_buffer_create().</td>
</tr>
<tr>
<td>timeout</td>
<td>The number of clock ticks to wait for the registration to occur (see note). Passing TM_INFINITE causes the caller to block indefinitely until the object is registered. Passing TM_NONBLOCK causes the service to return immediately without waiting if the object is not registered on entry.</td>
</tr>
</tbody>
</table>
Returns

Zero is returned upon success. Otherwise:

- -EINVAL is returned if \texttt{bf} is not a valid buffer descriptor.

### 6.79.3.2 \texttt{rt_buffer_clear()} 

```c
int rt_buffer_clear (RT_BUFFER * bf )
```

Clear an IPC buffer.

This routine empties a buffer from any data.

**Parameters**

- \texttt{bf} The buffer descriptor.

**Returns**

Zero is returned upon success. Otherwise:

- -EINVAL is returned if \texttt{bf} is not a valid buffer descriptor.

**Tags**

unrestricted, switch-primary
6.79 Buffer services

6.79.3.3 rt_buffer_create()

```c
int rt_buffer_create (  
    RT_BUFFER ∗ bf,  
    const char ∗ name,  
    size_t bufsz,  
    int mode )
```

Create an IPC buffer.

This routine creates an IPC object that allows tasks to send and receive data asynchronously via a memory buffer. Data may be of an arbitrary length, albeit this IPC is best suited for small to medium-sized messages, since data always have to be copied to the buffer during transit. Large messages may be more efficiently handled by message queues (RT_QUEUE).

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bf</code></td>
<td>The address of a buffer descriptor which can be later used to identify uniquely the created object, upon success of this call.</td>
</tr>
<tr>
<td><code>name</code></td>
<td>An ASCII string standing for the symbolic name of the buffer. When non-NULL and non-empty, a copy of this string is used for indexing the created buffer into the object registry.</td>
</tr>
<tr>
<td><code>bufsz</code></td>
<td>The size of the buffer space available to hold data. The required memory is obtained from the main heap.</td>
</tr>
<tr>
<td><code>mode</code></td>
<td>The buffer creation mode. The following flags can be OR'ed into this bitmask, each of them affecting the new buffer:</td>
</tr>
</tbody>
</table>

- B_FIFO makes tasks pend in FIFO order for reading data from the buffer.
- B_PRIO makes tasks pend in priority order for reading data from the buffer.

This parameter also applies to tasks blocked on the buffer’s write side (see `rt_buffer_write()`).

**Returns**

Zero is returned upon success. Otherwise:

- -EINVAL is returned if `mode` is invalid or `bufsz` is zero.
- -ENOMEM is returned if the system fails to get memory from the main heap in order to create the buffer.
- -EEXIST is returned if the `name` is conflicting with an already registered buffer.
- -EPERM is returned if this service was called from an asynchronous context.

**Tags**

mode-unrestricted, switch-secondary

**Note**

Buffers can be shared by multiple processes which belong to the same Xenomai session.
6.79.3.4 rt_buffer_delete()

```c
int rt_buffer_delete (RT_BUFFER * bf )
```

Delete an IPC buffer.

This routine deletes a buffer object previously created by a call to rt_buffer_create().

Parameters

- `bf` The buffer descriptor.

Returns

Zero is returned upon success. Otherwise:

- -EINVAL is returned if `bf` is not a valid buffer descriptor.
- -EPERM is returned if this service was called from an asynchronous context.

Tags

- mode-unrestricted, switch-secondary

6.79.3.5 rt_buffer_inquire()

```c
int rt_buffer_inquire (RT_BUFFER * bf,
                        RT_BUFFER_INFO * info )
```

Query buffer status.

This routine returns the status information about the specified buffer.

Parameters

- `bf` The buffer descriptor.
- `info` A pointer to the returnbuffer* to copy the information to.

Returns

Zero is returned and status information is written to the structure pointed at by `info` upon success. Otherwise:

- -EINVAL is returned if `bf` is not a valid buffer descriptor.
6.79 Buffer services

Tags

unrestricted, switch-primary

6.79.3.6 rt_buffer_read()

ssize_t rt_buffer_read (  
    RT_BUFFER * bf,  
    void * ptr,  
    size_t len,  
    RTIME timeout ) [inline], [static]

Read from an IPC buffer (with relative scalar timeout).

This routine is a variant of rt_buffer_read_timed() accepting a relative timeout specification expressed
as a scalar value.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bf</td>
<td>The buffer descriptor.</td>
</tr>
<tr>
<td>ptr</td>
<td>A pointer to a memory area which will be written upon success with the received data.</td>
</tr>
<tr>
<td>len</td>
<td>The length in bytes of the memory area pointed to by ptr.</td>
</tr>
<tr>
<td>timeout</td>
<td>A delay expressed in clock ticks.</td>
</tr>
</tbody>
</table>

Tags

xthread-nowait, switch-primary

References rt_buffer_read_timed().

6.79.3.7 rt_buffer_read_timed()

ssize_t rt_buffer_read_timed (  
    RT_BUFFER * bf,  
    void * ptr,  
    size_t len,  
    const struct timespec * abs_timeout )

Read from an IPC buffer.

This routine reads the next message from the specified buffer. If no message is available on entry, the
caller is allowed to block until enough data is written to the buffer, or a timeout elapses.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bf</td>
<td>The buffer descriptor.</td>
</tr>
</tbody>
</table>
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>ptr</em></td>
<td>A pointer to a memory area which will be written upon success with the received data.</td>
</tr>
<tr>
<td><em>len</em></td>
<td>The length in bytes of the memory area pointed to by <em>ptr</em>. Under normal circumstances, <em>rt_buffer_read_timed()</em> only returns entire messages as specified by the <em>len</em> argument, or an error value. However, short reads are allowed when a potential deadlock situation is detected (see note below).</td>
</tr>
<tr>
<td><em>abs_timeout</em></td>
<td>An absolute date expressed in clock ticks, specifying a time limit to wait for a message to be available from the buffer (see note). Passing NULL causes the caller to block indefinitely until enough data is available. Passing <code>{ .tv_sec = 0, .tv_nsec = 0 }</code> causes the service to return immediately without blocking in case not enough data is available.</td>
</tr>
</tbody>
</table>

Returns

The number of bytes read from the buffer is returned upon success. Otherwise:

- **-ETIMEDOUT** is returned if *abs_timeout* is reached before a complete message arrives.
- **-EWOULDBLOCK** is returned if *abs_timeout* is `{ .tv_sec = 0, .tv_nsec = 0 }` and not enough data is immediately available on entry to form a complete message.
- **-EINVAL** is returned if *bf* is not a valid buffer descriptor, or *len* is greater than the actual buffer length.
- **-EIDRM** is returned if *bf* is deleted while the caller was waiting for data. In such event, *bf* is no more valid upon return of this service.
- **-EPERM** is returned if this service should block, but was not called from a Xenomai thread.

Note

A short read (i.e. fewer bytes returned than requested by *len*) may happen whenever a pathological use of the buffer is encountered. This condition only arises when the system detects that one or more writers are waiting for sending data, while a reader would have to wait for receiving a complete message at the same time. For instance, consider the following sequence, involving a 1024-byte buffer (*bf*) and two threads:

```
writer thread > rt_write_buffer(&bf, ptr, 1, TM_INFINITE); (one byte to read, 1023 bytes available for sending)
writer thread > rt_write_buffer(&bf, ptr, 1024, TM_INFINITE); (writer blocks - no space for another 1024-byte message)
reader thread > rt_read_buffer(&bf, ptr, 1024, TM_INFINITE); (short read - a truncated (1-byte) message is returned)
```

In order to prevent both threads to wait for each other indefinitely, a short read is allowed, which may be completed by a subsequent call to *rt_buffer_read()* or *rt_buffer_read_until()*. If that case arises, thread priorities, buffer and/or message lengths should likely be fixed, in order to eliminate such condition.

Tags

*thread-nowait*, *switch-primary*

Referenced by *rt_buffer_read()*, and *rt_buffer_read_until()*.
6.79.3.8 rt_buffer_read_until()

```c
ssize_t rt_buffer_read_until ( 
    RT_BUFFER * bf, 
    void * ptr, 
    size_t len, 
    RTIME abs_timeout ) [inline], [static]
```

Read from an IPC buffer (with absolute scalar timeout).

This routine is a variant of `rt_buffer_read_timed()` accepting an absolute timeout specification expressed as a scalar value.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bf</code></td>
<td>The buffer descriptor.</td>
</tr>
<tr>
<td><code>ptr</code></td>
<td>A pointer to a memory area which will be written upon success with the received data.</td>
</tr>
<tr>
<td><code>len</code></td>
<td>The length in bytes of the memory area pointed to by <code>ptr</code>.</td>
</tr>
<tr>
<td><code>abs_timeout</code></td>
<td>An absolute date expressed in clock ticks.</td>
</tr>
</tbody>
</table>

Tags

- xthread-nowait, switch-primary

References `rt_buffer_read_timed()`.

6.79.3.9 rt_buffer_unbind()

```c
int rt_buffer_unbind ( 
    RT_BUFFER * bf )
```

Unbind from an IPC buffer.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bf</code></td>
<td>The buffer descriptor.</td>
</tr>
</tbody>
</table>

This routine releases a previous binding to an IPC buffer. After this call has returned, the descriptor is no more valid for referencing this object.

Tags

- thread-unrestricted
6.79.3.10  rt_buffer_write()

ssize_t rt_buffer_write (  
    RT_BUFFER ∗ bf,  
    const void ∗ ptr,  
    size_t len,  
    RTIME timeout )  [inline], [static]

Write to an IPC buffer (with relative scalar timeout).

This routine is a variant of rt_buffer_write_timed() accepting a relative timeout specification expressed as a scalar value.

Parameters

<table>
<thead>
<tr>
<th>bf</th>
<th>The buffer descriptor.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ptr</td>
<td>The address of the message data to be written to the buffer.</td>
</tr>
<tr>
<td>len</td>
<td>The length in bytes of the message data.</td>
</tr>
<tr>
<td>timeout</td>
<td>A delay expressed in clock ticks.</td>
</tr>
</tbody>
</table>

Tags

xthread-nowait, switch-primary

References rt_buffer_write_timed().

6.79.3.11  rt_buffer_write_timed()

ssize_t rt_buffer_write_timed (  
    RT_BUFFER ∗ bf,  
    const void ∗ ptr,  
    size_t len,  
    const struct timespec ∗ abs_timeout )

Write to an IPC buffer.

This routine writes a message to the specified buffer. If not enough buffer space is available on entry to hold the message, the caller is allowed to block until enough room is freed, or a timeout elapses, whichever comes first.

Parameters

<table>
<thead>
<tr>
<th>bf</th>
<th>The buffer descriptor.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ptr</td>
<td>The address of the message data to be written to the buffer.</td>
</tr>
<tr>
<td>len</td>
<td>The length in bytes of the message data. Zero is a valid value, in which case the buffer is left untouched, and zero is returned to the caller.</td>
</tr>
<tr>
<td>abs_timeout</td>
<td>An absolute date expressed in clock ticks, specifying a time limit to wait for enough buffer space to be available to hold the message (see note). Passing NULL causes the caller to block indefinitely until enough buffer space is available. Passing { .tv_sec = 0, .tv_nsec = 0 } causes the service to return immediately without blocking in case of buffer space shortage.</td>
</tr>
</tbody>
</table>
Returns

The number of bytes written to the buffer is returned upon success. Otherwise:

- -ETIMEDOUT is returned if the absolute \texttt{abs\_timeout} date is reached before enough buffer space is available to hold the message.
- -EWOULDBLOCK is returned if \texttt{abs\_timeout} is \{ \texttt{.tv\_sec = 0, .tv\_nsec = 0} \} and no buffer space is immediately available on entry to hold the message.
- -EINTR is returned if \texttt{rt\_task\_unblock()} was called for the current task before enough buffer space became available to hold the message.
- -EINVAL is returned if \texttt{bf} is not a valid buffer descriptor, or \texttt{len} is greater than the actual buffer length.
- -EIDRM is returned if \texttt{bf} is deleted while the caller was waiting for buffer space. In such event, \texttt{bf} is no more valid upon return of this service.
- -EPERM is returned if this service should block, but was not called from a Xenomai thread.

Tags

\texttt{xthread\_nowait, switch\_primary}

Note

\texttt{abs\_timeout} is interpreted as a multiple of the Alchemy clock resolution (see \texttt{--alchemy\_clock\_resolution} option, defaults to 1 nanosecond).

Referenced by \texttt{rt\_buffer\_write()}, and \texttt{rt\_buffer\_write\_until()}.

6.79.3.12  \texttt{rt\_buffer\_write\_until()}

\begin{verbatim}
ssize_t rt_buffer_write_until ( 
    RT_BUFFER * bf, 
    const void * ptr, 
    size_t len, 
    RTIME abs_timeout ) [inline], [static]
\end{verbatim}

Write to an IPC buffer (with absolute scalar timeout).

This routine is a variant of \texttt{rt\_buffer\_write\_timed()} accepting an absolute timeout specification expressed as a scalar value.

Parameters

<table>
<thead>
<tr>
<th>bf</th>
<th>The buffer descriptor.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ptr</td>
<td>The address of the message data to be written to the buffer.</td>
</tr>
<tr>
<td>len</td>
<td>The length in bytes of the message data.</td>
</tr>
<tr>
<td>abs_timeout</td>
<td>An absolute date expressed in clock ticks.</td>
</tr>
</tbody>
</table>
Tags

xthread-nowait, switch-primary

References rt_buffer_write_timed().
6.80 Condition variable services

POSIXish condition variable mechanism.

Collaboration diagram for Condition variable services:

Data Structures

- struct RT_COND_INFO
  
  Condition variable status descriptor.

Functions

- int rt_cond_create (RT_COND *cond, const char *name)
  
  Create a condition variable.

- int rt_cond_delete (RT_COND *cond)
  
  Delete a condition variable.

- int rt_cond_signal (RT_COND *cond)
  
  Signal a condition variable.

- int rt_cond_broadcast (RT_COND *cond)
  
  Broadcast a condition variable.

- int rt_cond_wait_timed (RT_COND *cond, RT_MUTEX *mutex, const struct timespec *abs←timeout)
  
  Wait on a condition variable.

- static int rt_cond_wait_until (RT_COND *cond, RT_MUTEX *mutex, RTIME timeout)
  
  Wait on a condition variable (with absolute scalar timeout).

- static int rt_cond_wait (RT_COND *cond, RT_MUTEX *mutex, RTIME timeout)
  
  Wait on a condition variable (with relative scalar timeout).

- int rt_cond_inquire (RT_COND *cond, RT_COND_INFO *info)
  
  Query condition variable status.

- int rt_cond_bind (RT_COND *cond, const char *name, RTIME timeout)
  
  Bind to a condition variable.

- int rt_cond_unbind (RT_COND *cond)
  
  Unbind from a condition variable.
6.80.1 Detailed Description

POSIXish condition variable mechanism.

A condition variable is a synchronization mechanism which allows tasks to suspend execution until some
predicate on some arbitrary shared data is satisfied.

The basic operations on conditions are: signal the condition (when the predicate becomes true), and
wait for the condition, blocking the task execution until another task signals the condition. A condition
variable must always be associated with a mutex, to avoid a well-known race condition where a task
prepares to wait on a condition variable and another task signals the condition just before the first task
actually waits on it.

6.80.2 Function Documentation

6.80.2.1 rt_cond_bind()

```
int rt_cond_bind(
    RT_COND * cond,
    const char * name,
    RTIME timeout )
```

Bind to a condition variable.

This routine creates a new descriptor to refer to an existing condition variable identified by its symbolic
name. If the object not exist on entry, the caller may block until a condition variable of the given name is
created.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| `cond`    | The address of a condition variable descriptor filled in by the operation. Contents of this
            memory is undefined upon failure. |
| `name`    | A valid NULL-terminated name which identifies the condition variable to bind to. This
            string should match the object name argument passed to `rt_cond_create()`. |
| `timeout` | The number of clock ticks to wait for the registration to occur (see note). Passing
            `TM_INFINITE` causes the caller to block indefinitely until the object is registered. Passing
            `TM_NONBLOCK` causes the service to return immediately without waiting if the object is
            not registered on entry. |

Returns

Zero is returned upon success. Otherwise:

- -EINTR is returned if `rt_task_unblock()` was called for the current task before the retrieval has
  completed.
- -EWOULDBLOCK is returned if `timeout` is equal to `TM_NONBLOCK` and the searched object is
  not registered on entry.
6.80 Condition variable services

- ETIMEDOUT is returned if the object cannot be retrieved within the specified amount of time.
- EPERM is returned if this service should block, but was not called from a Xenomai thread.

Tags

xthread-nowait, switch-primary

Note

The timeout value is interpreted as a multiple of the Alchemy clock resolution (see -alchemy-clock-resolution option, defaults to 1 nanosecond).

6.80.2.2 rt_cond_broadcast()

int rt_cond_broadcast (    
    RT_COND * cond )

Broadcast a condition variable.

All tasks currently waiting on the condition variable are immediately unblocked.

Parameters

| cond | The condition variable descriptor. |

Returns

Zero is returned upon success. Otherwise:

- EINVAL is returned if cond is not a valid condition variable descriptor.

Tags

unrestricted, switch-primary

6.80.2.3 rt_cond_create()

int rt_cond_create (    
    RT_COND * cond,    
    const char * name )

Create a condition variable.

Create a synchronization object which allows tasks to suspend execution until some predicate on shared data is satisfied.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>cond</strong></td>
<td>The address of a condition variable descriptor which can be later used to identify uniquely the created object, upon success of this call.</td>
</tr>
<tr>
<td><strong>name</strong></td>
<td>An ASCII string standing for the symbolic name of the condition variable. When non-NULL and non-empty, a copy of this string is used for indexing the created condition variable into the object registry.</td>
</tr>
</tbody>
</table>

Returns

Zero is returned upon success. Otherwise:

- **ENOMEM** is returned if the system fails to get memory from the main heap in order to create the condition variable.
- **EEXIST** is returned if the `name` is conflicting with an already registered condition variable.
- **EPERM** is returned if this service was called from an asynchronous context.

Tags

- `mode-unrestricted`, `switch-secondary`

Note

Condition variables can be shared by multiple processes which belong to the same Xenomai session.

Attention

If the underlying threading library does not support `pthread_condattr_setclock()`, timings with Alchemy condition variables will be based on `CLOCK_REALTIME`, and may therefore be affected by updates to the system date (e.g. NTP). This typically concerns legacy setups based on the linuxthreads library. In the normal case, timings are based on `CLOCK_MONOTONIC`.

6.80.2.4  `rt_cond_delete()`

```c
int rt_cond_delete (RT_COND *cond )
```

Delete a condition variable.

This routine deletes a condition variable object previously created by a call to `rt_cond_create()`.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>cond</strong></td>
<td>The condition variable descriptor.</td>
</tr>
</tbody>
</table>
Returns

Zero is returned upon success. Otherwise:

- -EINVAL is returned if `alarm` is not a valid condition variable descriptor.
- -EPERM is returned if this service was called from an asynchronous context.
- -EBUSY is returned upon an attempt to destroy the object referenced by `cond` while it is referenced (for example, while being used in a `rt_cond_wait()`, `rt_cond_wait_timed()` or `rt_cond_wait_until()` by another task).

Tags

`mode-unrestricted, switch-secondary`

6.80.2.5  `rt_cond_inquire()`

```c
int rt_cond_inquire ( RT_COND * cond,
                    RT_COND_INFO * info )
```

Query condition variable status.

This routine returns the status information about the specified condition variable.

Parameters

| `cond` | The condition variable descriptor. |
| `info` | A pointer to the returnbuffer to copy the information to. |

Returns

Zero is returned and status information is written to the structure pointed at by `info` upon success. Otherwise:

- -EINVAL is returned if `cond` is not a valid condition variable descriptor.

Tags

`unrestricted, switch-primary`

6.80.2.6  `rt_cond_signal()`

```c
int rt_cond_signal ( RT_COND * cond )
```

Signal a condition variable.

If the condition variable `cond` is pended, this routine immediately unblocks the first waiting task (by queuing priority order).
Parameters

cond | The condition variable descriptor.

Returns

Zero is returned upon success. Otherwise:

- EINVAL is returned if cond is not a valid condition variable descriptor.

Tags

unrestricted, switch-primary

6.80.2.7 rt_cond_unbind()

int rt_cond_unbind (RT_COND *cond)

Unbind from a condition variable.

Parameters

cond | The condition variable descriptor.

This routine releases a previous binding to a condition variable. After this call has returned, the descriptor is no more valid for referencing this object.

Tags

thread-unrestricted

6.80.2.8 rt_cond_wait()

int rt_cond_wait (RT_COND *cond,
                 RT_MUTEX *mutex,
                 RTIME timeout) [inline], [static]

Wait on a condition variable (with relative scalar timeout).

This routine is a variant of rt_cond_wait_timed() accepting a relative timeout specification expressed as a scalar value.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cond</td>
<td>The condition variable descriptor.</td>
</tr>
<tr>
<td>mutex</td>
<td>The address of the mutex serializing the access to the shared data.</td>
</tr>
<tr>
<td>timeout</td>
<td>A delay expressed in clock ticks.</td>
</tr>
</tbody>
</table>

Tags

xthread-only, switch-primary

6.80.2.9  rt_cond_wait_timed()

```
int rt_cond_wait_timed (
    RT_COND * cond,
    RT_MUTEX * mutex,
    const struct timespec * abs_timeout )
```

Wait on a condition variable.

This service atomically releases the mutex and blocks the calling task, until the condition variable `cond` is signaled or a timeout occurs, whichever comes first. The mutex is re-acquired before returning from this service.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cond</td>
<td>The condition variable descriptor.</td>
</tr>
<tr>
<td>mutex</td>
<td>The address of the mutex serializing the access to the shared data.</td>
</tr>
<tr>
<td>abs_timeout</td>
<td>An absolute date expressed in clock ticks, specifying a time limit to wait for the condition variable to be signaled (see note). Passing NULL causes the caller to block indefinitely.</td>
</tr>
</tbody>
</table>

Returns

Zero is returned upon success. Otherwise:

- -ETIMEDOUT is returned if `abs_timeout` is reached before the condition variable is signaled.
- -EWOULDBLOCK is returned if `abs_timeout` is `{ .tv_sec = 0, .tv_nsec = 0 }`.
- -EINTR is returned if `rt_task_unblock()` was called for the current task.
- -EINVAL is returned if `cond` is not a valid condition variable descriptor.
- -EIDRM is returned if `cond` is deleted while the caller was waiting on the condition variable. In such event, `cond` is no more valid upon return of this service.
- -EPERM is returned if this service should block, but was not called from a Xenomai thread.

Tags

xthread-only, switch-primary
Note

`abs_timeout` is interpreted as a multiple of the Alchemy clock resolution (see `--alchemy-clock-resolution` option, defaults to 1 nanosecond).

6.80.2.10  `rt_cond_wait_until()`

```c
int rt_cond_wait_until (  
    RT_COND * cond,  
    RT_MUTEX * mutex,  
    RTIME abs_timeout ) [inline], [static]
```

Wait on a condition variable (with absolute scalar timeout).

This routine is a variant of `rt_cond_wait_timed()` accepting an `abs_timeout` specification expressed as a scalar value.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cond</code></td>
<td>The condition variable descriptor.</td>
</tr>
<tr>
<td><code>mutex</code></td>
<td>The address of the mutex serializing the access to the shared data.</td>
</tr>
<tr>
<td><code>abs_timeout</code></td>
<td>An absolute date expressed in clock ticks.</td>
</tr>
</tbody>
</table>

Tags

xthread-only, switch-primary
6.81 Event flag group services

Inter-task notification mechanism based on discrete flags.

Collaboration diagram for Event flag group services:

![Alchemy API to Event flag group services diagram]

Data Structures

- `struct RT_EVENT_INFO`
  
  Event status descriptor.

Macros

- `#define EV_PRIO 0x1 /* Pend by task priority order. */`
  
  Creation flags.

- `#define EV_ANY 0x1 /* Disjunctive wait. */`
  
  Operation flags.

Functions

- `int rt_event_delete (RT_EVENT *event)`
  
  Delete an event flag group.

- `int rt_event_wait_timed (RT_EVENT *event, unsigned int mask, unsigned int *mask_r, int mode, const struct timespec *abs_timeout)`
  
  Wait for an arbitrary set of events.

- `static int rt_event_wait_until (RT_EVENT *event, unsigned int mask, unsigned int *mask_r, int mode, RTIME timeout)`
  
  Wait for an arbitrary set of events (with absolute scalar timeout).

- `static int rt_event_wait (RT_EVENT *event, unsigned int mask, unsigned int *mask_r, int mode, RTIME timeout)`
  
  Wait for an arbitrary set of events (with relative scalar timeout).

- `int rt_event_inquire (RT_EVENT *event, RT_EVENT_INFO *info)`
  
  Query event flag group status.

- `int rt_event_bind (RT_EVENT *event, const char *name, RTIME timeout)`
  
  Bind to an event flag group.

- `int rt_event_unbind (RT_EVENT *event)`
  
  Unbind from an event flag group.

- `int rt_event_create (RT_EVENT *event, const char *name, unsigned int ivalue, int mode)`
  
  Create an event flag group.

- `int rt_event_signal (RT_EVENT *event, unsigned int mask)`
  
  Signal an event.

- `int rt_event_clear (RT_EVENT *event, unsigned int mask, unsigned int *mask_r)`
  
  Clear event flags.
6.81.1 Detailed Description

Inter-task notification mechanism based on discrete flags.

An event flag group is a synchronization object represented by a long-word structure; every available bit in this word represents a user-defined event flag.

When a bit is set, the associated event is said to have occurred. Xenomai tasks can use this mechanism to signal the occurrence of particular events to other tasks.

Tasks can either wait for events to occur in a conjunctive manner (all awaited events must have occurred to satisfy the wait request), or in a disjunctive way (at least one of the awaited events must have occurred to satisfy the wait request).

6.81.2 Macro Definition Documentation

6.81.2.1 EV_ANY

#define EV_ANY 0x1 /* Disjunctive wait. */

Operation flags.

6.81.2.2 EV_PRIO

#define EV_PRIO 0x1 /* Pend by task priority order. */

Creation flags.

6.81.3 Function Documentation

6.81.3.1 rt_event_bind()

int rt_event_bind (  
    RT_EVENT * event,  
    const char * name,  
    RTIME timeout )

Bind to an event flag group.

This routine creates a new descriptor to refer to an existing event flag group identified by its symbolic name. If the object does not exist on entry, the caller may block until an event flag group of the given name is created.
6.81 Event flag group services

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event</td>
<td>The address of an event flag group descriptor filled in by the operation. Contents of this memory is undefined upon failure.</td>
</tr>
<tr>
<td>name</td>
<td>A valid NULL-terminated name which identifies the event flag group to bind to. This string should match the object name argument passed to <code>rt_event_create()</code>.</td>
</tr>
<tr>
<td>timeout</td>
<td>The number of clock ticks to wait for the registration to occur (see note). Passing <code>TM_INFINITE</code> causes the caller to block indefinitely until the object is registered. Passing <code>TM_NONBLOCK</code> causes the service to return immediately without waiting if the object is not registered on entry.</td>
</tr>
</tbody>
</table>

Returns

Zero is returned upon success. Otherwise:

- `-EINTR` is returned if `rt_task_unblock()` was called for the current task before the retrieval has completed.
- `-EWOULDBLOCK` is returned if `timeout` is equal to `TM_NONBLOCK` and the searched object is not registered on entry.
- `-ETIMEDOUT` is returned if the object cannot be retrieved within the specified amount of time.
- `-EPERM` is returned if this service should block, but was not called from a Xenomai thread.

Tags

`xthread-nowait, switch-primary`

Note

The `timeout` value is interpreted as a multiple of the Alchemy clock resolution (see `--alchemy-clock-resolution` option, defaults to 1 nanosecond).

### 6.81.3.2 rt_event_clear()

```c
int rt_event_clear (  
    RT_EVENT * event,  
    unsigned int mask,  
    unsigned int * mask_r )
```

Clear event flags.

This routine clears a set of flags from `event`.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event</td>
<td>The event descriptor.</td>
</tr>
<tr>
<td>mask</td>
<td>The set of event flags to be cleared.</td>
</tr>
<tr>
<td>mask←</td>
<td>If non-NULL, <code>mask_r</code> is the address of a memory location which will receive the previous value of the event flag group before the flags are cleared.</td>
</tr>
</tbody>
</table>
Returns

Zero is returned upon success. Otherwise:

- -EINVAL is returned if `event` is not a valid event flag group descriptor.

Tags

`unrestricted`, `switch-primary`

6.81.3.3 `rt_event_create()`

```c
int rt_event_create (    RT_EVENT ∗event,    const char ∗name,    unsigned int ivalue,    int mode )
```

Create an event flag group.

Event groups provide for task synchronization by allowing a set of flags (or "events") to be waited for and posted atomically. An event group contains a mask of received events; an arbitrary set of event flags can be pended or posted in a single operation.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>event</code></td>
<td>The address of an event descriptor which can be later used to identify uniquely the created object, upon success of this call.</td>
</tr>
<tr>
<td><code>name</code></td>
<td>An ASCII string standing for the symbolic name of the event. When non-NULL and non-empty, a copy of this string is used for indexing the created event into the object registry.</td>
</tr>
<tr>
<td><code>ivalue</code></td>
<td>The initial value of the group's event mask.</td>
</tr>
<tr>
<td><code>mode</code></td>
<td>The event group creation mode. The following flags can be OR'ed into this bitmask:</td>
</tr>
</tbody>
</table>

- EV_FIFO makes tasks pend in FIFO order on the event flag group.
- EV_PRIO makes tasks pend in priority order on the event flag group.

Returns

Zero is returned upon success. Otherwise:

- -EINVAL is returned if `mode` is invalid.
- -ENOMEM is returned if the system fails to get memory from the main heap in order to create the event flag group.
- -EEXIST is returned if the `name` is conflicting with an already registered event flag group.
• -EPERM is returned if this service was called from an asynchronous context.

Tags

mode-unrestricted, switch-secondary

Note

Event flag groups can be shared by multiple processes which belong to the same Xenomai session.

6.81.3.4 rt_event_delete()

int rt_event_delete (RT_EVENT *event)

Delete an event flag group.

This routine deletes a event flag group previously created by a call to rt_event_create().

Parameters

| event | The event descriptor. |

Returns

Zero is returned upon success. Otherwise:

• -EINVAL is returned if event is not a valid event flag group descriptor.
• -EPERM is returned if this service was called from an asynchronous context.

Tags

mode-unrestricted, switch-secondary

6.81.3.5 rt_event_inquire()

int rt_event_inquire (RT_EVENT *event,
                       RT_EVENT_INFO *info)

Query event flag group status.

This routine returns the status information about event.
Parameters

<table>
<thead>
<tr>
<th>event</th>
<th>The event descriptor.</th>
</tr>
</thead>
<tbody>
<tr>
<td>info</td>
<td>A pointer to the return buffer to copy the information to.</td>
</tr>
</tbody>
</table>

Returns

Zero is returned and status information is written to the structure pointed at by info upon success. Otherwise:

- EINVAL is returned if event is not a valid event flag group descriptor.

Tags

unrestricted, switch-primary

6.81.3.6 rt_event_signal()

int rt_event_signal (RT_EVENT * event, unsigned int mask)

Signal an event.

Post a set of flags to event. All tasks having their wait request satisfied as a result of this operation are immediately readied.

Parameters

<table>
<thead>
<tr>
<th>event</th>
<th>The event descriptor.</th>
</tr>
</thead>
<tbody>
<tr>
<td>mask</td>
<td>The set of events to be posted.</td>
</tr>
</tbody>
</table>

Returns

Zero is returned upon success. Otherwise:

- EINVAL is returned if event is not an event flag group descriptor.

Tags

unrestricted, switch-primary
6.81.3.7  rt_event_unbind()

```c
int rt_event_unbind ( 
   RT_EVENT * event )
```

Unbind from an event flag group.
Parameters

| event  | The event descriptor. |

This routine releases a previous binding to an event flag group. After this call has returned, the descriptor is no more valid for referencing this object.

Tags

thread-unrestricted

6.81.3.8 rt_event_wait()

```c
int rt_event_wait (  
   RT_EVENT ∗ event,  
   unsigned int mask,  
   unsigned int ∗ mask_r,  
   int mode,  
   RTIME timeout ) [inline], [static]
```

Wait for an arbitrary set of events (with relative scalar timeout).

This routine is a variant of `rt_event_wait_timed()` accepting a relative timeout specification expressed as a scalar value.

Parameters

| event  | The event descriptor. |
| mask   | The set of bits to wait for. |
| mask←_r | The value of the event mask at the time the task was readied. |
| mode   | The pend mode. |
| timeout | A delay expressed in clock ticks. |

Tags

xthread-nowait, switch-primary

References `rt_event_wait_timed()`.

6.81.3.9 rt_event_wait_timed()

```c
int rt_event_wait_timed (  
   RT_EVENT ∗ event,  
```
6.81 Event flag group services

```c
unsigned int mask,
unsigned int *mask_r,
int mode,
const struct timespec *abs_timeout)
```

Wait for an arbitrary set of events.

Waits for one or more events to be signaled in `event`, or until a timeout elapses.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>event</code></td>
<td>The event descriptor.</td>
</tr>
<tr>
<td><code>mask</code></td>
<td>The set of bits to wait for. Passing zero causes this service to return immediately with a success value; the current value of the event mask is also copied to <code>mask_r</code>.</td>
</tr>
<tr>
<td><code>mask_r</code></td>
<td>The value of the event mask at the time the task was readied.</td>
</tr>
<tr>
<td><code>mode</code></td>
<td>The pend mode. The following flags can be OR’ed into this bitmask, each of them affecting the operation:</td>
</tr>
</tbody>
</table>

- EV_ANY makes the task pend in disjunctive mode (i.e. OR); this means that the request is fulfilled when at least one bit set into `mask` is set in the current event mask.
- EV_ALL makes the task pend in conjunctive mode (i.e. AND); this means that the request is fulfilled when at all bits set into `mask` are set in the current event mask.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>abs_timeout</code></td>
<td>An absolute date expressed in clock ticks, specifying a time limit to wait for the request to be satisfied (see note). Passing NULL causes the caller to block indefinitely until the request is satisfied. Passing <code>{ .tv_sec = 0, .tv_nsec = 0 }</code> causes the service to return without blocking in case the request cannot be satisfied immediately.</td>
</tr>
</tbody>
</table>

**Returns**

Zero is returned upon success. Otherwise:

- -ETIMEDOUT is returned if `abs_timeout` is reached before the request is satisfied.
- -EWOULDBLOCK is returned if `abs_timeout` is `{ .tv_sec = 0, .tv_nsec = 0 }` and the requested flags are not set on entry to the call.
- -EINVAL is returned if `mode` is invalid, or `event` is not a valid event flag group descriptor.
- -EIDRM is returned if `event` is deleted while the caller was sleeping on it. In such a case, `event` is no more valid upon return of this service.
- -EPERM is returned if this service should block, but was not called from a Xenomai thread.

**Tags**

xthread-nowait, switch-primary

Generated by Doxygen
Note

`abs_timeout` value is interpreted as a multiple of the Alchemy clock resolution (see `--alchemy-clock-resolution` option, defaults to 1 nanosecond).

Referenced by `rt_event_wait()`, and `rt_event_wait_until()`.

### 6.81.3.10 rt_event_wait_until()

```c
int rt_event_wait_until ( 
    RT_EVENT * event, 
    unsigned int mask, 
    unsigned int * mask_r, 
    int mode, 
    RTIME abs_timeout ) [inline], [static]
```

Wait for an arbitrary set of events (with absolute scalar timeout).

This routine is a variant of `rt_event_wait_timed()` accepting an absolute timeout specification expressed as a scalar value.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>event</code></td>
<td>The event descriptor.</td>
</tr>
<tr>
<td><code>mask</code></td>
<td>The set of bits to wait for.</td>
</tr>
<tr>
<td><code>mask_r</code></td>
<td>The value of the event mask at the time the task was readied.</td>
</tr>
<tr>
<td><code>mode</code></td>
<td>The pend mode.</td>
</tr>
<tr>
<td><code>abs_timeout</code></td>
<td>An absolute date expressed in clock ticks.</td>
</tr>
</tbody>
</table>

**Tags**

`xthread-nowait, switch-primary`

References `rt_event_wait_timed()`.
6.82 Heap management services

Region of memory dedicated to real-time allocation.

Collaboration diagram for Heap management services:

```
Alchemy API  Heap management services
```

Data Structures

- struct **RT_HEAP_INFO**
  
  Heap status descriptor.

Macros

- #define **H_PRIO** 0x1 /* Pend by task priority order. */
  
  Creation flags.

Functions

- int **rt_heap_create** (RT_HEAP *heap, const char *name, size_t heapsize, int mode)
  
  Create a heap.
- int **rt_heap_delete** (RT_HEAP *heap)
  
  Delete a heap.
- int **rt_heap_alloc_timed** (RT_HEAP *heap, size_t size, const struct timespec *abs_timeout, void ***blockp)
  
  Allocate a block from a heap.
- static int **rt_heap_alloc_until** (RT_HEAP *heap, size_t size, RTIME timeout, void ***blockp)
  
  Allocate a block from a heap (with absolute scalar timeout).
- static int **rt_heap_alloc** (RT_HEAP *heap, size_t size, RTIME timeout, void ***blockp)
  
  Allocate a block from a heap (with relative scalar timeout).
- int **rt_heap_free** (RT_HEAP *heap, void *block)
  
  Release a block to a heap.
- int **rt_heap_inquire** (RT_HEAP *heap, RT_HEAP_INFO *info)
  
  Query heap status.
- int **rt_heap_bind** (RT_HEAP *heap, const char *name, RTIME timeout)
  
  Bind to a heap.
- int **rt_heap_unbind** (RT_HEAP *heap)
  
  Unbind from a heap.
6.82.1 Detailed Description

Region of memory dedicated to real-time allocation.

Heaps are regions of memory used for dynamic memory allocation in a time-bounded fashion. Blocks of memory are allocated and freed in an arbitrary order and the pattern of allocation and size of blocks is not known until run time.

6.82.2 Macro Definition Documentation

6.82.2.1 H_PRIO

#define H_PRIO 0x1 /* Pend by task priority order. */

Creation flags.

6.82.3 Function Documentation

6.82.3.1 rt_heap_alloc()

int rt_heap_alloc (  
    RT_HEAP * heap,  
    size_t size,  
    RTIME timeout,  
    void ** blockp ) [inline], [static]

Allocate a block from a heap (with relative scalar timeout).

This routine is a variant of rt_heap_alloc_timed() accepting a relative timeout specification expressed as a scalar value.

Tags

xthread-nowait, switch-primary

References rt_heap_alloc_timed().

6.82.3.2 rt_heap_alloc_timed()

int rt_heap_alloc_timed (  
    RT_HEAP * heap,  
    size_t size,  
    const struct timespec * abs_timeout,  
    void ** blockp )

Allocate a block from a heap.

This service allocates a block from a given heap, or returns the address of the single memory segment if H_SINGLE was mentioned in the creation mode to rt_heap_create(). When not enough memory is available on entry to this service, tasks may be blocked until their allocation request can be fulfilled.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>heap</td>
<td>The requested size (in bytes) of the block. If the heap is managed as a single-block area (H_SINGLE), this value can be either zero, or the same value given to <code>rt_heap_create()</code>. In that case, the same block covering the entire heap space is returned to all callers of this service.</td>
</tr>
<tr>
<td>size</td>
<td>The requested size (in bytes) of the block. If the heap is managed as a single-block area (H_SINGLE), this value can be either zero, or the same value given to <code>rt_heap_create()</code>. In that case, the same block covering the entire heap space is returned to all callers of this service.</td>
</tr>
<tr>
<td>abs_timeout</td>
<td>An absolute date expressed in clock ticks, specifying a time limit to wait for a block of the requested size to be available from the heap (see note). Passing NULL causes the caller to block indefinitely until a block is available. Passing <code>{ .tv_sec = 0, .tv_nsec = 0 }</code> causes the service to return immediately without blocking in case not block is available.</td>
</tr>
<tr>
<td>blockp</td>
<td>A pointer to a memory location which will be written upon success with the address of the allocated block, or the start address of the single memory segment. In the former case, the block can be freed using <code>rt_heap_free()</code>.</td>
</tr>
</tbody>
</table>

Returns

Zero is returned upon success. Otherwise:

- **-ETIMEDOUT** is returned if `abs_timeout` is reached before a block is available.
- **-EWOULDBLOCK** is returned if `abs_timeout` is equal to `{ .tv_sec = 0, .tv_nsec = 0 }` and no block is immediately available on entry to fulfill the allocation request.
- **-EINTR** is returned if `rt_task_unblock()` was called for the current task before a block became available.
- **-EINVAL** is returned if `heap` is not a valid heap descriptor, or `heap` is managed as a single-block area (i.e. H_SINGLE mode) and `size` is non-zero but does not match the original heap size passed to `rt_heap_create()`.
- **-EIDRM** is returned if `heap` is deleted while the caller was waiting for a block. In such event, `heap` is no more valid upon return of this service.
- **-EPERM** is returned if this service should block, but was not called from a Xenomai thread.

Tags

`xthread-nowait, switch-primary`

Note

If shared multi-processing is enabled (i.e. `--enable-pshared` was passed to the configure script), requests for a block size larger than twice the allocation page size are rounded up to the next page size. The allocation page size is currently 512 bytes long (HOBJ_PAGE_SIZE), which means that any request larger than 1k will be rounded up to the next 512 byte boundary. `abs_timeout` is interpreted as a multiple of the Alchemy clock resolution (see `--alchemy-clock-resolution` option, defaults to 1 nanosecond).

Referenced by `rt_heap_alloc()`, and `rt_heap_alloc_until()`. 

Generated by Doxygen
6.82.3.3  rt_heap_alloc_until()

int rt_heap_alloc_until (RT_HEAP * heap,
                    size_t size,
                    RTIME abs_timeout,
                    void ** blockp) [inline], [static]

Allocate a block from a heap (with absolute scalar timeout).

This routine is a variant of rt_heap_alloc_timed() accepting an absolute timeout specification expressed
as a scalar value.

Tags
  xthread-nowait, switch-primary

References rt_heap_alloc_timed().

6.82.3.4  rt_heap_bind()

int rt_heap_bind (RT_HEAP * heap,
                    const char * name,
                    RTIME timeout)

Bind to a heap.

This routine creates a new descriptor to refer to an existing heap identified by its symbolic name. If the
object does not exist on entry, the caller may block until a heap of the given name is created.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| heap      | The address of a heap descriptor filled in by the operation. Contents of this memory is
           | undefined upon failure. |
| name      | A valid NULL-terminated name which identifies the heap to bind to. This string should
           | match the object name argument passed to rt_heap_create(). |
| timeout   | The number of clock ticks to wait for the registration to occur (see note). Passing
            | TM_INFINITE causes the caller to block indefinitely until the object is registered. Passing
            | TM_NONBLOCK causes the service to return immediately without waiting if the object is
            | not registered on entry. |

Returns

Zero is returned upon success. Otherwise:

- -EINTR is returned if rt_task_unblock() was called for the current task before the retrieval has
  completed.
• -EWOULDBLOCK is returned if *timeout* is equal to TM_NONBLOCK and the searched object is not registered on entry.
• -ETIMEDOUT is returned if the object cannot be retrieved within the specified amount of time.
• -EPERM is returned if this service should block, but was not called from a Xenomai thread.

**Tags**

xthread-nowait, switch-primary

**Note**

The *timeout* value is interpreted as a multiple of the Alchemy clock resolution (see –alchemy-clock-resolution option, defaults to 1 nanosecond).

### 6.82.3.5 rt_heap_create()

```c
int rt_heap_create (  
   RT_HEAP ∗ heap,  
   const char ∗ name,  
   size_t heapsz,  
   int mode )
```

Create a heap.

This routine creates a memory heap suitable for time-bounded allocation requests of RAM chunks. When not enough memory is available, tasks may be blocked until their allocation request can be fulfilled.

By default, heaps support allocation of multiple blocks of memory in an arbitrary order. However, it is possible to ask for single-block management by passing the H_SINGLE flag into the mode parameter, in which case the entire memory space managed by the heap is made available as a unique block. In this mode, all allocation requests made through rt_heap_alloc() will return the same block address, pointing at the beginning of the heap memory.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>heap</td>
<td>The address of a heap descriptor which can be later used to identify uniquely the created object, upon success of this call.</td>
</tr>
<tr>
<td>name</td>
<td>An ASCII string standing for the symbolic name of the heap. When non-NULL and non-empty, a copy of this string is used for indexing the created heap into the object registry.</td>
</tr>
<tr>
<td>heapsz</td>
<td>The size (in bytes) of the memory pool, blocks will be claimed and released to. This area is not extensible, so this value must be compatible with the highest memory pressure that could be expected. The valid range is between 1 byte and 2Gb.</td>
</tr>
<tr>
<td>mode</td>
<td>The heap creation mode. The following flags can be OR'ed into this bitmask, each of them affecting the new heap:</td>
</tr>
</tbody>
</table>

• H_FIFO makes tasks pend in FIFO order on the heap when waiting for available blocks.
- H_PRIO makes tasks pend in priority order on the heap when waiting for available blocks.
- H_SINGE causes the entire heap space to be managed as a single memory block.

Returns
Zero is returned upon success. Otherwise:

- EINVAL is returned if mode is invalid, or heapsz is zero or larger than 2Gb.
- ENOMEM is returned if the system fails to get memory from the main heap in order to create the heap.
- EEXIST is returned if the name is conflicting with an already registered heap.
- EPERM is returned if this service was called from an asynchronous context.

Tags
mode-unrestricted, switch-secondary

Note
Heaps can be shared by multiple processes which belong to the same Xenomai session.

6.82.3.6 rt_heap_delete()

int rt_heap_delete ( RT_HEAP * heap )

Delete a heap.

This routine deletes a heap object previously created by a call to rt_heap_create(), releasing all tasks currently blocked on it.

Parameters

| heap | The heap descriptor. |

Returns
Zero is returned upon success. Otherwise:

- EINVAL is returned if heap is not a valid heap descriptor.
- EPERM is returned if this service was called from an asynchronous context.

Tags
mode-unrestricted, switch-secondary
6.82.3.7 rt_heap_free()

```c
int rt_heap_free ( RT_HEAP * heap, void * block )
```

Release a block to a heap.

This service should be used to release a block to the heap it belongs to. An attempt to fulfill the request of every task blocked on `rt_heap_alloc()` is made once `block` is returned to the memory pool.

**Parameters**

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>heap</td>
<td>The heap descriptor.</td>
</tr>
<tr>
<td>block</td>
<td>The address of the block to free.</td>
</tr>
</tbody>
</table>

**Returns**

Zero is returned upon success. Otherwise:

- `EINVAL` is returned if `heap` is not a valid heap descriptor, or `block` is not a valid block previously allocated by the `rt_heap_alloc()` service from `heap`.

**Tags**

unrestricted, switch-primary

6.82.3.8 rt_heap_inquire()

```c
int rt_heap_inquire ( RT_HEAP * heap, RT_HEAP_INFO * info )
```

Query heap status.

This routine returns the status information about `heap`.

**Parameters**

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>heap</td>
<td>The heap descriptor.</td>
</tr>
<tr>
<td>info</td>
<td>A pointer to the returnbuffer to copy the information to.</td>
</tr>
</tbody>
</table>

**Returns**

Zero is returned and status information is written to the structure pointed at by `info` upon success. Otherwise:

- `EINVAL` is returned if `heap` is not a valid heap descriptor.
6.82.3.9  rt_heap_unbind()

int rt_heap_unbind (  
    RT_HEAP * heap  )

Unbind from a heap.

Parameters

| heap | The heap descriptor. |

This routine releases a previous binding to a heap. After this call has returned, the descriptor is no more valid for referencing this object.

Tags

- thread-unrestricted
6.83 Alchemy API

A programming interface reminiscent from traditional RTOS APIs.

Collaboration diagram for Alchemy API:

- Heap management services
- Message pipe services
- Semaphore services
- Message queue services
- Task management services
- Buffer services
- Event flag group services
- Condition variable services
- Alarm services
- Timer management services
- Mutex services

Modules

- **Alarm services**
  General-purpose watchdog timers.
- **Buffer services**
  Lightweight FIFO IPC mechanism.
- **Condition variable services**
  
  POSIXish condition variable mechanism.

- **Event flag group services**
  
  Inter-task notification mechanism based on discrete flags.

- **Heap management services**
  
  Region of memory dedicated to real-time allocation.

- **Mutex services**
  
  POSIXish mutual exclusion services.

- **Message pipe services**
  
  Two-way communication channel between Xenomai & Linux domains.

- **Message queue services**
  
  real-time IPC mechanism for sending messages of arbitrary size

- **Semaphore services**
  
  Counting semaphore IPC mechanism.

- **Task management services**
  
  Services dealing with preemptive multi-tasking.

- **Timer management services**
  
  Services for reading and spinning on the hardware timer.

### 6.83.1 Detailed Description

A programming interface reminiscent from traditional RTOS APIs.

This interface is an evolution of the former native API available with the Xenomai 2.x series.
6.84 Mutex services

POSIXish mutual exclusion services.

Collaboration diagram for Mutex services:

![Collaboration diagram](image)

Data Structures

- struct RT_MUTEX_INFO
  Mutex status descriptor.

Functions

- int rt_mutex_create (RT_MUTEX *mutex, const char *name)
  Create a mutex.
- int rt_mutex_delete (RT_MUTEX *mutex)
  Delete a mutex.
- int rt_mutex_acquire_timed (RT_MUTEX *mutex, const struct timespec *abs_timeout)
  Acquire/lock a mutex (with absolute timeout date).
- static int rt_mutex_acquire_until (RT_MUTEX *mutex, RTIME timeout)
  Acquire/lock a mutex (with absolute scalar timeout).
- static int rt_mutex_acquire (RT_MUTEX *mutex, RTIME timeout)
  Acquire/lock a mutex (with relative scalar timeout).
- int rt_mutex_release (RT_MUTEX *mutex)
  Release/unlock a mutex.
- int rt_mutex_inquire (RT_MUTEX *mutex, RT_MUTEX_INFO *info)
  Query mutex status.
- int rt_mutex_bind (RT_MUTEX *mutex, const char *name, RTIME timeout)
  Bind to a mutex.
- int rt_mutex_unbind (RT_MUTEX *mutex)
  Unbind from a mutex.

6.84.1 Detailed Description

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

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

Xenomai mutex services enforce a priority inheritance protocol in order to solve priority inversions.
6.84.2 Function Documentation

6.84.2.1 rt_mutex_acquire()

```c
int rt_mutex_acquire (  
    RT_MUTEX *mutex,  
    RTIME timeout ) [inline], [static]
```

Acquire/lock a mutex (with relative scalar timeout).

This routine is a variant of `rt_mutex_acquire_timed()` accepting a relative timeout specification expressed as a scalar value.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>mutex</code></td>
<td>The mutex descriptor.</td>
</tr>
<tr>
<td><code>timeout</code></td>
<td>A delay expressed in clock ticks.</td>
</tr>
</tbody>
</table>

Tags

- xthread-only, switch-primary

6.84.2.2 rt_mutex_acquire_timed()

```c
int rt_mutex_acquire_timed (  
    RT_MUTEX *mutex,  
    const struct timespec *abs_timeout )
```

Acquire/lock a mutex (with absolute timeout date).

Attempt to lock a mutex. The calling task is blocked until the mutex is available, in which case it is locked again before this service returns. Xenomai mutexes are implicitly recursive and implement the priority inheritance protocol.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>mutex</code></td>
<td>The mutex descriptor.</td>
</tr>
<tr>
<td><code>abs_timeout</code></td>
<td>An absolute date expressed in clock ticks, specifying a time limit to wait for the mutex to be available (see note). Passing NULL the caller to block indefinitely. Passing <code>{ .tv_sec = 0, .tv_nsec = 0 }</code> causes the service to return immediately without blocking in case <code>mutex</code> is already locked by another task.</td>
</tr>
</tbody>
</table>
Returns

Zero is returned upon success. Otherwise:

- **-ETIMEDOUT** is returned if `abs_timeout` is reached before the mutex is available.
- **-EWOULDBLOCK** is returned if `timeout` is `{ .tv_sec = 0, .tv_nsec = 0 }` and the mutex is not immediately available.
- **-EINTR** is returned if `rt_task_unblock()` was called for the current task.
- **-EINVAL** is returned if `mutex` is not a valid mutex descriptor.
- **-EIDRM** is returned if `mutex` is deleted while the caller was waiting on it. In such event, `mutex` is no more valid upon return of this service.
- **-EPERM** is returned if this service should block, but was not called from a Xenomai thread.

Tags

`xthread-only, switch-primary`

Side effects

Over the Cobalt core, an Alchemy task with priority zero keeps running in primary mode until it releases the mutex, at which point it is switched back to secondary mode automatically.

Note

`abs_timeout` is interpreted as a multiple of the Alchemy clock resolution (see `–alchemy-clock-resolution` option, defaults to 1 nanosecond).

6.84.2.3  *rt_mutex_acquire_until()*

```c
int rt_mutex_acquire_until ( 
    RT_MUTEX * mutex, 
    RTIME abs_timeout ) [inline], [static]
```

Acquire/lock a mutex (with absolute scalar timeout).

This routine is a variant of `rt_mutex_acquire_timed()` accepting an absolute timeout specification expressed as a scalar value.

Parameters

<table>
<thead>
<tr>
<th><code>mutex</code></th>
<th>The mutex descriptor.</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>abs_timeout</code></td>
<td>An absolute date expressed in clock ticks.</td>
</tr>
</tbody>
</table>

Tags

`xthread-only, switch-primary`
6.84.2.4  rt_mutex_bind()

```c
int rt_mutex_bind ( 
    RT_MUTEX * mutex, 
    const char * name, 
    RTIME timeout )
```

Bind to a mutex.

This routine creates a new descriptor to refer to an existing mutex identified by its symbolic name. If the object not exist on entry, the caller may block until a mutex of the given name is created.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mutex</td>
<td>The address of a mutex descriptor filled in by the operation. Contents of this memory is undefined upon failure.</td>
</tr>
<tr>
<td>name</td>
<td>A valid NULL-terminated name which identifies the mutex to bind to. This string should match the object name argument passed to rt_mutex_create().</td>
</tr>
<tr>
<td>timeout</td>
<td>The number of clock ticks to wait for the registration to occur (see note). Passing TM_INFINITE causes the caller to block indefinitely until the object is registered. Passing TM_NONBLOCK causes the service to return immediately without waiting if the object is not registered on entry.</td>
</tr>
</tbody>
</table>

Returns

Zero is returned upon success. Otherwise:

- -EINTR is returned if rt_task_unblock() was called for the current task before the retrieval has completed.
- -EWOULDBLOCK is returned if timeout is equal to TM_NONBLOCK and the searched object is not registered on entry.
- -ETIMEDOUT is returned if the object cannot be retrieved within the specified amount of time.
- -EPERM is returned if this service should block, but was not called from a Xenomai thread.

Tags

xthread-nowait, switch-primary

Note

The timeout value is interpreted as a multiple of the Alchemy clock resolution (see –alchemy-clock-resolution option, defaults to 1 nanosecond).
6.84 Mutex services

### 6.84.2.5 rt_mutex_create()

```c
int rt_mutex_create (RT_MUTEX *mutex, const char *name )
```

Create a mutex.

Create a mutual exclusion object that allows multiple tasks to synchronize access to a shared resource. A mutex is left in an unlocked state after creation.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mutex</td>
<td>The address of a mutex descriptor which can be later used to identify uniquely the created object, upon success of this call.</td>
</tr>
<tr>
<td>name</td>
<td>An ASCII string standing for the symbolic name of the mutex. When non-NULL and non-empty, a copy of this string is used for indexing the created mutex into the object registry.</td>
</tr>
</tbody>
</table>

**Returns**

Zero is returned upon success. Otherwise:

- -ENOMEM is returned if the system fails to get memory from the main heap in order to create the mutex.
- -EEXIST is returned if the name is conflicting with an already registered mutex.
- -EPERM is returned if this service was called from an asynchronous context.

**Tags**

`mode-unrestricted, switch-secondary`

**Note**

Mutexes can be shared by multiple processes which belong to the same Xenomai session.

### 6.84.2.6 rt_mutex_delete()

```c
int rt_mutex_delete (RT_MUTEX *mutex )
```

Delete a mutex.

This routine deletes a mutex object previously created by a call to `rt_mutex_create()`. 
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| mutex     | The mutex descriptor.

Returns

Zero is returned upon success. Otherwise:

- -EINVAL is returned if alarm is not a valid mutex descriptor.
- -EPERM is returned if this service was called from an asynchronous context.
- -EBUSY is returned upon an attempt to destroy the object referenced by mutex while it is referenced (for example, while being used in a rt_mutex_acquire(), rt_mutex_acquire_timed() or rt_mutex_acquire_until() by another task).

Tags

mode-unrestricted, switch-secondary

6.84.2.7 rt_mutex_inquire()

```
int rt_mutex_inquire (  
              RT_MUTEX * mutex,  
              RT_MUTEX_INFO * info )
```

Query mutex status.

This routine returns the status information about the specified mutex.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mutex</td>
<td>The mutex descriptor.</td>
</tr>
<tr>
<td>info</td>
<td>A pointer to the return buffer* to copy the information to.</td>
</tr>
</tbody>
</table>

Returns

Zero is returned and status information is written to the structure pointed at by info upon success. Otherwise:

- -EINVAL is returned if mutex is not a valid mutex descriptor.
- -EPERM is returned if this service is called from an interrupt context.

Tags

xthread-only, switch-primary
6.84 Mutex services

6.84.2.8  
rt_mutex_release()

```c
int rt_mutex_release (RT_MUTEX *mutex)
```

Release/unlock a mutex.

This routine releases a mutex object previously locked by a call to `rt_mutex_acquire()` or `rt_mutex_acquire_until()`. If the mutex is pended, the first waiting task (by priority order) is immediately unblocked and transferred the ownership of the mutex; otherwise, the mutex is left in an unlocked state.

**Parameters**
- `mutex` | The mutex descriptor.

**Returns**
- Zero is returned upon success. Otherwise:
  - `-EINVAL` is returned if `mutex` is not a valid mutex descriptor.
  - `-EPERM` is returned if `mutex` is not owned by the current task, or more generally if this service was called from a context which cannot own any mutex (e.g. interrupt context).

**Tags**
- `xthread-only, switch-primary`

6.84.2.9  
rt_mutex_unbind()

```c
int rt_mutex_unbind (RT_MUTEX *mutex)
```

Unbind from a mutex.

**Parameters**
- `mutex` | The mutex descriptor.

This routine releases a previous binding to a mutex. After this call has returned, the descriptor is no more valid for referencing this object.
6.85 Message pipe services

Two-way communication channel between Xenomai & Linux domains.

Collaboration diagram for Message pipe services:

Macros

- \#define P_MINOR_AUTO XNPIPE_MINOR_AUTO  
  Creation flags.
- \#define P_URGENT XNPIPE_URGENT  
  Operation flags.

Functions

- int rt_pipe_delete (RT_PIPE ∗pipe)  
  Delete a message pipe.
- ssize_t rt_pipe_read_timed (RT_PIPE ∗pipe, void ∗buf, size_t size, const struct timespec ∗abs_←timeout)  
  Read a message from a pipe.
- static ssize_t rt_pipe_read_until (RT_PIPE ∗pipe, void ∗buf, size_t size, RTIME timeout)  
  Read from a pipe (with absolute scalar timeout).
- static ssize_t rt_pipe_read (RT_PIPE ∗pipe, void ∗buf, size_t size, RTIME timeout)  
  Read from a pipe (with relative scalar timeout).
- ssize_t rt_pipe_write (RT_PIPE ∗pipe, const void ∗buf, size_t size, int mode)  
  Write a message to a pipe.
- ssize_t rt_pipe_stream (RT_PIPE ∗pipe, const void ∗buf, size_t size)  
  Stream bytes through a pipe.
- int rt_pipe_bind (RT_PIPE ∗pipe, const char ∗name, RTIME timeout)  
  Bind to a message pipe.
- int rt_pipe_unbind (RT_PIPE ∗pipe)  
  Unbind from a message pipe.
- int rt_pipe_create (RT_PIPE ∗pipe, const char ∗name, int minor, size_t poolsize)  
  Create a message pipe.
6.85.1 Detailed Description

Two-way communication channel between Xenomai & Linux domains.

A message pipe is a two-way communication channel between Xenomai threads and normal Linux threads using regular file I/O operations on a pseudo-device. Pipes can be operated in a message-oriented fashion so that message boundaries are preserved, and also in byte-oriented streaming mode from real-time to normal Linux threads for optimal throughput.

Xenomai threads open their side of the pipe using the rt_pipe_create() service; regular Linux threads do the same by opening one of the /dev/rtpN special devices, where N is the minor number agreed upon between both ends of each pipe.

In addition, named pipes are available through the registry support, which automatically creates a symbolic link from entries under /proc/xenomai/registry/rtipc/xdpp to the corresponding special device file.

Note
Alchemy's message pipes are fully based on the XDDP protocol available from the RTDM/ipc driver.

6.85.2 Macro Definition Documentation

6.85.2.1 P_MINOR_AUTO

#define P_MINOR_AUTO XNPIPE_MINOR_AUTO

Creation flags.

6.85.2.2 P_URGENT

#define P_URGENT XNPIPE_URGENT

Operation flags.

6.85.3 Function Documentation

6.85.3.1 rt_pipe_bind()

int rt_pipe_bind ( RT_PIPE * pipe, const char * name, RTIME timeout )

Bind to a message pipe.

This routine creates a new descriptor to refer to an existing message pipe identified by its symbolic name. If the object does not exist on entry, the caller may block until a pipe of the given name is created.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pipe</td>
<td>The address of a pipe descriptor filled in by the operation. Contents of this memory is undefined upon failure.</td>
</tr>
<tr>
<td>name</td>
<td>A valid NULL-terminated name which identifies the pipe to bind to. This string should match the object name argument passed to rt_pipe_create().</td>
</tr>
<tr>
<td>timeout</td>
<td>The number of clock ticks to wait for the registration to occur (see note). Passing TM_INFINITE causes the caller to block indefinitely until the object is registered. Passing TM_NONBLOCK causes the service to return immediately without waiting if the object is not registered on entry.</td>
</tr>
</tbody>
</table>

Returns

Zero is returned upon success. Otherwise:

- -EINTR is returned if rt_task_unblock() was called for the current task before the retrieval has completed.
- -EWOULDBLOCK is returned if timeout is equal to TM_NONBLOCK and the searched object is not registered on entry.
- -ETIMEDOUT is returned if the object cannot be retrieved within the specified amount of time.
- -EPERM is returned if this service should block, but was not called from a Xenomai thread.

Tags

xthread-nowait

Note

The timeout value is interpreted as a multiple of the Alchemy clock resolution (see –alchemy-clock-resolution option, defaults to 1 nanosecond).

6.85.3.2 rt_pipe_create()

int rt_pipe_create (  
    RT_PIPE * pipe,  
    const char * name,  
    int minor,  
    size_t poolsize )

Create a message pipe.

This service opens a bi-directional communication channel for exchanging messages between Xenomai threads and regular Linux threads. Pipes natively preserve message boundaries, but can also be used in byte-oriented streaming mode from Xenomai to Linux.

rt_pipe_create() always returns immediately, even if no thread has opened the associated special device file yet. On the contrary, the non real-time side could block upon attempt to open the special device file until rt_pipe_create() is issued on the same pipe from a Xenomai thread, unless O_NONBLOCK was given to the open(2) system call.
Named pipes are supported through the use of the registry. Passing a valid `name` parameter when creating a message pipe causes a symbolic link to be created from `/proc/xenomai/registry/rtipc/xddp/name` to the associated special device (i.e. `/dev/rtp*`), so that the specific `minor` information does not need to be known from those processes for opening the proper device file. In such a case, both sides of the pipe only need to agree upon a symbolic name to refer to the same data path, which is especially useful whenever the `minor` number is picked up dynamically using an adaptive algorithm, such as passing `P_MINOR_AUTO` as `minor` value.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pipe</code></td>
<td>The address of a pipe descriptor which can be later used to identify uniquely the created object, upon success of this call.</td>
</tr>
<tr>
<td><code>name</code></td>
<td>An ASCII string standing for the symbolic name of the pipe. When non-NULL and non-empty, a copy of this string is used for indexing the created pipe into the object registry.</td>
</tr>
<tr>
<td><code>minor</code></td>
<td>The minor number of the device associated with the pipe. Passing <code>P_MINOR_AUTO</code> causes the minor number to be auto-allocated. In such a case, a symbolic link will be automatically created from <code>/proc/xenomai/registry/rtipc/xddp/name</code> to the allocated pipe device entry. Valid minor numbers range from 0 to <code>CONFIG_XENO_OPT_PIPE_NRDEV-1</code>.</td>
</tr>
<tr>
<td><code>poolsize</code></td>
<td>Specifies the size of a dedicated buffer pool for the pipe. Passing 0 means that all message allocations for this pipe are performed on the Cobalt core heap.</td>
</tr>
</tbody>
</table>

Returns

The `minor` number assigned to the connection is returned upon success. Otherwise:

- `ENOMEM` is returned if the system fails to get memory from the main heap in order to create the pipe.
- `ENODEV` is returned if `minor` is different from `P_MINOR_AUTO` and is not a valid minor number.
- `EEXIST` is returned if the `name` is conflicting with an already registered pipe.
- `EBUSY` is returned if `minor` is already open.
- `EPERM` is returned if this service was called from an asynchronous context.

Tags

`mode-unrestricted`, `switch-secondary`

6.85.3.3 rt_pipe_delete()

```c
int rt_pipe_delete ( RT_PIPE * pipe )
```

Delete a message pipe.

This routine deletes a pipe object previously created by a call to `rt_pipe_create()`. All resources attached to that pipe are automatically released, all pending data is flushed.
Parameters

| pipe  | The pipe descriptor. |

Returns

Zero is returned upon success. Otherwise:

- -EINVAL is returned if pipe is not a valid pipe descriptor.
- -EIDRM is returned if pipe is a closed pipe descriptor.
- -EPERM is returned if this service was called from an asynchronous context.

Tags

mode-unrestricted, switch-secondary

6.85.3.4 rt_pipe_read()

ssize_t rt_pipe_read ( 
    RT_PIPE ∗ pipe,
    void ∗ buf,
    size_t size,
    RTIME timeout ) [inline], [static]

Read from a pipe (with relative scalar timeout).

This routine is a variant of rt_queue_read_timed() accepting a relative timeout specification expressed as a scalar value.

Parameters

| pipe  | The pipe descriptor. |
| buf   | A pointer to a memory area which will be written upon success with the message received. |
| size  | The count of bytes from the received message to read up into buf. If size is lower than the actual message size, -ENOBUFS is returned since the incompletely received message would be lost. If size is zero, this call returns immediately with no other action. |
| timeout | A delay expressed in clock ticks. |

Tags

xthread-nowait, switch-primary

References rt_pipe_read_timed().
6.85 Message pipe services

6.85.3.5 rt_pipe_read_timed()

ssize_t rt_pipe_read_timed(
    RT_PIPE *pipe,
    void *buf,
    size_t size,
    const struct timespec *abs_timeout)

Read a message from a pipe.

This service reads the next available message from a given pipe.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pipe</td>
<td>The pipe descriptor.</td>
</tr>
<tr>
<td>buf</td>
<td>A pointer to a memory area which will be written upon success with the message received.</td>
</tr>
<tr>
<td>size</td>
<td>The count of bytes from the received message to read up into buf. If size is lower than the actual message size, -ENOBUSF is returned since the incompletely received message would be lost. If size is zero, this call returns immediately with no other action.</td>
</tr>
<tr>
<td>abs_timeout</td>
<td>An absolute date expressed in clock ticks, specifying a time limit to wait for a message to be available from the pipe (see note). Passing NULL causes the caller to block indefinitely until a message is available. Passing { .tv_sec = 0, .tv_nsec = 0 } causes the service to return immediately without blocking in case no message is available.</td>
</tr>
</tbody>
</table>

Returns

The number of bytes available from the received message is returned upon success. Otherwise:

- -ETIMEDOUT is returned if abs_timeout is reached before a message arrives.
- -EWOULDBLOCK is returned if abs_timeout is { .tv_sec = 0, .tv_nsec = 0 } and no message is immediately available on entry to the call.
- -EINTR is returned if rt_task_unblock() was called for the current task before a message was available.
- -EINVAL is returned if pipe is not a valid pipe descriptor.
- -EIDRM is returned if pipe is deleted while the caller was waiting for a message. In such event, pipe is no more valid upon return of this service.
- -EPERM is returned if this service should block, but was not called from a Xenomai thread.

Tags

xthread-nowait, switch-primary

Note

abs_timeout is interpreted as a multiple of the Alchemy clock resolution (see –alchemy-clock-resolution option, defaults to 1 nanosecond).

Referenced by rt_pipe_read(), and rt_pipe_read_until().
6.85.3.6 rt_pipe_read_until()

_ssize_t rt_pipe_read_until (  
   RT_PIPE ∗ pipe,  
   void ∗ buf,  
   size_t size,  
   RTIME abs_timeout ) [inline], [static]

Read from a pipe (with absolute scalar timeout).

This routine is a variant of _rt_queue_read_timed() accepting an absolute timeout specification expressed as a scalar value.

**Parameters**

<table>
<thead>
<tr>
<th>Pipe</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pipe</td>
<td>The pipe descriptor.</td>
</tr>
<tr>
<td>buf</td>
<td>A pointer to a memory area which will be written upon success with the message received.</td>
</tr>
<tr>
<td>size</td>
<td>The count of bytes from the received message to read up into buf. If size is lower than the actual message size, -ENOBUFS is returned since the incompletely received message would be lost. If size is zero, this call returns immediately with no other action.</td>
</tr>
<tr>
<td>abs_timeout</td>
<td>An absolute date expressed in clock ticks.</td>
</tr>
</tbody>
</table>

**Tags**

xthread-nowait, switch-primary

**References** rt_pipe_read_timed().

6.85.3.7 rt_pipe_stream()

_ssize_t rt_pipe_stream (  
   RT_PIPE ∗ pipe,  
   const void ∗ buf,  
   size_t size )

Stream bytes through a pipe.

This service writes a sequence of bytes to be received from the associated special device. Unlike rt.pipe.send(), this service does not preserve message boundaries. Instead, an internal buffer is filled on the fly with the data, which will be consumed as soon as the receiver wakes up.

Data buffers sent by the rt_pipe_stream() service are always transmitted in FIFO order (i.e. P_NORMAL mode).

**Parameters**

<table>
<thead>
<tr>
<th>Pipe</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pipe</td>
<td>The pipe descriptor.</td>
</tr>
<tr>
<td>buf</td>
<td>The address of the first data byte to send. The data will be copied to an internal buffer before transmission.</td>
</tr>
<tr>
<td>size</td>
<td>The size in bytes of the buffer. Zero is a valid value, in which case the service returns immediately without sending any data.</td>
</tr>
</tbody>
</table>
Returns

The number of bytes sent upon success; this value may be lower than \texttt{size}, depending on the available space in the internal buffer. Otherwise:

- \texttt{-EINVAL} is returned if \texttt{mode} is invalid or \texttt{pipe} is not a pipe descriptor.
- \texttt{-ENOMEM} is returned if not enough buffer space is available to complete the operation.
- \texttt{-EIDRM} is returned if \texttt{pipe} is a closed pipe descriptor.

Note

Writing data to a pipe before any peer has opened the associated special device is allowed. The output will be buffered until then, only restricted by the available memory in the associated buffer pool (see \texttt{rt_pipe_create()}).

Tags
\texttt{xcontext, switch-primary}

6.85.3.8 rt_pipe_unbind()

\texttt{int rt_pipe_unbind (}
\texttt{ RT_PIPE * pipe )}

Unbind from a message pipe.

Parameters

\begin{description}
\item[\textit{pipe}] The pipe descriptor.
\end{description}

This routine releases a previous binding to a message pipe. After this call has returned, the descriptor is no more valid for referencing this object.

Tags
\texttt{thread-unrestricted}

6.85.3.9 rt_pipe_write()

\texttt{ssize_t rt_pipe_write (}
\texttt{ RT_PIPE * pipe,}
\texttt{ const void * buf,}
\texttt{ size_t size,}
\texttt{ int mode )}

Generated by Doxygen
Write a message to a pipe.

This service writes a complete message to be received from the associated special device. `rt_pipe_write()` always preserves message boundaries, which means that all data sent through a single call of this service will be gathered in a single read(2) operation from the special device.

This service differs from `rt_pipe_send()` in that it accepts a pointer to the raw data to be sent, instead of a canned message buffer.

**Parameters**

<table>
<thead>
<tr>
<th>pipe</th>
<th>The pipe descriptor.</th>
</tr>
</thead>
<tbody>
<tr>
<td>buf</td>
<td>The address of the first data byte to send. The data will be copied to an internal buffer before transmission.</td>
</tr>
<tr>
<td>size</td>
<td>The size in bytes of the message (payload data only). Zero is a valid value, in which case the service returns immediately without sending any message.</td>
</tr>
<tr>
<td>mode</td>
<td>A set of flags affecting the operation:</td>
</tr>
</tbody>
</table>

- P_URGENT causes the message to be prepended to the output queue, ensuring a LIFO ordering.
- P_NORMAL causes the message to be appended to the output queue, ensuring a FIFO ordering.

**Returns**

Upon success, this service returns `size`. Upon error, one of the following error codes is returned:

- -EINVAL is returned if `mode` is invalid or `pipe` is not a pipe descriptor.
- -ENOMEM is returned if not enough buffer space is available to complete the operation.
- -EIDRM is returned if `pipe` is a closed pipe descriptor.

**Note**

Writing data to a pipe before any peer has opened the associated special device is allowed. The output will be buffered until then, only restricted by the available memory in the associated buffer pool (see `rt_pipe_create()`).

**Tags**

xcontext, switch-primary
6.86 Message queue services

real-time IPC mechanism for sending messages of arbitrary size

Collaboration diagram for Message queue services:

```
Alchemy API  Message queue services
```

Data Structures

- `struct RT_QUEUE_INFO`
  Queue status descriptor.

Macros

- `#define Q_PRIO 0x1 /* Pend by task priority order. */`
  Creation flags.

Functions

- `int rt_queue_create (RT_QUEUE *queue, const char *name, size_t poolsize, size_t qlimit, int mode)`
  Create a message queue.
- `int rt_queue_delete (RT_QUEUE *queue)`
  Delete a message queue.
- `void * rt_queue_alloc (RT_QUEUE *queue, size_t size)`
  Allocate a message buffer.
- `int rt_queue_free (RT_QUEUE *queue, void *buf)`
  Free a message buffer.
- `int rt_queue_send (RT_QUEUE *queue, const void *buf, size_t size, int mode)`
  Send a message to a queue.
- `ssize_t rt_queue_receive_timed (RT_QUEUE *queue, void **bufp, const struct timespec *abs_timeout)`
  Receive a message from a queue (with absolute timeout date).
- `static ssize_t rt_queue_receive_until (RT_QUEUE *queue, void **bufp, RTIME timeout)`
  Receive from a queue (with absolute scalar timeout).
- `static ssize_t rt_queue_receive (RT_QUEUE *queue, void **bufp, RTIME timeout)`
  Receive from a queue (with relative scalar timeout).
- `ssize_t rt_queue_read_timed (RT_QUEUE *queue, void *buf, size_t size, const struct timespec *abs_timeout)`
  Read from a queue.
• static ssize_t rt_queue_read_until (RT_QUEUE *queue, void *buf, size_t size, RTIME timeout)
  Read from a queue (with absolute scalar timeout).
• static ssize_t rt_queue_read (RT_QUEUE *queue, void *buf, size_t size, RTIME timeout)
  Read from a queue (with relative scalar timeout).
• int rt_queue_flush (RT_QUEUE *queue)
  Flush pending messages from a queue.
• int rt_queue_inquire (RT_QUEUE *queue, RT_QUEUE_INFO *info)
  Query queue status.
• int rt_queue_bind (RT_QUEUE *queue, const char *name, RTIME timeout)
  Bind to a message queue.
• int rt_queue_unbind (RT_QUEUE *queue)
  Unbind from a message queue.

6.86.1 Detailed Description

real-time IPC mechanism for sending messages of arbitrary size

Message queueing is a method by which real-time tasks can exchange or pass data through a Xenomai-managed queue of messages. Messages can vary in length and be assigned different types or usages. A message queue can be created by one task and used by multiple tasks that send and/or receive messages to the queue.

6.86.2 Macro Definition Documentation

6.86.2.1 Q_PRIO

#define Q_PRIO 0x1 /* Pend by task priority order. */

Creation flags.

6.86.3 Function Documentation

6.86.3.1 rt_queue_alloc() 

void * rt_queue_alloc ( 
  RT_QUEUE * q, 
  size_t size )

Allocate a message buffer.

This service allocates a message buffer from the queue's internal pool. This buffer can be filled in with payload information, prior enqueuing it by a call to rt_queue_send(). When used in pair, these services provide a zero-copy interface for sending messages.
Parameters

<table>
<thead>
<tr>
<th>q</th>
<th>The queue descriptor.</th>
</tr>
</thead>
<tbody>
<tr>
<td>size</td>
<td>The requested size in bytes of the buffer. Zero is an acceptable value, which means that the message conveys no payload; in this case, the receiver will get a zero-sized message.</td>
</tr>
</tbody>
</table>

Returns

The address of the allocated buffer upon success, or NULL if the call fails.

Tags

unrestricted, switch-primary

6.86.3.2  rt_queue_bind()

int rt_queue_bind ( 
    RT_QUEUE * q, 
    const char * name, 
    RTIME timeout )

Bind to a message queue.

This routine creates a new descriptor to refer to an existing message queue identified by its symbolic name. If the object does not exist on entry, the caller may block until a queue of the given name is created.

Parameters

<table>
<thead>
<tr>
<th>q</th>
<th>The address of a queue descriptor filled in by the operation. Contents of this memory is undefined upon failure.</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>A valid NULL-terminated name which identifies the queue to bind to. This string should match the object name argument passed to rt_queue_create().</td>
</tr>
<tr>
<td>timeout</td>
<td>The number of clock ticks to wait for the registration to occur (see note). Passing TM_INFINITE causes the caller to block indefinitely until the object is registered. Passing TM_NONBLOCK causes the service to return immediately without waiting if the object is not registered on entry.</td>
</tr>
</tbody>
</table>

Returns

Zero is returned upon success. Otherwise:

- EINTR is returned if rt_task_unblock() was called for the current task before the retrieval has completed.
- EWOULDBLOCK is returned if timeout is equal to TM_NONBLOCK and the searched object is not registered on entry.
- ETIMEDOUT is returned if the object cannot be retrieved within the specified amount of time.
-EPERM is returned if this service should block, but was not called from a Xenomai thread.

**Tags**

`xthread-nowait, switch-primary`

**Note**

The `timeout` value is interpreted as a multiple of the Alchemy clock resolution (see `--alchemy-clock-resolution` option, defaults to 1 nanosecond).

### 6.86.3.3 rt_queue_create()

```c
int rt_queue_create (  
    RT_QUEUE ∗ q,  
    const char ∗ name,  
    size_t poolsize,  
    size_t qlimit,  
    int mode )
```

Create a message queue.

Create a message queue object which allows multiple tasks to exchange data through the use of variable-sized messages. A message queue is created empty.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>q</code></td>
<td>The address of a queue descriptor which can be later used to identify uniquely the created object, upon success of this call.</td>
</tr>
<tr>
<td><code>name</code></td>
<td>An ASCII string standing for the symbolic name of the queue. When non-NULL and non-empty, a copy of this string is used for indexing the created queue into the object registry.</td>
</tr>
<tr>
<td><code>poolsize</code></td>
<td>The size (in bytes) of the message buffer pool to be pre-allocated for holding messages. Message buffers will be claimed and released to this pool. The buffer pool memory cannot be extended. See note.</td>
</tr>
<tr>
<td><code>qlimit</code></td>
<td>This parameter allows to limit the maximum number of messages which can be queued at any point in time, sending to a full queue begets an error. The special value <code>Q_UNLIMITED</code> can be passed to disable the limit check.</td>
</tr>
<tr>
<td><code>mode</code></td>
<td>The queue creation mode. The following flags can be OR’ed into this bitmask, each of them affecting the new queue:</td>
</tr>
</tbody>
</table>

- `Q_FIFO` makes tasks pend in FIFO order on the queue for consuming messages.
- `Q_PRIO` makes tasks pend in priority order on the queue.
Returns

Zero is returned upon success. Otherwise:

- -EINVAL is returned if mode is invalid or poolsize is zero.
- -ENOMEM is returned if the system fails to get memory from the main heap in order to create the queue.
- -EEXIST is returned if the name is conflicting with an already registered queue.
- -EPERM is returned if this service was called from an asynchronous context.

Tags

mode-unrestricted, switch-secondary

Note

Queues can be shared by multiple processes which belong to the same Xenomai session. Each message pending into the queue consumes four long words plus the actual payload size, aligned to the next long word boundary. e.g. a 6 byte message on a 32 bit platform would require 24 bytes of storage into the pool.

When qlimit is given (i.e. different from Q_UNLIMITED), this overhead is accounted for automatically, so that qlimit messages of poolsize / qlimit bytes can be stored into the pool concurrently. Otherwise, poolsize is increased by 5% internally to cope with such overhead.

6.86.3.4 rt_queue_delete()

int rt_queue_delete (RT_QUEUE *q)

Delete a message queue.

This routine deletes a queue object previously created by a call to rt_queue_create(). All resources attached to that queue are automatically released, including all pending messages.

Parameters

| q | The queue descriptor. |

Returns

Zero is returned upon success. Otherwise:

- -EINVAL is returned if q is not a valid queue descriptor.
- -EPERM is returned if this service was called from an asynchronous context.

Tags

mode-unrestricted, switch-secondary
### 6.86.3.5 rt_queue_flush()

```c
int rt_queue_flush (RT_QUEUE * q)
```

Flush pending messages from a queue.

This routine flushes all messages currently pending in a queue, releasing all message buffers appropriately.

**Parameters**

- `q` The queue descriptor.

**Returns**

Zero is returned upon success. Otherwise:

- -EINVAL is returned if `q` is not a valid queue descriptor.

**Tags**

unrestricted, switch-primary

### 6.86.3.6 rt_queue_free()

```c
int rt_queue_free (RT_QUEUE * q, void * buf)
```

Free a message buffer.

This service releases a message buffer to the queue’s internal pool.

**Parameters**

- `q` The queue descriptor.
- `buf` The address of the message buffer to free. Even zero-sized messages carrying no payload data must be freed, since they are assigned a valid memory space to store internal information.

**Returns**

Zero is returned upon success, or -EINVAL if `buf` is not a valid message buffer previously allocated by the `rt_queue_alloc()` service, or the caller did not get ownership of the message through a successful return from `rt_queue_receive()`.
6.86 Message queue services

Tags

unrestricted, switch-primary

6.86.3.7 rt_queue_inquire()

```c
int rt_queue_inquire ( 
    RT_QUEUE * q, 
    RT_QUEUE_INFO * info )
```

Query queue status.
This routine returns the status information about the specified queue.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>q</td>
<td>The queue descriptor.</td>
</tr>
<tr>
<td>info</td>
<td>A pointer to the return buffer to copy the information to.</td>
</tr>
</tbody>
</table>

Returns

Zero is returned and status information is written to the structure pointed at by info upon success. Otherwise:

- EINVAL is returned if q is not a valid queue descriptor.

Tags

unrestricted, switch-primary

6.86.3.8 rt_queue_read()

```c
ssize_t rt_queue_read ( 
    RT_QUEUE * q, 
    void * buf, 
    size_t size, 
    RTIME timeout ) [inline], [static]
```

Read from a queue (with relative scalar timeout).

This routine is a variant of rt_queue_read_timed() accepting a relative timeout specification expressed as a scalar value.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>q</td>
<td>The queue descriptor.</td>
</tr>
<tr>
<td>buf</td>
<td>A pointer to a memory area which will be written upon success with the received message payload.</td>
</tr>
<tr>
<td>size</td>
<td>The length in bytes of the memory area pointed to by buf.</td>
</tr>
<tr>
<td>timeout</td>
<td>A delay expressed in clock ticks.</td>
</tr>
</tbody>
</table>
Tags

xthread-nowait, switch-primary

References rt_queue_read_timed().

6.86.3.9 rt_queue_read_timed()

ssize_t rt_queue_read_timed(
    RT_QUEUE *q,
    void *buf,
    size_t size,
    const struct timespec *abs_timeout)

Read from a queue.

This service reads the next available message from a given queue.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>q</td>
<td>The queue descriptor.</td>
</tr>
<tr>
<td>buf</td>
<td>A pointer to a memory area which will be written upon success with the received message payload. The internal message buffer conveying the data is automatically freed by this call. If --enable-pshared is enabled in the configuration, buf must have been obtained from the Xenomai memory allocator via xnmalloc() or any service based on it, such as rt_heap_alloc().</td>
</tr>
<tr>
<td>size</td>
<td>The length in bytes of the memory area pointed to by buf. Messages larger than size are truncated appropriately.</td>
</tr>
<tr>
<td>abs_timeout</td>
<td>An absolute date expressed in clock ticks, specifying a time limit to wait for a message to be available from the queue (see note). Passing NULL causes the caller to block indefinitely until a message is available. Passing { .tv_sec = 0, .tv_nsec = 0 } causes the service to return immediately without blocking in case no message is available.</td>
</tr>
</tbody>
</table>

Returns

The number of bytes copied to buf is returned upon success. Zero is a possible value corresponding to a zero-sized message passed to rt_queue_send() or rt_queue_write(). Otherwise:

- ETIMEDOUT is returned if abs_timeout is reached before a message arrives.
- EWOULDBLOCK is returned if abs_timeout is { .tv_sec = 0, .tv_nsec = 0 } and no message is immediately available on entry to the call.
- EINTR is returned if rt_task_unblock() was called for the current task before a message was available.
- EINVAL is returned if q is not a valid queue descriptor.
- EIDRM is returned if q is deleted while the caller was waiting for a message. In such event, q is no more valid upon return of this service.
- EPERM is returned if this service should block, but was not called from a Xenomai thread.
Tags

\texttt{xthread-nowait, switch-primary}

Note

\textit{abs\_timeout} is interpreted as a multiple of the Alchemy clock resolution (see \texttt{--alchemy-clock-resolution} option, defaults to 1 nanosecond).

Referenced by \texttt{rt\_queue\_read()}, and \texttt{rt\_queue\_read\_until()}.

\subsection{6.86.3.10 \texttt{rt\_queue\_read\_until()}}

\begin{verbatim}
ssize_t rt_queue_read_until (RT_QUEUE * q, void * buf, size_t size, RTIME abs_timeout) [inline], [static]
\end{verbatim}

Read from a queue (with absolute scalar timeout).

This routine is a variant of \texttt{rt\_queue\_read\_timed()} accepting an absolute timeout specification expressed as a scalar value.

Parameters

\begin{tabular}{|l|p{10cm}|}
\hline
\texttt{q} & The queue descriptor. \\
\texttt{buf} & A pointer to a memory area which will be written upon success with the received message payload. \\
\texttt{size} & The length in bytes of the memory area pointed to by \texttt{buf}. \\
\texttt{abs\_timeout} & An absolute date expressed in clock ticks. \\
\hline
\end{tabular}

Tags

\texttt{xthread-nowait, switch-primary}

References \texttt{rt\_queue\_read\_timed()}. 

\subsection{6.86.3.11 \texttt{rt\_queue\_receive()}}

\begin{verbatim}
ssize_t rt_queue_receive (RT_QUEUE * q, void ** bufp, RTIME timeout) [inline], [static]
\end{verbatim}

Receive from a queue (with relative scalar timeout).

This routine is a variant of \texttt{rt\_queue\_receive\_timed()} accepting a relative timeout specification expressed as a scalar value.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>q</code></td>
<td>The queue descriptor.</td>
</tr>
<tr>
<td><code>bufp</code></td>
<td>A pointer to a memory location which will be written with the address of the received message.</td>
</tr>
<tr>
<td><code>timeout</code></td>
<td>A delay expressed in clock ticks.</td>
</tr>
</tbody>
</table>

Tags

`xthread-nowait, switch-primary`

References `rt_queue_receive_timed()`.

6.86.3.12 `rt_queue_receive_timed()`

```c
ssize_t rt_queue_receive_timed (  
    RT_QUEUE *q,  
    void **bufp,  
    const struct timespec *abs_timeout )
```

Receive a message from a queue (with absolute timeout date).

This service receives the next available message from a given queue.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>q</code></td>
<td>The queue descriptor.</td>
</tr>
<tr>
<td><code>bufp</code></td>
<td>A pointer to a memory location which will be written with the address of the received message, upon success. Once consumed, the message space should be freed using <code>rt_queue_free()</code>.</td>
</tr>
<tr>
<td><code>abs_timeout</code></td>
<td>An absolute date expressed in clock ticks, specifying a time limit to wait for a message to be available from the queue (see note). Passing NULL causes the caller to block indefinitely until a message is available. Passing <code>{ .tv_sec = 0, .tv_nsec = 0 }</code> causes the service to return immediately without blocking in case no message is available.</td>
</tr>
</tbody>
</table>

Returns

The number of bytes available from the received message is returned upon success. Zero is a possible value corresponding to a zero-sized message passed to `rt_queue_send()` or `rt_queue_write()`. Otherwise:

- `-ETIMEDOUT` is returned if `abs_timeout` is reached before a message arrives.
- `-EWOULDBLOCK` is returned if `abs_timeout` is `{ .tv_sec = 0, .tv_nsec = 0 }` and no message is immediately available on entry to the call.
- `-EINTR` is returned if `rt_task_unblock()` was called for the current task before a message was available.
6.86 Message queue services

- EINVAL is returned if \( q \) is not a valid queue descriptor.
- EIDRM is returned if \( q \) is deleted while the caller was waiting for a message. In such event, \( q \) is no more valid upon return of this service.
- EPERM is returned if this service should block, but was not called from a Xenomai thread.

Tags

xthread-nowait, switch-primary

Note

abs_timeout is interpreted as a multiple of the Alchemy clock resolution (see –alchemy-clock-resolution option, defaults to 1 nanosecond).

Referenced by rt_queue_receive(), and rt_queue_receive_until().

6.86.3.13 rt_queue_receive_until()

ssize_t rt_queue_receive_until (  
  RT_QUEUE * q,  
  void ** bufp,  
  RTIME abs_timeout ) [inline], [static]

Receive from a queue (with absolute scalar timeout).

This routine is a variant of rt_queue_receive_timed() accepting an absolute timeout specification expressed as a scalar value.

Parameters

| \( q \) | The queue descriptor. |
| \( bufp \) | A pointer to a memory location which will be written with the address of the received message. |
| abs_timeout | An absolute date expressed in clock ticks. |

Tags

xthread-nowait, switch-primary

References rt_queue_receive_timed().

6.86.3.14 rt_queue_send()

int rt_queue_send (  
  RT_QUEUE * q,  

Generated by Doxygen
const void * buf,
size_t size,
int mode )

Send a message to a queue.

This service sends a complete message to a given queue. The message must have been allocated by a previous call to rt_queue_alloc().

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>q</td>
<td>The queue descriptor.</td>
</tr>
<tr>
<td>buf</td>
<td>The address of the message buffer to be sent. The message buffer must have been allocated using the rt_queue_alloc() service. Once passed to rt_queue_send(), the memory pointed to by buf is no more under the control of the sender and thus should not be referenced by it anymore; deallocation of this memory must be handled on the receiving side.</td>
</tr>
<tr>
<td>size</td>
<td>The actual size in bytes of the message, which may be lower than the allocated size for the buffer obtained from rt_queue_alloc(). Zero is a valid value, in which case an empty message will be sent.</td>
</tr>
<tr>
<td>mode</td>
<td>A set of flags affecting the operation:</td>
</tr>
<tr>
<td></td>
<td>• Q_URGENT causes the message to be prepended to the message queue, ensuring a LIFO ordering.</td>
</tr>
<tr>
<td></td>
<td>• Q_NORMAL causes the message to be appended to the message queue, ensuring a FIFO ordering.</td>
</tr>
<tr>
<td></td>
<td>• Q_BROADCAST causes the message to be sent to all tasks currently waiting for messages. The message is not copied; a reference count is maintained instead so that the message will remain valid until the last receiver releases its own reference using rt_queue_free(), after which the message space will be returned to the queue's internal pool.</td>
</tr>
</tbody>
</table>

Returns

Upon success, this service returns the number of receivers which got awaken as a result of the operation. If zero is returned, no task was waiting on the receiving side of the queue, and the message has been enqueued. Upon error, one of the following error codes is returned:

- -EINVAL is returned if q is not a message queue descriptor, mode is invalid, or buf is NULL.
- -ENOMEM is returned if queuing the message would exceed the limit defined for the queue at creation.

Tags

unrestricted, switch-primary

6.86.3.15 rt_queue_unbind()

int rt_queue_unbind (  
    RT_QUEUE * q )

Unbind from a message queue.
Parameters

| q  | The queue descriptor. |

This routine releases a previous binding to a message queue. After this call has returned, the descriptor is no more valid for referencing this object.

Tags

- thread-unrestricted
6.87 Semaphore services

Counting semaphore IPC mechanism.

Collaboration diagram for Semaphore services:

![Collaboration diagram for Semaphore services](image)

Data Structures

- `struct RT_SEM_INFO`
  Semaphore status descriptor.

Macros

- `#define S_PRIO 0x1 /* Pend by task priority order. */`
  Creation flags.

Functions

- `int rt_sem_create (RT_SEM *sem, const char *name, unsigned long icount, int mode)`
  Create a counting semaphore.

- `int rt_sem_delete (RT_SEM *sem)`
  Delete a semaphore.

- `int rt_sem_p_timed (RT_SEM *sem, const struct timespec *abs_timeout)`
  Pend on a semaphore.

- `static int rt_sem_p_until (RT_SEM *sem, RTIME timeout)`
  Pend on a semaphore (with absolute scalar timeout).

- `static int rt_sem_p (RT_SEM *sem, RTIME timeout)`
  Pend on a semaphore (with relative scalar timeout).

- `int rt_sem_v (RT_SEM *sem)`
  Signal a semaphore.

- `int rt_sem_broadcast (RT_SEM *sem)`
  Broadcast a semaphore.

- `int rt_sem_inquire (RT_SEM *sem, RT_SEM_INFO *info)`
  Query semaphore status.

- `int rt_sem_bind (RT_SEM *sem, const char *name, RTIME timeout)`
  Bind to a semaphore.

- `int rt_sem_unbind (RT_SEM *sem)`
  Unbind from a semaphore.
6.87 Semaphore services

6.87.1 Detailed Description

Counting semaphore IPC mechanism.

A counting semaphore is a synchronization object for controlling the concurrency level allowed in accessing a resource from multiple real-time tasks, based on the value of a count variable accessed atomically. The semaphore is used through the P ("Proberen", from the Dutch "test and decrement") and V ("Verhogen", increment) operations. The P operation decrements the semaphore count by one if non-zero, or waits until a V operation is issued by another task. Conversely, the V operation releases a resource by incrementing the count by one, unblocking the heading task waiting on the P operation if any. Waiting on a semaphore may cause a priority inversion.

If no more than a single resource is made available at any point in time, the semaphore enforces mutual exclusion and thus can be used to serialize access to a critical section. However, mutexes should be used instead in order to prevent priority inversions, based on the priority inheritance protocol.

6.87.2 Macro Definition Documentation

6.87.2.1 S_PRIO

#define S_PRIO 0x1 /* Pend by task priority order. */

Creation flags.

6.87.3 Function Documentation

6.87.3.1 rt_sem_bind()

int rt_sem_bind ( 
    RT_SEM *sem,
    const char *name,
    RTIME timeout )

Bind to a semaphore.

This routine creates a new descriptor to refer to an existing semaphore identified by its symbolic name. If the object does not exist on entry, the caller may block until a semaphore of the given name is created.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sem</td>
<td>The address of a semaphore descriptor filled in by the operation. Contents of this memory is undefined upon failure.</td>
</tr>
<tr>
<td>name</td>
<td>A valid NULL-terminated name which identifies the semaphore to bind to. This string should match the object name argument passed to rt_sem_create().</td>
</tr>
<tr>
<td>timeout</td>
<td>The number of clock ticks to wait for the registration to occur (see note). Passing TM_INFINITE causes the caller to block indefinitely until the object is registered. Passing TM_NONBLOCK causes the service to return immediately without waiting if the object is not registered on entry.</td>
</tr>
</tbody>
</table>

Generated by Doxygen
Returns

Zero is returned upon success. Otherwise:

- -EINTR is returned if `rt_task_unblock()` was called for the current task before the retrieval has completed.
- -EWOULDBLOCK is returned if `timeout` is equal to TM_NONBLOCK and the searched object is not registered on entry.
- -ETIMEDOUT is returned if the object cannot be retrieved within the specified amount of time.
- -EPERM is returned if this service should block, but was not called from a Xenomai thread.

Tags

`xthread-nowait, switch-primary`

Note

The `timeout` value is interpreted as a multiple of the Alchemy clock resolution (see –alchemy-clock-resolution option, defaults to 1 nanosecond).

6.87.3.2  `rt_sem_broadcast()`

```c
int rt_sem_broadcast ( RT_SEM *sem )
```

Broadcast a semaphore.

All tasks currently waiting on the semaphore are immediately unblocked. The semaphore count is set to zero.

Parameters

- `sem` The semaphore descriptor.

Returns

Zero is returned upon success. Otherwise:

- -EINVAL is returned if `sem` is not a valid semaphore descriptor.

Tags

`unrestricted`
6.87.3.3  rt_sem_create()

```c
int rt_sem_create ( 
    RT_SEM * sem, 
    const char * name, 
    unsigned long icount, 
    int mode )
```

Create a counting semaphore.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sem</code></td>
<td>The address of a semaphore descriptor which can be later used to identify uniquely the created object, upon success of this call.</td>
</tr>
<tr>
<td><code>name</code></td>
<td>An ASCII string standing for the symbolic name of the semaphore. When non-NULL and non-empty, a copy of this string is used for indexing the created semaphore into the object registry.</td>
</tr>
<tr>
<td><code>icount</code></td>
<td>The initial value of the counting semaphore.</td>
</tr>
<tr>
<td><code>mode</code></td>
<td>The semaphore creation mode. The following flags can be OR'ed into this bitmask:</td>
</tr>
</tbody>
</table>

- S_FIFO makes tasks pend in FIFO order on the semaphore.
- S_PRIO makes tasks pend in priority order on the semaphore.
- S_PULSE causes the semaphore to behave in "pulse" mode. In this mode, the V (signal) operation attempts to release a single waiter each time it is called, without incrementing the semaphore count, even if no waiter is pending. For this reason, the semaphore count in pulse mode remains zero.

Returns

Zero is returned upon success. Otherwise:

- -EINVAL is returned if `icount` is non-zero and S_PULSE is set in `mode`, or `mode` is otherwise invalid.
- -ENOMEM is returned if the system fails to get memory from the main heap in order to create the semaphore.
- -EEXIST is returned if the `name` is conflicting with an already registered semaphore.
- -EPERM is returned if this service was called from an asynchronous context.

Tags

- mode-unrestricted, switch-secondary

Note

Semaphores can be shared by multiple processes which belong to the same Xenomai session.
6.87.3.4  rt_sem_delete()

int rt_sem_delete (  
    RTSEM * sem  )

Delete a semaphore.

This routine deletes a semaphore previously created by a call to rt_sem_create().
Parameters

| sem  | The semaphore descriptor. |

Returns

Zero is returned upon success. Otherwise:

- -EINVAL is returned if `sem` is not a valid semaphore descriptor.
- -EPERM is returned if this service was called from an asynchronous context.

Tags

mode-unrestricted, switch-secondary

6.87.3.5  rt_sem_inquire()

```c
int rt_sem_inquire (
    RT_SEM * sem,
    RT_SEM_INFO * info )
```

Query semaphore status.

This routine returns the status information about the specified semaphore.

Parameters

| sem  | The semaphore descriptor. |
| info | A pointer to the returnbuffer" to copy the information to. |

Returns

Zero is returned and status information is written to the structure pointed at by `info` upon success. Otherwise:

- -EINVAL is returned if `sem` is not a valid semaphore descriptor.

Tags

unrestricted
6.87.3.6 rt_sem_p()

```c
int rt_sem_p (
    RT_SEM *sem,
    RTIME timeout ) [inline], [static]
```

Pend on a semaphore (with relative scalar timeout).

This routine is a variant of `rt_sem_p_timed()` accepting a relative timeout specification expressed as a scalar value.

**Parameters**

- **sem**: The semaphore descriptor.
- **timeout**: A delay expressed in clock ticks.

**Tags**

- xthread-nowait, switch-primary

**References** `rt_sem_p_timed()`.

6.87.3.7 rt_sem_p_timed()

```c
int rt_sem_p_timed (
    RT_SEM *sem,
    const struct timespec *abs_timeout )
```

Pend on a semaphore.

Test and decrement the semaphore count. If the semaphore value is greater than zero, it is decremented by one and the service immediately returns to the caller. Otherwise, the caller is blocked until the semaphore is either signaled or destroyed, unless a non-blocking operation was required.

**Parameters**

- **sem**: The semaphore descriptor.
- **abs_timeout**: An absolute date expressed in clock ticks, specifying a time limit to wait for the request to be satisfied (see note). Passing NULL causes the caller to block indefinitely until the request is satisfied. Passing `{ .tv_sec = 0, .tv_nsec = 0 }` causes the service to return without blocking in case the request cannot be satisfied immediately.

**Returns**

- Zero is returned upon success. Otherwise:

  - `-ETIMEDOUT` is returned if `abs_timeout` is reached before the request is satisfied.
• -EWOULDBLOCK is returned if `abs_timeout` is `{ .tv_sec = 0, .tv_nsec = 0 }` and the semaphore count is zero on entry.

• -EINTR is returned if `rt_task_unblock()` was called for the current task before the request is satisfied.

• -EINVAL is returned if `sem` is not a valid semaphore descriptor.

• -EIDRM is returned if `sem` is deleted while the caller was sleeping on it. In such a case, `sem` is no more valid upon return of this service.

• -EPERM is returned if this service should block, but was not called from a Xenomai thread.

Tags

  xthread-nowait, switch-primary

Note

`abs_timeout` is interpreted as a multiple of the Alchemy clock resolution (see `-alchemy-clock-resolution` option, defaults to 1 nanosecond).

Referenced by `rt_sem_p()`, and `rt_sem_p_until()`.

6.87.3.8 `rt_sem_p_until()`

```c
int rt_sem_p_until ( 
    RT_SEM * sem, 
    RTIME abs_timeout ) [inline], [static]
```

Pend on a semaphore (with absolute scalar timeout).

This routine is a variant of `rt_sem_p_timed()` accepting an absolute timeout specification expressed as a scalar value.

Parameters

| `sem` | The semaphore descriptor. |
| `abs_timeout` | An absolute date expressed in clock ticks. |

Tags

  xthread-nowait, switch-primary

References `rt_sem_p_timed()`.
6.87.3.9  rt_sem_unbind()

int rt_sem_unbind (  
    RT_SEM * sem )

Unbind from a semaphore.

Parameters

    sem    The semaphore descriptor.

This routine releases a previous binding to a semaphore. After this call has returned, the descriptor is no more valid for referencing this object.

Tags

    thread-unrestricted

6.87.3.10  rt_sem_v()

int rt_sem_v (  
    RT_SEM * sem )

Signal a semaphore.

If the semaphore is pended, the task heading the wait queue is immediately unblocked. Otherwise, the semaphore count is incremented by one, unless the semaphore is used in "pulse" mode (see rt_sem←_create()).

Parameters

    sem    The semaphore descriptor.

Returns

    Zero is returned upon success. Otherwise:

    • -EINVAL is returned if sem is not a valid semaphore descriptor.

Tags

    unrestricted
6.88 Task management services

Services dealing with preemptive multi-tasking.

Collaboration diagram for Task management services:

![Collaboration diagram](image)

Data Structures

- struct RT_TASK_INFO
  Task status descriptor.

Macros

- #define T_LOPRIO 0
  Task priorities.
- #define T_LOCK ____THREAD_M_LOCK
  Task mode bits.
- #define T_WARNSW ____THREAD_M_WARNSW
  Cobalt only, nop over Mercury.

Functions

- int rt_task_delete (RT_TASK *task)
  Delete a real-time task.
- int rt_task_set_affinity (RT_TASK *task, const cpu_set_t *cpus)
  Set CPU affinity of real-time task.
- int rt_task_start (RT_TASK *task, void(*entry)(void *arg), void *arg)
  Start a real-time task.
- int rt_task_shadow (RT_TASK *task, const char *name, int prio, int mode)
  Turn caller into a real-time task.
- int rt_task_join (RT_TASK *task)
  Wait on the termination of a real-time task.
- int rt_task_wait_period (unsigned long *overruns_r)
  Wait for the next periodic release point.
- int rt_task_sleep (RTIME delay)
  Delay the current real-time task (with relative delay).
- int rt_task_sleep_until (RTIME date)
  Delay the current real-time task (with absolute wake-up date).
- int rt_task_same (RT_TASK *task1, RT_TASK *task2)
Compare real-time task descriptors.

- `int rt_task_suspend (RT_TASK *task)`
  Suspend a real-time task.
- `int rt_task_resume (RT_TASK *task)`
  Resume a real-time task.
- `RT_TASK * rt_task_self (void)`
  Retrieve the current task descriptor.
- `int rt_task_set_priority (RT_TASK *task, int prio)`
  Change the base priority of a real-time task.
- `int rt_task_set_mode (int clrmask, int setmask, int *mode_r)`
  Change the current task mode.
- `int rt_task_yield (void)`
  Manual round-robin.
- `int rt_task_unblock (RT_TASK *task)`
  Unblock a real-time task.
- `int rt_task_slice (RT_TASK *task, RTIME quantum)`
  Set a task's round-robin quantum.
- `int rt_task_inquire (RT_TASK *task, RT_TASK_INFO *info)`
  Retrieve information about a real-time task.
- `ssize_t rt_task_send_timed (RT_TASK *task, RT_TASK_MCB *mcb_s, RT_TASK_MCB *mcb_r, const struct timespec *abs_timeout)`
  Send a message to a real-time task.
- `static ssize_t rt_task_send_until (RT_TASK *task, RT_TASK_MCB *mcb_s, RT_TASK_MCB *mcb_r, RTIME timeout)`
  Send a message to a real-time task (with absolute scalar timeout).
- `static ssize_t rt_task_send (RT_TASK *task, RT_TASK_MCB *mcb_s, RT_TASK_MCB *mcb_r, RTIME timeout)`
  Send a message to a real-time task (with relative scalar timeout).
- `int rt_task_receive_timed (RT_TASK_MCB *mcb_r, const struct timespec *abs_timeout)`
  Receive a message from a real-time task.
- `static int rt_task_receive_until (RT_TASK_MCB *mcb_r, RTIME timeout)`
  Receive a message from a real-time task (with absolute scalar timeout).
- `static int rt_task_receive (RT_TASK_MCB *mcb_r, RTIME timeout)`
  Receive a message from a real-time task (with relative scalar timeout).
- `int rt_task_reply (int flowid, RT_TASK_MCB *mcb_s)`
  Reply to a remote task message.
- `int rt_task_bind (RT_TASK *task, const char *name, RTIME timeout)`
  Bind to a task.
- `int rt_task_unbind (RT_TASK *task)`
  Unbind from a task.
- `int rt_task_create (RT_TASK *task, const char *name, int stksize, int prio, int mode)`
  Create a task with Alchemy personality.
- `int rt_task_set_periodic (RT_TASK *task, RTIME idate, RTIME period)`
  Make a real-time task periodic.
- `int rt_task_spawn (RT_TASK *task, const char *name, int stksize, int prio, int mode, void *(*entry)(void *arg), void *arg)`
  Create and start a real-time task.
6.88 Task management services

6.88.1 Detailed Description

Services dealing with preemptive multi-tasking.

Each Alchemy task is an independent portion of the overall application code embodied in a C procedure, which executes on its own stack context.

6.88.2 Macro Definition Documentation

6.88.2.1 T_LOCK

#define T_LOCK __THREAD_M_LOCK

Task mode bits.
Referenced by rt_task_set_mode().

6.88.2.2 T_LOPRIO

#define T_LOPRIO 0

Task priorities.

6.88.2.3 T_WARNNSW

#define T_WARNNSW __THREAD_M_WARNNSW

Cobalt only, nop over Mercury.
Referenced by rt_task_set_mode().

6.88.3 Function Documentation

6.88.3.1 rt_task_bind()

int rt_task_bind ( 
    RT_TASK * task, 
    const char * name, 
    RTIME timeout )

Bind to a task.
This routine creates a new descriptor to refer to an existing Alchemy task identified by its symbolic name. If the object does not exist on entry, the caller may block until a task of the given name is created.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>task</strong></td>
<td>The address of a task descriptor filled in by the operation. Contents of this memory is undefined upon failure.</td>
</tr>
<tr>
<td><strong>name</strong></td>
<td>A valid NULL-terminated name which identifies the task to bind to. This string should match the object name argument passed to <code>rt_task_create()</code>, or <code>rt_task_shadow()</code>.</td>
</tr>
<tr>
<td><strong>timeout</strong></td>
<td>The number of clock ticks to wait for the registration to occur (see note). Passing <code>TM_INFINITY</code> causes the caller to block indefinitely until the object is registered. Passing <code>TM_NONBLOCK</code> causes the service to return immediately without waiting if the object is not registered on entry.</td>
</tr>
</tbody>
</table>

Returns

Zero is returned upon success. Otherwise:

- **-EINTR** is returned if `rt_task_unblock()` was called for the current task before the retrieval has completed.
- **-EWOULDBLOCK** is returned if `timeout` is equal to `TM_NONBLOCK` and the searched object is not registered on entry.
- **-ETIMEDOUT** is returned if the object cannot be retrieved within the specified amount of time.
- **-EPERM** is returned if this service should block, but was not called from a Xenomai thread.

Tags

xthread-nowait, switch-primary

Note

The `timeout` value is interpreted as a multiple of the Alchemy clock resolution (see –alchemy-clock-resolution option, defaults to 1 nanosecond).

6.88.3.2  `rt_task_create()`

```c
int rt_task_create ( 
    RT_TASK * task, 
    const char * name, 
    int stksize, 
    int prio, 
    int mode )
```

Create a task with Alchemy personality.

This service creates a task with access to the full set of Alchemy services. If `prio` is non-zero, the new task belongs to Xenomai's real-time FIFO scheduling class, aka `SCHED_FIFO`. If `prio` is zero, the task belongs to the regular `SCHED_OTHER` class.

Creating tasks with zero priority is useful for running non real-time processes which may invoke blocking real-time services, such as pending on a semaphore, reading from a message queue or a buffer, and so on.

Once created, the task is left dormant until it is actually started by `rt_task_start()`.
Parameters

<table>
<thead>
<tr>
<th>task</th>
<th>The address of a task descriptor which can be later used to identify uniquely the created object, upon success of this call.</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>An ASCII string standing for the symbolic name of the task. When non-NULL and non-empty, a copy of this string is used for indexing the created task into the object registry.</td>
</tr>
<tr>
<td>stksize</td>
<td>The size of the stack (in bytes) for the new task. If zero is passed, a system-dependent default size will be substituted.</td>
</tr>
<tr>
<td>prio</td>
<td>The base priority of the new task. This value must be in the [0 .. 99] range, where 0 is the lowest effective priority.</td>
</tr>
<tr>
<td>mode</td>
<td>The task creation mode. The following flags can be OR’ed into this bitmask:</td>
</tr>
</tbody>
</table>

- **T_JOINABLE** allows another task to wait on the termination of the new task. `rt_task_join()` shall be called for this task to clean up any resources after its termination.
- **T_LOCK** causes the new task to lock the scheduler prior to entering the user routine specified by `rt_task_start()`. A call to `rt_task_set_mode()` from the new task is required to drop this lock.
- When running over the Cobalt core, **T_WARNSW** causes the SIGDEBUG signal to be sent to the current task whenever it switches to the secondary mode. This feature is useful to detect unwanted migrations to the Linux domain. This flag has no effect over the Mercury core.

Returns

Zero is returned upon success. Otherwise:

- **EINVAL** is returned if either `prio`, `mode` or `stksize` are invalid.
- **ENOMEM** is returned if the system fails to get memory from the main heap in order to create the task.
- **EEXIST** is returned if the `name` is conflicting with an already registered task.

Tags

**mode-unrestricted, switch-secondary**

Side effects

- When running over the Cobalt core:
  - calling `rt_task_create()` causes SCHED_FIFO tasks to switch to secondary mode.
  - members of Xenomai’s SCHED_FIFO class running in the primary domain have utmost priority over all Linux activities in the system, including Linux interrupt handlers.
- When running over the Mercury core, the new task belongs to the regular POSIX SCHED_FIFO class.

Note

Tasks can be referred to from multiple processes which all belong to the same Xenomai session.

Examples:

`cross-link.c`. 
6.88.3.3  rt_task_delete()

int rt_task_delete (  
    RT_TASK * task )

Delete a real-time task.

This call terminates a task previously created by rt_task_create().

Tasks created with the T_JOINABLE flag shall be joined by a subsequent call to rt_task_join() once successfully deleted, to reclaim all resources.

Parameters

| task | The task descriptor. |

Returns

Zero is returned upon success. Otherwise:

- -EINVAL is returned if task is not a valid task descriptor.
- -EPERM is returned if task is NULL and this service was called from an invalid context. In addition, this error is always raised when this service is called from asynchronous context, such as a timer/alarm handler.

Tags

mode-unrestricted, switch-secondary

Note

The caller must be an Alchemy task if task is NULL.

Examples:

cross-link.c.

6.88.3.4  rt_task_inquire()

int rt_task_inquire (  
    RT_TASK * task,  
    RT_TASK_INFO * info )

Retrieve information about a real-time task.

Return various information about an Alchemy task. This service may also be used to probe for task existence.
**6.88 Task management services**

### task descriptor

<table>
<thead>
<tr>
<th><strong>task</strong></th>
<th>The task descriptor. If <em>task</em> is NULL, the information about the current task is returned.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>info</strong></td>
<td>The address of a structure the task information will be written to. Passing NULL is valid, in which case the system is only probed for existence of the specified task.</td>
</tr>
</tbody>
</table>

**Returns**

Zero is returned if the task exists. In addition, if *info* is non-NULL, it is filled in with task information.

- **EINVAL** is returned if *task* is not a valid task descriptor, or if *prio* is invalid.
- **EPERM** is returned if *task* is NULL and this service was called from an invalid context.

**Tags**

*mode-unrestricted, switch-primary*

**Note**

The caller must be an Alchemy task if *task* is NULL.

---

**6.88.3.5 rt_task_join()**

```c
int rt_task_join (RT_TASK *task)
```

Wait on the termination of a real-time task.

This service blocks the caller in non-real-time context until *task* has terminated. All resources are released after successful completion of this service.

The specified task must have been created by the same process that wants to join it, and the T JOINABLE mode flag must have been set on creation to **rt_task_create()**.

Tasks created with the T JOINABLE flag shall be joined by a subsequent call to **rt_task_join()** once successfully deleted, to reclaim all resources.

**Parameters**

| **task** | The task descriptor. |

**Returns**

Zero is returned upon success. Otherwise:

- **EINVAL** is returned if *task* is not a valid task descriptor.
- EINVAL is returned if the task was not created with T_JOINABLE set or some other task is already waiting on the termination.

- EDEADLK is returned if task refers to the caller.

- ESRCH is returned if task no longer exists or refers to task created by a different process.

Tags

mode-unrestricted, switch-primary

Note

After successful completion of this service, it is neither required nor valid to additionally invoke rt_task_delete() on the same task.

6.88.3.6 rt_task_receive()

ssize_t rt_task_receive (  
  RT_TASK_MCB *mcb_r,  
  RTIME timeout ) [inline], [static]

Receive a message from a real-time task (with relative scalar timeout).

This routine is a variant of rt_task_receive_timed() accepting a relative timeout specification expressed as a scalar value.

Parameters

<table>
<thead>
<tr>
<th>mcb_r</th>
<th>The address of a message control block referring to the receive message area.</th>
</tr>
</thead>
<tbody>
<tr>
<td>timeout</td>
<td>A delay expressed in clock ticks.</td>
</tr>
</tbody>
</table>

Tags

xthread-only, switch-primary

References rt_task_receive_timed().

6.88.3.7 rt_task_receive_timed()

int rt_task_receive_timed (  
  RT_TASK_MCB *mcb_r,  
  const struct timespec *abs_timeout )

Receive a message from a real-time task.
This service is part of the synchronous message passing support available to Alchemy tasks. The caller receives a variable-sized message from another task. The sender is blocked until the caller invokes `rt_task_reply()` to finish the transaction.

A basic message control block is used to store the location and size of the data area to receive from the client, in addition to a user-defined operation code.

### Parameters

| `mcb_r` | The address of a message control block referring to the receive message area. The fields from this control block should be set as follows:
|---|

- `mcb_r->data` should contain the address of a buffer large enough to collect the data sent by the remote task;
- `mcb_r->size` should contain the size in bytes of the buffer space pointed at by `mcb_r->data`. If `mcb_r->size` is lower than the actual size of the received message, no data copy takes place and `-ENOBUFS` is returned to the caller. See note.

Upon return, `mcb_r->opcode` will contain the operation code sent from the remote task using `rt_task_send()`.

### Parameters

| `abs_timeout` | The number of clock ticks to wait for receiving a message (see note). Passing NULL causes the caller to block indefinitely until a remote task eventually sends a message. Passing `{ .tv_sec = 0, .tv_nsec = 0 }` causes the service to return immediately without waiting if no remote task is currently waiting for sending a message.
|---|

### Returns

A strictly positive value is returned upon success, representing a flow identifier for the opening transaction; this token should be passed to `rt_task_reply()`, in order to send back a reply to and unblock the remote task appropriately. Otherwise:

- `-EPERM` is returned if this service was called from an invalid context.
- `-EINTR` is returned if `rt_task_unblock()` was called for the current task before a message was received.
- `-ENOBUFFS` is returned if `mcb_r` does not point at a message area large enough to collect the remote task’s message.
- `-EWOULDBLOCK` is returned if `abs_timeout` is `{ .tv_sec = 0, .tv_nsec = 0 }` and no remote task is currently waiting for sending a message to the caller.
- `-ETIMEDOUT` is returned if no message was received within the `timeout`.

### Tags

`xthread-only, switch-primary`
Note

*abs_timeout* is interpreted as a multiple of the Alchemy clock resolution (see `-alchemy-clock-resolution` option, defaults to 1 nanosecond).

Referenced by `rt_task_receive()`, and `rt_task_receive_until()`.

### 6.88.3.8 rt_task_receive_until()

```c
ssize_t rt_task_receive_until (  
    RT_TASK_MCB *mcb_r,  
    RTIME abs_timeout  ) [inline], [static]
```

Receive a message from a real-time task (with absolute scalar timeout).

This routine is a variant of `rt_task_receive_timed()` accepting an absolute timeout specification expressed as a scalar value.

**Parameters**

<table>
<thead>
<tr>
<th>mcb_r</th>
<th>The address of a message control block referring to the receive message area.</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs_timeout</td>
<td>An absolute date expressed in clock ticks.</td>
</tr>
</tbody>
</table>

**Tags**

`xthread-only, switch-primary`

References `rt_task_receive_timed()`.

### 6.88.3.9 rt_task_reply()

```c
int rt_task_reply (  
    int flowid,  
    RT_TASK_MCB *mcb_s )
```

Reply to a remote task message.

This service is part of the synchronous message passing support available to Alchemy tasks. The caller sends a variable-sized message back to a remote task, in response to this task's initial message received by a call to `rt_task_receive()`. As a consequence of calling `rt_task_reply()`, the remote task will be unblocked from the `rt_task_send()` service.

A basic message control block is used to store the location and size of the data area to send back, in addition to a user-defined status code.
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flowid</td>
<td>The flow identifier returned by a previous call to <code>rt_task_receive()</code> which uniquely identifies the current transaction.</td>
</tr>
<tr>
<td>mcb_s</td>
<td>The address of an optional message control block referring to the message to be sent back. If <code>mcb_s</code> is NULL, the remote will be unblocked without getting any reply data. When <code>mcb_s</code> is valid, the fields from this control block should be set as follows:</td>
</tr>
</tbody>
</table>

- `mcb_s->data` should contain the address of the payload data to send to the remote task.
- `mcb_s->size` should contain the size in bytes of the payload data pointed at by `mcb_s->data`. Zero is a legitimate value, and indicates that no payload data will be transferred. In the latter case, `mcb_s->data` will be ignored.
- `mcb_s->opcode` is an opaque status code carried during the message transfer the caller can fill with any appropriate value. It will be made available "as is" to the remote task into the status code field by the `rt_task_send()` service. If `mcb_s` is NULL, Zero will be returned to the remote task into the status code field.

### Returns

Zero is returned upon success. Otherwise:

- `EINVAL` is returned if `flowid` is invalid.
- `ENXIO` is returned if `flowid` does not match the expected identifier returned from the latest call of the current task to `rt_task_receive()`, or if the remote task stopped waiting for the reply in the meantime (e.g. the remote could have been deleted or forcibly unblocked).
- `ENOBUFS` is returned if the reply data referred to by `mcb_s` is larger than the reply area mentioned by the remote task when calling `rt_task_send()`. In such a case, the remote task also receives `ENOBUFS` on return from `rt_task_send()`.
- `EPERM` is returned if this service was called from an invalid context.

### Tags

`xthread-only, switch-primary`

---

#### 6.88.3.10 `rt_task_resume()`

```c
int rt_task_resume ( RT_TASK * task )
```

Resume a real-time task.

Forcibly resume the execution of a task which was previously suspended by a call to `rt_task_suspend()`, if the suspend nesting count decrements to zero.
Parameters

| task | The task descriptor. |

Returns

Zero is returned upon success. Otherwise:

- EINVAL is returned if task is not a valid task descriptor.

Tags

unrestricted, switch-primary

Note

Blocked and suspended task states are cumulative. Therefore, resuming a task currently waiting on a synchronization object (e.g., semaphore, queue) does not make it eligible for scheduling until the awaited resource is eventually acquired, or a timeout elapses.

6.88.3.11 rt_task_same()

int rt_task_same ( 
    RT_TASK * task1, 
    RT_TASK * task2 )

Compare real-time task descriptors.

This predicate returns true if task1 and task2 refer to the same task.

Parameters

| task1 | First task descriptor to compare. |
| task2 | Second task descriptor to compare. |

Returns

A non-zero value is returned if both descriptors refer to the same task, zero otherwise.

Tags

unrestricted
6.88 Task management services

6.88.3.12 rt_task_self()

```c
RT_TASK * rt_task_self ( void )
```

Retrieve the current task descriptor.

Return the address of the current Alchemy task descriptor.

Returns

The address of the task descriptor referring to the current Alchemy task is returned upon success, or NULL if not called from a valid Alchemy task context.

Tags

- xthread-only

6.88.3.13 rt_task_send()

```c
ssize_t rt_task_send ( RT_TASK * task,
                      RT_TASK_MCB * mcb_s,
                      RT_TASK_MCB * mcb_r,
                      RTIME timeout ) [inline], [static]
```

Send a message to a real-time task (with relative scalar timeout).

This routine is a variant of `rt_task_send_timed()` accepting a relative timeout specification expressed as a scalar value.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>task</code></td>
<td>The task descriptor.</td>
</tr>
<tr>
<td><code>mcb_s</code></td>
<td>The address of the message control block referring to the message to be sent.</td>
</tr>
<tr>
<td><code>mcb_r</code></td>
<td>The address of an optional message control block referring to the reply message area.</td>
</tr>
<tr>
<td><code>timeout</code></td>
<td>A delay expressed in clock ticks.</td>
</tr>
</tbody>
</table>

Tags

- xthread-only, switch-primary

References `rt_task_send_timed()`. 
6.88.3.14  rt_task_send_timed()

ssize_t rt_task_send_timed (  
    RT_TASK * task,  
    RT_TASK_MCB * mcb_s,  
    RT_TASK_MCB * mcb_r,  
    const struct timespec * abs_timeout )

Send a message to a real-time task.

This service is part of the synchronous message passing support available to Alchemy tasks. The caller sends a variable-sized message to another task, waiting for the remote to receive the initial message by a call to rt_task_receive(), then reply to it using rt_task_reply().

A basic message control block is used to store the location and size of the data area to send or retrieve upon reply, in addition to a user-defined operation code.

Parameters

<table>
<thead>
<tr>
<th>task</th>
<th>The task descriptor.</th>
</tr>
</thead>
<tbody>
<tr>
<td>mcb_s</td>
<td>The address of the message control block referring to the message to be sent. The fields from this control block should be set as follows:</td>
</tr>
<tr>
<td>mcb_r</td>
<td>The address of an optional message control block referring to the reply message area. If mcb_r is NULL and a reply is sent back by the remote task, the reply message will be discarded, and -ENOBUFS will be returned to the caller. When mcb_r is valid, the fields from this control block should be set as follows:</td>
</tr>
</tbody>
</table>

- mcb_s->data should contain the address of the payload data to send to the remote task.
- mcb_s->size should contain the size in bytes of the payload data pointed at by mcb_s->data. Zero is a legitimate value, and indicates that no payload data will be transferred. In the latter case, mcb_s->data will be ignored.
- mcb_s->opcode is an opaque operation code carried during the message transfer, the caller can fill with any appropriate value. It will be made available "as is" to the remote task into the operation code field by the rt_task_receive() service.

Parameters

| mcb_r   | The address of an optional message control block referring to the reply message area. If mcb_r is NULL and a reply is sent back by the remote task, the reply message will be discarded, and -ENOBUFS will be returned to the caller. When mcb_r is valid, the fields from this control block should be set as follows: |

- mcb_r->data should contain the address of a buffer large enough to collect the reply data from the remote task.
- mcb_r->size should contain the size in bytes of the buffer space pointed at by mcb_r->data. If mcb_r->size is lower than the actual size of the reply message, no data copy takes place and -ENOBUFS is returned to the caller.

Upon return, mcb_r->opcode will contain the status code sent back from the remote task using rt_task_reply(), or zero if unspecified.
Parameters

| abs_timeout | An absolute date expressed in clock ticks, specifying a time limit to wait for the recipient task to reply to the initial message (see note). Passing NULL causes the caller to block indefinitely until a reply is received. Passing \{ .tv_sec = 0, .tv_nsec = 0 \} causes the service to return without blocking in case the recipient task is not waiting for messages at the time of the call. |

Returns

A positive value is returned upon success, representing the length (in bytes) of the reply message returned by the remote task. Zero is a success status, meaning either that \( mcb_r \) was NULL on entry, or that no actual message was passed to the remote call to \texttt{rt_task_reply}(). Otherwise:

- \texttt{-EINVAL} is returned if \texttt{task} is not a valid task descriptor.
- \texttt{-EPERM} is returned if this service was called from an invalid context.
- \texttt{-ENOBUFFS} is returned if \texttt{mcb_r} does not point at a message area large enough to collect the remote task’s reply. This includes the case where \texttt{mcb_r} is NULL on entry, despite the remote task attempts to send a reply message.
- \texttt{-EWOULDBLOCK} is returned if \texttt{abs\_timeout} is \{ .tv\_sec = 0, .tv\_nsec = 0 \} and the recipient \texttt{task} is not currently waiting for a message on the \texttt{rt\_task\_receive}() service.
- \texttt{-EIDRM} is returned if \texttt{task} has been deleted while waiting for a reply.
- \texttt{-EINTR} is returned if \texttt{rt\_task\_unblock}() was called for the current task before any reply was received from the recipient \texttt{task}.

Tags

xthread-only, switch-primary

Note

\texttt{abs\_timeout} is interpreted as a multiple of the Alchemy clock resolution (see \texttt{--alchemy\_clock\_resolution} option, defaults to 1 nanosecond).

Referenced by \texttt{rt\_task\_send()}, and \texttt{rt\_task\_send\_until}().

6.88.3.15 \texttt{rt\_task\_send\_until}()

\begin{verbatim}
ssize_t rt_task_send_until (  
    RT_TASK * task,  
    RT_TASK_MCB * mcb_s,  
    RT_TASK_MCB * mcb_r,  
    RTIME abs_timeout ) [inline], [static]
\end{verbatim}

Send a message to a real-time task (with absolute scalar timeout).

This routine is a variant of \texttt{rt\_task\_send\_timed}() accepting an absolute timeout specification expressed as a scalar value.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>task</code></td>
<td>The task descriptor.</td>
</tr>
<tr>
<td><code>mcb_s</code></td>
<td>The address of the message control block referring to the message to be sent.</td>
</tr>
<tr>
<td><code>mcb_r</code></td>
<td>The address of an optional message control block referring to the reply message area.</td>
</tr>
<tr>
<td><code>abs_timeout</code></td>
<td>An absolute date expressed in clock ticks.</td>
</tr>
</tbody>
</table>

Tags

xthread-only, switch-primary

References `rt_task_send_timed()`.

6.88.3.16  `rt_task_set_affinity()`

```c
int rt_task_set_affinity (RT_TASK *task, const cpu_set_t *cpus )
```

Set CPU affinity of real-time task.

This calls makes `task` affine to the set of CPUs defined by `cpus`.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>task</code></td>
<td>The task descriptor. If <code>task</code> is NULL, the CPU affinity of the current task is changed.</td>
</tr>
<tr>
<td><code>cpus</code></td>
<td>The set of CPUs <code>task</code> should be affine to.</td>
</tr>
</tbody>
</table>

Returns

Zero is returned upon success. Otherwise:

- EINVAL is returned if `task` is NULL but the caller is not a Xenomai task, or if `task` is non-NULL but not a valid task descriptor.
- EINVAL is returned if `cpus` contains no processors that are currently physically on the system and permitted to the process according to any restrictions that may be imposed by the "cpuset" mechanism described in cpuset(7).

Tags

mode-unrestricted, switch-secondary

Note

The caller must be an Alchemy task if `task` is NULL.
6.88 Task management services

6.88.3.17 rt_task_set_mode()

```c
int rt_task_set_mode (  
    int clrmask,  
    int setmask,  
    int * mode_r  
)
```

Change the current task mode.

Each Alchemy task has a set of internal flags determining several operating conditions. `rt_task_set_mode()` takes a bitmask of mode bits to clear for disabling the corresponding modes for the current task, and another one to set for enabling them. The mode bits which were previously in effect before the change can be returned upon request.

The following bits can be part of the bitmask:

- **T_LOCK** causes the current task to lock the scheduler on the current CPU, preventing all further involuntary task switches on this CPU. Clearing this bit unlocks the scheduler.
- Only when running over the Cobalt core:
  - **T_WARNSW** causes the SIGDEBUG signal to be sent to the current task whenever it switches to the secondary mode. This feature is useful to detect unwanted migrations to the Linux domain.
  - **T_CONFORMING** can be passed in `setmask` to switch the current Alchemy task to its preferred runtime mode. The only meaningful use of this switch is to force a real-time task back to primary mode (see note). Any other use leads to a nop.

These two last flags have no effect over the Mercury core, and are simply ignored.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clrmask</td>
<td>A bitmask of mode bits to clear for the current task, before <code>setmask</code> is applied. Zero is an acceptable value which leads to a no-op.</td>
</tr>
<tr>
<td>setmask</td>
<td>A bitmask of mode bits to set for the current task. Zero is an acceptable value which leads to a no-op.</td>
</tr>
<tr>
<td>mode_r</td>
<td>If non-NULL, <code>mode_r</code> must be a pointer to a memory location which will be written upon success with the previous set of active mode bits. If NULL, the previous set of active mode bits will not be returned.</td>
</tr>
</tbody>
</table>

**Returns**

Zero is returned upon success. Otherwise:

- **-EINVAL** is returned if `task` is not a valid task descriptor, or if any bit from `clrmask` or `setmask` is invalid.
- **-EPERM** is returned if this service was called from an invalid context.

**Tags**

`xthread-only, switch-primary`
Note
The caller must be an Alchemy task.
Forcing the task mode using the T_CONFORMING bit from user code is almost always wrong,
since the Xenomai/cobalt core handles mode switches internally when/if required. Most often,
manual mode switching from applications introduces useless overhead. This mode bit is part of
the API only to cover rare use cases in middleware code based on the Alchemy interface.

References T_LOCK, and T_WARNSW.

6.88.3.18  rt_task_set_periodic()

int rt_task_set_periodic (  
    RT_TASK * task,  
    RTIME idate,  
    RTIME period )  

Make a real-time task periodic.

Make a task periodic by programing its first release point and its period in the processor time line. task
should then call rt_task_wait_period() to sleep until the next periodic release point in the processor
timeline is reached.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task</td>
<td>The task descriptor. If task is NULL, the current task is made periodic. task must belong the current process.</td>
</tr>
<tr>
<td>idate</td>
<td>The initial (absolute) date of the first release point, expressed in clock ticks (see note). If idate is equal to TM_NOW, the current system date is used.</td>
</tr>
<tr>
<td>period</td>
<td>The period of the task, expressed in clock ticks (see note). Passing TM_INFINITE stops the task's periodic timer if enabled, then returns successfully.</td>
</tr>
</tbody>
</table>

Returns
Zero is returned upon success. Otherwise:

- -EINVAL is returned if task is NULL but the caller is not a Xenomai task, or if task is non-NULL but not a valid task descriptor.
- -ETIMEDOUT is returned if idate is different from TM_INFINITE and represents a date in the past.

Tags
mode-unrestricted, switch-primary

Note
The caller must be an Alchemy task if task is NULL.
Over Cobalt, -EINVAL is returned if period is different from TM_INFINITE but shorter than the user scheduling latency value for the target system, as displayed by /proc/xenomai/latency.
The idate and period values are interpreted as a multiple of the Alchemy clock resolution (see --alchemy-clock-resolution option, defaults to 1 nanosecond).
Attention

Unlike its Xenomai 2.x counterpart, \texttt{rt_task_set_periodic()} will NOT block task until \texttt{idade} is reached. The first beat in the periodic timeline should be awaited for by a call to \texttt{rt_task_wait_period()}.

Examples:

\texttt{cross-link.c}.

6.88.3.19 \texttt{rt_task_set_priority()}

\begin{verbatim}
int rt_task_set_priority ( 
    RT_TASK * task, 
    int prio )
\end{verbatim}

Change the base priority of a real-time task.

The base priority of a task defines the relative importance of the work being done by each task, which gains control of the CPU accordingly.

Changing the base priority of a task does not affect the priority boost the target task might have obtained as a consequence of a priority inheritance undergoing.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{task}</td>
<td>The task descriptor. If \texttt{task} is NULL, the priority of the current task is changed.</td>
</tr>
<tr>
<td>\texttt{prio}</td>
<td>The new priority. This value must range from \texttt{T_LOPRIO .. T_HIPRIO} (inclusive) where \texttt{T_LOPRIO} is the lowest effective priority.</td>
</tr>
</tbody>
</table>

Returns

Zero is returned upon success. Otherwise:

- \texttt{EINVAL} is returned if \texttt{task} is not a valid task descriptor, or if \texttt{prio} is invalid.
- \texttt{EPERM} is returned if \texttt{task} is NULL and this service was called from an invalid context.

Tags

\texttt{mode-unrestricted, switch-primary}

Note

The caller must be an Alchemy task if \texttt{task} is NULL.

Assigning the same priority to a running or ready task moves it to the end of its priority group, thus causing a manual round-robin.
6.88.3.20  rt_task_shadow()

int rt_task_shadow(
    RT_TASK * task,
    const char * name,
    int prio,
    int mode )

Turn caller into a real-time task.

Set the calling thread personality to the Alchemy API, enabling the full set of Alchemy services. Upon
success, the caller is no more a regular POSIX thread, but a Xenomai-extended thread.

If *prio* is non-zero, the new task moves to Xenomai's real-time FIFO scheduling class, aka SCHED_FIFO.
If *prio* is zero, the task moves to the regular SCHED_OTHER class.

Running Xenomai tasks with zero priority is useful for running non real-time processes which may invoke
blocking real-time services, such as pending on a semaphore, reading from a message queue or a buffer,
and so on.

Parameters

<table>
<thead>
<tr>
<th>task</th>
<th>If non-NULL, the address of a task descriptor which can be later used to identify uniquely the task, upon success of this call. If NULL, no descriptor is returned.</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>An ASCII string standing for the symbolic name of the task. When non-NULL and non-empty, a copy of this string is used for indexing the task into the object registry.</td>
</tr>
<tr>
<td>prio</td>
<td>The base priority of the task. This value must be in the [0 .. 99] range, where 0 is the lowest effective priority.</td>
</tr>
<tr>
<td>mode</td>
<td>The task shadowing mode. The following flags can be OR'ed into this bitmask:</td>
</tr>
</tbody>
</table>

- T_LOCK causes the current task to lock the scheduler before returning to the caller, preventing all further involuntary task switches on the current CPU. A call to rt_task_set_mode() from the current task is required to drop this lock.

- When running over the Cobalt core, T_WARNSW causes the SIGDEBUG signal to be sent to the current task whenever it switches to the secondary mode. This feature is useful to detect unwanted migrations to the Linux domain. This flag has no effect over the Mercury core.

Returns

Zero is returned upon success. Otherwise:

- EINVAL is returned if *prio* is invalid.
- ENOMEM is returned if the system fails to get memory from the main heap in order to create the task extension.
- EEXIST is returned if the *name* is conflicting with an already registered task.
- EBUSY is returned if the caller is not a regular POSIX thread.

Tags

pthread-only, switch-secondary, switch-primary
Side effects

Over Cobalt, if the caller is a plain POSIX thread, it is turned into a Xenomai shadow thread, with full access to all Cobalt services. The caller always returns from this service in primary mode.

Note

Tasks can be referred to from multiple processes which all belong to the same Xenomai session.

Examples:

 rtcanrecv.c, and rtcansend.c.

6.88.3.21 rt_task_sleep()

```c
int rt_task_sleep (
    RTIME delay )
```

Delay the current real-time task (with relative delay).

This routine is a variant of `rt_task_sleep_until()` accepting a relative timeout specification.

Parameters

| delay | A relative delay expressed in clock ticks (see note). A zero delay causes this service to return immediately to the caller with a success status. |

Returns

See `rt_task_sleep_until()`.

Tags

xthread-only, switch-primary

Note

The delay value is interpreted as a multiple of the Alchemy clock resolution (see –alchemy-clock-resolution option, defaults to 1 nanosecond).

Examples:

cross-link.c, and rtcansend.c.
6.88.3.22  rt_task_sleep_until()

```c
int rt_task_sleep_until (RTIME date)
```

Delay the current real-time task (with absolute wakeup date).

Delay the execution of the calling task until a given date is reached. The caller is put to sleep, and does not consume any CPU time in such a state.
6.88 Task management services

Parameters

| date | An absolute date expressed in clock ticks, specifying a wakeup date (see note). As a special case, TM_INFINITE is an acceptable value that causes the caller to block indefinitely, until rt_task Unblock() is called against it. Otherwise, any wake up date in the past causes the task to return immediately. |

Returns

Zero is returned upon success. Otherwise:

- -EINTR is returned if rt_task Unblock() was called for the current task.
- -ETIMEDOUT is returned if date has already elapsed.
- -EPERM is returned if this service was called from an invalid context.

Tags

xthread-only, switch-primary

Note

The caller must be an Alchemy task if task is NULL.

The date value is interpreted as a multiple of the Alchemy clock resolution (see –alchemy-clock-resolution option, defaults to 1 nanosecond).

6.88.3.23 rt_task_slice()

int rt_task_slice (  
    RT_TASK * task,  
    RTIME quantum )

Set a task's round-robin quantum.

Set the time credit allotted to a task undergoing the round-robin scheduling. If quantum is non-zero, rt_task_slice() also refills the current quantum for the target task, otherwise, time-slicing is stopped for that task.

In other words, rt_task_slice() should be used to toggle round-robin scheduling for an Alchemy task.

Parameters

| task | The task descriptor. If task is NULL, the time credit of the current task is changed. task must belong to the current process. |
| quantum | The round-robin quantum for the task expressed in clock ticks (see note). |
Returns

Zero is returned upon success. Otherwise:

- -EINVAL is returned if `task` is not a valid task descriptor, or if `prio` is invalid.
- -EPERM is returned if `task` is NULL and this service was called from an invalid context.

Tags

`mode-unrestricted`, `switch-primary`

Note

The caller must be an Alchemy task if `task` is NULL.
The `quantum` value is interpreted as a multiple of the Alchemy clock resolution (see `--alchemy-clock-resolution` option, defaults to 1 nanosecond).

6.88.3.24 rt_task_spawn()

```c
int rt_task_spawn (  
    RT_TASK * task,  
    const char * name,  
    int stksize,  
    int prio,  
    int mode,  
    void(*)(void *arg) entry,  
    void * arg )
```

Create and start a real-time task.

This service spawns a task by combining calls to `rt_task_create()` and `rt_task_start()` for the new task.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>task</code></td>
<td>The address of a task descriptor which can be later used to identify uniquely the created object, upon success of this call.</td>
</tr>
<tr>
<td><code>name</code></td>
<td>An ASCII string standing for the symbolic name of the task. When non-NULL and non-empty, a copy of this string is used for indexing the created task into the object registry.</td>
</tr>
<tr>
<td><code>stksize</code></td>
<td>The size of the stack (in bytes) for the new task. If zero is passed, a system-dependent default size will be substituted.</td>
</tr>
<tr>
<td><code>prio</code></td>
<td>The base priority of the new task. This value must be in the [0 .. 99] range, where 0 is the lowest effective priority.</td>
</tr>
<tr>
<td><code>mode</code></td>
<td>The task creation mode. See <code>rt_task_create()</code>.</td>
</tr>
<tr>
<td><code>entry</code></td>
<td>The address of the task entry point.</td>
</tr>
<tr>
<td><code>arg</code></td>
<td>A user-defined opaque argument <code>entry</code> will receive.</td>
</tr>
</tbody>
</table>
Returns

See \texttt{rt\_task\_create()}.

Tags

\texttt{mode-unrestricted, switch-secondary}

Side effects

see \texttt{rt\_task\_create()}.

6.88.3.25 \texttt{rt\_task\_start()}

\begin{verbatim}
int rt_task_start(
    RT\_TASK * task,
    void(*)(void *arg) entry,
    void * arg )
\end{verbatim}

Start a real-time task.

This call starts execution of a task previously created by \texttt{rt\_task\_create()}. This service causes the
started task to leave the initial dormant state.

Parameters

\begin{tabular}{|l|l|}
\hline
\textit{task} & The task descriptor. \\
\textit{entry} & The address of the task entry point. \\
\textit{arg} & A user-defined opaque argument \textit{entry} will receive. \\
\hline
\end{tabular}

Returns

Zero is returned upon success. Otherwise:

- \texttt{-EINVAL} is returned if \textit{task} is not a valid task descriptor.

Tags

\texttt{mode-unrestricted, switch-primary}

Note

Starting an already started task leads to a \texttt{nop}, returning a success status.

Examples:

\begin{verbatim}
cross-link.c.
\end{verbatim}
6.88.3.26  rt_task_suspend()

```c
int rt_task_suspend (  
    RT_TASK * task )
```

Suspend a real-time task.

Forcibly suspend the execution of a task. This task will not be eligible for scheduling until it is explicitly resumed by a call to `rt_task_resume()`. In other words, the suspended state caused by a call to `rt_task_suspend()` is cumulative with respect to the delayed and blocked states caused by other services, and is managed separately from them.

A nesting count is maintained so that `rt_task_suspend()` and `rt_task_resume()` must be used in pairs.

Receiving a Linux signal causes the suspended task to resume immediately.

Parameters

<table>
<thead>
<tr>
<th>task</th>
<th>The task descriptor. If <code>task</code> is NULL, the current task is suspended.</th>
</tr>
</thead>
</table>

Returns

Zero is returned upon success. Otherwise:

- -EINVAL is returned if `task` is NULL but the caller is not a Xenomai task, or if `task` is non-NULL but not a valid task descriptor.
- -EINTR is returned if a Linux signal has been received by the caller if suspended.
- -EPERM is returned if `task` is NULL and this service was called from an invalid context.

Tags

`mode-unrestricted, switch-primary`

Note

The caller must be an Alchemy task if `task` is NULL.

Blocked and suspended task states are cumulative. Therefore, suspending a task currently waiting on a synchronization object (e.g. semaphore, queue) holds its execution until it is resumed, despite the awaited resource may have been acquired, or a timeout has elapsed in the meantime.

6.88.3.27  rt_task_unbind()

```c
int rt_task_unbind (  
    RT_TASK * task )
```

Unbind from a task.
6.88 Task management services

Parameters

| task | The task descriptor. |

This routine releases a previous binding to an Alchemy task. After this call has returned, the descriptor is no more valid for referencing this object.

Tags

- thread-unrestricted

6.88.3.28 rt_task_unblock()

```c
int rt_task_unblock (RT_TASK * task )
```

Unblock a real-time task.

Break the task out of any wait it is currently in. This call clears all delay and/or resource wait condition for the target task.

However, `rt_task_unblock()` does not resume a task which has been forcibly suspended by a previous call to `rt_task_suspend()`. If all suspensive conditions are gone, the task becomes eligible anew for scheduling.

Parameters

| task | The task descriptor. |

Returns

Zero is returned upon success. Otherwise:

- EINVAL is returned if `task` is not a valid task descriptor.

Tags

- unrestricted, switch-primary

6.88.3.29 rt_task_wait_period()

```c
int rt_task_wait_period (unsigned long * overruns_r )
```

Wait for the next periodic release point.

Delay the current task until the next periodic release point is reached. The periodic timer should have been previously started for `task` by a call to `rt_task_set_periodic()`.
Parameters

| overruns_r | If non-NULL, overruns_r shall be a pointer to a memory location which will be written with the count of pending overruns. This value is written to only when \texttt{rt_task\_wait\_period()} returns -ETIMEDOUT or success. The memory location remains unmodified otherwise. If NULL, this count will not be returned. |

Returns

Zero is returned upon success. If overruns_r is non-NULL, zero is written to the pointed memory location. Otherwise:

- EWOULDBLOCK is returned if \texttt{rt_task\_set\_periodic()} was not called for the current task.
- EINVAL is returned if \texttt{rt_task\_unblock()} was called for the waiting task before the next periodic release point was reached. In this case, the overrun counter is also cleared.
- ETIMEDOUT is returned if a timer overrun occurred, which indicates that a previous release point was missed by the calling task. If overruns_r is non-NULL, the count of pending overruns is written to the pointed memory location.
- EPERM is returned if this service was called from an invalid context.

Tags

xthread-only, switch-primary

Note

If the current release point has already been reached at the time of the call, the current task immediately returns from this service with no delay.

Examples:

cross-link.c.

6.88.3.30 rt_task_yield()

int rt_task_yield ( void )

Manual round-robin.

Move the current task to the end of its priority group, so that the next equal-priority task in ready state is switched in.

Returns

Zero is returned upon success. Otherwise:

- EPERM is returned if this service was called from an invalid context.

Tags

xthread-only, switch-primary
6.89 Timer management services

Services for reading and spinning on the hardware timer.

Collaboration diagram for Timer management services:

![Diagram showing Alchemy API and Timer management services]

Data Structures

- **struct rt_timer_info**
  
  *Timer status descriptor.*

Typedefs

- **typedef struct rt_timer_info RT_TIMER_INFO**
  
  *Timer status descriptor.*

Functions

- **static RTIME rt_timer_read (void)**
  
  *Return the current system time.*

- **SRTIME rt_timer_ns2ticks (SRTIME ns)**
  
  *Convert nanoseconds to Alchemy clock ticks.*

- **SRTIME rt_timer_ticks2ns (SRTIME ticks)**
  
  *Convert Alchemy clock ticks to nanoseconds.*

- **void rt_timer_inquire (RT_TIMER_INFO *info)**
  
  *Inquire about the Alchemy clock.*

- **void rt_timer_spin (RTIME ns)**
  
  *Busy wait burning CPU cycles.*

6.89.1 Detailed Description

Services for reading and spinning on the hardware timer.

6.89.2 Typedef Documentation
6.89.2.1 RT_TIMER_INFO

typedef struct rt_timer_info RT_TIMER_INFO

Timer status descriptor.
This structure reports information about the Alchemy clock, returned by a call to rt_timer_inquire().

6.89.3 Function Documentation

6.89.3.1 rt_timer_inquire()

void rt_timer_inquire ( RT_TIMER_INFO * info )

Inquire about the Alchemy clock.
Return status information about the Alchemy clock.

Parameters

| info | The address of a structure to fill with the clock information. |

Tags

unrestricted

References rt_timer_info::period.

6.89.3.2 rt_timer_ns2ticks()

SRTIME rt_timer_ns2ticks ( SRTIME ns )

Convert nanoseconds to Alchemy clock ticks.
Convert a count of nanoseconds to Alchemy clock ticks. This routine operates on signed nanosecond values. This is the converse call to rt_timer_ticks2ns().

Parameters

| ns | The count of nanoseconds to convert. |
Returns

The corresponding value expressed in clock ticks of the Alchemy clock. The resolution of the Alchemy clock can be set using the –alchemy-clock-resolution option when starting the application process (defaults to 1 nanosecond).

Tags

unrestricted

Examples:

cross-link.c, and rtcansend.c.

6.89.3.3 rt_timer_read()

RTIME rt_timer_read (
    void ) [inline], [static]

Return the current system time.

Return the current time maintained by the Xenomai core clock.

Returns

The current time expressed in clock ticks (see note).

Tags

unrestricted

Note

The time value is a multiple of the Alchemy clock resolution (see –alchemy-clock-resolution option, defaults to 1 nanosecond).

Examples:

cross-link.c.

6.89.3.4 rt_timer_spin()

void rt_timer_spin ( 
    RTIME ns )

Busy wait burning CPU cycles.

Enter a busy waiting loop for a count of nanoseconds.

Since this service is always called with interrupts enabled, the caller might be preempted by other real-time activities, therefore the actual delay might be longer than specified.
Parameters

**ns** | The time to wait expressed in nanoseconds.

Tags

*unrestricted*

### 6.89.3.5  
rt_timer_ticks2ns()

```c
SRTIME rt_timer_ticks2ns ( 
    SRTIME ns )
```

Convert Alchemy clock ticks to nanoseconds.

Convert a count of Alchemy clock ticks to nanoseconds. This routine operates on signed nanosecond values. This is the converse call to *rt_timer_ns2ticks()*.

Parameters

**ns** | The count of nanoseconds to convert.

Returns

The corresponding value expressed in nanoseconds. The resolution of the Alchemy clock can be set using the `--alchemy-clock-resolution` option when starting the application process (defaults to 1 nanosecond).

Tags

*unrestricted*
A VxWorks® emulation library on top of Xenomai.

The emulator mimicks the behavior described in the public documentation of the WIND 5.x API for the following class of services:

- taskLib, taskInfoLib, taskHookLib
- semLib, msgQLib, wdLib, memPartLib
- intLib, tickLib, sysLib (partial)
- errnoLib, lstLib, kernelLib (partial)
6.91 pSOS® emulator

A pSOS® emulation library on top of Xenomai.

The emulator mimicks the behavior described in the public documentation of the pSOS 2.x API for the following class of services:

- Tasks, Events, Queues, Semaphores
- Partitions, Regions, Timers
6.92 Transition Kit

A set of wrappers and services easing the transition from Xenomai 2.x to 3.x.

- `int COMPAT__rt_task_create (RT_TASK *task, const char *name, int stksize, int prio, int mode)`
  Create a real-time task (compatibility service).
- `int COMPAT__rt_task_set_periodic (RT_TASK *task, RTIME idate, RTIME period)`
  Make a real-time task periodic (compatibility service).
- `int COMPAT__rt_alarm_create (RT_ALARM *alarm, const char *name)`
  Create an alarm object (compatibility service).
- `int rt_alarm_wait (RT_ALARM *alarm)`
  Wait for the next alarm shot (compatibility service).
- `int COMPAT__rt_event_create (RT_EVENT *event, const char *name, unsigned long ivalue, int mode)`
  Create an event flag group.
- `int COMPAT__rt_event_signal (RT_EVENT *event, unsigned long mask)`
  Signal an event.
- `int COMPAT__rt_event_clear (RT_EVENT *event, unsigned long mask, unsigned long *mask_r)`
  Clear event flags.
- `int COMPAT__rt_pipe_create (RT_PIPE *pipe, const char *name, int minor, size_t poolsize)`
  Create a message pipe.
- `int pthread_make_periodic_np (pthread_t thread, struct timespec *starttp, struct timespec *periodtp)`
  Make a thread periodic (compatibility service).
- `int pthread_wait_np (unsigned long *overruns_r)`
  Wait for the next periodic release point (compatibility service).

6.92.1 Detailed Description

A set of wrappers and services easing the transition from Xenomai 2.x to 3.x.

This interface provides a source compatibility layer for building applications based on the Xenomai 2.x posix and native APIs over Xenomai 3.x.

6.92.2 Function Documentation

6.92.2.1 COMPAT__rt_alarm_create()

```c
int COMPAT__rt_alarm_create (  
    RT_ALARM *alarm,  
    const char *name )
```

Create an alarm object (compatibility service).

This routine creates an object triggering an alarm routine at a specified time in the future. Alarms can be periodic or oneshot, depending on the reload interval value passed to `rt_alarm_start()`. A task can wait for timeouts using the `rt_alarm_wait()` service.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>alarm</strong></td>
<td>The address of an alarm descriptor which can be later used to identify uniquely the created object, upon success of this call.</td>
</tr>
<tr>
<td><strong>name</strong></td>
<td>An ASCII string standing for the symbolic name of the alarm. When non-NULL and non-empty, a copy of this string is used for indexing the created alarm into the object registry.</td>
</tr>
</tbody>
</table>

Returns

Zero is returned upon success. Otherwise:

- -ENOMEM is returned if the system fails to get memory from the local pool in order to create the alarm.
- -EEXIST is returned if the name is conflicting with an already registered alarm.
- -EPERM is returned if this service was called from an asynchronous context.

Tags

thread-unrestricted, switch-secondary

Note

Alarms are process-private objects and thus cannot be shared by multiple processes, even if they belong to the same Xenomai session.

**Deprecated** This is a compatibility service from the Transition Kit.

6.92.2.2 COMPAT__rt_event_clear()

```c
int COMPAT__rt_event_clear ( 
    RT_EVENT * event, 
    unsigned long mask, 
    unsigned long * mask_r )
```

Clear event flags.

This call is the legacy form of the `rt_event_clear()` service, using a long event mask. The new form uses a regular integer to hold the event mask instead.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>event</strong></td>
<td>The event descriptor.</td>
</tr>
<tr>
<td><strong>mask</strong></td>
<td>The set of event flags to be cleared.</td>
</tr>
<tr>
<td><strong>mask→_r</strong></td>
<td>If non-NULL, mask_r is the address of a memory location which will receive the previous value of the event flag group before the flags are cleared.</td>
</tr>
</tbody>
</table>
Returns

Zero is returned upon success. Otherwise:

- -EINVAL is returned if event is not a valid event flag group descriptor.

Tags

unrestricted, switch-primary

Deprecated This is a compatibility service from the Transition Kit.

6.92.2.3 COMPAT__rt_event_create()

```
int COMPAT__rt_event_create (
    RT_EVENT ∗ event,
    const char ∗ name,
    unsigned long ivalue,
    int mode )
```

Create an event flag group.

This call is the legacy form of the rt_event_create() service, using a long event mask. The new form
uses a regular integer to hold the event mask instead.

Parameters

| event | The address of an event descriptor which can be later used to identify uniquely the created object, upon success of this call. |
| name | An ASCII string standing for the symbolic name of the event. When non-NULL and non-empty, a copy of this string is used for indexing the created event into the object registry. |
| ivalue | The initial value of the group's event mask. |
| mode | The event group creation mode. The following flags can be OR'ed into this bitmask: |

- EV_FIFO makes tasks pend in FIFO order on the event flag group.
- EV_PRIIO makes tasks pend in priority order on the event flag group.

Returns

Zero is returned upon success. Otherwise:

- -EINVAL is returned if mode is invalid.
- -ENOMEM is returned if the system fails to get memory from the main heap in order to create the event flag group.
- EEXIST is returned if the name is conflicting with an already registered event flag group.
- EPERM is returned if this service was called from an asynchronous context.

Tags
- thread-unrestricted, switch-secondary

Note
Event flag groups can be shared by multiple processes which belong to the same Xenomai session.

Deprecated This is a compatibility service from the Transition Kit.

6.92.2.4 COMPAT__rt_event_signal()

```c
int COMPAT__rt_event_signal (
    RT_EVENT * event,
    unsigned long mask )
```

Signal an event.

This call is the legacy form of the `rt_event_signal()` service, using a long event mask. The new form uses a regular integer to hold the event mask instead.

Parameters

<table>
<thead>
<tr>
<th>event</th>
<th>The event descriptor.</th>
</tr>
</thead>
<tbody>
<tr>
<td>mask</td>
<td>The set of events to be posted.</td>
</tr>
</tbody>
</table>

Returns

Zero is returned upon success. Otherwise:

- EINVAL is returned if event is not an event flag group descriptor.

Tags
- unrestricted, switch-primary

Deprecated This is a compatibility service from the Transition Kit.
6.92.2.5 COMPAT__rt_pipe_create()

```c
int COMPAT__rt_pipe_create(
    RT_PIPE * pipe,
    const char * name,
    int minor,
    size_t poolsize )
```

Create a message pipe.

This call is the legacy form of the `rt_pipe_create()` service, which returns a zero status upon success. The new form returns the `minor` number assigned to the connection instead, which is useful when `P_MINOR_AUTO` is specified in the call (see the discussion about the `minor` parameter).

This service opens a bi-directional communication channel for exchanging messages between Xenomai threads and regular Linux threads. Pipes natively preserve message boundaries, but can also be used in byte-oriented streaming mode from Xenomai to Linux.

`rt_pipe_create()` always returns immediately, even if no thread has opened the associated special device file yet. On the contrary, the non real-time side could block upon attempt to open the special device file until `rt_pipe_create()` is issued on the same pipe from a Xenomai thread, unless `O_NONBLOCK` was given to the `open(2)` system call.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pipe</code></td>
<td>The address of a pipe descriptor which can be later used to identify uniquely the created object, upon success of this call.</td>
</tr>
<tr>
<td><code>name</code></td>
<td>An ASCII string standing for the symbolic name of the pipe. When non-NULL and non-empty, a copy of this string is used for indexing the created pipe into the object registry.</td>
</tr>
<tr>
<td><code>minor</code></td>
<td>The minor number of the device associated with the pipe. Passing <code>P_MINOR_AUTO</code> causes the minor number to be auto-allocated. In such a case, a symbolic link will be automatically created from <code>/proc/xenomai/registry/rtipc/xddp/name</code> to the allocated pipe device entry. Valid minor numbers range from 0 to <code>CONFIG_XENO_OPT_PIPE_NRDEV-1</code>.</td>
</tr>
<tr>
<td><code>poolsize</code></td>
<td>Specifies the size of a dedicated buffer pool for the pipe. Passing 0 means that all message allocations for this pipe are performed on the Cobalt core heap.</td>
</tr>
</tbody>
</table>

Named pipes are supported through the use of the registry. Passing a valid `name` parameter when creating a message pipe causes a symbolic link to be created from `/proc/xenomai/registry/rtipc/xddp/name` to the associated special device (i.e. `/dev/rtp*`), so that the specific `minor` information does not need to be known from those processes for opening the proper device file. In such a case, both sides of the pipe only need to agree upon a symbolic name to refer to the same data path, which is especially useful whenever the `minor` number is picked up dynamically using an adaptive algorithm, such as passing `P_MINOR_AUTO` as `minor` value.

**Returns**

This compatibility call returns zero upon success. Otherwise:

- `-ENOMEM` is returned if the system fails to get memory from the main heap in order to create the pipe.
• -ENODEV is returned if \textit{minor} is different from P\_MINOR\_AUTO and is not a valid minor number.
• -EEXIST is returned if the \textit{name} is conflicting with an already registered pipe.
• -EBUSY is returned if \textit{minor} is already open.
• -EPERM is returned if this service was called from an asynchronous context.

Tags

\texttt{thread-unrestricted, switch-secondary}

6.92.2.6 \texttt{COMPAT\_rt_task\_create()}

\begin{verbatim}
int COMPAT__rt_task_create (  
    RT_TASK * task,  
    const char * name,  
    int stksize,  
    int prio,  
    int mode )
\end{verbatim}

Create a real-time task (compatibility service).

This service creates a task with access to the full set of Xenomai real-time services. If \textit{prio} is non-zero, the new task belongs to Xenomai's real-time FIFO scheduling class, aka SCHED\_FIFO. If \textit{prio} is zero, the task belongs to the regular SCHED\_OTHER class.

Creating tasks with zero priority is useful for running non real-time processes which may invoke blocking real-time services, such as pending on a semaphore, reading from a message queue or a buffer, and so on.

Once created, the task is left dormant until it is actually started by \texttt{rt\_task\_start()}.

Parameters

\begin{tabular}{|l|l|}
\hline
\textit{task} & The address of a task descriptor which can be later used to identify uniquely the created object, upon success of this call. \\
\hline
\textit{name} & An ASCII string standing for the symbolic name of the task. When non-NULL and non-empty, a copy of this string is used for indexing the created task into the object registry. \\
\hline
\textit{stksize} & The size of the stack (in bytes) for the new task. If zero is passed, a system-dependent default size will be substituted. \\
\hline
\textit{prio} & The base priority of the new task. This value must be in the [0 .. 99] range, where 0 is the lowest effective priority. \\
\hline
\textit{mode} & The task creation mode. The following flags can be OR'ed into this bitmask: \\
\hline
\end{tabular}

\begin{itemize}
\item T\_FPU allows the task to use the FPU whenever available on the platform. This flag may be omitted, as it is automatically set when a FPU is present on the platform, cleared otherwise.
\item T\_SUSP causes the task to start in suspended mode. In such a case, the thread will have to be explicitly resumed using the \texttt{rt\_task\_resume()} service for its execution to actually begin.
\end{itemize}
- T_CPU(cpuid) makes the new task affine to CPU # cpuid. CPU identifiers range from 0 to 7 (inclusive).
- T_JOINABLE allows another task to wait on the termination of the new task. rt_task_join() shall be called for this task to clean up any resources after its termination.

Passing T_FPU|T_CPU(1) in the mode parameter thus creates a task with FPU support enabled and which will be affine to CPU #1.

- When running over the Cobalt core, T_WARNSW causes the SIGDEBUG signal to be sent to the current task whenever it switches to the secondary mode. This feature is useful to detect unwanted migrations to the Linux domain. This flag has no effect over the Mercury core.

Returns

Zero is returned upon success. Otherwise:

- EINVAL is returned if either prio, mode or stksize are invalid.
- ENOMEM is returned if the system fails to get memory from the main heap in order to create the task.
- EEXIST is returned if the name is conflicting with an already registered task.

Tags

thread-unrestricted, switch-secondary

Side effects

- calling rt_task_create() causes SCHED_FIFO tasks to switch to secondary mode.
- members of Xenomai's SCHED_FIFO class running in the primary domain have utmost priority over all Linux activities in the system, including Linux interrupt handlers.

Note

Tasks can be referred to from multiple processes which all belong to the same Xenomai session.

Deprecated This is a compatibility service from the Transition Kit.

6.92.2.7 COMPAT__rt_task_set_periodic()

int COMPAT__rt_task_set_periodic (  
    RT_TASK * task,  
    RTIME  idate,  
    RTIME  period )

Make a real-time task periodic (compatibility service).

Make a task periodic by programing its first release point and its period in the processor time line. task should then call rt_task_wait_period() to sleep until the next periodic release point in the processor timeline is reached.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task</td>
<td>The task descriptor. If task is NULL, the current task is made periodic. task must belong the current process.</td>
</tr>
<tr>
<td>idate</td>
<td>The initial (absolute) date of the first release point, expressed in clock ticks (see note). If idate is equal to TM_NOW, the current system date is used. Otherwise, if task is NULL or equal to rt_task_self(), the caller is delayed until idate has elapsed.</td>
</tr>
<tr>
<td>period</td>
<td>The period of the task, expressed in clock ticks (see note). Passing TM_INFINITE stops the task's periodic timer if enabled, then returns successfully.</td>
</tr>
</tbody>
</table>

Returns

Zero is returned upon success. Otherwise:

- -EINVAL is returned if task is NULL but the caller is not a Xenomai task, or if task is non-NULL but not a valid task descriptor.
- -ETIMEDOUT is returned if idate is different from TM_INFINITE and represents a date in the past.

Tags

thread-unrestricted, switch-primary

Note

The caller must be an Alchemy task if task is NULL. Unlike the original Xenomai 2.x call, this emulation delays the caller until idate has elapsed only if task is NULL or equal to rt_task_self().

Side effects

Over Cobalt, -EINVAL is returned if period is different from TM_INFINITE but shorter than the user scheduling latency value for the target system, as displayed by /proc/xenomai/latency.

Note

The idate and period values are interpreted as a multiple of the Alchemy clock resolution (see –alchemy-clock-resolution option, defaults to 1 nanosecond).

Deprecated

This is a compatibility service from the Transition Kit.

6.92.2.8  pthread_make_periodic_np()

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

Make a thread periodic (compatibility service).

This service makes the POSIX thread periodic.
6.92 Transition Kit

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>thread</td>
<td>thread to arm a periodic timer for.</td>
</tr>
<tr>
<td>starttp</td>
<td>start time, expressed as an absolute value of the CLOCK_REALTIME clock.</td>
</tr>
<tr>
<td>periodtp</td>
<td>period, expressed as a time interval.</td>
</tr>
</tbody>
</table>

Returns

- 0 on success;
- an error number if:
  - ESRCH, thread is invalid.
  - ETIMEDOUT, the start time has already passed.
  - EPERM, the caller is not a Xenomai thread.
  - EINVAL, thread does not refer to the current thread.

Note

Unlike the original Xenomai 2.x call, this emulation does not delay the caller waiting for the first periodic release point. In addition, thread must be equal to pthread_self().

Deprecated

This service is a non-portable extension of the Xenomai 2.x POSIX interface, not available with Xenomai 3.x. Instead, Cobalt-based applications should set up a periodic timer using the timer_create(), timer_settime() call pair, then wait for release points via sigwaitinfo(). Overruns can be detected by looking at the siginfo.si_overrun field. Alternatively, applications may obtain a file descriptor referring to a Cobalt timer via the timerfd() call, and read() from it to wait for timeouts.

References timer_create(), and timer_settime().

6.92.2.9 pthread_wait_np()

int pthread_wait_np (
    unsigned long ∗ overruns_r )

Wait for the next periodic release point (compatibility service)

Delay the current thread until the next periodic release point is reached. The periodic timer should have been previously started for thread by a call to pthread_make_periodic_npt().

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>overruns− _r</td>
<td>If non-NULL, overruns_r shall be a pointer to a memory location which will be written with the count of pending overruns. This value is written to only when pthread_wait_np() returns ETIMEDOUT or success. The memory location remains unmodified otherwise. If NULL, this count will not be returned.</td>
</tr>
</tbody>
</table>
Returns

Zero is returned upon success. If `overruns_r` is non-NULL, zero is written to the pointed memory location. Otherwise:

- EWOULDBLOCK is returned if `pthread_make_periodic_np()` was not called for the current thread.
- EINTR is returned if `thread` was interrupted by a signal before the next periodic release point was reached.
- ETIMEDOUT is returned if a timer overrun occurred, which indicates that a previous release point was missed by the calling thread. If `overruns_r` is non-NULL, the count of pending overruns is written to the pointed memory location.
- EPERM is returned if this service was called from an invalid context.

Note

If the current release point has already been reached at the time of the call, the current thread immediately returns from this service with no delay.

Deprecated

This service is a non-portable extension of the Xenomai 2.x POSIX interface, not available with Xenomai 3.x. Instead, Cobalt-based applications should set up a periodic timer using the `timer_create()`, `timer_settime()` call pair, then wait for release points via `sigwaitinfo()`. Overruns can be detected by looking at the `siginfo.si_overrun` field. Alternatively, applications may obtain a file descriptor referring to a Cobalt timer via the `timerfd()` call, and `read()` from it to wait for timeouts.

6.92.2.10 `rt_alarm_wait()`

```c
int rt_alarm_wait ( RT_ALARM *alarm )
```

Wait for the next alarm shot (compatibility service).

This service allows the current task to suspend execution until the specified alarm triggers. The priority of the current task is raised above all other tasks - except those also undergoing an alarm wait.

Returns

Zero is returned upon success, after the alarm timed out. Otherwise:

- EINVAL is returned if `alarm` is not a valid alarm descriptor.
- EPERM is returned if this service was called from an invalid context.
- EINVAL is returned if `rt_task_unblock()` was called for the current task before the request is satisfied.
- EIDRM is returned if `alarm` is deleted while the caller was sleeping on it. In such a case, `alarm` is no more valid upon return of this service.

Tags

`xthread-only, switch-primary`

Deprecated

This is a compatibility service from the Transition Kit.
Chapter 7

Data Structure Documentation

7.1  a4l_channel Struct Reference

Structure describing some channel's characteristics.

Data Fields

- unsigned long flags
- unsigned long nb_bits

7.1.1  Detailed Description

Structure describing some channel's characteristics.

7.1.2  Field Documentation

7.1.2.1  flags

unsigned long a4l_channel::flags

Channel flags to define the reference.

7.1.2.2  nb_bits

unsigned long a4l_channel::nb_bits

Channel resolution.

Referenced by a4l_get_chan().

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

- include/cobalt/kernel/rdm/analogy/channel_range.h
7.2  a4l_channels_desc Struct Reference

Structure describing a channels set.

Collaboration diagram for a4l_channels_desc:

Data Fields

- unsigned long mode
- unsigned long length
- struct a4l_channel chans []

7.2.1 Detailed Description

Structure describing a channels set.

7.2.2 Field Documentation

7.2.2.1 chans

struct a4l_channel a4l_channels_desc::chans[]

Channels tab

Referenced by a4l_get_chan().
7.2.2.2 length

unsigned long a4l_channels_desc::length
Channels count

7.2.2.3 mode

unsigned long a4l_channels_desc::mode
Declaration mode (global or per channel)
Referenced by a4l_get_chan().
The documentation for this struct was generated from the following file:
- include/cobalt/kernel/rtdm/analogy/channel_range.h

7.3 a4l_cmd_desc Struct Reference

Structure describing the asynchronous instruction.

Data Fields
- unsigned char idx_subd
  Subdevice to which the command will be applied.
- unsigned long flags
  Command flags.
- unsigned int start_src
  Start trigger type.
- unsigned int start_arg
  Start trigger argument.
- unsigned int scan_begin_src
  Scan begin trigger type.
- unsigned int scan_begin_arg
  Scan begin trigger argument.
- unsigned int convert_src
  Convert trigger type.
- unsigned int convert_arg
  Convert trigger argument.
- unsigned int scan_end_src
  Scan end trigger type.
- unsigned int scan_end_arg
  Scan end trigger argument.
- unsigned int stop_src
  Stop trigger type.
- unsigned int stop_arg
  Stop trigger argument.
- unsigned char nb_chan
  Count of channels related with the command.
- unsigned int * chan_descs
  Tab containing channels descriptors.
- unsigned int data_len
  < cmd simulation valid stages (driver dependent)
- sampl_t * data
  Driver specific buffer pointer.
7.3.1 Detailed Description

Structure describing the asynchronous instruction.

See also

a4l_snd_command()

7.3.2 Field Documentation

7.3.2.1 data_len

unsigned int a4l_cmd_desc::data_len
< cmd simulation valid stages (driver dependent)
Driver specific buffer size

7.3.2.2 idx_subd

unsigned char a4l_cmd_desc::idx_subd
Subdevice to which the command will be applied.
The documentation for this struct was generated from the following file:

- include/rtdm/uapi/analogy.h

7.4 a4l_descriptor Struct Reference

Structure containing device-information useful to users.

Data Fields

- char board_name [A4L_NAMELEN]
  Board name.
- char driver_name [A4L_NAMELEN]
  Driver name.
- int nb_subd
  Subdevices count.
- int idx_read_subd
  Input subdevice index.
- int idx_write_subd
  Output subdevice index.
- int fd
  File descriptor.
- unsigned int magic
  Opaque field.
- int sbsize
  Data buffer size.
- void * sbdata
  Data buffer pointer.
7.4.1 Detailed Description

Structure containing device-information useful to users.

See also

a4l_get_desc()

7.4.2 Field Documentation

7.4.2.1 board_name

char a4l_descriptor::board_name[A4L_NAMELEN]

Board name.

7.4.2.2 driver_name

char a4l_descriptor::driver_name[A4L_NAMELEN]

Driver name.

7.4.2.3 fd

int a4l_descriptor::fd

File descriptor.

Referenced by a4l_close(), a4l_fill_desc(), a4l_get_bufsize(), a4l_mark_bufrw(), a4l_mmap(), a4l_open(), a4l_poll(), a4l_set_bufsize(), a4l_snd_cancel(), a4l_snd_command(), a4l_snd_insn(), and a4l_snd_insnlist().

7.4.2.4 idx_read_subd

int a4l_descriptor::idx_read_subd

Input subdevice index.
7.4.2.5 idx_write_subd

int a4l_descriptor::idx_write_subd

Output subdevice index.

7.4.2.6 magic

unsigned int a4l_descriptor::magic

Opaque field.

Referenced by a4l_fill_desc(), a4l_find_range(), a4l_get_chinfo(), a4l_get_rnginfo(), a4l_get_subdinfo(), and a4l_sys_desc().

7.4.2.7 nb_subd

int a4l_descriptor::nb_subd

Subdevices count.

7.4.2.8 sbdata

void* a4l_descriptor::sbdata

Data buffer pointer.

7.4.2.9 sbsize

int a4l_descriptor::sbsize

Data buffer size.

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

- include/rtdm/analogy.h

7.5 a4l_driver Struct Reference

Structure containing driver declaration data.
Data Fields

- `struct list_head list`
  List stuff.
- `struct module * owner`
  Pointer to module containing the code.
- `unsigned int flags`
  Type / status driver's flags.
- `char * board_name`
  Board name.
- `char * driver_name`
  Driver name
- `int privdata_size`
  Size of the driver's private data.
- `int( attach )(struct a4l_device *, struct a4l_link_desc *)`
  Attach procedure.
- `int( detach )(struct a4l_device *)`
  Detach procedure.

7.5.1 Detailed Description

Structure containing driver declaration data.

See also

- `rt_task_inquire()`

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

- `include/cobalt/kernel/rtdm/analogy/driver.h`

7.6 a4l_instruction Struct Reference

Structure describing the synchronous instruction.

Data Fields

- `unsigned int type`
  Instruction type.
- `unsigned int idx_subd`
  Subdevice to which the instruction will be applied.
- `unsigned int chan_desc`
  Channel descriptor.
- `unsigned int data_size`
  Size of the instruction data.
- `void * data`
  Instruction data.
7.6.1 Detailed Description

Structure describing the synchronous instruction.

See also

\texttt{a4l\_snd\_insn()}

7.6.2 Field Documentation

7.6.2.1 \texttt{idx\_subd}

\texttt{unsigned int a4l\_instruction::idx\_subd}

Subdevice to which the instruction will be applied.

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

- \texttt{include/rtdm/uapi/analogy.h}

7.7 \texttt{a4l\_instruction\_list} Struct Reference

Structure describing the list of synchronous instructions.

Collaboration diagram for \texttt{a4l\_instruction\_list}:

\begin{figure}
\begin{center}
\includegraphics[width=0.5\textwidth]{collaboration.png}
\end{center}
\end{figure}

Data Fields

- \texttt{unsigned int count}
  \textit{Instructions count.}
- \texttt{a4l\_insn\_t \* insns}
  \textit{Tab containing the instructions pointers.}
7.7.1 Detailed Description

Structure describing the list of synchronous instructions.

See also

a4l_snd_insnlist()

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

- include/rtdm/uapi/analogy.h

7.8 a4l_range Struct Reference

Structure describing a (unique) range.

Data Fields

- long min
- long max
- unsigned long flags

7.8.1 Detailed Description

Structure describing a (unique) range.

7.8.2 Field Documentation

7.8.2.1 flags

unsigned long a4l_range::flags

Range flags (unit, etc.)

7.8.2.2 max

long a4l_range::max

Maximal value
7.8.2.3 min

long a4l_range::min

Minimal value

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

- include/cobalt/kernel/rtdm/analogy/channel_range.h

7.9 a4l_subdevice Struct Reference

Structure describing the subdevice.

Collaboration diagram for a4l_subdevice:

```
    a4l_channel
       chans
    a4l_cmd_desc
    a4l_channels_desc
       cmd_mask
       chan_desc
    a4l_subdevice
```

Data Fields

- struct list_head list
  
  List stuff.
- struct a4l_device * dev
  
  Containing device.
- unsigned int idx
  
  Subdevice index.
- struct a4l_buffer * buf
  
  Linked buffer.
- unsigned long status
  
  Subdevice's status.
- unsigned long flags
Type flags.

- struct a4l_channels_desc * chan_desc
  Tab of channels descriptors pointers.
- struct a4l_rngdesc * rng_desc
  Tab of ranges descriptors pointers.
- struct a4l_cmd_desc * cmd_mask
  Command capabilities mask.

- int (* insn_read )(struct a4l_subdevice *, struct a4l_kernel_instruction *)
  Callback for the instruction "read".
- int (* insn_write )(struct a4l_subdevice *, struct a4l_kernel_instruction *)
  Callback for the instruction "write".
- int (* insn_bits )(struct a4l_subdevice *, struct a4l_kernel_instruction *)
  Callback for the instruction "bits".
- int (* insn_config )(struct a4l_subdevice *, struct a4l_kernel_instruction *)
  Callback for the configuration instruction.
- int (* do_cmd )(struct a4l_subdevice *, struct a4l_cmd_desc *)
  Callback for command handling.
- int (* do_cmdtest )(struct a4l_subdevice *, struct a4l_cmd_desc *)
  Callback for command checking.
- void (* cancel )(struct a4l_subdevice *)
  Callback for asynchronous transfer cancellation.
- void (* munge )(struct a4l_subdevice *, void *, unsigned long)
  Callback for munge operation.
- int (* trigger )(struct a4l_subdevice *, lsampl_t)
  Callback for trigger operation.
- char priv [0]
  Private data.

7.9.1 Detailed Description

Structure describing the subdevice.

See also

a4l_add_subd()

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

- include/cobalt/kernel/rtdm/analogy/subdevice.h

7.10 atomic_t Struct Reference

Copyright © 2011 Gilles Chanteperdrix gilles.chanteperdrix@xenomai.org.
7.10.1 Detailed Description

Copyright © 2011 Gilles Chanteperdrix gilles.chanteperdrix@xenomai.org.

Copyright © 2013 Philippe Gerum rpm@xenomai.org.

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

This library 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 Lesser General Public License for more details.

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

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

- include/boilerplate/atomic.h

7.11 can_bittime Struct Reference

Custom CAN bit-time definition.

Collaboration diagram for can_bittime:

![Collaboration Diagram]

Data Fields

- can_bittime_type_t type
  Type of bit-time definition.
- struct can_bittime_std std
  Standard bit-time.
- struct can_bittime_btr btr
  Hardware-specific BTR bit-time.
7.12 can_bittime_btr Struct Reference

Hardware-specific BTR bit-times.

Data Fields

- `uint8_t btr0`
  Bus timing register 0.
- `uint8_t btr1`
  Bus timing register 1.

7.12.1 Detailed Description

Hardware-specific BTR bit-times.

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

- include/rtdm/uapi/can.h

7.13 can_bittime_std Struct Reference

Standard bit-time parameters according to Bosch.

Data Fields

- `uint32_t brp`
  Baud rate prescaler.
- `uint8_t prop_seg`
  from 1 to 8
- `uint8_t phase_seg1`
  from 1 to 8
- `uint8_t phase_seg2`
  from 1 to 8
- `uint8_t sjw:7`
  from 1 to 4
- `uint8_t sam:1`
  1 - enable triple sampling

Generated by Doxygen
7.13.1 Detailed Description

Standard bit-time parameters according to Bosch.

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

- include/rtdm/uapi/can.h

7.14 can_filter Struct Reference

Filter for reception of CAN messages.

Data Fields

- `uint32_t can_id`
  CAN ID which must match with incoming IDs after passing the mask.
- `uint32_t can_mask`
  Mask which is applied to incoming IDs.

7.14.1 Detailed Description

Filter for reception of CAN messages.

This filter works as follows: A received CAN ID is AND'ed bitwise with `can_mask` and then compared to `can_id`. This also includes the `CAN_EFF_FLAG` and `CAN_RTR_FLAG` of `CAN_xxx_FLAG`. If this comparison is true, the message will be received by the socket. The logic can be inverted with the `can_id` flag `CAN_INV_FILTER`:

```c
if (can_id & CAN_INV_FILTER) {
    if ((received_can_id & can_mask) != (can_id & ~CAN_INV_FILTER))
        accept-message;
} else {
    if ((received_can_id & can_mask) == can_id)
        accept-message;
}
```

Multiple filters can be arranged in a filter list and set with `Sockopts`. If one of these filters matches a CAN ID upon reception of a CAN frame, this frame is accepted.

Examples:

- `can-rtt.c`, and `rcanrecv.c`.

7.14.2 Field Documentation
7.14.2.1 can_id

uint32_t can_filter::can_id

CAN ID which must match with incoming IDs after passing the mask.

The filter logic can be inverted with the flag CAN_INV_FILTER.

Examples:
   can-rtt.c, and rtcanrecv.c.

7.14.2.2 can_mask

uint32_t can_filter::can_mask

Mask which is applied to incoming IDs.

See CAN ID masks if exactly one CAN ID should come through.

Examples:
   can-rtt.c, and rtcanrecv.c.

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

   • include/rtdm/uapi/can.h

7.15 can_frame Struct Reference

Raw CAN frame.

Public Member Functions

   • uint8_t data [8] __attribute__((aligned(8)))

   Payload data bytes.

Data Fields

   • can_id_t can_id

   CAN ID of the frame.

   • uint8_t can_dlc

   Size of the payload in bytes.
7.15.1 Detailed Description

Raw CAN frame.
Central structure for receiving and sending CAN frames.
Examples:
    can-rtt.c, rtcanrecv.c, and rtcansend.c.

7.15.2 Field Documentation

7.15.2.1 can_id

\texttt{can_id_t can_frame::can_id}

CAN ID of the frame.
See CAN ID flags for special bits.
Examples:
    can-rtt.c, and rtcansend.c.

The documentation for this struct was generated from the following file:
- include/rtdm/uapi/can.h

7.16 can_ifreq Struct Reference

CAN interface request descriptor.
Collaboration diagram for can_ifreq:
7.17 macb_dma_desc Struct Reference

7.16.1 Detailed Description

CAN interface request descriptor.

Parameter block for submitting CAN control requests.

Examples:

- can-rtt.c, rtcanconfig.c, rtcanrecv.c, and rtcanwend.c.

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

- include/rtdm/uapi/can.h

7.17 macb_dma_desc Struct Reference

Hardware DMA descriptor.

Data Fields

- u32 addr
  - DMA address of data buffer.
- u32 ctrl
  - Control and status bits.

7.17.1 Detailed Description

Hardware DMA descriptor.

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

- kernel/drivers/net/drivers/rt_macb.h

7.18 macb_tx_skb Struct Reference

Data about an skb which is being transmitted.

Data Fields

- struct rtskb * skb
  - skb currently being transmitted
- dma_addr_t mapping
  - DMA address of the skb's data buffer.
7.18.1 Detailed Description

Data about an skb which is being transmitted.

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

- kernel/drivers/net/drivers/rt_macb.h

7.19 RT_ALARM_INFO Struct Reference

Alarm status descriptor.

Data Fields

- unsigned long expiries
  *Number of past expiries.*
- char name[XOBJECT_NAME_LEN]
  *Name of alarm object.*
- int active
  *Active flag.*

7.19.1 Detailed Description

Alarm status descriptor.

This structure reports various static and runtime information about a real-time alarm, returned by a call to rt_alarm_inquire().

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

- include/alchemy/alarm.h

7.20 RT_BUFFER_INFO Struct Reference

Buffer status descriptor.

Data Fields

- int iwaiters
  *Number of tasks waiting on the read side of the buffer for input data.*
- int owaiters
  *Number of tasks waiting on the write side of the buffer for sending out data.*
- size_t totalmem
  *Overall size of buffer (in bytes).*
- size_t availmem
  *Amount of memory currently available for holding more data.*
- char name[XOBJECT_NAME_LEN]
  *Name of the buffer.*
7.20.1 Detailed Description

Buffer status descriptor.

This structure reports various static and runtime information about a real-time buffer, returned by a call to \texttt{rt_buffer_inquire()}.

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

- include/alchemy(buffer.h)

7.21 RT_COND_INFO Struct Reference

Condition variable status descriptor.

Data Fields

- \texttt{char name [XNOBJECT_NAME_LEN]}
  
  \textit{Name of condition variable.}

7.21.1 Detailed Description

Condition variable status descriptor.

This structure reports various static and runtime information about a condition variable, returned by a call to \texttt{rt_cond_inquire()}.

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

- include/alchemy/cond.h

7.22 RT_EVENT_INFO Struct Reference

Event status descriptor.

Data Fields

- \texttt{unsigned int value}
  
  \textit{Current value of the event flag group.}
- \texttt{int nwaiters}
  
  \textit{Number of tasks currently waiting for events.}
- \texttt{char name [XNOBJECT_NAME_LEN]}
  
  \textit{Name of event flag group.}
7.22.1 Detailed Description

Event status descriptor.

This structure reports various static and runtime information about an event flag group, returned by a call to \texttt{rt_event_inquire()}.

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

- \texttt{include/alchemy/event.h}

7.23 RT_HEAP_INFO Struct Reference

Heap status descriptor.

Data Fields

- \texttt{int nwaiters}
  
  \textit{Number of tasks waiting for available memory in \texttt{rt_heap_alloc()}.

- \texttt{int mode}
  
  \textit{Creation mode flags as given to \texttt{rt_heap_create()}}.

- \texttt{size_t heapsize}
  
  \textit{Size of heap (in bytes) as given to \texttt{rt_heap_create()}}.

- \texttt{size_t usablemem}
  
  \textit{Maximum amount of memory available from the heap}.

- \texttt{size_t usedmem}
  
  \textit{Amount of heap memory currently consumed}.

- \texttt{char name [XNOBJECT_NAME_LEN]}
  
  \textit{Name of heap}.

7.23.1 Detailed Description

Heap status descriptor.

This structure reports various static and runtime information about a real-time heap, returned by a call to \texttt{rt_heap_inquire()}.

7.23.2 Field Documentation

7.23.2.1 heapsize

\texttt{size_t RT_HEAP_INFO::heapsize}

\textit{Size of heap (in bytes) as given to \texttt{rt_heap_create()}}.

The maximum amount of memory available from this heap may be larger, due to internal padding.
7.23.2.2 usablemem

size_t RT_HEAP_INFO::usablemem

Maximum amount of memory available from the heap.
This value accounts for the overhead of internal data structures required to maintain the heap.

7.23.2.3 usedmem

size_t RT_HEAP_INFO::usedmem

Amount of heap memory currently consumed.
info.usablemem - info.usedmem computes the current amount of free memory in the relevant heap.

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

- include/alchemy/heap.h

7.24 RT_MUTEX_INFO Struct Reference

Mutex status descriptor.

Data Fields

- RT_TASK owner
  
  Current mutex owner, or NO_ALCHEMY_TASK if unlocked.
- char name [XOBJECT_NAME_LEN]
  
  Name of mutex.

7.24.1 Detailed Description

Mutex status descriptor.

This structure reports various static and runtime information about a mutex, returned by a call to rt_mutex_inquire().

7.24.2 Field Documentation

Generated by Doxygen
7.24.2.1 owner

RT_TASK RT_MUTEX_INFO::owner

Current mutex owner, or NO_ALCHEMY_TASK if unlocked.

This information is in essence transient, and may not be valid anymore once used by the caller.

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

- include/alchemy/mutex.h

7.25 RT_QUEUE_INFO Struct Reference

Queue status descriptor.

Data Fields

- int nwaiters
  
  *Number of tasks currently waiting on the queue for messages.*

- int nmessages
  
  *Number of messages pending in queue.*

- int mode
  
  
  *Queue mode bits, as given to rt_queue_create().*

- size_t qlimit
  
  
  *Maximum number of messages in queue, zero if unlimited.*

- size_t poolsize
  
  
  *Size of memory pool for holding message buffers (in bytes).*

- size_t usedmem
  
  
  *Amount of memory consumed from the buffer pool.*

- char name [XOBJECT_NAME_LEN]
  
  *Name of message queue.*

7.25.1 Detailed Description

Queue status descriptor.

This structure reports various static and runtime information about a real-time queue, returned by a call to rt_queue_inquire().

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

- include/alchemy/queue.h

7.26 RT_SEM_INFO Struct Reference

Semaphore status descriptor.
Data Fields

- **unsigned long count**
  
  Current semaphore value.

- **int nwaiters**
  
  Number of tasks waiting on the semaphore.

- **char name [XOBJECT_NAME_LEN]**
  
  Name of semaphore.

### 7.26.1 Detailed Description

Semaphore status descriptor.

This structure reports various static and runtime information about a semaphore, returned by a call to `rt_sem_inquire()`.

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

- include/alchemy/sem.h

### 7.27 RT_TASK_INFO Struct Reference

Task status descriptor.

Data Fields

- **int prio**
  
  Task priority.

- **struct threadobj_stat stat**
  
  Task status.

- **char name [XOBJECT_NAME_LEN]**
  
  Name of task.

- **pid_t pid**
  
  Host pid.

### 7.27.1 Detailed Description

Task status descriptor.

This structure reports various static and runtime information about a real-time task, returned by a call to `rt_task_inquire()`.

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

- include/alchemy/task.h

Generated by Doxygen
Timer status descriptor.

Data Fields

- RTIME period
  
  *Clock resolution in nanoseconds.*

- RTIME date
  
  *Current monotonic date expressed in clock ticks.*

7.28.1 Detailed Description

Timer status descriptor.

This structure reports information about the Alchemy clock, returned by a call to `rt_timer_inquire()`.

7.28.2 Field Documentation

7.28.2.1 date

RTIME rt_timer_info::date

Current monotonic date expressed in clock ticks.

The duration of a tick depends on the Alchemy clock resolution for the process (see `--alchemy-clock-resolution` option, defaults to 1 nanosecond).

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

- include/alchemy/timer.h
7.29 rtdm_dev_context Struct Reference

Device context.

Collaboration diagram for rtdm_dev_context:

Data Fields

- `struct rtdm_device ∗ device`
  - Set of active device operation handlers.
- `char dev_private [0]`
  - Begin of driver defined context data structure.

7.29.1 Detailed Description

Device context.

A device context structure is associated with every open device instance. RTDM takes care of its creation and destruction and passes it to the operation handlers when being invoked.

Drivers can attach arbitrary data immediately after the official structure. The size of this data is provided via `rtdm_driver.context_size` during device registration.

7.29.2 Field Documentation
7.29.2.1 device

struct rtdm_device* rtdm_dev_context::device

Set of active device operation handlers.

Reference to owning device

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

- include/cobalt/kernel/rtdm/driver.h

7.30 rtdm_device Struct Reference

RTDM device.

Collaboration diagram for rtdm_device:

Data Fields

- struct rtdm_driver * driver
  
  Device driver.
- void * device_data
  
  Driver definable device data.
- const char * label
  
  Device label template for composing the device name.
- int minor
  
  Minor number of the device.
- struct {
      
  }

  Reserved area.
7.30.1 Detailed Description

RTDM device.

This descriptor describes a RTDM device instance. The structure holds runtime data, therefore it must reside in writable memory.

7.30.2 Field Documentation

7.30.2.1 struct {

    ... }

Reserved area.

7.30.2.2 driver

struct rtadm_driver* rtdm_device::driver

Device driver.

Referenced by rtdm_dev_unregister(), and udd_get_device().

7.30.2.3 label

const char* rtdm_device::label

Device label template for composing the device name.

A limited printf-like format string is assumed, with a provision for replacing the first d/i placeholder found in the string by the device minor number. It is up to the driver to actually mention this placeholder or not, depending on the naming convention for its devices. For named devices, the corresponding device node will automatically appear in the /dev/rtdm hierarchy with hotplug-enabled device filesystems (DEVTMPFS).
7.30.2.4 minor

int rtdm_device::minor

Minor number of the device.

If RTDM_FIXED_MINOR is present in the driver flags, the value stored in this field is used verbatim by rtdm_dev_register(). Otherwise, the RTDM core automatically assigns minor numbers to all devices managed by the driver referred to by driver, in order of registration, storing the resulting values into this field.

Device nodes created for named devices in the Linux /dev hierarchy are assigned this minor number.

The minor number of the current device handling an I/O request can be retrieved by a call to rtdm_fd←minor().

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

- include/cobalt/kernel/rtdm/driver.h

# 7.31 rtdm_device_info Struct Reference

Device information.

## Data Fields

- int device_flags
  
  Device flags, see Device Flags for details.

- int device_class
  
  Device class ID, see RTDM_CLASS_xxx.

- int device_sub_class
  
  Device sub-class, either RTDM_SUBCLASS GENERIC or a RTDM_SUBCLASS_xxx definition of the related Device Profile.

- int profile_version
  
  Supported device profile version.

# 7.31.1 Detailed Description

Device information.

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

- include/rtdm/uapi/rtdm.h
7.32  rtdm_driver Struct Reference

RTDM driver.

Collaboration diagram for rtdm_driver:

Data Fields

- struct rtdm_profile_info profile_info
  
  Class profile information.

- int device_flags
  
  Device flags, see Device Flags for details.

- size_t context_size
  
  Size of the private memory area the core should automatically allocate for each open file descriptor, which is usable for storing the context data associated to each connection.

- int protocol_family
  
  Protocol device identification: protocol family (PF_xxx)

- int socket_type
  
  Protocol device identification: socket type (SOCK_xxx)

- struct rtdm_fd_ops ops
  
  I/O operation handlers.

- struct rtdm_sm_ops smops
  
  State management handlers.

- int device_count
  
  Count of devices this driver manages.

- int base_minor
  
  Base minor for named devices.

- struct {
  }

  Reserved area.
7.32.1 Detailed Description

RTDM driver.

This descriptor describes a RTDM device driver. The structure holds runtime data, therefore it must reside in writable memory.

7.32.2 Field Documentation

7.32.2.1 base_minor

int rtdm_driver::base_minor

Base minor for named devices.

7.32.2.2 context_size

size_t rtdm_driver::context_size

Size of the private memory area the core should automatically allocate for each open file descriptor, which is usable for storing the context data associated to each connection.

The allocated memory is zero-initialized. The start of this area can be retrieved by a call to rtdm_fd_to_private().

7.32.2.3 device_count

int rtdm_driver::device_count

Count of devices this driver manages.

This value is used to allocate a chrdev region for named devices.

7.32.2.4 device_flags

int rtdm_driver::device_flags

Device flags, see Device Flags for details.
7.32.2.5 profile_info

struct rtdm_profile_info rtdm_driver::profile_info

Class profile information.

The RTDM_PROFILE_INFO() macro must be used for filling up this field.

Referenced by rtdm_drv_set_sysclass(), and udd_get_device().

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

- include/cobalt/kernel/rtdm/driver.h

7.33 rtdm_fd_ops Struct Reference

RTDM file operation descriptor.

Data Fields

- int(* open )(struct rtdm_fd *fd, int oflags)
  See rtdm_open_handler().
- int(* socket )(struct rtdm_fd *fd, int protocol)
  See rtdm_socket_handler().
- void(* close )(struct rtdm_fd *fd)
  See rtdm_close_handler().
- int(* ioctl_rt )(struct rtdm_fd *fd, unsigned int request, void __user *arg)
  See rtdm_ioctl_handler().
- int(* ioctl_nrt )(struct rtdm_fd *fd, unsigned int request, void __user *arg)
  See rtdm_ioctl_handler().
- ssize_t(* read_rt )(struct rtdm_fd *fd, void __user *buf, size_t size)
  See rtdm_read_handler().
- ssize_t(* read_nrt )(struct rtdm_fd *fd, void __user *buf, size_t size)
  See rtdm_read_handler().
- ssize_t(* write_rt )(struct rtdm_fd *fd, const void __user *buf, size_t size)
  See rtdm_write_handler().
- ssize_t(* write_nrt )(struct rtdm_fd *fd, const void __user *buf, size_t size)
  See rtdm_write_handler().
- ssize_t(* recvmsg_rt )(struct rtdm_fd *fd, struct user_msghdr *msg, int flags)
  See rtdm_recvmsg_handler().
- ssize_t(* recvmsg_nrt )(struct rtdm_fd *fd, struct user_msghdr *msg, int flags)
  See rtdm_recvmsg_handler().
- ssize_t(* sendmsg_rt )(struct rtdm_fd *fd, const struct user_msghdr *msg, int flags)
  See rtdm_sendmsg_handler().
- ssize_t(* sendmsg_nrt )(struct rtdm_fd *fd, const struct user_msghdr *msg, int flags)
  See rtdm_sendmsg_handler().
- int(* select )(struct rtdm_fd *fd, struct xnselector *selector, unsigned int type, unsigned int index)
  See rtdm_select_handler().
- int(* mmap )(struct rtdm_fd *fd, struct vm_area_struct *vma)
  See rtdm_mmap_handler().
- unsigned long(* get_unmapped_area )(struct rtdm_fd *fd, unsigned long len, unsigned long poff, unsigned long flags)
  See rtdm_get_unmapped_area_handler().
7.33.1 Detailed Description

RTDM file operation descriptor.

This structure describes the operations available with a RTDM device, defining handlers for submitting I/O requests. Those handlers are implemented by RTDM device drivers.

7.33.2 Field Documentation

7.33.2.1 close

```c
void(* rtdm_fd_ops::close) (struct rtdm_fd *fd)
```

See `rtdm_close_handler()`.

7.33.2.2 get_unmapped_area

```c
unsigned long(* rtdm_fd_ops::get_unmapped_area) (struct rtdm_fd *fd, unsigned long len, unsigned long pgoff, unsigned long flags)
```

See `rtdm_get_unmapped_area_handler()`.

7.33.2.3 ioctl_nrt

```c
int(* rtdm_fd_ops::ioctl_nrt) (struct rtdm_fd *fd, unsigned int request, void __user *arg)
```

See `rtdm_ioctl_handler()`.

7.33.2.4 ioctl_rt

```c
int(* rtdm_fd_ops::ioctl_rt) (struct rtdm_fd *fd, unsigned int request, void __user *arg)
```

See `rtdm_ioctl_handler()`.
7.33.2.5  mmap

int(* rtdm_fd_ops::mmap) (struct rtdm_fd *fd, struct vm_area_struct *vma)

See rtdm_mmap_handler().

7.33.2.6  open

int(* rtdm_fd_ops::open) (struct rtdm_fd *fd, int oflags)

See rtdm_open_handler().

7.33.2.7  read_nrt

ssize_t(* rtdm_fd_ops::read_nrt) (struct rtdm_fd *fd, void __user *buf, size_t size)

See rtdm_read_handler().

7.33.2.8  read_rt

ssize_t(* rtdm_fd_ops::read_rt) (struct rtdm_fd *fd, void __user *buf, size_t size)

See rtdm_read_handler().

7.33.2.9  recvmsg_nrt

ssize_t(* rtdm_fd_ops::recvmsg_nrt) (struct rtdm_fd *fd, struct user_msghdr *msg, int flags)

See rtdm_recvmsg_handler().

7.33.2.10  recvmsg_rt

ssize_t(* rtdm_fd_ops::recvmsg_rt) (struct rtdm_fd *fd, struct user_msghdr *msg, int flags)

See rtdm_recvmsg_handler().
7.33.2.11 select

int(* rtdm_fd_ops::select) (struct rtdm_fd *fd, struct xnselector *selector, unsigned int type, unsigned int index)

See rtdm_select_handler().

7.33.2.12 sendmsg_nrt

ssize_t(* rtdm_fd_ops::sendmsg_nrt) (struct rtdm_fd *fd, const struct user_msghdr *msg, int flags)

See rtdm_sendmsg_handler().

7.33.2.13 sendmsg_rt

ssize_t(* rtdm_fd_ops::sendmsg_rt) (struct rtdm_fd *fd, const struct user_msghdr *msg, int flags)

See rtdm_sendmsg_handler().

7.33.2.14 socket

int(* rtdm_fd_ops::socket) (struct rtdm_fd *fd, int protocol)

See rtdm_socket_handler().

7.33.2.15 write_nrt

ssize_t(* rtdm_fd_ops::write_nrt) (struct rtdm_fd *fd, const void __user *buf, size_t size)

See rtdm_write_handler().

7.33.2.16 write_rt

ssize_t(* rtdm_fd_ops::write_rt) (struct rtdm_fd *fd, const void __user *buf, size_t size)

See rtdm_write_handler().

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

- include/cobalt/kernel/rtdm/fd.h
7.34  rtdm_sm_ops Struct Reference

RTDM state management handlers.

Data Fields

- int(start)(struct rtdm_driver *drv)
  
  *Handler called upon transition to COBALT_STATE_WARMUP.*

- int(stop)(struct rtdm_driver *drv)
  
  *Handler called upon transition to COBALT_STATE_TEARDOWN.*

7.34.1  Detailed Description

RTDM state management handlers.

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

- include/cobalt/kernel/rtdm/driver.h

7.35  rtdm_spi_config Struct Reference

7.35.1  Detailed Description

Note

Copyright (C) 2016 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.

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

- include/rtdm/uapi/spi.h

7.36  rtipc_port_label Struct Reference

Port label information structure.
Data Fields

- char label [XNOBJECT_NAME_LEN]
  
  Port label string, null-terminated.

7.36.1 Detailed Description

Port label information structure.

Examples:

  bufp-label.c, iddp-label.c, and xddp-label.c.

7.36.2 Field Documentation

7.36.2.1 label

char rtipc_port_label::label[XNOBJECT_NAME_LEN]

Port label string, null-terminated.

Examples:

  bufp-label.c, iddp-label.c, and xddp-label.c.

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

- include/rtdm/uapi/ipc.h

7.37 rtser_config Struct Reference

Serial device configuration.
Data Fields

- **int config_mask**
  - *mask specifying valid fields, see RTSER_SET_xxx*
- **int baud_rate**
  - *baud rate, default RTSER_DEF_BAUD*
- **int parity**
  - *number of parity bits, see RTSER_xxx_PARITY*
- **int data_bits**
  - *number of data bits, see RTSER_xxx_BITS*
- **int stop_bits**
  - *number of stop bits, see RTSER_xxx_STOPB*
- **int handshake**
  - *handshake mechanisms, see RTSER_xxx_HAND*
- **int fifo_depth**
  - *reception FIFO interrupt threshold, see RTSER_FIFO_xxx*
- **nanosecs_rel_t rx_timeout**
  - *reception timeout, see RTSER_TIMEOUT_xxx for special values*
- **nanosecs_rel_t tx_timeout**
  - *transmission timeout, see RTSER_TIMEOUT_xxx for special values*
- **nanosecs_rel_t event_timeout**
  - *event timeout, see RTSER_TIMEOUT_xxx for special values*
- **int timestamp_history**
  - *enable timestamp history, see RTSER_xxx_TIMESTAMP_HISTORY*
- **int event_mask**
  - *event mask to be used with RTSER_RTIOC_WAIT_EVENT, see RTSER_EVENT_xxx*
- **int rs485**
  - *enable RS485 mode, see RTSER_RS485_xxx*

### 7.37.1 Detailed Description

Serial device configuration.

Examples:

`cross-link.c`

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

- `include/rtdm/uapi/serial.h`

### 7.38 rtser_event Struct Reference

Additional information about serial device events.
Data Fields

- int **events**
  - signalled events, see **RTSER_EVENT_xxx**
- int **rx_pending**
  - number of pending input characters
- **nanosecs_abs_t last_timestamp**
  - last interrupt timestamp
- **nanosecs_abs_t rxpend_timestamp**
  - reception timestamp of oldest character in input queue

### 7.38.1 Detailed Description

Additional information about serial device events.

Examples:

`cross-link.c`

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

- include/rtdm/uapi/serial.h

### 7.39 rtser_status Struct Reference

Serial device status.

Data Fields

- int **line_status**
  - line status register, see **RTSER_LSR_xxx**
- int **modem_status**
  - modem status register, see **RTSER_MSR_xxx**

### 7.39.1 Detailed Description

Serial device status.

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

- include/rtdm/uapi/serial.h

### 7.40 sockaddr_can Struct Reference

Socket address structure for the CAN address family.
Data Fields

- `sa_family_t can_family`  
  CAN address family, must be `AF_CAN`.
- `int can_ifindex`  
  Interface index of CAN controller.

7.40.1 Detailed Description

Socket address structure for the CAN address family.

Examples:

- `can-rtt.c`, `rtcanrecv.c`, and `rtcansend.c`.

7.40.2 Field Documentation

7.40.2.1 can_ifindex

`int sockaddr_can::can_ifindex`

Interface index of CAN controller.

See `SIOCGIFINDEX`.

Examples:

- `can-rtt.c`, `rtcanrecv.c`, and `rtcansend.c`.

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

- `include/rtdm/uapi/can.h`

7.41 sockaddr_ipc Struct Reference

Socket address structure for the RTIPC address family.

Data Fields

- `sa_family_t sipc_family`  
  RTIPC address family, must be `AF_RTIPC`.
- `rtipc_port_t sipc_port`  
  Port number.
7.41.1 Detailed Description

Socket address structure for the RTIPC address family.

Examples:

   *bufp-label.c, bufp-readwrite.c, iddp-label.c, iddp-sendrecv.c, xddp-echo.c, xddp-label.c, and xddp-stream.c.*

7.41.2 Field Documentation

7.41.2.1 sipc_port

```
rtipc_port_t sockaddr_ipc::sipc_port
```
Port number.

Examples:

   *bufp-label.c, bufp-readwrite.c, iddp-label.c, iddp-sendrecv.c, xddp-echo.c, xddp-label.c, and xddp-stream.c.*

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

- `include/rtdm/uapi/ipc.h`

7.42 udd_device Struct Reference

Collaboration diagram for udd_device:

[Generated by Doxygen]
Data Structures

- struct udd_reserved
  
  Reserved to the UDD core.

Data Fields

- const char * device_name
  
  Name of the device managed by the mini-driver, appears automatically in the /dev/rtdm namespace upon creation.

- int device_flags
  
  Additional device flags (e.g.

- int device_subclass
  
  Subclass code of the device managed by the mini-driver (see RTDM_SUBCLASS_XXX definition in the Device Profiles).

- int irq
  
  IRQ number.

- struct udd_memregion mem_regions [UDD_NR_MAPS]
  
  Array of memory regions defined by the device.

- int(* open )(struct rtdm_fd *fd, int oflags)
  
  Ancillary open() handler, optional.

- void(* close )(struct rtdm_fd *fd)
  
  Ancillary close() handler, optional.

- int(* ioctl )(struct rtdm_fd *fd, unsigned int request, void *arg)
  
  Ancillary ioctl() handler, optional.

- int(* mmap )(struct rtdm_fd *fd, struct vm_area_struct *vma)
  
  Ancillary mmap() handler for the mapper device, optional.

- int(* interrupt )(struct udd_device *udd)

7.42.1 Detailed Description

UDD device descriptor.

This descriptor defines the characteristics of a UDD-based mini-driver when registering via a call to udd_register_device().

7.42.2 Field Documentation

7.42.2.1 close

void(* udd_device::close) (struct rtdm_fd *fd)

Ancillary close() handler, optional.

See rtdm_close_handler().

Note

This handler is called from secondary mode only.
7.42.2.2 device_flags

```
int udd_device::device_flags
```

Additional device flags (e.g. RTDM_EXCLUSIVE) RTDM_NAMED_DEVICE may be omitted.

Referenced by udd_register_device().

7.42.2.3 device_subclass

```
int udd_device::device_subclass
```

Subclass code of the device managed by the mini-driver (see RTDM_SUBCLASS_xxx definition in the Device Profiles).

The main class code is pre-set to RTDM_CLASS_UDD.

7.42.2.4 interrupt

```
int(* udd_device::interrupt) (struct udd_device *udd)
```

Ancillary handler for receiving interrupts. This handler must be provided if the mini-driver hands over IRQ handling to the UDD core, by setting the irq field to a valid value, different from UDD_IRQ_CUSTOM and UDD_IRQ_NONE.

The ->interrupt() handler shall return one of the following status codes:

- RTDM_IRQ_HANDLED, if the mini-driver successfully handled the IRQ. This flag can be combined with RTDM_IRQ_DISABLE to prevent the Cobalt kernel from re-enabling the interrupt line upon return, otherwise it is re-enabled automatically.
- RTDM_IRQ_NONE, if the interrupt does not match any IRQ the mini-driver can handle.

Once the ->interrupt() handler has returned, the UDD core notifies user-space Cobalt threads waiting for IRQ events (if any).

Note

This handler is called from primary mode only.

Referenced by udd_register_device().
7.42.2.5 ioctl

\begin{verbatim}
int(* udd_device::ioctl) (struct rtdm_fd *fd, unsigned int request, void *arg)
\end{verbatim}

Ancillary ioctl() handler, optional.

See rtdm_ioctl_handler().

If this routine returns -ENOSYS, the default action implemented by the UDD core for the corresponding request will be applied, as if no ioctl handler had been defined.

Note

This handler is called from primary mode only.

7.42.2.6 irq

int udd_device::irq

IRQ number.

If valid, the UDD core manages the corresponding interrupt line, installing a base handler. Otherwise, a special value can be passed for declaring unmanaged IRQs.

Referenced by udd_disable_irq(), udd_enable_irq(), udd_register_device(), and udd_unregister_device().

7.42.2.7 mem_regions

struct udd_memregion udd_device::mem_regions[UDD_NR_MAPS]

Array of memory regions defined by the device.

The array can be sparse, with some entries bearing the UDD_MEM_NONE type interleaved with valid ones. See the discussion about UDD memory regions.

7.42.2.8 mmap

\begin{verbatim}
int(* udd_device::mmap) (struct rtdm_fd *fd, struct vm_area_struct *vma)
\end{verbatim}

Ancillary mmap() handler for the mapper device, optional.

See rtdm_mmap_handler(). The mapper device operates on a valid region defined in the mem_regions[] array. A pointer to the region can be obtained by a call to udd_get_region().

If this handler is NULL, the UDD core establishes the mapping automatically, depending on the memory type defined for the region.

Note

This handler is called from secondary mode only.
Open

int(∗ udd_device::open) (struct rtdm_fd ∗fd, int oflags)

Ancillary open() handler, optional.

See rtdm_open_handler().

Note

This handler is called from secondary mode only.

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

- include/cobalt/kernel/rtdm/udd.h

7.43 udd_memregion Struct Reference

Data Fields

- const char ∗ name
  Name of the region (informational but required)
- unsigned long addr
  Start address of the region.
- size_t len
  Length (in bytes) of the region.
- int type
  Type of the region.

7.43.1 Detailed Description

UDD memory region descriptor.

This descriptor defines the characteristics of a memory region declared to the UDD core by the mini-
driver. All valid regions should be declared in the udd_device.mem_regions[] array, invalid/unassigned
ones should bear the UDD_MEM_NONE type.

The UDD core exposes each region via the mmap(2) interface to the application. To this end, a com-
panion mapper device is created automatically when registering the mini-driver.

The mapper device creates special files in the RTDM namespace for reaching the individual regions,
which the application can open then map to its address space via the mmap(2) system call.

For instance, declaring a region of physical memory at index #2 of the memory region array could be
done as follows:
static struct udd_device udd;
static int foocard_pci_probe(struct pci_dev *dev, const struct pci_device_id *id) {
    udd.device_name = "foocard";
    ...
    udd.mem_regions[2].name = "ADC";
    udd.mem_regions[2].addr = pci_resource_start(dev, 1);
    udd.mem_regions[2].len = pci_resource_len(dev, 1);
    udd.mem_regions[2].type = UDD_MEM_PHYS;
    return udd_register_device(&udd);
}

This will make such region accessible via the mapper device using the following sequence of code, via
the default ->mmap() handler from the UDD core:

int fd, fdm;
void *p;

fd = open("/dev/foocard", O_RDWR);
fdm = open("/dev/foocard.mapper@2", O_RDWR);
p = mmap(NULL, 4096, PROT_READ|PROT_WRITE, 0, fdm, 0);

Note

No mapper device is created unless a valid region has been declared in the udd_device.mem_regions[] array.

7.43.2 Field Documentation

7.43.2.1 addr

unsigned long udd_memregion::addr

Start address of the region.
This may be a physical or virtual address, depending on the memory type.

7.43.2.2 len

size_t udd_memregion::len

Length (in bytes) of the region.
This value must be PAGE_SIZE aligned.

7.43.2.3 type

int udd_memregion::type

Type of the region.
See the discussion about UDD memory types for possible values.
The documentation for this struct was generated from the following file:

- include/cobalt/kernel/rtdm/udd.h
7.44  udd_device::udd_reserved Struct Reference

Reserved to the UDD core.

Collaboration diagram for udd_device::udd_reserved:

![Collaboration Diagram]

7.44.1  Detailed Description

Reserved to the UDD core.

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

- include/cobalt/kernel/rtdm/udd.h

7.45  udd_signotify Struct Reference

UDD event notification descriptor.

Data Fields

- pid_t pid
  
  *PID of the Cobalt thread to notify upon interrupt receipt.*

- int sig

  *Signal number to send to PID for notifying, which must be in the range [SIGRTMIN .

---

Generated by Doxygen
7.45 Detailed Description

UDD event notification descriptor.

This structure shall be used to pass the information required to enable/disable the notification by signal upon interrupt receipt.

If PID is zero or negative, the notification is disabled. Otherwise, the Cobalt thread whose PID is given will receive the Cobalt signal also mentioned, along with the count of interrupts at the time of the receipt stored in siginfo.si_int. A Cobalt thread must explicitly wait for notifications using the sigwaitinfo() or sigtimedwait() services (no asynchronous mode available).

7.45.2 Field Documentation

7.45.2.1 pid

pid_t udd_signotify::pid

PID of the Cobalt thread to notify upon interrupt receipt.

If pid is zero or negative, the notification is disabled.

7.45.2.2 sig

int udd_signotify::sig

Signal number to send to PID for notifying, which must be in the range [SIGRTMIN, SIGRTMAX] inclusive. This value is not considered if pid is zero or negative.

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

- include/rtdm/uapi/udd.h

7.46 xnheap::xnbucket Struct Reference

log2 bucket list

7.46.1 Detailed Description

log2 bucket list

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

- include/cobalt/kernel/heap.h
7.47  xnsched Struct Reference

Scheduling information structure.

Data Fields

- unsigned long status
- unsigned long lflags
- struct xnthread * curr
- int cpu
- cpumask_t resched
- struct xnsched_rt rt
- volatile unsigned inesting
- struct xntimer htimer
- struct xntimer rrbtimer

7.47.1  Detailed Description

Scheduling information structure.

7.47.2  Field Documentation

7.47.2.1  cpu

int xnsched::cpu

Mask of CPUs needing rescheduling.

Referenced by xnthread_set_periodic(), xntimer_grab_hardware(), and xntimer_release_hardware().

7.47.2.2  curr

struct xnthread* xnsched::curr

Owner CPU id.

7.47.2.3  htimer

struct xntimer xnsched::htimer

Round-robin timer.
7.47.2.4 inesting

volatile unsigned xnsched::inesting

Host timer.

7.47.2.5 lflags

unsigned long xnsched::lflags

Current thread.

7.47.2.6 resched

cpumask_t xnsched::resched

Context of built-in real-time class.

7.47.2.7 rrbtimer

struct xntimer xnsched::rrbtimer

Root thread control block.

7.47.2.8 rt

struct xnsched_rt xnsched::rt

Interrupt nesting level.

7.47.2.9 status

unsigned long xnsched::status

< Scheduler specific status bitmask. Scheduler specific local flags bitmask.

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

- include/cobalt/kernel/sched.h

7.48 xnvfile_lock_ops Struct Reference

Vfile locking operations.

Generated by Doxygen
Data Fields

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

7.48.1 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.

7.48.2 Field Documentation

7.48.2.1 get

int 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.

7.48.2.2 put

void 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:
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.

7.49.1 Detailed Description

Regular vfile iterator.

This structure defines an iterator over a regular vfile.

7.49.2 Field Documentation

7.49.2.1 pos

loff_t xnvfile_regular_iterator::pos

Current record position while iterating.

7.49.2.2 private

char xnvfile_regular_iterator::private[0]

Start of private area.

Use xnvfile_iterator_priv() to address it.
7.49.2.3  seq

struct seq_file* xnvfile_regular_iterator::seq

Backlink to the host sequential file supporting the vfile.

7.49.2.4  vfile

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/kernel/vfile.h

7.50  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)

7.50.1  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.

7.50.2  Field Documentation

7.50.2.1  begin

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.

7.50.2.2 end

```c
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.

7.50.2.3 next

```c
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.

7.50.2.4 rewind

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.
7.50.2.5 show

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

<table>
<thead>
<tr>
<th>it</th>
<th>A pointer to the current vfile iterator.</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>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 handler; the show handler should test this special value to output any header that fits, prior to receiving more calls with actual records.</td>
</tr>
</tbody>
</table>

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.

7.50.2.6 store

`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/kernel/vfile.h
7.51 xnvfile_rev_tag Struct Reference

Snapshot revision tag.

Data Fields

- int rev
  
  Current revision number.

7.51.1 Detailed Description

Snapshot revision tag.

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

7.51.2 Field Documentation

7.51.2.1 rev

int xnvfile_rev_tag::rev

Current revision number.

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

- include/cobalt/kernel/vfile.h

7.52 xnvfile_snapshot Struct Reference

Snapshot vfile descriptor.

Collaboration diagram for xnvfile_snapshot:
7.52.1 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/kernel/vfile.h

7.53 xnvfile_snapshot_iterator Struct Reference

Snapshot-driven vfile iterator.

Collaboration diagram for xnvfile_snapshot_iterator:

![Collaboration diagram](image)

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.*
7.53.1 Detailed Description

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

7.53.2 Field Documentation

7.53.2.1 databuf

caddr_t xnvfile_snapshot_iterator::databuf

Address of record buffer.

7.53.2.2 endfn

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

Buffer release handler.

7.53.2.3 nrdata

int xnvfile_snapshot_iterator::nrdata

Number of collected records.

7.53.2.4 private

char xnvfile_snapshot_iterator::private[0]

Start of private area.
Use xnvfile_iterator_priv() to address it.

7.53.2.5 seq

struct seq_file* xnvfile_snapshot_iterator::seq

Backlink to the host sequential file supporting the vfile.
7.53.2.6  vfile

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/kernel/vfile.h

7.54  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)

7.54.1 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.

7.54.2 Field Documentation

7.54.2.1  begin

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().

Generated by Doxygen
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.

7.54.2.2  end

```
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.

7.54.2.3  next

```
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

<table>
<thead>
<tr>
<th>it</th>
<th>A pointer to the current snapshot iterator.</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>A pointer to the record to fill in.</td>
</tr>
</tbody>
</table>

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.

7.54.2.4 rewind

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

<table>
<thead>
<tr>
<th>it</th>
<th>A pointer to the current snapshot iterator. Two useful information can be retrieved from this iterator in this context:</th>
</tr>
</thead>
</table>

- 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.

7.54.2.5 show

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

<table>
<thead>
<tr>
<th>it</th>
<th>A pointer to the current snapshot iterator.</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>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.</td>
</tr>
</tbody>
</table>

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.

7.54.2.6 store

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.

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

- include/cobalt/kernel/vfile.h
Chapter 8

File Documentation

8.1 include/cobalt/kernel/rtdm/analogy/channel_range.h File Reference

Analogy for Linux, channel, range related features.

Include dependency graph for channel_range.h:
This graph shows which files directly or indirectly include this file:

![Diagram showing file inclusions](image)

Data Structures

- **struct a4l_channel**
  
  *Structure describing some channel’s characteristics.*

- **struct a4l_channels_desc**
  
  *Structure describing a channels set.*

- **struct a4l_range**
  
  *Structure describing a (unique) range.*

Macros

- **#define A4L_CHAN_GLOBAL 0x10**
  
  *Internal use flag (must not be used by driver developer)*

- **#define A4L_RNG_GLOBAL 0x8**
  
  *Internal use flag (must not be used by driver developer)*
**#define RANGE(x, y)**  
Macro to declare a (unique) range with no unit defined.

**#define RANGE_V(x, y)**  
Macro to declare a (unique) range in Volt.

**#define RANGE_mA(x, y)**  
Macro to declare a (unique) range in milliAmpere.

**#define RANGE_ext(x, y)**  
Macro to declare a (unique) range in some external reference.

**#define A4L_RNG_GLOBAL_RNGDESC 0**  
Constant to define a ranges descriptor as global (inter-channel)

**#define A4L_RNG_PERCHAN_RNGDESC 1**  
Constant to define a ranges descriptor as specific for a channel.

**#define RNG_GLOBAL(x)**  
Macro to declare a ranges global descriptor in one line.

Channel reference

*Flags to define the channel's reference*

**#define A4L_CHAN_AREF_GROUND 0x1**  
Ground reference.

**#define A4L_CHAN_AREF_COMMON 0x2**  
Common reference.

**#define A4L_CHAN_AREF_DIFF 0x4**  
Differential reference.

**#define A4L_CHAN_AREF_OTHER 0x8**  
Misc reference.

Channels declaration mode

*Constant to define whether the channels in a descriptor are identical*

**#define A4L_CHAN_GLOBAL_CHANDESC 0**  
Global declaration, the set contains channels with similar characteristics.

**#define A4L_CHAN_PERCHAN_CHANDESC 1**  
Per channel declaration, the descriptor gathers different channels.

---

### 8.1.1 Detailed Description

Analogy for Linux, channel, range related features.

Copyright (C) 1997-2000 David A. Schleef ds@schleef.org Copyright (C) 2008 Alexis Berlemont alexis.berlemont@free.fr

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.
8.2 include/cobalt/kernel/rtdm/analogy/driver.h File Reference

Analogy for Linux, driver facilities.

Include dependency graph for driver.h:

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

Data Structures

- **struct a4l_driver**

  *Structure containing driver declaration data.*

8.2.1 Detailed Description

Analogy for Linux, driver facilities.
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.

**8.3 include/cobalt/kernel/rtdm/driver.h File Reference**

Real-Time Driver Model for Xenomai, driver API header.

Include dependency graph for driver.h:

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

---

**Data Structures**

- struct rtdm_dev_context
  - Device context.
- struct rtdm_sm_ops
  - RTDM state management handlers.
- struct rtdm_driver
  - RTDM driver.
- struct rtdm_device
  - RTDM device.
Macros

- `#define RTDM_MAX_MINOR 1024`
  
  Maximum number of named devices per driver.

- `#define cobalt_atomic_enter(__context)`
  
  Enter atomic section (dual kernel only)

- `#define cobalt_atomic_leave(__context)`
  
  Leave atomic section (dual kernel only)

- `#define RTDM_EXECUTE_ATOMICALLY(code_block)`
  
  Execute code block atomically (DEPRECATED)

- `#define RTDM_LOCK_UNLOCKED(__name) IPIPE_SPIN_LOCK_UNLOCKED`
  
  Static lock initialisation.

- `#define rtdm_lock_get_irqsave(__lock, __context) ((__context) = __rtdm_lock_get_irqsave(__lock))`
  
  Acquire lock and disable preemption, by stalling the head domain.

- `#define rtdm_lock_irqsave(__context) splhigh(__context)`
  
  Disable preemption locally.

- `#define rtdm_lock_irqrestore(__context) splexit(__context)`
  
  Restore preemption state.

- `#define rtdm_irq_get_arg(irq_handle, type) ((type *)irq_handle->cookie)`
  
  Retrieve IRQ handler argument.

Device Flags

Static flags describing a RTDM device

- `#define RTDM_EXCLUSIVE 0x0001`
  
  If set, only a single instance of the device can be requested by an application.

- `#define RTDM_FIXED_MINOR 0x0002`
  
  Use fixed minor provided in the rtdm_device description for registering.

- `#define RTDM_NAMED_DEVICE 0x0010`
  
  If set, the device is addressed via a clear-text name.

- `#define RTDM_PROTOCOL_DEVICE 0x0020`
  
  If set, the device is addressed via a combination of protocol ID and socket type.

- `#define RTDM_DEVICE_TYPE_MASK 0x00F0`
  
  Mask selecting the device type.

- `#define RTDM_SECURE_DEVICE 0x80000000`
  
  Flag indicating a secure variant of RTDM (not supported here)

RTDM_IRQTYPE_xxx

Interrupt registrations flags

- `#define RTDM_IRQTYPE_SHARED XN_IRQTYPE_SHARED`
  
  Enable IRQ-sharing with other real-time drivers.

- `#define RTDM_IRQTYPE_EDGE XN_IRQTYPE_EDGE`
  
  Mark IRQ as edge-triggered, relevant for correct handling of shared edge-triggered IRQs.

RTDM_IRQ_xxx

Return flags of interrupt handlers

- `#define RTDM_IRQ_NONE XN_IRQ_NONE`
  
  Unhandled interrupt.

- `#define RTDM_IRQ_HANDLED XN_IRQ_HANDLED`
  
  Denote handled interrupt.

- `#define RTDM_IRQ_DISABLE XN_IRQ_DISABLE`
  
  Denote disconnected interrupt.
Request interrupt disabling on exit.

Task Priority Range
Maximum and minimum task priorities

- #define RTDM_TASK_LOWEST_PRIORITY 0
- #define RTDM_TASK_HIGHEST_PRIORITY 99

Task Priority Modification
Raise or lower task priorities by one level

- #define RTDM_TASK_RAISE_PRIORITY (+1)
- #define RTDM_TASK_LOWER_PRIORITY (-1)

Typedefs

- typedef ipipe_spinlock_t rtdm_lock_t
  Lock variable.
- typedef unsigned long rtdm_lockctx_t
  Variable to save the context while holding a lock.
- typedef int(* rtdm_irq_handler_t) (rtdm_irq_t *irq_handle)
  Interrupt handler.
- typedef void(* rtdm_nrtsig_handler_t) (rtdm_nrtsig_t *nrt_sig, void *arg)
  Non-real-time signal handler.
- typedef void(* rtdm_timer_handler_t) (rtdm_timer_t *timer)
  Timer handler.
- typedef void(* rtdm_task_proc_t) (void *arg)
  Real-time task procedure.

Enumerations

RTDM_SELECTTYPE_xxx
Event types select can bind to

- enum rtdm_selecttype { RTDM_SELECTTYPE_READ = XNSELECT_READ, RTDM_SELECTTYPE_WRITE = XNSELECT_WRITE, RTDM_SELECTTYPE_EXCEPT = XNSELECT_EXCEPT }

RTDM_TIMERMODE_xxx
Timer operation modes

- enum rtdm_timer_mode { RTDM_TIMERMODE_RELATIVE = XN_RELATIVE, RTDM_TIMERMODE_ABSOLUTE = XN_ABSOLUTE, RTDM_TIMERMODE_REALTIME = XN_REALTIME }
Functions

- static void * rtdm_fd_to_private (struct rtdm_fd *fd)
  Locate the driver private area associated to a device context structure.
- static struct rtdm_fd * rtdm_private_to_fd (void *dev_private)
  Locate a device file descriptor structure from its driver private area.
- static bool rtdm_fd_is_user (struct rtdm_fd *fd)
  Tell whether the passed file descriptor belongs to an application.
- static struct rtdm_device * rtdm_fd_device (struct rtdm_fd *fd)
  Locate a device structure from a file descriptor.
- int rtdm_dev_register (struct rtdm_device *device)
  Register a RTDM device.
- void rtdm_dev_unregister (struct rtdm_device *device)
  Unregister a RTDM device.
- void rtdm_toseq_init (rtdm_toseq_t *timeout_seq, nanosecs_rel_t timeout)
  Initialise a timeout sequence.
- static void rtdm_lock_init (rtdm_lock_t *lock)
  Dynamic lock initialisation.
- static void rtdm_lock_get (rtdm_lock_t *lock)
  Acquire lock from non-preemptible contexts.
- static void rtdm_lock_put (rtdm_lock_t *lock)
  Release lock without preemption restoration.
- static void rtdm_lock_put_irqrestore (rtdm_lock_t *lock, rtdm_lockctx_t context)
  Release lock and restore preemption state.
- int rtdm_irq_request (rtdm_irq_t *irq_handle, unsigned int irq_no, rtdm_irq_handler_t handler, unsigned long flags, const char *device_name, void *arg)
  Register an interrupt handler.
- void rtdm_schedule_nrt_work (struct work_struct *lostage_work)
  Put a work task in Linux non real-time global workqueue from primary mode.
- int rtdm_timer_destroy (rtdm_timer_t *timer)
  Destroy a timer.
- int rtdm_timer_start (rtdm_timer_t *timer, nanosecs_abs_t expiry, nanosecs_rel_t interval, enum rtdm_timer_mode mode)
  Start a timer.
- void rtdm_timer_stop (rtdm_timer_t *timer)
  Stop a timer.
- int rtdm_task_init (rtdm_task_t *task, const char *name, rtdm_task_proc_t task_proc, void *arg, int priority, nanosecs_rel_t period)
  Initialise and start a real-time task.
- void rtdm_task_busy_sleep (nanosecs_rel_t delay)
  Busy-wait a specified amount of time.
- int rtdm_event_init (rtdm_event_t *event, unsigned long pending)
  Initialise an event.
- void rtdm_event_wait (rtdm_event_t *event)
  Wait on event occurrence.
- int rtdm_event_timedwait (rtdm_event_t *event, nanosecs_rel_t timeout, rtdm_toseq_t *timeout_seq)
  Wait on event occurrence with timeout.
- void rtdm_event_signal (rtdm_event_t *event)
  Signal an event occurrence.
- void rtdm_event_clear (rtdm_event_t *event)
Clear event state.

- `void rtdm_event_pulse (rtdm_event_t *event)`
  Signal an event occurrence to currently listening waiters.
- `void rtdm_event_destroy (rtdm_event_t *event)`
  Destroy an event.
- `void rtdm_sem_init (rtdm_sem_t *sem, unsigned long value)`
  Initialise a semaphore.
- `int rtdm_sem_down (rtdm_sem_t *sem)`
  Decrement a semaphore.
- `int rtdm_sem_timeddown (rtdm_sem_t *sem, nanosecs_rel_t timeout, rtdm_toseq_t *timeout_seq)`
  Decrement a semaphore with timeout.
- `void rtdm_sem_up (rtdm_sem_t *sem)`
  Increment a semaphore.
- `void rtdm_sem_destroy (rtdm_sem_t *sem)`
  Destroy a semaphore.
- `void rtdm_mutex_init (rtdm_mutex_t *mutex)`
  Initialise a mutex.
- `int rtdm_mutex_lock (rtdm_mutex_t *mutex)`
  Request a mutex.
- `int rtdm_mutex_timedlock (rtdm_mutex_t *mutex, nanosecs_rel_t timeout, rtdm_toseq_t *timeout_seq)`
  Request a mutex with timeout.
- `void rtdm_mutex_unlock (rtdm_mutex_t *mutex)`
  Release a mutex.
- `void rtdm_mutex_destroy (rtdm_mutex_t *mutex)`
  Destroy a mutex.
- `int rtdm_ratelimit (struct rtdm_ratelimit_state *rs, const char *func)`
  Enforces a rate limit.

RTDM profile information descriptor

RTDM profile information

This descriptor details the profile information associated to a RTDM class of device managed by a driver.

- `#define RTDM_CLASS_MAGIC 0x8284636c`
  Initializer for class profile information.
- `#define RTDM_PROFILE_INFO(__name, __id, __subid, __version)`
  Initializer for class profile information.
- `int rtdm_drv_set_sysclass (struct rtdm_driver *drv, struct class *cls)`
  Set the kernel device class of a RTDM driver.
8.3.1 Detailed Description

Real-Time Driver Model for Xenomai, driver API header.

Copyright (C) 2005-2007 Jan Kiszka jan.kiszka@web.de Copyright (C) 2005 Joerg Langenberg joerg.langenberg@gmx.net Copyright (C) 2008 Gilles Chanteperdrix gilles.chanteperdrix@xenomai.org.org Copyright (C) 2014 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.

8.3.2 Macro Definition Documentation

8.3.2.1 RTDM_CLASS_MAGIC

#define RTDM_CLASS_MAGIC 0x8284636c

Initializer for class profile information.

This macro must be used to fill in the class profile information field from a RTDM driver.

Parameters

<table>
<thead>
<tr>
<th>__name</th>
<th>Class name (unquoted).</th>
</tr>
</thead>
<tbody>
<tr>
<td>__id</td>
<td>Class major identification number (profile_version.class_id).</td>
</tr>
<tr>
<td>__subid</td>
<td>Class minor identification number (profile_versionsubclass_id).</td>
</tr>
<tr>
<td>__version</td>
<td>Profile version number.</td>
</tr>
</tbody>
</table>

Note

See Device Profiles.

8.3.2.2 RTDM_PROFILE_INFO

#define RTDM_PROFILE_INFO(
    __name,
__id,
__subid,
__version )

Value:

{
    .name = ( # __name ),
    .class_id = (__id),
    .subclass_id = (__subid),
    .version = (__version),
    .magic = ~RTDM_CLASS_MAGIC,
    .owner = THIS_MODULE,
    .kdev_class = NULL,
}

Initializer for class profile information.

This macro must be used to fill in the class profile information field from a RTDM driver.

Parameters

| __name | Class name (unquoted). |
| __id   | Class major identification number (profile_version.class_id). |
| __subid | Class minor identification number (profile_version.subclass_id). |
| __version | Profile version number. |

Note

See Device Profiles.

8.3.3 Function Documentation

8.3.3.1 rtdm_fd_device()

static struct rtdm_device* rtdm_fd_device (  
    struct rtdm_fd * fd )  [static]

Locate a device structure from a file descriptor.

Parameters

| in  | fd | File descriptor |

Returns

The address of the device structure to which this file descriptor is attached.

Referenced by udd_get_device().

Generated by Doxygen
8.3.3.2 rtdm_fd_is_user()

static bool rtdm_fd_is_user ( 
    struct rtdm_fd * fd ) [inline], [static]

Tell whether the passed file descriptor belongs to an application.

Parameters

**in** | **fd** | File descriptor

Returns

true if passed file descriptor belongs to an application, false otherwise.

8.3.3.3 rtdm_fd_to_private()

static void* rtdm_fd_to_private ( 
    struct rtdm_fd * fd ) [inline], [static]

Locate the driver private area associated to a device context structure.

Parameters

**in** | **fd** | File descriptor structure associated with opened device instance

Returns

The address of the private driver area associated to file descriptor.

8.3.3.4 rtdm_private_to_fd()

static struct rtdm_fd* rtdm_private_to_fd ( 
    void * dev_private ) [static]

Locate a device file descriptor structure from its driver private area.

Parameters

**in** | **dev_private** | Address of a private context area
Returns

The address of the file descriptor structure defining dev_private.

8.4 include/cobalt/kernel/rtdm/analogy/subdevice.h File Reference

Analogy for Linux, subdevice related features.

Include dependency graph for subdevice.h:
Data Structures

- struct `a4l_subdevice`

  Structure describing the subdevice.

8.4.1 Detailed Description

Analogy for Linux, subdevice related features.

Copyright (C) 1997-2000 David A. Schleef ds@schleef.org Copyright (C) 2008 Alexis Berlemont alexis.berlemont@free.fr

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.

8.5 `include/rtdm/can.h` File Reference

Include dependency graph for can.h:
This graph shows which files directly or indirectly include this file:

8.5.1 Detailed Description

Note

Copyright (C) 2006 Wolfgang Grandegger wg@grandegger.com
Copyright (C) 2005, 2006 Sebastian Smolorz Sebastian.Smolorz@stud.uni-hannover.de

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.

8.6 include/rtdm/uapi/can.h File Reference

Real-Time Driver Model for RT-Socket-CAN, CAN device profile header.
This graph shows which files directly or indirectly include this file:

```plaintext
include/rtdm/uapi/can.h
include/cobalt/kernel/rtdm/can.h
include/rtdm/can.h
include/trank/rtdm/rtcan.h
kernel/drivers/can/rtcan_socket.h
kernel/drivers/can/rtcan_list.h
```

**Data Structures**

- **struct can_bittime_std**
  
  Standard bit-time parameters according to Bosch.

- **struct can_bittime_btr**
  
  Hardware-specific BTR bit-times.

- **struct can_bittime**
  
  Custom CAN bit-time definition.

- **struct can_filter**
  
  Filter for reception of CAN messages.

- **struct sockaddr_can**
  
  Socket address structure for the CAN address family.

- **struct can_frame**
  
  Raw CAN frame.

- **struct can_ifreq**
  
  CAN interface request descriptor.

**Macros**

- **#define AF_CAN 29**
  
  CAN address family.

- **#define PF_CAN AF_CAN**
  
  CAN protocol family.

- **#define SOL_CAN_RAW 103**
  
  Generated by Doxygen
CAN socket levels.

CAN ID masks
Bit masks for masking CAN IDs

- #define CAN_EFF_MASK 0x1FFFFFFF
  Bit mask for extended CAN IDs.
- #define CAN_SFF_MASK 0x000007FF
  Bit mask for standard CAN IDs.

CAN ID flags
Flags within a CAN ID indicating special CAN frame attributes

- #define CAN_EFF_FLAG 0x80000000
  Extended frame.
- #define CAN_RTR_FLAG 0x40000000
  Remote transmission frame.
- #define CAN_ERR_FLAG 0x20000000
  Error frame (see Errors), not valid in struct can_filter.
- #define CAN_INV_FILTER CAN_ERR_FLAG
  Invert CAN filter definition, only valid in struct can_filter.

Particular CAN protocols
Possible protocols for the PF_CAN protocol family
Currently only the RAW protocol is supported.

- #define CAN_RAW 1
  Raw protocol of PF_CAN, applicable to socket type SOCK_RAW.

CAN controller modes
Special CAN controllers modes, which can be or’ed together.

Note
These modes are hardware-dependent. Please consult the hardware manual of the CAN controller for more detailed information.

- #define CAN_CTRLMODE_LISTENONLY 0x1
- #define CAN_CTRLMODE_LOOPBACK 0x2
- #define CAN_CTRLMODE_3_SAMPLES 0x4

Timestamp switches
Arguments to pass to RTCAN_RTIOC_TAKE_TIMESTAMP

- #define RTCAN_TAKE_NO_TIMESTAMPS 0
  Switch off taking timestamps.
- #define RTCAN_TAKE_TIMESTAMPS 1
  Do take timestamps.

RAW socket options
Setting and getting CAN RAW socket options.

- #define CAN_RAW_FILTER 0x1
  CAN filter definition.
- #define CAN_RAW_ERR_FILTER 0x2
CAN error mask.

- `#define CAN_RAW_LOOPBACK 0x3`
  CAN TX loopback.
- `#define CAN_RAW_RECV_OWN_MSGS 0x4`
  CAN receive own messages.

IOCTLs

CAN device IOCTLs

**Deprecated**  Passing `struct ifreq` as a request descriptor for CAN IOCTLs is still accepted for backward compatibility, however it is recommended to switch to `struct can_ifreq` at the first opportunity.

- `#define SIOCGIFINDEX defined_by_kernel_header_file`
  Get CAN interface index by name.
- `#define SIOCSCANBAUDRATE _IOW(RTIOC_TYPE_CAN, 0x01, struct can_ifreq)`
  Set baud rate.
- `#define SIOCGCANBAUDRATE _IOWR(RTIOC_TYPE_CAN, 0x02, struct can_ifreq)`
  Get baud rate.
- `#define SIOCSANCELTIMEOUT _IOW(RTIOC_TYPE_CAN, 0x03, struct can_ifreq)`
  Set custom bit time parameters.
- `#define SIOCGCANCELLTIMEOUT _IOWR(RTIOC_TYPE_CAN, 0x04, struct can_ifreq)`
  Get custom bit-time parameters.
- `#define SIOCSCANMODE _IW(RTIOC_TYPE_CAN, 0x05, struct can_ifreq)`
  Set operation mode of CAN controller.
- `#define SIOCGCANMACE _IW(RTIOC_TYPE_CAN, 0x06, struct can_ifreq)`
  Get current state of CAN controller.
- `#define SIOCSCANCTRLMODE _IOW(RTIOC_TYPE_CAN, 0x07, struct can_ifreq)`
  Set special controller modes.
- `#define SIOCGCANCTRLMODE _IOWR(RTIOC_TYPE_CAN, 0x08, struct can_ifreq)`
  Get special controller modes.
- `#define RTCAN_RTIOC_TAKE_TIMESTAMP _IOW(RTIOC_TYPE_CAN, 0x09, int)`
  Enable or disable storing a high precision timestamp upon reception of a CAN frame.
- `#define RTCAN_RTIOC_RCV_TIMEOUT _IOW(RTIOC_TYPE_CAN, 0x0A, nanosecs_rel_t)`
  Specify a reception timeout for a socket.
- `#define RTCAN_RTIOC_SND_TIMEOUT _IOW(RTIOC_TYPE_CAN, 0x0B, nanosecs_rel_t)`
  Specify a transmission timeout for a socket.

**Error mask**

Error class (mask) in `can_id` field of `struct can_frame` to be used with `CAN_RAW_ERR_FILTER`.

**Note:** Error reporting is hardware dependent and most CAN controllers report less detailed error conditions than the SJA1000.

**Note:** In case of a bus-off error condition (`CAN_ERR_BUSOFF`), the CAN controller is not restarted automatically. It is the application's responsibility to react appropriately, e.g. calling `CAN_MODE←START`.

**Note:** Bus error interrupts (`CAN_ERR_BUSERROR`) are enabled when an application is calling a `Recv` function on a socket listening on bus errors (using `CAN_RAW_ERR_FILTER`). After one bus error has occurred, the interrupt will be disabled to allow the application time for error processing and to efficiently avoid bus error interrupt flooding.

- `#define CAN_ERR_TX_TIMEOUT 0x00000001U`
  TX timeout (netdevice driver)
- `#define CAN_ERR_LOSTARB 0x00000002U`
  Lost arbitration (see `data[0]`)
- `#define CAN_ERR_CRTL 0x00000004U`
  Controller problems (see `data[1]`)

Generated by Doxygen
• #define CAN_ERR_PROT 0x00000008U
  Protocol violations (see data[2], data[3])
• #define CAN_ERR_TRX 0x00000010U
  Transceiver status (see data[4])
• #define CAN_ERR_ACK 0x00000020U
  Received no ACK on transmission.
• #define CAN_ERR_BUSOFF 0x00000040U
  Bus off.
• #define CAN_ERR_BUSERRO 0x00000080U
  Bus error (may flood)
• #define CAN_ERR_RESTARTED 0x00000100U
  Controller restarted.
• #define CAN_ERR_MASK 0x1FFFFFFFU
  Omit EFF, RTR, ERR flags.

Arbitration lost error
Error in the data[0] field of struct can_frame.
• #define CAN_ERR_LOSTARB_UNSPEC 0x00
  unspecified

Controller problems
Error in the data[1] field of struct can_frame.
• #define CAN_ERR_CRTL_UNSPEC 0x00
  unspecified
• #define CAN_ERR_CRTL_RX_OVERFLOW 0x01
  RX buffer overflow.
• #define CAN_ERR_CRTL_TX_OVERFLOW 0x02
  TX buffer overflow.
• #define CAN_ERR_CRTL_RX_WARNING 0x04
  reached warning level for RX errors
• #define CAN_ERR_CRTL_TX_WARNING 0x08
  reached warning level for TX errors
• #define CAN_ERR_CRTL_RX_PASSIVE 0x10
  reached passive level for RX errors
• #define CAN_ERR_CRTL_TX_PASSIVE 0x20
  reached passive level for TX errors

Protocol error type
Error in the data[2] field of struct can_frame.
• #define CAN_ERR_PROT_UNSPEC 0x00
  unspecified
• #define CAN_ERR_PROT_BIT 0x01
  single bit error
• #define CAN_ERR_PROT_FORM 0x02
  frame format error
• #define CAN_ERR_PROT_STUFF 0x04
  bit stuffing error
• #define CAN_ERR_PROT_BIT0 0x08
  unable to send dominant bit
• #define CAN_ERR_PROT_BIT1 0x10
  unable to send recessive bit
• #define CAN_ERR_PROT_OVERLOAD 0x20
  bus overload
• #define CAN_ERR_PROT_ACTIVE 0x40

Generated by Doxygen
active error announcement

- `#define CAN_ERR_PROT_TX 0x80`
  error occurred on transmission

Protocol error location
Error in the data[4] field of struct `can_frame`.

- `#define CAN_ERR_PROT_LOC_UNSPEC 0x00`
  unspecified
- `#define CAN_ERR_PROT_LOC_SOF 0x03`
  start of frame
- `#define CAN_ERR_PROT_LOC_ID28_21 0x02`
  ID bits 28 - 21 (SFF: 10 - 3)
- `#define CAN_ERR_PROT_LOC_ID20_18 0x06`
  ID bits 20 - 18 (SFF: 2 - 0)
- `#define CAN_ERR_PROT_LOC_SRTR 0x04`
  substitute RTR (SFF: RTR)
- `#define CAN_ERR_PROT_LOC_IDE 0x05`
  identifier extension
- `#define CAN_ERR_PROT_LOC_ID17_13 0x07`
  ID bits 17-13.
- `#define CAN_ERR_PROT_LOC_ID12_05 0x0F`
  ID bits 12-5.
- `#define CAN_ERR_PROT_LOC_ID04_00 0x0E`
  ID bits 4-0.
- `#define CAN_ERR_PROT_LOC_RTR 0x0C`
  RTR.
- `#define CAN_ERR_PROT_LOC_RES1 0x0D`
  reserved bit 1
- `#define CAN_ERR_PROT_LOC_RES0 0x09`
  reserved bit 0
- `#define CAN_ERR_PROT_LOC_DLC 0x0B`
  data length code
- `#define CAN_ERR_PROT_LOC_DATA 0x0A`
  data section
- `#define CAN_ERR_PROT_LOC_CRC_SEQ 0x08`
  CRC sequence.
- `#define CAN_ERR_PROT_LOC_CRC_DEL 0x18`
  CRC delimiter.
- `#define CAN_ERR_PROT_LOC_ACK 0x19`
  ACK slot.
- `#define CAN_ERR_PROT_LOC_ACK_DEL 0x1B`
  ACK delimiter.
- `#define CAN_ERR_PROT_LOC_EOF 0x1A`
  end of frame
- `#define CAN_ERR_PROT_LOC_INTERM 0x12`
  intermission
- `#define CAN_ERR_TRX_UNSPEC 0x00`
  0000 0000
- `#define CAN_ERR_TRX_CANH_NO_WIRE 0x04`
  0000 0100
- `#define CAN_ERR_TRX_CANH_SHORT_TO_BAT 0x05`
  0000 0101
- `#define CAN_ERR_TRX_CANH_SHORT_TO_VCC 0x06`
  0000 0110
- `#define CAN_ERR_TRX_CANH_SHORT_TO_GND 0x07`
  0000 0111
- `#define CAN_ERR_TRX_CANL_NO_WIRE 0x40`
8.6 include/rtdm/uapi/can.h File Reference

- #define CAN_ERR_TRX_CANL_SHORT_TO_BAT 0x50
- #define CAN_ERR_TRX_CANL_SHORT_TO_VCC 0x60
- #define CAN_ERR_TRX_CANL_SHORT_TO_GND 0x70
- #define CAN_ERR_TRX_CANL_SHORT_TO_CANH 0x80

Typedefs
- typedef uint32_t can_id_t
  Type of CAN id (see CAN_xxx_MASK and CAN_xxx_FLAG)
- typedef can_id_t can_err_mask_t
  Type of CAN error mask.
- typedef uint32_t can_baudrate_t
  Baudrate definition in bits per second.
- typedef enum CAN_BITTIME_TYPE can_bittime_type_t
  See CAN_BITTIME_TYPE.
- typedef enum CAN_MODE can_mode_t
  See CAN_MODE.
- typedef int can_ctrlmode_t
  See CAN_CTRLMODE.
- typedef enum CAN_STATE can_state_t
  See CAN_STATE.
- typedef struct can_filter can_filter_t
  Filter for reception of CAN messages.
- typedef struct can_frame can_frame_t
  Raw CAN frame.

Enumerations
- enum CAN_BITTIME_TYPE { CAN_BITTIME_STD, CAN_BITTIME_BTR }
  Supported CAN bit-time types.

CAN operation modes
Modes into which CAN controllers can be set
- enum CAN_MODE { CAN_MODE_STOP = 0, CAN_MODE_START, CAN_MODE_SLEEP }

CAN controller states
States a CAN controller can be in.
- enum CAN_STATE { CAN_STATE_ERROR_ACTIVE = 0, CAN_STATE_ACTIVE = 0, CAN_STATE_ERROR_WA-
  RRNING = 1, CAN_STATE_BUS_WARNING = 1, CAN_STATE_ERROR_PASSIVE = 2, CAN_STATE_BUS_PASSIVE = 2, CAN_STATE_BU-
  S_OFF, CAN_STATE_SCANNING_BAUDRATE, CAN_STATE_STOPPED, CAN_STATE_SLEEPING }

Generated by Doxygen
8.6.1 Detailed Description

Real-Time Driver Model for RT-Socket-CAN, CAN device profile header.

Note

Copyright (C) 2004, 2005, Robert Schwebel, Benedikt Spranger, Marc Kleine-Budde, Pengutronix
Copyright (C) 2005, 2006 Sebastian Smolorz Sebastian.Smolorz@stud.uni-hannover.de

This RTDM CAN device profile header is based on:

include/linux/can.h, include/linux/socket.h, net/can/pf_can.h in linux-can.patch, a CAN socket framework for Linux

Copyright (C) 2004, 2005, Robert Schwebel, Benedikt Spranger, Marc Kleine-Budde, Pengutronix

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

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

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

8.7 include/cobalt/kernel/rtdm/fd.h File Reference

Include dependency graph for fd.h:

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

- struct rtdm_fd_ops
  RTDM file operation descriptor.

Functions

- int rtdm_open_handler (struct rtdm_fd *fd, int oflags)
  Open handler for named devices.
- int rtdm_socket_handler (struct rtdm_fd *fd, int protocol)
  Socket creation handler for protocol devices.
- void rtdm_close_handler (struct rtdm_fd *fd)
  Close handler.
- int rtdm_ioctl_handler (struct rtdm_fd *fd, unsigned int request, void __user *arg)
  IOCTL handler.
- ssize_t rtdm_read_handler (struct rtdm_fd *fd, void __user *buf, size_t size)
  Read handler.
- ssize_t rtdm_write_handler (struct rtdm_fd *fd, const void __user *buf, size_t size)
  Write handler.
- ssize_t rtdm_recvmsg_handler (struct rtdm_fd *fd, struct user_msghdr *msg, int flags)
  Receive message handler.
- ssize_t rtdm_sendmsg_handler (struct rtdm_fd *fd, const struct user_msghdr *msg, int flags)
  Transmit message handler.
- int rtdm_select_handler (struct rtdm_fd *fd, struct xnselector *selector, unsigned int type, unsigned int index)
  Select handler.
- int rtdm_mmap_handler (struct rtdm_fd *fd, struct vm_area_struct *vma)
  Memory mapping handler.
- unsigned long rtdm_get_unmapped_area_handler (struct rtdm_fd *fd, unsigned long len, unsigned long pgoff, unsigned long flags)
  Allocate mapping region in address space.
- struct rtdm_fd * rtdm_fd_get (int ufd, unsigned int magic)
  Retrieve and lock a RTDM file descriptor.
- int rtdm_fd_lock (struct rtdm_fd *fd)
  Hold a reference on a RTDM file descriptor.
- void rtdm_fd_put (struct rtdm_fd *fd)
  Release a RTDM file descriptor obtained via rtdm_fd_get().
- void rtdm_fd_unlock (struct rtdm_fd *fd)
  Drop a reference on a RTDM file descriptor.
- int rtdm_fd_select (int ufd, struct xnselector *selector, unsigned int type)
  Bind a selector to specified event types of a given file descriptor.

8.7.1 Detailed Description

operation handlers

8.7.2 Function Documentation

Generated by Doxygen
8.7.2.1  rtdm_fd_get()

```c
struct rtdm_fd* rtdm_fd_get(
    int ufd,
    unsigned int magic)
```

Retrieve and lock a RTDM file descriptor.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>ufd</th>
<th>User-side file descriptor</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>magic</td>
<td>Magic word for lookup validation</td>
</tr>
</tbody>
</table>

**Returns**
Pointer to the RTDM file descriptor matching `ufd`, or `ERR_PTR(-EBADF)`.

**Note**
The file descriptor returned must be later released by a call to `rtdm_fd_put()`.

**Tags**
unrestricted

Referenced by `rtdm_fd_select()`.

8.7.2.2  rtdm_fd_lock()

```c
int rtdm_fd_lock(
    struct rtdm_fd* fd)
```

Hold a reference on a RTDM file descriptor.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>fd</th>
<th>Target file descriptor</th>
</tr>
</thead>
</table>

**Note**
`rtdm_fd_lock()` increments the reference counter of `fd`. You only need to call this function in special scenarios, e.g. when keeping additional references to the file descriptor that have different lifetimes. Only use `rtdm_fd_lock()` on descriptors that are currently locked via an earlier `rtdm_fd_get()` or while running a device operation handler.

**Tags**
unrestricted
8.7.2.3  rtdm_fd_put()

```c
void rtdm_fd_put ( 
    struct rtdm_fd * fd )
```

Release a RTDM file descriptor obtained via `rtdm_fd_get()`

Parameters

- **in** `fd`  RTDM file descriptor to release

Note

Every call to `rtdm_fd_get()` must be matched by a call to `rtdm_fd_put()`.

Tags

- unrestricted

Referenced by `rtdm_fd_select()`.

8.7.2.4  rtdm_fd_select()

```c
int rtdm_fd_select ( 
    int ufd, 
    struct xnselector * selector, 
    unsigned int type )
```

Bind a selector to specified event types of a given file descriptor.

This function is invoked by higher RTOS layers implementing select-like services. It shall not be called directly by RTDM drivers.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>ufd</th>
<th>User-side file descriptor to bind to</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td></td>
<td></td>
</tr>
<tr>
<td>in, out</td>
<td>selector</td>
<td>Selector object that shall be bound to the given event</td>
</tr>
<tr>
<td>in</td>
<td>type</td>
<td>Event type the caller is interested in</td>
</tr>
</tbody>
</table>

Returns

- 0 on success, otherwise:
  - -EBADF is returned if the file descriptor `ufd` cannot be resolved.
-EINVAL is returned if type is invalid.

Tags

**task-unrestricted**

References rtdm_fd_get(), rtdm_fd_put(), splnone, and spltest.

### 8.7.2.5 rtdm_fd_unlock()

```c
void rtdm_fd_unlock ( 
    struct rtdm_fd ∗ fd )
```

Drop a reference on a RTDM file descriptor.

Parameters

- **in fd** Target file descriptor

Note

Every call to rtdm_fd_lock() must be matched by a call to rtdm_fd_unlock().

Tags

**unrestricted**

### 8.8 include/rtdm/uapi/ipc.h File Reference

This file is part of the Xenomai project.

Include dependency graph for ipc.h:
This graph shows which files directly or indirectly include this file:

```
include/rtdm/uapi/ipc.h
include/cobalt/kernel
/include/rtdm/ipc.h
include/rtdm/ipc.h
include/trank/rtdm
/include/rtipc.h
```

Data Structures

- **struct rtipc_port_label**
  
  Port label information structure.

- **struct sockaddr_ipc**
  
  Socket address structure for the RTIPC address family.

Macros

**XDDP socket options**

Setting and getting XDDP socket options.

- `#define XDDP_LABEL 1`
  
  XDDP label assignment.

- `#define XDDP_POOLSZ 2`
  
  XDDP local pool size configuration.

- `#define XDDP_BUFSZ 3`
  
  XDDP streaming buffer size configuration.

- `#define XDDP_MONITOR 4`
  
  XDDP monitoring callback.

**XDDP events**

Specific events occurring on XDDP channels, which can be monitored via the XDDP_MONITOR socket option.

- `#define XDDP_EVTIN 1`
  
  Monitor writes to the non real-time endpoint.

- `#define XDDP_EVTOUT 2`
  
  Monitor reads from the non real-time endpoint.

- `#define XDDP_EVTDOWN 3`
  
  Monitor close from the non real-time endpoint.
• \#define XDDP_EVTNOBUF 4
  Monitor memory shortage for non real-time datagrams.

**IDDP socket options**

Setting and getting IDDP socket options.

• \#define IDDP_LABEL 1
  IDDP label assignment.
• \#define IDDP_POOLSZ 2
  IDDP local pool size configuration.

**BUFP socket options**

Setting and getting BUFP socket options.

• \#define BUFP_LABEL 1
  BUFP label assignment.
• \#define BUFP_BUFSZ 2
  BUFP buffer size configuration.

**Socket level options**

Setting and getting supported standard socket level options.

• \#define SO_SNDTIMEO defined_by_kernel_header_file
  IPCPROTO_IDDP and IPProto_BUFP protocols support the standard SO_SNDTIMEO socket option, from the SOL_SOCKET level.
• \#define SO_RCVTIMEO defined_by_kernel_header_file
  All RTIPC protocols support the standard SO_RCVTIMEO socket option, from the SOL_SOCKET level.

**Typedefs**

• typedef int16_t rtipc_port_t
  Port number type for the RTIPC address family.

**Enumerations**

**RTIPC protocol list**

protocols for the PF_RTIPC protocol family

• enum {
  IPCPROTO_IPC = 0, IPCPROTO_XDDP = 1, IPCPROTO_IDDP = 2, IPCPROTO_BUFP = 3
}
Functions

Supported operations

Standard socket operations supported by the RTIPC protocols.

- int socket__AF_RTIPC (int domain=AF_RTIPC, int type=SOCK_DGRAM, int protocol)
  
  Create an endpoint for communication in the AF_RTIPC domain.

- int close__AF_RTIPC (int sockfd)
  
  Close a RTIPC socket descriptor.

- int bind__AF_RTIPC (int sockfd, const struct sockaddr_ipc *addr, socklen_t addrlen)
  
  Bind a RTIPC socket to a port.

- int connect__AF_RTIPC (int sockfd, const struct sockaddr_ipc *addr, socklen_t addrlen)
  
  Initiate a connection on a RTIPC socket.

- int setsockopt__AF_RTIPC (int sockfd, int level, int optname, const void *optval, socklen_t optlen)
  
  Set options on RTIPC sockets.

- int getsockopt__AF_RTIPC (int sockfd, int level, int optname, void *optval, socklen_t *optlen)
  
  Get options on RTIPC sockets.

- ssize_t sendmsg__AF_RTIPC (int sockfd, const struct msghdr *msg, int flags)
  
  Send a message on a RTIPC socket.

- ssize_t recvmsg__AF_RTIPC (int sockfd, struct msghdr *msg, int flags)
  
  Receive a message from a RTIPC socket.

- int getsockname__AF_RTIPC (int sockfd, struct sockaddr_ipc *addr, socklen_t *addrlen)
  
  Get socket name.

- int getpeername__AF_RTIPC (int sockfd, struct sockaddr_ipc *addr, socklen_t *addrlen)
  
  Get socket peer.

8.8.1 Detailed Description

This file is part of the Xenomai project.

Note

Copyright (C) 2009 Philippe Gerum rpm@xenomai.org

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

This library 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 Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.
8.9  include/rtdm/rtdm.h File Reference

Include dependency graph for rtdm.h:

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

8.9.1  Detailed Description

Note

Copyright (C) 2005, 2006 Jan Kiszka jan.kiszka@web.de
Copyright (C) 2005 Joerg Langenberg joerg.langenberg@gmx.net

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

Xenomai is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even
the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

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

8.10  include/rtdm/uapi/rtdm.h File Reference

Real-Time Driver Model for Xenomai, user API header.

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

- struct rtdm_device_info
  
  Device information.

Macros

API Versioning

- #define RTDM_API_VER 9
  
  Common user and driver API version.
- #define RTDM_API_MIN_COMPAT_VER 9
  
  Minimum API revision compatible with the current release.

RTDM_TIMEOUT_xxx

Special timeout values

- #define RTDM_TIMEOUT_INFINITE 0
  
  Block forever.
- #define RTDM_TIMEOUT_NONE (-1)
  
  Any negative timeout means non-blocking.

RTDM_CLASS_xxx

Device classes

- #define RTDM_CLASS_PARPORT 1
- #define RTDM_CLASS_SERIAL 2
- #define RTDM_CLASS_CAN 3
- #define RTDM_CLASS_NETWORK 4
- #define RTDM_CLASS_RTMAC 5
- #define RTDM_CLASS_TESTING 6
- #define RTDM_CLASS_RTIPC 7
- #define RTDM_CLASS_COBALT 8
- #define RTDM_CLASS_UDD 9
- #define RTDM_CLASS_MEMORY 10
- #define RTDM_CLASS_GPIO 11
- #define RTDM_CLASS_SPI 12
- #define RTDM_CLASS_MISC 223
- #define RTDM_CLASS_EXPERIMENTAL 224
- #define RTDM_CLASS_MAX 255

Device Naming

Maximum length of device names (excluding the final null character)

- #define RTDM_MAX_DEVNAME_LEN 31

RTDM_PURGE_xxx_BUFFER

Flags selecting buffers to be purged

- #define RTDM_PURGE_RX_BUFFER 0x0001
- #define RTDM_PURGE_TX_BUFFER 0x0002

Common IOCTLs

The following IOCTLs are common to all device rtdm_profiles.

- #define RTIOC_DEVICE_INFO _IOR(RTIOC_TYPE_COMMON, 0x00, struct rtdm_device_info)
  
  Retrieve information about a device or socket.
- #define RTIOC_PURGE _IOW(RTIOC_TYPE_COMMON, 0x10, int)
  
  Purge internal device or socket buffers.
Typedefs

- typedef uint64_t nanosecs_abs_t
  
  *RTDM type for representing absolute dates.*

- typedef int64_t nanosecs_rel_t
  
  *RTDM type for representing relative intervals.*

- typedef struct rtdm_device_info rtdm_device_info_t
  
  *Device information.*

8.10.1 Detailed Description

Real-Time Driver Model for Xenomai, user API header.

Note

Copyright (C) 2005, 2006 Jan Kiszka jan.kiszka@web.de
Copyright (C) 2005 Joerg Langenberg joerg.langenberg@gmx.net

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

This library 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 Lesser General Public License for more details.

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

8.11 include/rtdm/serial.h File Reference

Real-Time Driver Model for Xenomai, serial device profile header.

Include dependency graph for serial.h:
8.11.1 Detailed Description

Real-Time Driver Model for Xenomai, serial device profile header.

Note

Copyright (C) 2005-2007 Jan Kiszka jan.kiszka@web.de

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

Xenomai is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with Xenomai; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
This graph shows which files directly or indirectly include this file:

![Graph showing file inclusion]

- include/rtdm/uapi/serial.h
- include/cobalt/kernel
- /rtdm/serial.h
- include/rtdm/serial.h
- include/trank/rtdm
- /rtserial.h

### Data Structures

- **struct rtser_config**
  
  Serial device configuration.

- **struct rtser_status**
  
  Serial device status.

- **struct rtser_event**
  
  Additional information about serial device events.

### Macros

- **#define RTSER_RTIOC_BREAK_CTL _IOR(RTIOC_TYPE_SERIAL, 0x06, int)**
  
  Set or clear break on UART output line.

**RTSER_DEF_BAUD**

*Default baud rate*

- **#define RTSER_DEF_BAUD 9600**

**RTSER_xxx_PARITY**

*Number of parity bits*

- **#define RTSER_NO_PARITY 0x00**
- **#define RTSER_ODD_PARITY 0x01**
- **#define RTSER_EVEN_PARITY 0x03**
- **#define RTSER_DEF_PARITY RTSER_NO_PARITY**

**RTSER_xxx_BITS**

*Number of data bits*
- #define RTSER_5_BITS 0x00
- #define RTSER_6_BITS 0x01
- #define RTSER_7_BITS 0x02
- #define RTSER_8_BITS 0x03
- #define RTSER_DEF_BITS RTSER_8_BITS

### RTSER_xxx_STOPB

*Number of stop bits*

- #define RTSER_1_STOPB 0x00  
  *valid only in combination with 5 data bits*
- #define RTSER_1_5_STOPB 0x01  
  *valid only in combination with 5 data bits*
- #define RTSER_2_STOPB 0x01  
  *valid only in combination with 5 data bits*
- #define RTSER_DEF_STOPB RTSER_1_STOPB  
  *valid only in combination with 5 data bits*

### RTSER_xxx_HAND

*Handshake mechanisms*

- #define RTSER_NO_HAND 0x00
- #define RTSER_RTSCTS_HAND 0x01
- #define RTSER_DEF_HAND RTSER_NO_HAND

### RTSER_RS485_xxx

*RS485 mode with automatic RTS handling*

- #define RTSER_RS485_DISABLE 0x00
- #define RTSER_RS485_ENABLE 0x01
- #define RTSER_DEF_RS485 RTSER_RS485_DISABLE

### RTSER_FIFO_xxx

*Reception FIFO interrupt threshold*

- #define RTSER_FIFO_DEPTH_1 0x00
- #define RTSER_FIFO_DEPTH_4 0x40
- #define RTSER_FIFO_DEPTH_8 0x80
- #define RTSER_FIFO_DEPTH_14 0xC0
- #define RTSER_DEF_FIFO_DEPTH RTSER_FIFO_DEPTH_1

### RTSER_TIMEOUT_xxx

*Special timeout values, see also RTDM_TIMEOUT_xxx*

- #define RTSER_TIMEOUT_INFINITE RTDM_TIMEOUT_INFINITE
- #define RTSER_TIMEOUT_NONE RTDM_TIMEOUT_NONE
- #define RTSER_DEF_TIMEOUT RTDM_TIMEOUT_INFINITE

### RTSER_xxx_TIMESTAMP_HISTORY

*Timestamp history control*

- #define RTSER_RX_TIMESTAMP_HISTORY 0x01
- #define RTSER_DEF_TIMESTAMP_HISTORY 0x00
RTSER_EVENT_xxx

Events bits

- #define RTSER_EVENT_RXPEND 0x01
- #define RTSER_EVENT_ERRPEND 0x02
- #define RTSER_EVENT_MODEMHI 0x04
- #define RTSER_EVENT_MODEMLO 0x08
- #define RTSER_EVENT_TXEMPTY 0x10
- #define RTSER_DEF_EVENT_MASK 0x00

RTSER_SET_xxx

Configuration mask bits

- #define RTSER_SET_BAUD 0x0001
- #define RTSER_SET_PARITY 0x0002
- #define RTSER_SET_DATA_BITS 0x0004
- #define RTSER_SET_STOP_BITS 0x0008
- #define RTSER_SET_HANDSHAKE 0x0010
- #define RTSER_SET_FIFO_DEPTH 0x0020
- #define RTSER_SET_TIMEOUT_RX 0x0100
- #define RTSER_SET_TIMEOUT_TX 0x0200
- #define RTSER_SET_TIMEOUT_EVENT 0x0400
- #define RTSER_SET_TIMESTAMP_HISTORY 0x0800
- #define RTSER_SET_EVENT_MASK 0x1000
- #define RTSER_SET_RS485 0x2000

RTSER_LSR_xxx

Line status bits

- #define RTSER_LSR_DATA 0x01
- #define RTSER_LSR_OVERRUN_ERR 0x02
- #define RTSER_LSR_PARITY_ERR 0x04
- #define RTSER_LSR_FRAMING_ERR 0x08
- #define RTSER_LSR_BREAK_IND 0x10
- #define RTSER_LSR_THR_EMTPY 0x20
- #define RTSER_LSR_TRANSM_EMPTY 0x40
- #define RTSER_LSR_FIFO_ERR 0x80
- #define RTSER_SOFT_OVERRUN_ERR 0x0100

RTSER_MSR_xxx

Modem status bits

- #define RTSER_MSR_DCTS 0x01
- #define RTSER_MSR_DDSR 0x02
- #define RTSER_MSR_TERI 0x04
- #define RTSER_MSR_DDCD 0x08
- #define RTSER_MSR_CTS 0x10
- #define RTSER_MSR_DSR 0x20
- #define RTSER_MSR_RI 0x40
- #define RTSER_MSR_DCD 0x80

RTSER_MCR_xxx

Modem control bits

- #define RTSER_MCR_DTR 0x01
- #define RTSER_MCR_RTS 0x02
- #define RTSER_MCR_OUT1 0x04
- #define RTSER_MCR_OUT2 0x08
- #define RTSER_MCR_LOOP 0x10
Sub-Classes of RTDM_CLASS_SERIAL

- #define RTDM_SUBCLASS_16550A 0

IOCTLS

Serial device IOCTLs

- #define RTSER_RTIOC_GET_CONFIG _IOR(RTIOC_TYPE_SERIAL, 0x00, struct rtser_config)
  Get serial device configuration.
- #define RTSER_RTIOC_SET_CONFIG _IOW(RTIOC_TYPE_SERIAL, 0x01, struct rtser_config)
  Set serial device configuration.
- #define RTSER_RTIOC_GET_STATUS _IOR(RTIOC_TYPE_SERIAL, 0x02, struct rtser_status)
  Get serial device status.
- #define RTSER_RTIOC_GET_CONTROL _IOR(RTIOC_TYPE_SERIAL, 0x03, int)
  Get serial device’s modem control register.
- #define RTSER_RTIOC_SET_CONTROL _IOW(RTIOC_TYPE_SERIAL, 0x04, int)
  Set serial device’s modem control register.
- #define RTSER_RTIOC_WAIT_EVENT _IOR(RTIOC_TYPE_SERIAL, 0x05, struct rtser_event)
  Wait on serial device events according to previously set mask.

RTSER_BREAK_xxx

Break control

- #define RTSER_BREAK_CLR 0x00
  Serial device configuration.
- #define RTSER_BREAK_SET 0x01
  Serial device configuration.
- #define RTIOC_TYPE_SERIAL RTDM_CLASS_SERIAL
  Serial device configuration.
- typedef struct rtser_config rtser_config_t
  Serial device configuration.
- typedef struct rtser_status rtser_status_t
  Serial device status.
- typedef struct rtser_event rtser_event_t
  Additional information about serial device events.

8.12.1 Detailed Description

Real-Time Driver Model for Xenomai, serial device profile header.

Note

Copyright (C) 2005-2007 Jan Kiszka jan.kiszka@web.de

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

This library 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 Lesser General Public License for more details.

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

8.12.2.1  RTSER_RTIOC_BREAK_CTL

#define RTSER_RTIOC_BREAK_CTL _IOR(RTIOC_TYPE_SERIAL, 0x06, int)

Set or clear break on UART output line.

Parameters

| in | arg | RTSER_BREAK_SET or RTSER_BREAK_CLR (int) |

Returns

0 on success, otherwise negative error code

Tags

  task-unrestricted

Note

A set break condition may also be cleared on UART line reconfiguration.

8.12.2.2  RTSER_RTIOC_GET_CONFIG

#define RTSER_RTIOC_GET_CONFIG _IOR(RTIOC_TYPE_SERIAL, 0x00, struct rtser_config)

Get serial device configuration.

Parameters

| out | arg | Pointer to configuration buffer (struct rtser_config) |

Returns

0 on success, otherwise negative error code

Tags

  task-unrestricted
8.12.2.3 RTSER_RTIOC_GET_CONTROL

#define RTSER_RTIOC_GET_CONTROL _IOR(RTIOC_TYPE_SERIAL, 0x03, int)

Get serial device's modem control register.

Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>arg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pointer to variable receiving the content (int, see RTSER_MCR_xxx)</td>
<td></td>
</tr>
</tbody>
</table>

Returns

0 on success, otherwise negative error code

Tags

task-unrestricted

8.12.2.4 RTSER_RTIOC_GET_STATUS

#define RTSER_RTIOC_GET_STATUS _IOR(RTIOC_TYPE_SERIAL, 0x02, struct rtser_status)

Get serial device status.

Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>arg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pointer to status buffer (struct rtser_status)</td>
<td></td>
</tr>
</tbody>
</table>

Returns

0 on success, otherwise negative error code

Tags

task-unrestricted

Note

The error states RTSER_LSR_OVERRUN_ERR, RTSER_LSR_PARITY_ERR, RTSER_LSR_FRAMING_ERR, and RTSER_SOFT_OVERRUN_ERR that may have occurred during previous read accesses to the device will be saved for being reported via this IOCTL. Upon return from RTSER_RTIOC_GET_STATUS, the saved state will be cleared.

8.12.2.5 RTSER_RTIOC_SET_CONFIG

#define RTSER_RTIOC_SET_CONFIG _IOW(RTIOC_TYPE_SERIAL, 0x01, struct rtser_config)

Set serial device configuration.
Parameters

| in | arg | Pointer to configuration buffer (struct rtser_config) |

Returns

0 on success, otherwise:

- -EPERM is returned if the caller’s context is invalid, see note below.
- -ENOMEM is returned if a new history buffer for timestamps cannot be allocated.

Tags

- task-unrestricted

Note

If rtser_config contains a valid timestamp_history and the addressed device has been opened in non-real-time context, this IOCTL must be issued in non-real-time context as well. Otherwise, this command will fail.

Examples:

cross-link.c.

8.12.2.6 RTSER_RTIOC_SET_CONTROL

#define RTSER_RTIOC_SET_CONTROL _IOW(RTIOC_TYPE_SERIAL, 0x04, int)

Set serial device’s modem control register.

Parameters

| in | arg | New control register content (int, see RTSER_MCR_xxx) |

Returns

0 on success, otherwise negative error code

Tags

- task-unrestricted
#define RTSER_RTIOC_WAIT_EVENT _IOR(RTIOC_TYPE_SERIAL, 0x05, struct rtser_event)

Wait on serial device events according to previously set mask.

Parameters

---

out | arg | Pointer to event information buffer (struct rtser_event)

Returns

- 0 on success, otherwise:
  - -EBUSY is returned if another task is already waiting on events of this device.
  - -EBADF is returned if the file descriptor is invalid or the device has just been closed.

Tags

- mode-unrestricted

Examples:

- cross-link.c.

8.13 include/rtdm/testing.h File Reference

Real-Time Driver Model for Xenomai, testing device profile header.

Include dependency graph for testing.h:
This graph shows which files directly or indirectly include this file:

8.13.1 Detailed Description

Real-Time Driver Model for Xenomai, testing device profile header.

Note

Copyright (C) 2005 Jan Kiszka jan.kiszka@web.de

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

Xenomai is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

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

8.14 include/rtdm/uapi/testing.h File Reference

Real-Time Driver Model for Xenomai, testing device profile header.

Include dependency graph for testing.h:
This graph shows which files directly or indirectly include this file:

```
#include/rtdm/uapi/testing.h
#include/cobalt/kernel
  /rtdm/testing.h
#include/rtdm/testing.h
#include/trank/rtdm
  /rttesting.h
```

Macros

### Sub-Classes of RTDM\_CLASS\_TESTING

- `#define RTDM\_SUBCLASS\_TIMERBENCH 0`
  subclass name: "timerbench"
- `#define RTDM\_SUBCLASS\_IRQBENCH 1`
  subclass name: "irqbench"
- `#define RTDM\_SUBCLASS\_SWITCHTEST 2`
  subclass name: "switchtest"
- `#define RTDM\_SUBCLASS\_RTDMTEST 3`
  subclass name: "rtdm"

### IOCTLs

#### Testing device IOCTLs

- `#define RTTST\_RTIOC\_INTERM\_BENCH\_RES _IOW(RTIIOC\_TYPE\_TESTING, 0x00, struct rttst\_intern\_bench\_res)`
- `#define RTTST\_RTIOC\_TMBENCH\_START _IOW(RTIIOC\_TYPE\_TESTING, 0x10, struct rttst\_tmbench\_config)`
- `#define RTTST\_RTIOC\_TMBENCH\_STOP _IOW(RTIIOC\_TYPE\_TESTING, 0x11, struct rttst\_overall\_bench\_res)`
- `#define RTTST\_RTIOC\_SWTEST\_SET\_TASKS\_COUNT _IOW(RTIIOC\_TYPE\_TESTING, 0x30, __u32)`
- `#define RTTST\_RTIOC\_SWTEST\_SET\_CPU _IOW(RTIIOC\_TYPE\_TESTING, 0x31, __u32)`
- `#define RTTST\_RTIOC\_SWTEST\_REGISTER\_UTASK _IOW(RTIIOC\_TYPE\_TESTING, 0x32, struct rttst\_swtest\_task)`
- `#define RTTST\_RTIOC\_SWTEST\_CREATE\_KTASK _IOW(RTIIOC\_TYPE\_TESTING, 0x33, struct rttst\_swtest\_task)`
- `#define RTTST\_RTIOC\_SWTEST\_PEND _IOR(RTIIOC\_TYPE\_TESTING, 0x34, struct rttst\_swtest\_dir)`
- `#define RTTST\_RTIOC\_SWTEST\_SWITCH\_TO _IOR(RTIIOC\_TYPE\_TESTING, 0x35, struct rttst\_swtest\_dir)`
• \#define RTTST_RTIIOC_SWTEST_GET_SWITCHES_COUNT _IOR(RTIOC_TYPE_TESTING, 0x36, __u32)
• \#define RTTST_RTIIOC_SWTEST_GET_LAST_ERROR _IOR(RTIOC_TYPE_TESTING, 0x37, struct rttst_swtest_error)
• \#define RTTST_RTIIOC_SWTEST_SET_PAUSE _IOW(RTIOC_TYPE_TESTING, 0x38, __u32)
• \#define RTTST_RTIIOC_RTDM_DEFER_CLOSE _IOW(RTIOC_TYPE_TESTING, 0x40, __u32)
• \#define RTTST_RTIIOC_RTDM_ACTOR_GET_CPU _IOR(RTIOC_TYPE_TESTING, 0x41, __u32)
• \#define RTTST_RTIIOC_RTDM_PING_PRIMARY _IOR(RTIOC_TYPE_TESTING, 0x42, __u32)
• \#define RTTST_RTIIOC_RTDM_PING_SECONDARY _IOR(RTIOC_TYPE_TESTING, 0x43, __u32)

8.14.1 Detailed Description

Real-Time Driver Model for Xenomai, testing device profile header.

Note
Copyright (C) 2005 Jan Kiszka jan.kiszka@web.de

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

This library 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 Lesser General Public License for more details.

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

8.15 include/cobalt/kernel/rtdm/udd.h File Reference

Copyright (C) 2014 Philippe Gerum rpm@xenomai.org

Include dependency graph for udd.h:
Data Structures

- struct udd_memregion
- struct udd_device
- struct udd_device::udd_reserved
  
  Reserved to the UDD core.

Macros

- #define UDD_IRQ_NONE 0
  
  No IRQ managed.
- #define UDD_IRQ_CUSTOM (-1)
  
  IRQ directly managed from the mini-driver on top of the UDD core.

Memory types for mapping

Types of memory for mapping

The UDD core implements a default ->mmap() handler which first attempts to hand over the request to the corresponding handler defined by the mini-driver. If not present, the UDD core establishes the mapping automatically, depending on the memory type defined for the region.

- #define UDD_MEM_NONE 0
  
  No memory region.
- #define UDD_MEM_PHYS 1
  
  Physical I/O memory region.
- #define UDD_MEM_LOGICAL 2
  
  Kernel logical memory region (e.g.
- #define UDD_MEM_VIRTUAL 3
  
  Virtual memory region with no direct physical mapping (e.g.

Functions

- int udd_register_device (struct udd_device *udd)
  
  Register a UDD device.
- int udd_unregister_device (struct udd_device *udd)
  
  Unregister a UDD device.
- struct udd_device * udd_get_device (struct rtdm_fd *fd)
  
  RTDM file descriptor to target UDD device.
- void udd_notify_event (struct udd_device *udd)
  
  Notify an IRQ event for an unmanaged interrupt.
- void udd_enable_irq (struct udd_device *udd, rtdm_event_t *done)
  
  Enable the device IRQ line.
- void udd_disable_irq (struct udd_device *udd, rtdm_event_t *done)
  
  Disable the device IRQ line.
8.15.1 Detailed Description

Copyright (C) 2014 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.

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.

8.16 include/rtdm/uapi/udd.h File Reference

This file is part of the Xenomai project.

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

Data Structures

- **struct udd_signotify**
  
  *UDD event notification descriptor.*

Macros

**UDD_IOCTL**

* IOCTL requests

- #define **UDD_RTIOC_IRQEN** _IO(RTDM_CLASS_UDD, 0)
  
  *Enable the interrupt line.*

- #define **UDD_RTIOC_IRQDIS** _IO(RTDM_CLASS_UDD, 1)
  
  *Disable the interrupt line.*

- #define **UDD_RTIOC_IRQSIG** _IOW(RTDM_CLASS_UDD, 2, struct udd_signotify)
  
  *Enable/Disable signal notification upon interrupt event.*
8.16.1 Detailed Description

This file is part of the Xenomai project.

Author

Copyright (C) 2014 Philippe Gerum rpm@xenomai.org

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

This library 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 Lesser General Public License for more details.

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

8.17 include/rtdm/analogy.h File Reference

Analogy for Linux, library facilities.

Include dependency graph for analogy.h:

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

Data Structures

- `struct a4l_descriptor`

  Structure containing device-information useful to users.
Macros

**ANALOGY_xxx_DESC**

Constants used as argument so as to define the description depth to recover

- `#define A4L_BSC_DESC 0x0
  BSC stands for basic descriptor (device data)
- `#define A4L_CPLX_DESC 0x1
  CPLX stands for complex descriptor (subdevice + channel + range data)

### 8.17.1 Detailed Description

Analogy for Linux, library facilities.

**Note**

Copyright (C) 1997-2000 David A. Schleef [ds@schleef.org](mailto:ds@schleef.org)
Copyright (C) 2008 Alexis Berlemont [alexis.berlemont@free.fr](mailto:alexis.berlemont@free.fr)

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

This library 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 Lesser General Public License for more details.

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

### 8.18 include/rtdm/uapi/analogy.h File Reference

Analogy for Linux, UAPI bits.

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

- struct a4l_cmd_desc
  Structure describing the asynchronous instruction.
- struct a4l_instruction
  Structure describing the synchronous instruction.
- struct a4l_instruction_list
  Structure describing the list of synchronous instructions.

Macros

- #define A4L_RNG_FACTOR 1000000
  Constant for internal use only (must not be used by driver developer).
- #define A4L_RNG_VOLT_UNIT 0x0
  Volt unit range flag.
- #define A4L_RNG_MAMP_UNIT 0x1
  MilliAmpere unit range flag.
- #define A4L_RNG_NO_UNIT 0x2
  No unit range flag.
- #define A4L_RNG_EXT_UNIT 0x4
  External unit range flag.
- #define A4L_RNG_UNIT(x)
  Macro to retrieve the range unit from the range flags.
- #define A4L_INSN_WAIT_MAX 100000
  Maximal wait duration.

ANALOGY_CMD_xxx

Common command flags definitions

- #define A4L_CMD_SIMUL 0x1
  Do not execute the command, just check it.
- #define A4L_CMD_BULK 0x2
  Perform data recovery / transmission in bulk mode.
- #define A4L_CMD_WRITE 0x4
  Perform a command which will write data to the device.

TRIG_xxx

Command triggers flags definitions

- #define TRIG_NONE 0x00000001
  Never trigger.
- #define TRIG_NOW 0x00000002
  Trigger now + N ns.
- #define TRIG_FOLLOW 0x00000004
  Trigger on next lower level trig.
- #define TRIG_TIME 0x00000008
  Trigger at time N ns.
- #define TRIG_TIMER 0x00000010
  Trigger at rate N ns.
- #define TRIG_COUNT 0x00000020
  Trigger when count reaches N.
- #define TRIG_EXT 0x00000040

Generated by Doxygen
Trigger on external signal N.

- #define TRIG_INT 0x00000080
  Trigger on analogy-internal signal N.
- #define TRIG_OTHER 0x00000100
  Driver defined trigger.
- #define TRIG_WAKE_EOS 0x0020
  Wake up on end-of-scan.
- #define TRIG_ROUND_MASK 0x00030000
  Trigger not implemented yet.
- #define TRIG_ROUND_NEAREST 0x00000000
  Trigger not implemented yet.
- #define TRIG_ROUND_DOWN 0x00010000
  Trigger not implemented yet.
- #define TRIG_ROUND_UP 0x00020000
  Trigger not implemented yet.
- #define TRIG_ROUND_UP_NEXT 0x00030000
  Trigger not implemented yet.

Channel macros

Specific precompilation macros and constants useful for the channels descriptors tab located in the command structure

- #define CHAN(a) ((a) & 0xffff)
  Channel indication macro.
- #define RNG(a) (((a) & 0xff) << 16)
  Range definition macro.
- #define AREF(a) (((a) & 0x03) << 24)
  Reference definition macro.
- #define FLAGS(a) ((a) & CR_FLAGS_MASK)
  Flags definition macro.
- #define PACK(a, b, c) (a | RNG(b) | AREF(c))
  Channel + range + reference definition macro.
- #define PACK_FLAGS(a, b, c, d) (PACK(a, b, c) | FLAGS(d))
  Channel + range + reference + flags definition macro.
- #define AREF_GROUND 0x00
  Analog reference is analog ground.
- #define AREF_COMMON 0x01
  Analog reference is analog common.
- #define AREF_DIFF 0x02
  Analog reference is differential.
- #define AREF_OTHER 0x03
  Analog reference is undefined.

Subdevices types

Flags to define the subdevice type

- #define A4L_SUBD_UNUSED (A4L_SUBD_MASK_SPECIAL|0x1)
  Unused subdevice.
- #define A4L_SUBD_AI (A4L_SUBD_MASK_READ|0x2)
  Analog input subdevice.
- #define A4L_SUBD_AO (A4L_SUBD_MASK_WRITE|0x4)
  Analog output subdevice.
- #define A4L_SUBD_DI (A4L_SUBD_MASK_READ|0x8)
  Digital input subdevice.
- #define A4L_SUBD_DO (A4L_SUBD_MASK_WRITE|0x10)
  Digital output subdevice.
- #define A4L_SUBD_DIO (A4L_SUBD_MASK_SPECIAL|0x20)
Digital input/output subdevice.
• #define A4L_SUBD_COUNTER (A4L_SUBD_MASK_SPECIAL|0x40)
  Counter subdevice.
• #define A4L_SUBD_TIMER (A4L_SUBD_MASK_SPECIAL|0x80)
  Timer subdevice.
• #define A4L_SUBD_MEMORY (A4L_SUBD_MASK_SPECIAL|0x100)
  Memory, EEPROM, DPRAM.
• #define A4L_SUBD_CALIB (A4L_SUBD_MASK_SPECIAL|0x200)
  Calibration subdevice DACs.
• #define A4L_SUBD_PROC (A4L_SUBD_MASK_SPECIAL|0x400)
  Processor, DSP.
• #define A4L_SUBD_SERIAL (A4L_SUBD_MASK_SPECIAL|0x800)
  Serial IO subdevice.
• #define A4L_SUBD_TYPES
  Mask which gathers all the types.

Subdevice features
Flags to define the subdevice's capabilities
• #define A4L_SUBD_CMD 0x1000
  The subdevice can handle command (i.e it can perform asynchronous acquisition)
• #define A4L_SUBD_MMAP 0x8000
  The subdevice support mmap operations (technically, any driver can do it; however, the developer
  might want that his driver must be accessed through read / write.

Subdevice status
Flags to define the subdevice's status
• #define A4L_SUBD_BUSY_NR 0
  The subdevice is busy, a synchronous or an asynchronous acquisition is occuring.
• #define A4L_SUBD_BUSY (1 << A4L_SUBD_BUSY_NR)
  The subdevice is busy, a synchronous or an asynchronous acquisition is occuring.
• #define A4L_SUBD_CLEAN_NR 1
  The subdevice is about to be cleaned in the middle of the detach procedure.
• #define A4L_SUBD_CLEAN (1 << A4L_SUBD_CLEAN_NR)
  The subdevice is busy, a synchronous or an asynchronous acquisition is occuring.

Instruction type
Flags to define the type of instruction
• #define A4L_INSN_READ (0 | A4L_INSN_MASK_READ)
  Read instruction.
• #define A4L_INSN_WRITE (1 | A4L_INSN_MASK_WRITE)
  Write instruction.
• #define A4L_INSN_BITS
  "Bits" instruction
• #define A4L_INSN_CONFIG
  Configuration instruction.
• #define A4L_INSN_GTOD
  Get time instruction.
• #define A4L_INSN_WAIT
  Wait instruction.
• #define A4L_INSN_INTTRIG
  Trigger instruction (to start asynchronous acquisition)
Configuration instruction type

Values to define the type of configuration instruction

- #define A4L_INSNS_CONFIG_DIO_INPUT 0
- #define A4L_INSNS_CONFIG_DIO_OUTPUT 1
- #define A4L_INSNS_CONFIG_DIO_OPENDRAIN 2
- #define A4L_INSNS_CONFIG_ANALOG_TRIG 16
- #define A4L_INSNS_CONFIG_ALT_SOURCE 20
- #define A4L_INSNS_CONFIG_BLOCK_SIZE 22
- #define A4L_INSNS_CONFIG_TIMER_1 23
- #define A4L_INSNS_CONFIG_FILTER 24
- #define A4L_INSNS_CONFIG_CHANGE_NOTIFY 25
- #define A4L_INSNS_CONFIG_SERIAL_CLOCK 26
- #define A4L_INSNS_CONFIG_BIDIRECTIONAL_DATA 27
- #define A4L_INSNS_CONFIG_DIO_QUERY 28
- #define A4L_INSNS_CONFIG_PWM_OUTPUT 29
- #define A4L_INSNS_CONFIG_GET_PWM_OUTPUT 30
- #define A4L_INSNS_CONFIG_ARM 31
- #define A4L_INSNS_CONFIG_DISARM 32
- #define A4L_INSNS_CONFIG_GET_COUNTER_STATUS 33
- #define A4L_INSNS_CONFIG_RESET 34
- #define A4L_INSNS_CONFIG_GPCT_SINGLE_PULSE_GENERATOR 1001 /* Use CTR as single pulsae generator */
- #define A4L_INSNS_CONFIG_GPCT_PULSE_TRAIN_GENERATOR 1002 /* Use CTR as pulsetrain generator */
- #define A4L_INSNS_CONFIG_GPCT_QUADRATURE_ENCODER 1003 /* Use the counter as encoder */
- #define A4L_INSNS_CONFIG_SET_GATE_SRC 2001 /* Set gate source */
- #define A4L_INSNS_CONFIG_GET_GATE_SRC 2002 /* Get gate source */
- #define A4L_INSNS_CONFIG_SET_CLOCK_SRC 2003 /* Set master clock source */
- #define A4L_INSNS_CONFIG_GET_CLOCK_SRC 2004 /* Get master clock source */
- #define A4L_INSNS_CONFIG_SET_OTHER_SRC 2005 /* Set other source */
- #define A4L_INSNS_CONFIG_SET_COUNTER_MODE 4097
- #define A4L_INSNS_CONFIG_SET_ROUTING 4099
- #define A4L_INSNS_CONFIG_GET_ROUTING 4109

Counter status bits

Status bits for INSN_CONFIG_GET_COUNTER_STATUS

- #define A4L_COUNTER_ARMED 0x1
- #define A4L_COUNTER_COUNTING 0x2
- #define A4L_COUNTER_TERMINAL_COUNT 0x4

IO direction

Values to define the IO polarity

- #define A4L_INPUT 0
- #define A4L_OUTPUT 1
- #define A4L_OPENDRAIN 2

Events types

Values to define the Analogy events. They might used to send some specific events through the instruction interface.

- #define A4L_EV_START 0x00040000
- #define A4L_EV_SCAN_BEGIN 0x00080000
- #define A4L_EV_CONVERT 0x00100000
- #define A4L_EV_SCAN_END 0x00200000
- #define A4L_EV_STOP 0x00400000

Generated by Doxygen
8.18.1 Detailed Description

Analogy for Linux, UAPI bits.

Note

Copyright (C) 1997-2000 David A. Schleef ds@schleef.org
Copyright (C) 2008 Alexis Berlemont alexis.berlemont@free.fr

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

This library 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 Lesser General Public License for more details.

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

8.18.2 Macro Definition Documentation

8.18.2.1 A4L_RNG_FACTOR

#define A4L_RNG_FACTOR 1000000

Constant for internal use only (must not be used by driver developer).

8.19 lib/analogy/calibration.h File Reference

Analogy for Linux, internal calibration declarations.

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

Analogy for Linux, internal calibration declarations.

Note

Copyright (C) 2014 Jorge A Ramirez-Ortiz jro@xenomai.org

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

This library 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 Lesser General Public License for more details.

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

8.20 lib/analogy/internal.h File Reference

Analogy for Linux, internal declarations.

Include dependency graph for internal.h:
8.20.1 Detailed Description

Analogy for Linux, internal declarations.

Note

Copyright (C) 2008 Alexis Berlemont alexis.berlemont@free.fr

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

This library 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 Lesser General Public License for more details.

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

8.21 lib/analogy/async.c File Reference

Analogy for Linux, command, transfer, etc.

Include dependency graph for async.c:

```
Functions
- int a4l_snd_command (a4l_desc_t *dsc, a4l_cmd_t *cmd)
  Send a command to an Analoy device.
- int a4l_snd_cancel (a4l_desc_t *dsc, unsigned int idx_subd)
  Cancel an asynchronous acquisition.
- int a4l_set_bufsize (a4l_desc_t *dsc, unsigned int idx_subd, unsigned long size)
  Change the size of the asynchronous buffer.
- int a4l_get_bufsize (a4l_desc_t *dsc, unsigned int idx_subd, unsigned long *size)
  Get the size of the asynchronous buffer.
- int a4l_mark_bufrw (a4l_desc_t *dsc, unsigned int idx_subd, unsigned long cur, unsigned long *new)
  Update the asynchronous buffer state.
- int a4l_poll (a4l_desc_t *dsc, unsigned int idx_subd, unsigned long ms_timeout)
  Get the available data count.
- int a4l_mmap (a4l_desc_t *dsc, unsigned int idx_subd, unsigned long size, void **ptr)
  Map the asynchronous ring-buffer into a user-space.
- int a4l_async_read (a4l_desc_t *dsc, void *buf, size_t nbyte, unsigned long ms_timeout)
  Perform asynchronous read operation on the analog input subdevice.
- int a4l_async_write (a4l_desc_t *dsc, void *buf, size_t nbyte, unsigned long ms_timeout)
  Perform asynchronous write operation on the analog input subdevice.
```
8.21.1 Detailed Description

Analogy for Linux, command, transfer, etc.
related features

Note
Copyright (C) 1997-2000 David A. Schleef ds@schleef.org
Copyright (C) 2008 Alexis Berlemont alexis.berlemont@free.fr

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

This library 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 Lesser General Public License for more details.

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

8.22 lib/analogy/calibration.c File Reference

Analogy for Linux, device, subdevice, etc.

Include dependency graph for calibration.c:

Functions

- int a4l_read_calibration_file (char *name, struct a4l_calibration_data *data)
  Read the analogy generated calibration file.
- int a4l_get_softcal_converter (struct a4l_polynomial *converter, int subd, int chan, int range, struct a4l_calibration_data *data)
  Get the polynomial that will be use for the software calibration.
- int a4l_rawtodcal (a4l_chinfo_t *chan, double *dst, void *src, int cnt, struct a4l_polynomial *converter)
  Convert raw data (from the driver) to calibrated double units.
- int a4l_dcaltoraw (a4l_chinfo_t *chan, void *dst, double *src, int cnt, struct a4l_polynomial *converter)
  Convert double values to raw calibrated data using polynomials.
8.22.1 Detailed Description

Analogy for Linux, device, subdevice, etc.
related features

Note

Copyright (C) 1997-2000 David A. Schleef ds@schleef.org
Copyright (C) 2014 Jorge A. Ramirez-Ortiz jro@xenomai.org

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

This library 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 Lesser General Public License for more details.

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

8.23 lib/analogy/descriptor.c File Reference

Analogy for Linux, descriptor related features.

Include dependency graph for descriptor.c:

![Dependency Graph](image)

Functions

- int `a4l_sys_desc` (int fd, a4l_desc_t *dsc, int pass)
  Get a descriptor on an attached device.
- int `a4l_open` (a4l_desc_t *dsc, const char *fname)
  Open an Analogy device and basically fill the descriptor.
- int `a4l_close` (a4l_desc_t *dsc)
  Close the Analogy device related with the descriptor.
- int `a4l_fill_desc` (a4l_desc_t *dsc)
  Fill the descriptor with subdevices, channels and ranges data.
- int `a4l_get_subdinfo` (a4l_desc_t *dsc, unsigned int subd, a4l_sbinfo_t **info)
  Get an information structure on a specified subdevice.
- int `a4l_get_chinfo` (a4l_desc_t *dsc, unsigned int subd, unsigned int chan, a4l_chinfo_t **info)
  Get an information structure on a specified channel.
- int `a4l_get_rnginfo` (a4l_desc_t *dsc, unsigned int subd, unsigned int chan, unsigned int rng, a4l_rnginfo_t **info)
  Get an information structure on a specified range.
8.23.1 Detailed Description

Analogy for Linux, descriptor related features.

Note

Copyright (C) 1997-2000 David A. Schleef ds@schleef.org
Copyright (C) 2008 Alexis Berlemont alexis.berlemont@free.fr

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

This library 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 Lesser General Public License for more details.

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

8.24 lib/analogy/range.c File Reference

Analogy for Linux, range related features.

Include dependency graph for range.c:

---

Functions

- `int a4l_sizeof_chan (a4l_chinfo_t *chan)`
  
  Get the size in memory of an acquired element.

- `int a4l_sizeof_subd (a4l_sbinfo_t *subd)`
  
  Get the size in memory of a digital acquired element.

- `int a4l_find_range (a4l_desc_t *dsc, unsigned int idx_subd, unsigned int idx_chan, unsigned long unit, double min, double max, a4l_rnginfo_t **rng)`
  
  Find the most suitable range.

- `int a4l_rawtoul (a4l_chinfo_t *chan, unsigned long *dst, void *src, int cnt)`
  
  Unpack raw data (from the driver) into unsigned long values.

- `int a4l_rawtof (a4l_chinfo_t *chan, a4l_rnginfo_t *rng, float *dst, void *src, int cnt)`
  
  Convert raw data (from the driver) to float-typed samples.
8.25 lib/analogy/root_leaf.h File Reference

Analogy for Linux, root / leaf system.

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

```
lib/analogy/root_leaf.h
lib/analogy/descriptor.c
```

---

**Detailed Description**

Analogy for Linux, range related features.

**Note**

Copyright (C) 1997-2000 David A. Schleef ds@schleef.org
Copyright (C) 2008 Alexis Berlemont alexis.berlemont@free.fr

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

This library 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 Lesser General Public License for more details.

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

---

- int a4l_rawtod (a4l_chinfo_t *chan, a4l_rnginfo_t *rng, double *dst, void *src, int cnt)
  Convert raw data (from the driver) to double-typed samples.
- int a4l_ultoraw (a4l_chinfo_t *chan, void *dst, unsigned long *src, int cnt)
  Pack unsigned long values into raw data (for the driver)
- int a4l_ftoraw (a4l_chinfo_t *chan, a4l_rnginfo_t *rng, void *dst, float *src, int cnt)
  Convert float-typed samples to raw data (for the driver)
- int a4l_dtoraw (a4l_chinfo_t *chan, a4l_rnginfo_t *rng, void *dst, double *src, int cnt)
  Convert double-typed samples to raw data (for the driver)
8.25.1 Detailed Description

Analogy for Linux, root / leaf system.

Note

Copyright (C) 1997-2000 David A. Schleef ds@schleef.org
Copyright (C) 2008 Alexis Berlemont alexis.berlemont@free.fr

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

This library 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 Lesser General Public License for more details.

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

8.26 lib/analogy/sync.c File Reference

Analogy for Linux, instruction related features.

Include dependency graph for sync.c:

![Dependency Graph](image)

Functions

- **int a4l_snd_insnlist (a4l_desc_t *dsc, a4l_insnlst_t *arg)**
  
  Perform a list of synchronous acquisition misc operations.

- **int a4l_snd_insn (a4l_desc_t *dsc, a4lInsn_t *arg)**
  
  Perform a synchronous acquisition misc operation.

- **int a4l_sync_write (a4l_desc_t *dsc, unsigned int idx_subd, unsigned int chan_desc, unsigned int ns_delay, void *buf, size_t nbyte)**
  
  Perform a synchronous acquisition write operation.

- **int a4l_sync_read (a4l_desc_t *dsc, unsigned int idx_subd, unsigned int chan_desc, unsigned int ns_delay, void *buf, size_t nbyte)**
  
  Perform a synchronous acquisition read operation.

- **int a4l_sync_dio (a4l_desc_t *dsc, unsigned int idx_subd, void *mask, void *buf)**
  
  Perform a synchronous acquisition digital acquisition.

- **int a4l_config_subd (a4l_desc_t *dsc, unsigned int idx_subd, unsigned int type,...)**
  
  Configure a subdevice.
8.26.1 Detailed Description

Analogy for Linux, instruction related features.

Note

Copyright (C) 1997-2000 David A. Schleef ds@schleef.org
Copyright (C) 2008 Alexis Berlemont alexis.berlemont@free.fr

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

This library 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 Lesser General Public License for more details.

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

8.27 lib/analogy/sys.c File Reference

Analogy for Linux, descriptor related features.

Include dependency graph for sys.c:

![Dependency Graph](image)

Functions

- int **a4l_sys_open**(const char *fname)
  
  Open an Analogy device.

- int **a4l_sys_close**(int fd)
  
  Close an Analogy device.

- int **a4l_sys_read**(int fd, void *buf, size_t nbyte)
  
  Read from an Analogy device.

- int **a4l_sys_write**(int fd, void *buf, size_t nbyte)
  
  Write to an Analogy device.

- int **a4l_sys_attach**(int fd, a4l_lnkdesc_t *arg)
  
  Attach an Analogy device to a driver.

- int **a4l_sys_detach**(int fd)
  
  Detach an Analogy device from a driver.

- int **a4l_sys_bufcfg**(int fd, unsigned int idx_subd, unsigned long size)
  
  Configure the buffer size.
8.27.1 Detailed Description

Analogy for Linux, descriptor related features.

Note

Copyright (C) 1997-2000 David A. Schleef ds@schleef.org
Copyright (C) 2008 Alexis Berlemont alexis.berlemont@free.fr

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

This library 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 Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.
Chapter 9

Example Documentation

9.1 bufp-label.c

/*
 * Copyright (C) 2009 Philippe Gerum <rpm@xenomai.org>.
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2 of the License, or (at your option) any later version.
 * This library 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
 * Lesser General Public License for more details.
 * You should have received a copy of the GNU Lesser General Public
 * License along with this library; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.
 *
 * BUFP-based client/server demo, using the read(2)/write(2)
 * system calls to exchange data over a socket.
 *
 * In this example, two sockets are created. A server thread (reader)
 * is bound to a real-time port and receives a stream of bytes sent to
 * this port from a client thread (writer).
 */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <string.h>
#include <pthread.h>
#include <errno.h>
#include <rtdm/ipc.h>

pthread_t svtid, cltid;

#define BUFP_PORT_LABEL "bufp-demo"

static const char *msg[] = {
    "Surfing With The Alien",
    "Lords of Karma",
    "Banana Mango",
    "Psycho Monkey",
    "Luminous Flesh Giants",
    "Moroccan Sunset",
    "Satch Boogie",
    "Flying In A Blue Dream",
    "Ride",
    "Summer Song",
    "Speed Of Light",
    "Crystal Planet",
    "Raspberry Jam Delta-V",
    "Champagne?",
    "Clouds Race Across The Sky",
    "Engines Of Creation"
};
static void fail(const char *reason)
{
    perror(reason);
    exit(EXIT_FAILURE);
}

static void *server(void *arg)
{
    struct rtipc_port_label plabel;
    struct sockaddr_ipc saddr;
    char buf[128];
    size_t bufsz;
    int ret, s;
    s = socket(AF_RTIIPC, SOCK_DGRAM, IPCPROTO_BUFF);
    if (s < 0)
        fail("socket");
    
    /*
    * Set a 16k buffer for the server endpoint. This
    * configuration must be done prior to binding the socket to a
    * port.
    */
    bufsz = 16384; /* bytes */
    ret = setsockopt(s, SOL_BUFF, BUFP_BUFSZ,
                     &bufsz, sizeof(bufsz));
    if (ret)
        fail("setsockopt");
    
    /*
    * Set a port label. This name will be registered when
    * binding, in addition to the port number (if given).
    */
    strcpy(plabel.label, BUFP_PORT_LABEL);
    ret = setsockopt(s, SOL_BUFF, BUFP_LABEL,
                     &plabel, sizeof(plabel));
    if (ret)
        fail("setsockopt");
    
    /*
    * Bind the socket to the port. Assign that port a label, so
    * that peers may use a descriptive information to locate
    * it. Labeled ports will appear in the
    * /proc/xenomai/registry/rtipc/bufp directory once the socket
    * is bound.
    */
    saddr.sipc_family = AF_RTIIPC;
    saddr.sipc_port = -1;
    ret = bind(s, (struct sockaddr *)&saddr, sizeof(saddr));
    if (ret)
        fail("bind");
    
    for (;;) {
        ret = read(s, buf, sizeof(buf));
        if (ret < 0) {
            close(s);
            fail("read");
        }
        printf("%s: received %d bytes,
               \"%.*s\"\n", __FUNCTION__, ret, ret, buf);
    }
    return NULL;
}

static void *client(void *arg)
{
    struct rtipc_port_label plabel;
    struct sockaddr_ipc svaddr;
    int ret, s, n = 0, len;
    struct timespec ts;
    s = socket(AF_RTIIPC, SOCK_DGRAM, IPCPROTO_BUFF);
    if (s < 0)
        fail("socket");
    
    /*
    * Set the port label. This name will be used to find the peer
    * when connecting, instead of the port number. The label must
    * be set _after_ the socket is bound to the port, so that
    * BUFP does not try to register this label for the client
    * port as well (like the server thread did).
    */
}
strcpy(label.label, BUFP_PORT_LABEL);
ret = setsockopt(s, SOL_BUFP, BUFP_LABEL,
&plabel, sizeof(plabel));
if (ret)
    fail("setsockopt");
memset(&svsaddr, 0, sizeof(svsaddr));
svsaddr.sipc_family = AF_RTIPC;
svsaddr.sipc_port = -1; /* Tell BUFP to search by label. */
ret = connect(s, (struct sockaddr *)&svsaddr, sizeof(svsaddr));
if (ret)
    fail("connect");
for (;;) {
    len = strlen(msg[n]);
    ret = write(s, msg[n], len);
    if (ret < 0) {
        close(s);
        fail("write");
    }
    printf("%s: sent %d bytes,\n" __FUNCTION__, ret, ret, msg[n]);
    n = (n + 1) % (sizeof(msg) / sizeof(msg[0]));
    /* We run in full real-time mode (i.e. primary mode),
    * so we have to let the system breathe between two
    * iterations.
    */
    ts.tv_sec = 0;
    ts.tv_nsec = 500000000; /* 500 ms */
    clock_nanosleep(CLOCK_REALTIME, 0, &ts, NULL);
}
return NULL;
}
int main(int argc, char **argv)
{
    struct sched_param svparam = { .sched_priority = 71 };
    struct sched_param clparam = { .sched_priority = 70 };
    pthread_attr_t svattr, clattr;
    sigset_t set;
    int sig;
    sigemptyset(&set);
    sigaddset(&set, SIGINT);
    sigaddset(&set, SIGTERM);
    sigaddset(&set, SIGHUP);
    pthread_sigmask(SIG_BLOCK, &set, NULL);

    pthread_attr_init(&svattr);
    pthread_attr_setdetachstate(&svattr, PTHREAD_CREATE_JOINABLE);
    pthread_attr_setschedpolicy(&svattr, SCHED_FIFO);
    pthread_attr_setschedparam(&svattr, &svparam);

    errno = pthread_create(&svtid, &svattr, &server, NULL);
    if (errno)
        fail("pthread_create");

    pthread_attr_init(&clattr);
    pthread_attr_setdetachstate(&clattr, PTHREAD_CREATE_JOINABLE);
    pthread_attr_setschedpolicy(&clattr, SCHED_FIFO);
    pthread_attr_setschedparam(&clattr, &clparam);

    errno = pthread_create(&cltid, &clattr, &client, NULL);
    if (errno)
        fail("pthread_create");

    sigwait(&set, &sig);
    pthread_cancel(svtid);
    pthread_join(svtid, NULL);
    pthread_cancel(cltid);
    pthread_join(cltid, NULL);

    return 0;
}/*

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

This library 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 Lesser General Public License for more details.

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

BUFF-based client/server demo, using the read(2)/write(2) system calls to exchange data over a socket.

In this example, two sockets are created. A server thread (reader) is bound to a real-time port and receives a stream of bytes sent to this port from a client thread (writer).

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <string.h>
#include <pthread.h>
#include <errno.h>
#include <rtdm/ipc.h>

pthread_t svtid, cltid;
#define BUFP_SVPORT 12

static const char *msg[] = {
    "Surfing With The Alien",
    "Lords of Karma",
    "Banana Mango",
    "Psycho Monkey",
    "Luminous Flesh Giants",
    "Moroccan Sunset",
    "Satch Boogie",
    "Flying In A Blue Dream",
    "Ride",
    "Summer Song",
    "Speed Of Light",
    "Crystal Planet",
    "Raspberry Jam Delta-V",
    "Champagne?",
    "Clouds Race Across The Sky",
    "Engines Of Creation"
};

static void fail(const char *reason)
{
    perror(reason);
    exit(EXIT_FAILURE);
}

static void *server(void *arg)
{
    struct sockaddr_ipc saddr;
    char buf[128];
    size_t bufsz;
    int ret, s;
    s = socket(AF_RTIPC, SOCK_DGRAM, IPCPROTO_BUFP);
    if (s < 0)
        fail("socket");
    
    /* Set a 16k buffer for the server endpoint. This configuration must be done prior to binding the socket to a port. */
    bufsz = 16384; /* bytes */
    ret = setsockopt(s, SOL_BUFP, BUFP_BUFSIZE, &bufsz, sizeof(bufsz));
    if (ret)
        fail("setsockopt");
    
    saddr.sipc_family = AF_RTIPC;
    saddr.sipc_port = BUFP_SVPORT;
    ret = bind(s, (struct sockaddr *)&saddr, sizeof(saddr));
    if (ret)
        fail("bind");
    
    pthread_t svtid, cltid;
    pthread_create(&svtid, NULL, server, arg);
    pthread_create(&cltid, NULL, server, arg);
    
    /* Wait for both server threads to complete. */
    pthread_join(svtid, NULL);
    pthread_join(cltid, NULL);
    return NULL;
}

int main(int argc, char **argv)
{
    if (argc < 2)
        fail("Requires a port number as argument.");
    
    int port = atoi(argv[1]);
    
    struct sockaddr_ipc addr;
    addr.sipc_family = AF_RTIPC;
    addr.sipc_port = port;
    
    return server(&addr);
}
```
if (ret)
    fail("bind");

for (;;) {
    ret = read(s, buf, sizeof(buf));
    if (ret < 0) {
        close(s);
        fail("read");
    }
    printf("%s: received %d bytes, \"%.*s\"\n", __FUNCTION__, ret, ret, buf);
}
return NULL;
}

static void *client(void *arg)
{
    struct sockaddr_ipc svsaddr;
    int ret, s, n = 0, len;
    struct timespec ts;
    s = socket(AF_RTIPC, SOCK_DGRAM, IPCPROTO_BUFP);
    if (s < 0)
        fail("socket");
    memset(&svsaddr, 0, sizeof(svsaddr));
    svsaddr.sipc_family = AF_RTIPC;
    svsaddr.sipc_port = BUFP_SVPORT;
    ret = connect(s, (struct sockaddr *)&svsaddr, sizeof(svsaddr));
    if (ret)
        fail("connect");

    for (;;) {
        len = strlen(msg[n]);
        ret = write(s, msg[n], len);
        if (ret < 0) {
            close(s);
            fail("write");
        }
        printf("%s: sent %d bytes, \"%.*s\"\n", __FUNCTION__, ret, ret, msg[n]);
        n = (n + 1) % (sizeof(msg) / sizeof(msg[0]));
        /*
         * We run in full real-time mode (i.e. primary mode),
         * so we have to let the system breathe between two
         * iterations.
         */
        ts.tv_sec = 0;
        ts.tv_nsec = 500000000; /* 500 ms */
        clock_nanosleep(CLOCK_REALTIME, 0, &ts, NULL);
    }
    return NULL;
}

int main(int argc, char **argv)
{
    struct sched_param svparam = {.sched_priority = 71 };  
    struct sched_param clparam = {.sched_priority = 70 };  
    pthread_attr_t svattr, clattr;
    sigset_t set;
    int sig;

    sigemptyset(&set);
    sigaddset(&set, SIGINT);
    sigaddset(&set, SIGTERM);
    sigaddset(&set, SIGUSR1);
    pthread_setsigmask(SIG_BLOCK, &set, NULL);

    pthread_attr_init(&svattr);
    pthread_attr_setdetachstate(&svattr, PTHREAD_CREATE_JOINABLE);
    pthread_attr_setinheritsched(&svattr, PTHREAD_EXPLICIT_SCHED);
    pthread_attr_setschedpolicy(&svattr, SCHED_FIFO);
    pthread_attr_setschedparam(&svattr, &svparam);
    errno = pthread_create(&svtid, &svattr, &server, NULL);
    if (errno)
        fail("pthread_create");

    pthread_attr_init(&clattr);
    pthread_attr_setdetachstate(&clattr, PTHREAD_CREATE_JOINABLE);
    pthread_attr_setinheritsched(&clattr, PTHREAD_EXPLICIT_SCHED);
    pthread_attr_setschedpolicy(&clattr, SCHED_FIFO);
    pthread_attr_setschedparam(&clattr, &clparam);
    errno = pthread_create(&cltid, &clattr, &client, NULL);
}
if (errno)
  fail("pthread_create");
sigwait(&set, &sig);
pthread_cancel(svtid);
pthread_cancel(cltid);
pthread_join(svtid, NULL);
pthread_join(cltid, NULL);
  return 0;
}

9.3 can-rtt.c

/*
 * Round-Trip-Time Test - sends and receives messages and measures the
 * time in between.
 *
 * Copyright (C) 2006 Wolfgang Grandegger <wg@grandegger.com>
 * Based on RTnet's examples/xenomai/posix/rtt-sender.c.
 * Copyright (C) 2002 Ulrich Marx <marx@kammer.uni-hannover.de>
 * 2002 Marc Kleine-Budde <kleine-budde@gmx.de>
 * 2006 Jan Kiszka <jan.kiszka@web.de>
 *
 * 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., 675 Mass Ave, Cambridge, MA 02139, USA.
 *
 * The program sends out CAN messages periodically and copies the current
 * time-stamp to the payload. At reception, that time-stamp is compared
 * with the current time to determine the round-trip time. The jitter
 * values are printer out regularly. Concurrent tests can be carried out
 * by starting the program with different message identifiers. It is also
 * possible to use this program on a remote system as simple repeater to
 * loopback messages.
 */
#include <errno.h>
#include <mq.h>
#include <signal.h>
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <limits.h>
#include <getopt.h>
#include <memory.h>
#include <netinet/in.h>
#include <net/if.h>
#include <sys/ioctl.h>
#include <rtdm/can.h>
#include <xenomai/init.h>

#define NSEC_PER_SEC 1000000000
static unsigned int cycle = 10000; /* 10 ms */
static canid_t can_id = 0x1;
static pthread_t txthread, rxthread;
static int txsock, rxsock;
static int overruns;
static int repeater;
static rtt_stat { long long rtt;
void application_usage(void)
{
fprintf(stderr, "usage: %s [options] <tx-can-interface> <rx-can-interface>:\n", get_program_name());
fprintf(stderr, "  -r, --repeater Repeater, send back received messages\n"
"  -i, --id=ID CAN Identifier (default = 0x1)\n"
"  -c, --cycle Cycle time in us (default = 1000us)\n");
}

static void *transmitter(void *arg)
{
struct sched_param param = { .sched_priority = 80 };
struct timespec next_period;
struct can_frame frame;
long long *rtt_time = (long long *)&frame.data, t;
/* Pre-fill CAN frame */
frame.can_id = can_id;
frame.can_dlc = sizeof(*rtt_time);
pthread_setname_np(pthread_self(), "rtcan_rtt_transmitter");
pthread_setschedparam(pthread_self(), SCHED_FIFO, &param);
clock_gettime(CLOCK_MONOTONIC, &next_period);
while(1) {
next_period.tv_nsec += cycle * 1000;
while (next_period.tv_nsec >= NSEC_PER_SEC) {
next_period.tv_nsec -= NSEC_PER_SEC;
next_period.tv_sec++;
}

clock_nanosleep(CLOCK_MONOTONIC, TIMER_ABSTIME, &next_period, NULL);
if (rxcount != txcount) {
overruns++;
continue;
}

clock_gettime(CLOCK_MONOTONIC, &time);
t = (long long)time.tv_sec * NSEC_PER_SEC + time.tv_nsec;
memcpy(rtt_time, &t, sizeof(t));
/* Transmit the message containing the local time */
if (send(txsock, (void *)&frame, sizeof(struct can_frame), 0) < 0) {
if (errno == EAGAIN) {
printf("terminating transmitter thread\n");
else {
return NULL;
}
}
}
}

static void *receiver(void *arg)
{
struct sched_param param = { .sched_priority = 82 };
struct timespec time;
struct can_frame frame;
long long *rtt_time = (long long *)frame.data, t;
struct rtt_stat rtt_stat = {0, 1000000000000000000LL, -1000000000000000000LL,
0, 0, 0};
pthread_setname_np(pthread_self(), "rtcan_rtt_receiver");
pthread_setschedparam(pthread_self(), SCHED_FIFO, &param);
rtt_stat.counts_per_sec = 1000000 / cycle;
while(1) {
if (recv(rxsock, (void *)&frame, sizeof(struct can_frame), 0) < 0) {
if (errno == EAGAIN) {
printf("terminating receiver thread\n");
else {
return NULL;
}
}
if (repeater) {
    /* Transmit the message back as is */
    if (send(txsock, (void *)&frame, sizeof(struct can_frame), 0) < 0) {
        if (errno == EBADF)
            printf("terminating transmitter thread\n");
        else
            perror("send failed");
        return NULL;
    }
    txcount++;
} else {
    memcpy(&t, rtt_time, sizeof(t));
    clock_gettime(CLOCK_MONOTONIC, &time);
    if (rxcount > 0) {
        rtt_stat.rtt = ((long long)time.tv_sec * 1000000000LL +
            time.tv_nsec - t);
        rtt_stat.rtt_sum += rtt_stat.rtt;
        if (rtt_stat.rtt < rtt_stat.rtt_min)
            rtt_stat.rtt_min = rtt_stat.rtt;
        if (rtt_stat.rtt > rtt_stat.rtt_max)
            rtt_stat.rtt_max = rtt_stat.rtt;
    }
    rxcount++;
    if ((rxcount % rtt_stat.counts_per_sec) == 0) {
        mq_send(mq, (char *)&rtt_stat, sizeof(rtt_stat), 0);
        rtt_stat.rtt_sum_last = rtt_stat.rtt_sum;
    }
}

static void catch_signal(int sig)
{
    mq_close(mq);
    close(rxsock);
    close(txsock);
}

int main(int argc, char *argv[])
{
    struct sched_param param = { .sched_priority = 1 };
    pthread_attr_t tattr;
    struct mq_attr mqattr;
    struct sockaddr_can rxaddr, txaddr;
    struct can_filter rxfilter[1];
    struct rtt_stat rtt_stat;
    char mqname[32];
    char *txdev, *rxdev;
    struct can_ifreq ifr;
    int ret, opt;
    struct option long_options[] = {
        { "id", required_argument, 0, 'i' },
        { "cycle", required_argument, 0, 'c' },
        { "repeater", no_argument, 0, 'r' },
        { 0, 0, 0, 0 }
    };
    while ((opt = getopt_long(argc, argv, "ri:c:",
            long_options, NULL)) != -1) {
        switch (opt) {
            case 'c':
                cycle = atoi(optarg);
                break;
            case 'i':
                can_id = strtoul(optarg, NULL, 0);
                break;
            case 'r':
                repeater = 1;
                break;
            default:
                fprintf(stderr, "Unknown option '\n"
            exit(-1);
        }
    }
    printf("%d %d\n", optind, argc);
    if (optind + 2 != argc) {
        xenomai_usage();
        exit(0);
    }
txdev = argv[optind];
rxdev = argv[optind + 1];

/* Create and configure RX socket */
if ((rxsock = socket(PF_CAN, SOCK_RAW, CAN_RAW)) < 0) {
    perror("RX socket failed");
    return -1;
}

strncpy(ifr.ifr_name, rxdev, IFNAMSIZ);
printf("RX rxsock=%d, ifr_name=%s
", rxsock, ifr.ifr_name);
if (ioctl(rxsock, SIOCGIFINDEX, &ifr) < 0) {
    perror("RX ioctl SIOCGIFINDEX failed");
    goto failure1;
}

/* We only want to receive our own messages */
rxfilter[0].can_id = can_id;
rxfilter[0].can_mask = 0xff;
if (setsockopt(rxsock, SOL_CAN_RAW, CAN_RAW_FILTER,
        &rxfilter, sizeof(struct can_filter)) < 0) {
    perror("RX setsockopt CAN_RAW_FILTER failed");
    goto failure1;
}

memset(&rxaddr, 0, sizeof(rxaddr));
rxaddr.can_ifindex = ifr.ifr_ifindex;
rxaddr.can_family = AF_CAN;
if (bind(rxsock, (struct sockaddr *)&rxaddr, sizeof(rxaddr)) < 0) {
    perror("RX bind failed\n");
    goto failure1;
}

/* Create and configure TX socket */
if (strcmp(rxdev, txdev) == 0) {
    txsock = rxsock;
} else {
    if ((txsock = socket(PF_CAN, SOCK_RAW, 0)) < 0) {
        perror("TX socket failed");
        goto failure1;
    }
    strncpy(ifr.ifr_name, txdev, IFNAMSIZ);
    printf("TX txsock=%d, ifr_name=%s
", txsock, ifr.ifr_name);
    if (ioctl(txsock, SIOCGIFINDEX, &ifr) < 0) {
        perror("TX ioctl SIOCGIFINDEX failed");
        goto failure2;
    }
    memset(&txaddr, 0, sizeof(txaddr));
    txaddr.can_ifindex = ifr.ifr_ifindex;
    txaddr.can_family = AF_CAN;
    if (bind(txsock, (struct sockaddr *)&txaddr, sizeof(txaddr)) < 0) {
        perror("TX bind failed\n");
        goto failure2;
    }
}

signal(SIGTERM, catch_signal);
signal(SIGINT, catch_signal);
signal(SIGHUP, catch_signal);

printf("Round-Trip-Time test %s -> %s with CAN ID 0x%x
", argv[optind], argv[optind + 1], can_id);
printf("Cycle time: %d us\n", cycle);
printf("All RTT timing figures are in us.\n");

/* Create statistics message queue */
snprintf(mqname, sizeof(mqname), "/rtcan_rtt-%d", getpid());
mqattr.mq_flags = 0;
mqattr.mq_maxmsg = 100;
mqattr.mq_msgsize = sizeof(struct rtt_stat);
mq = mq_open(mqname, O_RDWR | O_CREAT | O_EXCL, 0600, &mqattr);
if (mq == (mqd_t)-1) {
    perror("opening message queue failed");
    goto failure2;
}
/* Create receiver RT-thread */
pthread_attr_init(&thattr);
pthread_attr_setdetachstate(&thattr, PTHREAD_CREATE_JOINABLE);
ret = pthread_create(&rxthread, &thattr, &receiver, NULL);
if (ret) {
  fprintf(stderr, "%s: pthread_create(receiver) failed\n",
          strerror(-ret));
  goto failure3;
}

/* Create transmitter RT-thread */
if (!repeater) {
  ret = pthread_create(&txthread, &thattr, &transmitter, NULL);
  if (ret) {
    fprintf(stderr, "%s: pthread_create(transmitter) failed\n",
            strerror(-ret));
    goto failure4;
  }
}

pthread_setschedparam(pthread_self(), SCHED_FIFO, &param);

while (1) {
  long long rtt_avg;
  ret = mq_receive(mq, (char *)&rtt_stat, sizeof(rtt_stat), NULL);
  if (ret != sizeof(rtt_stat)) {
    if (ret < 0) {
      if (errno == EBADF)
        printf("terminating mq_receive\n");
      else
        perror("mq_receive failed");
    } else
      fprintf(stderr, "mq_receive returned invalid length %d\n", ret);
    break;
  }

  if (repeater) {
    printf("%ld\n", rxcount);
  } else {
    rtt_avg = ((rtt_stat.rtt_sum - rtt_stat.rtt_sum_last) / 
               rtt_stat.counts_per_sec);
    printf("%ld %ld %ld %ld %ld %ld\n", rxcount, 
           (long)(rtt_stat.rtt / 1000), (long)(rtt_avg / 1000), 
           (long)(rtt_stat.rtt_min / 1000), 
           (long)(rtt_stat.rtt_max / 1000), 
           overruns);
  }
}

/* This call also leaves primary mode, required for socket cleanup. */
printf("shutting down\n");

/* Important: First close the sockets! */
close(rxsock);
close(txsock);
pthread_join(txthread, NULL);
pthread_cancel(rxthread);
pthread_join(rxthread, NULL);
return 0;

failure4:
  pthread_cancel(rxthread);
pthread_join(rxthread, NULL);
failure3:
  mq_close(mq);
failure2:
  close(txsock);
failure1:
  close(rxsock);
return 1;
9.4 cross-link.c

/*
 * cross-link.c
 *
 * Userspace test program (Xenomai alchemy skin) for RTDM-based UART drivers
 * Copyright 2005 by Joerg Langenberg <joergel75@gmx.net>
 * Updates by Jan Kiszka <jan.kiszka@web.de>
 *
 * 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., 675 Mass Ave, Cambridge, MA 02139, USA.
 */
#include <stdio.h>
#include <signal.h>
#include <unistd.h>
#include <sys/mman.h>
#include <alchemy/task.h>
#include <alchemy/timer.h>
#include <rtdm/serial.h>
#define MAIN_PREFIX "main : ">
#define WTASK_PREFIX "write_task: ">
#define RTASK_PREFIX "read_task: ">
#define WRITE_FILE "/dev/rtdm/rtser0"
#define READ_FILE "/dev/rtdm/rtser1"
int read_fd = -1;
int write_fd = -1;
unsigned int read_state = 0;
unsigned int write_state = 0;
static const struct rtser_config read_config = {
    .config_mask = 0xFFFF,
    .baud_rate = 115200,
    .parity = RTSER_DEF_PARITY,
    .data_bits = RTSER_DEF_BITS,
    .stop_bits = RTSER_DEF_STOPB,
    .handshake = RTSER_DEF_HAND,
    .fifo_depth = RTSER_DEF_FIFO_DEPTH,
    .rx_timeout = RTSER_DEF_TIMEOUT,
    .tx_timeout = RTSER_DEF_TIMEOUT,
    .event_timeout = 1000000000, /* 1 s */
    .timestamp_history = RTSER_RX_TIMESTAMP_HISTORY,
    .event_mask = RTSER_EVENT_RXPEND,
};
static const struct rtser_config write_config = {
    .config_mask = RTSER_SET_BAUD | RTSER_SET_TIMESTAMP_HISTORY,
    .baud_rate = 115200,
    .timestamp_history = RTSER_DEF_TIMESTAMP_HISTORY,
    /* the rest implicitly remains default */
};
static int close_file( int fd, char *name)
{
    int err, i=0;
    do {
        i++;
        err = close(fd);
        switch (err) {
        case -EAGAIN:
            printf(MAIN_PREFIX "%s -> EAGAIN (%d times)\n", ",
        default:
            return -1;
        }
    }

Generated by Doxygen
/**
 * Example Documentation
 */

void rt_task_sleep(unsigned int ms)
{
    rt_task_sleep((ms * 1000)); /* wait $ms *ms */
    break;
}

case 0:
    printf(MAIN_PREFIX "%s -> closed\n", name);
    break;
default:
    printf(MAIN_PREFIX "%s -> %s\n", name, strerror(errno));
    break;
}
} while (err == -EAGAIN && i < 10);

return err;
}

static void cleanup_all(void)
{
    if (read_state & STATE_FILE_OPENED) {
        close_file(read_fd, READ_FILE " (read)");
        read_state &= ~STATE_FILE_OPENED;
    }

    if (write_state & STATE_FILE_OPENED) {
        close_file(write_fd, WRITE_FILE " (write)";
        write_state &= ~STATE_FILE_OPENED;
    }

    if (write_state & STATE_TASK_CREATED) {
        printf(MAIN_PREFIX "delete write_task\n");
        rt_task_delete(&write_task);
        write_state &= ~STATE_TASK_CREATED;
    }

    if (read_state & STATE_TASK_CREATED) {
        printf(MAIN_PREFIX "delete read_task\n");
        rt_task_delete(&read_task);
        read_state &= ~STATE_TASK_CREATED;
    }
}

static void catch_signal(int sig)
{
    cleanup_all();
    printf(MAIN_PREFIX "exit\n");
    return;
}

static void write_task_proc(void *arg)
{
    int err;
    RTIME write_time;
    ssize_t sz = sizeof(RTIME);
    int written = 0;

    err = rt_task_set_periodic(NULL, TM_NOW,
        rt_timer_ns2ticks(write_task_period_ns));
    if (err) {
        printf(WTASK_PREFIX "error on set periodic, %s\n",
            strerror(-err));
        goto exit_write_task;
    }

    while (1) {
        err = rt_task_wait_period(NULL);
        if (err) {
            printf(WTASK_PREFIX "error on rt_task_wait_period, %s\n",
                strerror(-err));
            break;
        }
        write_time = rt_timer_read();

        written = write(write_fd, &write_time, sz);
        if (written < 0 ) {
            printf(WTASK_PREFIX "error on write, %s\n",
                strerror(errno));
            break;
        } else if (written != sz) {
            printf(WTASK_PREFIX "only %d / %zd byte transmitted\n",
                written, sz);
            break;
        }
    }

exit_write_task:

if ((write_state & STATE_FILE_OPENED) &&
close_file(write_fd, WRITE_FILE " (write") == 0)
write_state &= ~STATE_FILE_OPENED;

printf(WTASK_PREFIX "exit\n");
}

static void read_task_proc(void *arg)
{
int err;
int nr = 0;
RTIME read_time = 0;
RTIME write_time = 0;
RTIME irq_time = 0;
ssize_t sz = sizeof(RTIME);
int rd = 0;
struct rtser_event rx_event;

printf(" Nr | write->irq | irq->read | write->read |\n");
printf("-----------------------------------------------------------\n");

"* We are in secondary mode now due to printf, the next
* blocking Xenomai or driver call will switch us back
* (here: RTSER_RTIOC_WAIT_EVENT).
*/
while (1) {
    /* waiting for event */
    err = ioctl(read_fd, RTSER_RTIOC_WAIT_EVENT, &rx_event);
    if (err) {
        printf(RTASK_PREFIX "error on RTSER_RTIOC_WAIT_EVENT, %s\n",
               strerror(errno));
        if (err == -ETIMEDOUT)
            continue;
        break;
    }
    irq_time = rx_event.rxpend_timestamp;
    rd = read(read_fd, &write_time, sz);
    if (rd == sz) {
        read_time = rt_timer_read();
        printf("%3d |%16llu |%16llu |%16llu\n", nr,
               irq_time - write_time,
               read_time - irq_time,
               read_time - write_time);
        nr++;
    } else if (rd < 0 ) {
        printf(RTASK_PREFIX "error on read, code %s\n",
               strerror(errno));
        break;
    } else {
        printf(RTASK_PREFIX "only %d / %zd byte received \n", rd, sz);
        break;
    }
}

if ((read_state & STATE_FILE_OPENED) &&
close_file(read_fd, READ_FILE " (read") == 0)
read_state &= ~STATE_FILE_OPENED;

printf(RTASK_PREFIX "exit\n");
}

int main(int argc, char* argv[])
{
int err = 0;
signal(SIGTERM, catch_signal);
signal(SIGINT, catch_signal);

"* open rtser */
write_fd = open( WRITE_FILE, 0);
if (write_fd < 0) {
    printf(MAIN_PREFIX "can't open %s (write), %s\n",
           WRITE_FILE, strerror(errno));
    goto error;
}
write_state |= STATE_FILE_OPENED;
printf(MAIN_PREFIX "write-file opened\n");

/* writing config */
err = ioctl(write_fd, RTSER_RTIOC_SET_CONFIG, &write_config);
if (err) {
    printf(MAIN_PREFIX "error while RTSER_RTIOC_SET_CONFIG, %s\n",
           strerror(errno));
    goto exit:
}

exit:
return 0;
}

Generated by Doxygen
    strerror(errno));
    goto error;
}

/* open rtser1 */
read_fd = open(READ_FILE, 0);
if (read_fd < 0) {
    printf(MAIN_PREFIX "can’t open %s (read), %s\n", READ_FILE, 
           strerror(errno));
    goto error;
}
read_state |= STATE_FILE_OPENED;
printf(MAIN_PREFIX "read-file opened\n");
/* writing read-config */
err = ioctl(read_fd, RTSER_RTIOC_SET_CONFIG, &read_config);
if (err) {
    printf(MAIN_PREFIX "error while ioctl, %s\n", 
           strerror(errno));
    goto error;
}
printf(MAIN_PREFIX "read-config written\n");
/* create write_task */
err = rt_task_create(&write_task, "write_task", 0, 50, 0);
if (err) {
    printf(MAIN_PREFIX "failed to create write_task, %s\n", 
           strerror(-err));
    goto error;
}
write_state |= STATE_TASK_CREATED;
printf(MAIN_PREFIX "write-task created\n");
/* start write_task */
printf(MAIN_PREFIX "starting write-task\n");
err = rt_task_start(&write_task, &write_task_proc, NULL);
if (err) {
    printf(MAIN_PREFIX "failed to start write_task, %s\n", 
           strerror(-err));
    goto error;
}
/* create read_task */
err = rt_task_create(&read_task, "read_task", 0, 51, 0);
if (err) {
    printf(MAIN_PREFIX "failed to create read_task, %s\n", 
           strerror(-err));
    goto error;
}
read_state |= STATE_TASK_CREATED;
printf(MAIN_PREFIX "read-task created\n");
/* start read_task */
printf(MAIN_PREFIX "starting read-task\n");
err = rt_task_start(&read_task, &read_task_proc, NULL);
if (err) {
    printf(MAIN_PREFIX "failed to start read_task, %s\n", 
           strerror(-err));
    goto error;
}
for (;;) 
    pause();
return 0;
error:
    cleanup_all();
    return err;
}

9.5  iddp-label.c

/*
 * Copyright (C) 2009 Philippe Gerum <rpm@xenomai.org>.
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2 of the License, or (at your option) any later version.
*/
This library 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 
Lesser General Public License for more details.

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

IDDP-based client/server demo, using the write(2)/recvfrom(2) 
system calls to exchange data over a socket.

In this example, two sockets are created. A server thread (reader) 
is bound to a labeled real-time port and receives datagrams sent to 
this port from a client thread (writer). The client thread attaches 
to the port opened by the server using a labeled connection 
request. The client socket is bound to a different port, only to 
provide a valid peer name; this is optional.

ASCII labels can be attached to bound ports, in order to connect 
sockets to them in a more descriptive way than using plain numeric 
port values.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <string.h>
#include <pthread.h>
#include <errno.h>
#include <rtdm/ipc.h>
```

```
pthread_t svtid, cltid;
#define IDDP_CLPORT 27
#define IDDP_PORT_LABEL "iddp-demo"
```

```
static const char *msg[] = {
  "Surfing With The Alien",
  "Lords of Karma",
  "Banana Mango",
  "Psycho Monkey",
  "Luminous Flesh Giants",
  "Moroccan Sunset",
  "Satch Boogie",
  "Flying In A Blue Dream",
  "Ride",
  "Summer Song",
  "Speed Of Light",
  "Crystal Planet",
  "Raspberry Jam Delta-V",
  "Champagne?",
  "Clouds Race Across The Sky",
  "Engines Of Creation"
};
```

```
static void fail(const char *reason) 
{
  perror(reason);
  exit(EXIT_FAILURE);
}
```

```
static void *server(void *arg)
{
    struct sockaddr_ipc saddr, claddr;
    struct rtipc_port_label plabel;
    socklen_t addrlen;
    char buf[128];
    int ret, s;

    s = socket(AF_RTPC, SOCK_DGRAM, IPCPROTO_IDDP);
    if (s < 0)
        fail("socket");
    /*
    * We will use Xenomai's system heap for datagram, so no
    * IDDP_POOLSZ required here.
    */
    /*
    * Set a port label. This name will be registered when
    * binding, in addition to the port number (if given).
    */
    strcpy(plabel.label, IDDP_PORT_LABEL);
    ```
ret = setsockopt(s, SOL_IDDP, IDDP_LABEL, &plabel, sizeof(plabel));

if (ret)
    fail("setsockopt");

/*
 * Bind the socket to the port. Assign that port a label, so
 * that peers may use a descriptive information to locate
 * it. Labeled ports will appear in the
 * /proc/xenomai/registry/rtipc/iddp directory once the socket
 * is bound.
 *
 * saddr.sipc_port specifies the port number to use. If -1 is
 * passed, the IDDP driver will auto-select an idle port.
 */

saddr.sipc_family = AF_RTIPC;
saddr.sipc_port = -1; /* Pick next free */
ret = bind(s, (struct sockaddr *)&saddr, sizeof(saddr));
if (ret)
    fail("bind");

for (;;) {
    addrlen = sizeof(saddr);
    ret = recvfrom(s, buf, sizeof(buf), 0, (struct sockaddr *)&claddr, &addrlen);
    if (ret < 0) {
        close(s);
        fail("recvfrom");
    } else {
        printf("%s: received %d bytes, \"%.*s\" from port %d
", __FUNCTION__, ret, ret, buf, claddr.sipc_port);
    }
}

return NULL;

static void *client(void *arg)
{
    struct sockaddr_ipc svsaddr, clsaddr;
    struct rtipc_port_label plabel;
    int ret, s, n = 0, len;
    struct timespec ts;

    s = socket(AF_RTIPC, SOCK_DGRAM, IPCPROTO_IDDP);
    if (s < 0)
        fail("socket");

    /*
    * Set a name on the client socket. This is strictly optional,
    * and only done here for the purpose of getting back a
    * different port number in recvfrom().
    */
    clsaddr.sipc_family = AF_RTIPC;
    clsaddr.sipc_port = IDDP_CLPORT;
    ret = bind(s, (struct sockaddr *)&clsaddr, sizeof(clsaddr));
    if (ret)
        fail("bind");

    /*
    * Set the port label. This name will be used to find the peer
    * when connecting, instead of the port number. The label must
    * be set _after_ the socket is bound to the port, so that
    * IDDP does not try to register this label for the client
    * port as well (like the server thread did).
    */
    strcpy(plabel.label, IDDP_PORT_LABEL);
    ret = setsockopt(s, SOL_IDDP, IDDP_LABEL, &plabel, sizeof(plabel));
    if (ret)
        fail("setsockopt");

    memset(&svsaddr, 0, sizeof(svsaddr));
    svsaddr.sipc_family = AF_RTIPC;
    svsaddr.sipc_port = -1; /* Tell IDDP to search by label. */
    ret = connect(s, (struct sockaddr *)&svsaddr, sizeof(svsaddr));
    if (ret)
        fail("connect");

    for (;;) {
        len = strlen(msg[n]);
        /* Send to default destination we connected to. */
        ret = write(s, msg[n], len);
        if (ret < 0) {
            close(s);
            fail("sendto");
        }
        printf("%s: sent %d bytes, \"%.*s\"\n", __FUNCTION__, ret, ret, buf, 0);
    }
    return NULL;
}
__FUNCTION__, ret, ret, msg[n]);
/*
 * We run in full real-time mode (i.e., primary node),
 * so we have to let the system breathe between two
 * iterations.
 */
      ts.tv_sec = 0;
      ts.tv_nsec = 500000000; /* 500 ms */
      clock_nanosleep(CLOCK_REALTIME, 0, &ts, NULL);
}

return NULL;
}

int main(int argc, char **argv)
{
  struct sched_param svparam = {.sched_priority = 71 };  
  struct sched_param clparam = {.sched_priority = 70 };  
  pthread_attr_t svattr, clattr;
  sigset_t set;
  int sig;
  sigemptyset(&set);
  sigaddset(&set, SIGINT);
  sigaddset(&set, SIGTERM);
  sigaddset(&set, SIGHUP);
  pthread_sigmask(SIG_BLOCK, &set, NULL);
  pthread_attr_init(&svattr);
  pthread_attr_setdetachstate(&svattr, PTHREAD_CREATE_JOINABLE);
  pthread_attr_setschedpolicy(&svattr, SCHED_FIFO);
  pthread_attr_setschedparam(&svattr, &svparam);
  errno = pthread_create(&svtid, &svattr, &server, NULL);
  if (errno)
      fail("pthread_create");
  pthread_attr_init(&clattr);
  pthread_attr_setdetachstate(&clattr, PTHREAD_CREATE_JOINABLE);
  pthread_attr_setschedpolicy(&clattr, SCHED_FIFO);
  pthread_attr_setschedparam(&clattr, &clparam);
  errno = pthread_create(&cltid, &clattr, &client, NULL);
  if (errno)
      fail("pthread_create");
  sigwait(&set, &sig);
  pthread_cancel(svtid);
  pthread_join(svtid, NULL);
  pthread_cancel(cltid);
  pthread_join(cltid, NULL);
  return 0;
}

9.6  iddp-sendrecv.c

/*
 * Copyright (C) 2009 Philippe Gerum <rpm@xenomai.org>.
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2 of the License, or (at your option) any later version.
 * This library 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
 * Lesser General Public License for more details.
 * You should have received a copy of the GNU Lesser General Public
 * License along with this library; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.
 *
 * IDDP-based client/server demo, using the sendto(2)/recvfrom(2)
 * system calls to exchange data over a socket.
 */
In this example, two sockets are created. A server thread (reader) is bound to a real-time port and receives datagrams sent to this port from a client thread (writer). The client socket is bound to a different port, only to provide a valid peer name; this is optional.

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <string.h>
#include <pthread.h>
#include <errno.h>
#include <rtdm/ipc.h>

pthread_t svtid, cltid;
#define IDDP_SVPORT 12
#define IDDP_CLPORT 13

static const char *msg[] = {
    "Surfing With The Alien",
    "Lords of Karma",
    "Banana Mango",
    "Psycho Monkey",
    "Luminous Flesh Giants",
    "Moroccan Sunset",
    "Satch Boogie",
    "Flying In A Blue Dream",
    "Ride",
    "Summer Song",
    "Speed Of Light",
    "Crystal Planet",
    "Raspberry Jam Delta-V",
    "Champagne",
    "Clouds Race Across The Sky",
    "Engines Of Creation"
};

static void fail(const char *reason)
{
    perror(reason);
    exit(EXIT_FAILURE);
}

static void *server(void *arg)
{
    struct sockaddr_ipc saddr, claddr;
    socklen_t addrlen;
    char buf[128];
    size_t poolsz;
    int ret, s;

    s = socket(AF_RTIPC, SOCK_DGRAM, IPCPROTO_IDDP);
    if (s < 0)
        fail("socket");
    
    /*
     * Set a local 32k pool for the server endpoint. Memory needed to convey datagrams will be pulled from this pool, instead of Xenomai’s system pool.
     */
    poolsz = 32768; /* bytes */
    ret = setsockopt(s, SOL_IDDP, IDDP_POOLSZ,
                     &poolsz, sizeof(poolsz));
    if (ret)
        fail("setsockopt");

    saddr.sipc_family = AF_RTIPC;
    saddr.sipc_port = IDDP_SVPORT;
    ret = bind(s, (struct sockaddr *)&saddr, sizeof(saddr));
    if (ret)
        fail("bind");

    for (;;) {
        addrlen = sizeof(saddr);
        ret = recvfrom(s, buf, sizeof(buf), 0,
                        (struct sockaddr *)&claddr, &addrlen);
        if (ret < 0) {
            close(s);
            fail("recvfrom");
        }
        printf("%s: received %d bytes, \"%s\" from port %d\n",
               __FUNCTION__, ret, ret, buf, claddr.sipc_port);
    }

    return NULL;
```
static void *client(void *arg)
{
    struct sockaddr_ipc svsaddr, clsaddr;
    int ret, s, n = 0, len;
    struct timespec ts;
    
    s = socket(AF_RTIPC, SOCK_DGRAM, IPCPROTO_IDDP);
    if (s < 0)
        fail("socket");
    clsaddr.sipc_family = AF_RTIPC;
    clsaddr.sipc_port = IDDP_CLPORT;
    ret = bind(s, (struct sockaddr *)&clsaddr, sizeof(clsaddr));
    if (ret)
        fail("bind");
    svsaddr.sipc_family = AF_RTIPC;
    svsaddr.sipc_port = IDDP_SVPORT;
    for (;;) {
        len = strlen(msg[n]);
        ret = sendto(s, msg[n], len, 0,
                     (struct sockaddr *)&svsaddr, sizeof(svsaddr));
        if (ret < 0) {
            close(s);
            fail("sendto");
        }
        printf("%s: sent %d bytes, \n", __FUNCTION__, ret, ret, msg[n]);
        n = (n + 1) % (sizeof(msg) / sizeof(msg[0]));
        /*
         * We run in full real-time mode (i.e. primary mode),
         * so we have to let the system breathe between two
         * iterations.
         */
        ts.tv_sec = 0;
        ts.tv_nsec = 500000000; /* 500 ms */
        clock_nanosleep(CLOCK_REALTIME, 0, &ts, NULL);
    }
    return NULL;
}

int main(int argc, char **argv)
{
    struct sched_param svparam = {.sched_priority = 71};
    struct sched_param cliparam = {.sched_priority = 70};
    pthread_attr_t svattr, clattr;
    sigset_t set;
    int sig;
    sigemptyset(&set);
    sigaddset(&set, SIGINT);
    sigaddset(&set, SIGTERM);
    sigaddset(&set, SIGHUP);
    pthread_sigmask(SIG_BLOCK, &set, NULL);
    pthread_attr_init(&svattr);
    pthread_attr_setdetachstate(&svattr, PTHREAD_CREATE_JOINABLE);
    pthread_attr_setinheritsched(&svattr, PTHREAD_EXPLICIT_SCHED);
    pthread_attr_setschedpolicy(&svattr, SCHED_FIFO);
    pthread_attr_setschedparam(&svattr, &svparam);
    errno = pthread_create(&svtid, &svattr, &server, NULL);
    if (errno)
        fail("pthread_create");
    pthread_attr_init(&clattr);
    pthread_attr_setdetachstate(&clattr, PTHREAD_CREATE_JOINABLE);
    pthread_attr_setinheritsched(&clattr, PTHREAD_EXPLICIT_SCHED);
    pthread_attr_setschedpolicy(&clattr, SCHED_FIFO);
    pthread_attr_setschedparam(&clattr, &clparam);
    errno = pthread_create(&cltid, &clattr, &client, NULL);
    if (errno)
        fail("pthread_create");
    sigwait(&set, &sig);
    pthread_cancel(svtid);
    pthread_cancel(cltid);
    pthread_join(svtid, NULL);
    pthread_join(cltid, NULL);
    return 0;
}
/
* Program to configuring the CAN controller
* Copyright (C) 2006 Wolfgang Grandegger <wg@grandegger.com>
* Copyright (C) 2005, 2006 Sebastian Smolorz
* <Sebastian.Smolorz@stud.uni-hannover.de>
* 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., 675 Mass Ave, Cambridge, MA 02139, USA.
*/

#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>
#include <string.h>
#include <time.h>
#include <errno.h>
#include <getopt.h>
#include <sys/mman.h>
#include <rtdm/can.h>

static void print_usage(char *prg)
{
    fprintf(stderr,
"Usage: %s <can-interface> [Options] [up|down|start|stop|sleep]\n"
"Options:\n"
" -v, --verbose be verbose\n"
" -h, --help this help\n"
" -c, --ctrlmode=CTRLMODE listenonly, loopback or none\n"
" -b, --baudrate=BPS baudrate in bits/sec\n"
" -B, --bittime=BTR0:BTR1 BTR or standard bit-time\n"
" -b, --bittime=BRP:PROP_SEG:PHASE_SEG1:PHASE_SEG2:SJW:SAM\n",
prg);
}

static can_baudrate_t string_to_baudrate(char *str)
{
    can_baudrate_t baudrate;
    if (sscanf(str, "%i", &baudrate) != 1)
        return -1;
    return baudrate;
}

static int string_to_mode(char *str)
{
    if ( !strcmp(str, "up") || !strcmp(str, "start") )
        return CAN_MODE_START;
    else if ( !strcmp(str, "down") || !strcmp(str, "stop") )
        return CAN_MODE_STOP;
    else if ( !strcmp(str, "sleep") )
        return CAN_MODE_SLEEP;
    return -EINVAL;
}

static int string_to_ctrlmode(char *str)
{
    if ( !strcmp(str, "listenonly") )
        return CAN_CTRLMODE_LISTENONLY;
    else if ( !strcmp(str, "loopback") )
        return CAN_CTRLMODE_LOOPBACK;
    else if ( !strcmp(str, "none") )
        return 0;
    return -1;
}

int main(int argc, char *argv[])
{

char ifname[16];
int can_fd = -1;
int new_baudrate = -1;
int new_mode = -1;
int new_ctrlmode = 0, set_ctrlmode = 0;
int verbose = 0;
bittime_count = 0, bittime_data[6];
struct can_ifreq ifr;
struct can_bittime *bittime;
int opt, ret;
char* ptr;

struct option long_options[] = {
    { "help", no_argument, 0, 'h' },
    { "verbose", no_argument, 0, 'v' },
    { "baudrate", required_argument, 0, 'b' },
    { "bittime", required_argument, 0, 'B' },
    { "ctrlmode", required_argument, 0, 'c' },
    { 0, 0, 0, 0 },
};

while ((opt = getopt_long(argc, argv, "hvb:B:c:",
                           long_options, NULL)) != -1) {
    switch (opt) {
    case 'h':
        print_usage(argv[0]);
        exit(0);
        break;
    case 'v':
        verbose = 1;
        break;
    case 'b':
        new_baudrate = string_to_baudrate(optarg);
        if (new_baudrate == -1) {
            print_usage(argv[0]);
            exit(0);
        }
        break;
    case 'B':
        ptr = optarg;
        while (1) {
            bittime_data[bittime_count++] = strtoul(ptr, NULL, 0);
            if (!(ptr = strchr(ptr, ':')))
                break;
            ptr++;
        }
        if (bittime_count != 2 && bittime_count != 6) {
            print_usage(argv[0]);
            exit(0);
        }
        break;
    case 'c':
        ret = string_to_ctrlmode(optarg);
        if (ret == -1) {
            print_usage(argv[0]);
            exit(0);
        }
        new_ctrlmode |= ret;
        set_ctrlmode = 1;
        break;
    break;
    case 'B':
        ptr = optarg;
        while (1) {
            bittime_data[bittime_count++] = strtoul(ptr, NULL, 0);
            if (!strchr(ptr, ':'))
                break;
            ptr++;
        }
        if (bittime_count != 2 && bittime_count != 6) {
            print_usage(argv[0]);
            exit(0);
        }
        break;
    default:
        fprintf(stderr, "Unknown option %c\n", opt);
        exit(0);
        break;
    }
/* Get CAN interface name */
if (optind == argc - 1 && optind != argc - 2) {
    print_usage(argv[0]);
    return 0;
}
strncpy(ifname, argv[optind], IFNAMSIZ);
strncpy(ifr.ifr_name, ifname, IFNAMSIZ);
if (optind == argc - 2) { /* Get mode setting */
    new_mode = string_to_mode(argv[optind + 1]);
    if (verbose)
        printf("mode: %s (%#x)\n", argv[optind + 1], new_mode);
    if (new_mode < 0) {
        print_usage(argv[0]);
    }
return 0;
}
}
can_fd = socket(PF_CAN, SOCK_RAW, CAN_RAW);
if (can_fd < 0) {
    fprintf(stderr, "Cannot open RTDM CAN socket. Maybe driver not loaded? \n");
    return can_fd;
}
ret = ioctl(can_fd, SIOCGIFINDEX, &ifr);
if (ret) {
    fprintf(stderr,"Can't get interface index for %s, errno = %d", ifname, errno);
    return ret;
}
if (new_baudrate != -1) {
    if (verbose)
        printf("baudrate: %d\n", new_baudrate);
    ifr.ifr_ifru.baudrate = new_baudrate;
    ret = ioctl(can_fd, SIOCSCANBAUDRATE, &ifr);
    if (ret) {
        goto abort;
    }
}
if (bittime_count) {
    bittime = &ifr.ifr_ifru.bittime;
    if (bittime_count == 2) {
        bittime->type = CAN_BITTIME_BTR;
        bittime->btr.btr0 = bittime_data[0];
        bittime->btr.btr1 = bittime_data[1];
        if (verbose)
            printf("bit-time: btr0=0x%02x btr1=0x%02x\n", bittime->btr.btr0, bittime->btr.btr1);
    } else {
        bittime->type = CAN_BITTIME_STD;
        bittime->std.brp = bittime_data[0];
        bittime->std.prop_seg = bittime_data[1];
        bittime->std.phase_seg1 = bittime_data[2];
        bittime->std.phase_seg2 = bittime_data[3];
        bittime->std.sjw = bittime_data[4];
        bittime->std.sam = bittime_data[5];
        if (verbose)
            printf("bit-time: brp=%d prop_seg=%d phase_seg1=%d "
                "phase_seg2=%d sjw=%d sam=%d\n",
                bittime->std.brp,
                bittime->std.prop_seg,
                bittime->std.phase_seg1,
                bittime->std.phase_seg2,
                bittime->std.sjw,
                bittime->std.sam);
    }
    ret = ioctl(can_fd, SIOCSCANCUSTOMBITTIME, &ifr);
    if (ret) {
        goto abort;
    }
}
if (set_ctrlmode != 0) {
    ifr.ifr_ifru.ctrlmode = new_ctrlmode;
    if (verbose)
        printf("ctrlmode: %#x\n", new_ctrlmode);
    ret = ioctl(can_fd, SIOCSCANCTRLMODE, &ifr);
    if (ret) {
        goto abort;
    }
}
if (new_mode != -1) {
    ifr.ifr_ifru.mode = new_mode;
    ret = ioctl(can_fd, SIOCSCANMODE, &ifr);
    if (ret) {
        goto abort;
    }
}
close(can_fd);
return 0;
abort:
close(can_fd);
return ret;
# include <stdio.h>
# include <stdlib.h>
# include <signal.h>
# include <unistd.h>
# include <time.h>
# include <errno.h>
# include <getopt.h>

# include <alchemy/task.h>
# include <rtdm/can.h>

static void print_usage(char *prg)
{
    fprintf(stderr,
            "Usage: %s [can-interface] [Options]\n"       
            "Options:\n"       
            "  -f --filter=id:mask... apply filter\n"       
            "  -e --error=mask receive error messages\n"       
            "  -t, --timeout=MS timeout in ms\n"       
            "  -T, --timestamp with absolute timestamp\n"       
            "  -R, --timestamp-rel with relative timestamp\n"       
            "  -v, --verbose be verbose\n"       
            "  -p, --print=MODULO print every MODULO message\n"       
            "  -h, --help this help\n","  prg);
}

typedef int optind, opterr, optopt;

static int s = -1, verbose = 0, print = 1;
static nanosecs_rel_t timeout = 0, with_timestamp = 0, timestamp_rel = 0;

RT_TASK rt_task_desc;

#define BUF_SIZ 255
#define MAX_FILTER 16

struct sockaddr_can recv_addr;
struct can_filter recv_filter[MAX_FILTER];
static int filter_count = 0;

static int add_filter(u_int32_t id, u_int32_t mask)
{
    if (filter_count >= MAX_FILTER)
        return -1;
    recv_filter[filter_count].can_id = id;
    recv_filter[filter_count].can_mask = mask;
    printf("Filter #%d: id=0x%08x mask=0x%08x\n", filter_count, id, mask);
    filter_count++;
    return 0;
}

static void cleanup(void)
{
    int ret;
    if (verbose)
        printf("Cleaning up...\n");

    if (s == -1) {
        ret = close(s);
        s = -1;
    }
    if (ret) {
        fprintf(stderr, "close: %s\n", strerror(-ret));
        exit(EXIT_SUCCESS);
    }
}

static void cleanup_and_exit(int sig)
{
    if (verbose)
        printf("Signal %d received\n", sig);
    cleanup();
    exit(0);
}

static void rt_task(void)
{
    int i, ret, count = 0;
    struct can_frame frame;
}
struct sockaddr_can addr;
socklen_t addrlen = sizeof(addr);
struct msghdr msg;
struct iovec iov;
nanosecs_abs_t timestamp, timestamp_prev = 0;

if (with_timestamp) {
    msg.msg_iov = &iov;
    msg.msg_iovlen = 1;
    msg.msg_name = (void *)&addr;
    msg.msg_namelen = sizeof(struct sockaddr_can);
    msg.msg_control = (void *)&timestamp;
    msg.msg_controllen = sizeof(nanosecs_abs_t);
}

while (1) {
    if (with_timestamp) {
        iov.iov_base = (void *)&frame;
        iov.iov_len = sizeof(can_frame_t);
        ret = recvmsg(s, &msg, 0);
    } else
        ret = recvfrom(s, (void *)&frame, sizeof(can_frame_t), 0,
                      (struct sockaddr *)&addr, &addrlen);
    if (ret < 0) {
        switch (ret) {
        case -ETIMEDOUT:
            if (verbose)
                printf("recv: timed out");
            continue;
        case -EBADF:
            if (verbose)
                printf("recv: aborted because socket was closed");
            break;
        default:
            fprintf(stderr, "recv: \%s\n", strerror(-ret));
            break;
        }
    } else
        count++;
}

int main(int argc, char **argv)
{
    int opt, ret;
    u_int32_t id, mask;
    u_int32_t err_mask = 0;
    struct can_ifreq ifr;
    char *ptr;
    char name[32];

    struct option long_options[] = {
        { "help", no_argument, 0, 'h' },
        { "verbose", no_argument, 0, 'v' },
        { "filter", required_argument, 0, 'f' },
    },
while ((opt = getopt_long(argc, argv, "hve:t:p:RT", long_options, NULL)) != -1) {
    switch (opt) {
    case 'h':
        print_usage(argv[0]);
        exit(0);
        break;
    case 'p':
        print = strtoul(optarg, NULL, 0);
        break;
    case 'v':
        verbose = 1;
        break;
    case 'e':
        err_mask = strtoul(optarg, NULL, 0);
        break;
    case 'f':
        ptr = optarg;
        while (1) {
            id = strtoul(ptr, NULL, 0);
            ptr = strchr(ptr, ':');
            if (!ptr) {
                fprintf(stderr, "filter must be applied in the form id:mask[:id:mask]...\n"");
                exit(1);
            }
            ptr++;
            mask = strtoul(ptr, NULL, 0);
            ptr = strchr(ptr, ':');
            add_filter(id, mask);
            if (!ptr)
                break;
            ptr++;
        }
        break;
    case 't':
        timeout = (nanosecs_rel_t)strtoul(optarg, NULL, 0) * 1000000;
        break;
    case 'R':
        timestamp_rel = 1;
    case 'T':
        with_timestamp = 1;
        break;
    default:
        fprintf(stderr, "Unknown option %c\n", opt);
        break;
    }
}
ret = socket(PF_CAN, SOCK_RAW, CAN_RAW);
if (ret < 0) {
    fprintf(stderr, "socket: %s\n", strerror(-ret));
    return -1;
}
s = ret;
if (argv[optind] == NULL) {
    if (verbose)
        printf("interface all\n");
    ifr.ifr_ifindex = 0;
} else {
    if (verbose)
        printf("interface %s\n", argv[optind]);
    strncpy(ifr.ifr_name, argv[optind], IFNAMSIZ);
    if (verbose)
        printf("s=%d, ifr_name=%s\n", s, ifr.ifr_name);
    ret = ioctl(s, SIOCGETINDEX, &ifr);
    if (ret < 0) {
        fprintf(stderr, "ioctl GET_IFINDEX: %s\n", strerror(-ret));
    } else {
        if (verbose)
            printf("interface %s\n", argv[optind]);
    }
goto failure;
}
}

if (err_mask) {
    ret = setsockopt(s, SOL_CAN_RAW, CAN_RAW_ERR_FILTER,
                    &err_mask, sizeof(err_mask));
    if (ret < 0) {
        fprintf(stderr, "setsockopt: %s\n", strerror(-ret));
        goto failure;
    }
    if (verbose)
        printf("Using err_mask=%#x\n", err_mask);
}

if (filter_count) {
    ret = setsockopt(s, SOL_CAN_RAW, CAN_RAW_FILTER,
                     &recv_filter, filter_count * sizeof(struct can_filter));
    if (ret < 0) {
        fprintf(stderr, "setsockopt: %s\n", strerror(-ret));
        goto failure;
    }
    recv_addr.can_family = AF_CAN;
    recv_addr.can_ifindex = ifr.ifr_ifindex;
    ret = bind(s, (struct sockaddr *)&recv_addr,
               sizeof(struct sockaddr_can));
    if (ret < 0) {
        fprintf(stderr, "bind: %s\n", strerror(-ret));
        goto failure;
    }
    if (timeout) {
        if (verbose)
            printf("Timeout: %lld ns\n", (long long)timeout);
        ret = ioctl(s, RTCAN_RTIOC_RCV_TIMEOUT, &timeout);
        if (ret) {
            fprintf(stderr, "ioctl RCV_TIMEOUT: %s\n", strerror(-ret));
            goto failure;
        }
    }
    if (with_timestamp) {
        ret = ioctl(s, RTCAN_RTIOC_TAKE_TIMESTAMP,
                    RTCAN_TAKE_TIMESTAMPS);
        if (ret) {
            fprintf(stderr, "ioctl TAKE_TIMESTAMP: %s\n", strerror(-ret));
            goto failure;
        }
    }
    snprintf(name, sizeof(name), "rtcanrecv-%d", getpid());
    ret = rt_task_shadow(&rt_task_desc, name, 0, 0);
    if (ret) {
        fprintf(stderr, "rt_task_shadow: %s\n", strerror(-ret));
        goto failure;
    }
    rt_task();
    /* never returns */
}

9.9 rtcansend.c

#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>
#include <time.h>
#include <errno.h>
#include <getopt.h>
#include <alchemy/task.h>
#include <alchemy/timer.h>

Generated by Doxygen
```c
#include <rtdm/can.h>

extern int optind, opterr, optopt;

static void print_usage(char *prg)
{
    fprintf(stderr,
        "Usage: %s <can-interface> [Options] <can-msg>\n"<can-msg> can consist of up to 8 bytes given as a space separated list\n"
        "Options:\n"-i, --identifier=ID CAN Identifier (default = 1)\n"-r --rtr send remote request\n"-e --extended send extended frame\n"-l --loop=COUNT send message COUNT times\n"-c, --count message count in data\n"-d, --delay=MS delay in ms (default = 1ms)\n"-s, --send use send instead of sendto\n"-t, --timeout=MS timeout in ms\n"-l, --loopback=0|1 switch local loopback off or on\n"-v, --verbose be verbose\n"-p, --print=MODULO print every MODULO message\n"-h, --help this help\n",
    prg);
}

RT_TASK rt_task_desc;

static int s=-1, dlc=0, rtr=0, extended=0, verbose=0, loops=1;
static SRTIME delay=1000000;
static int count=0, print=1, use_send=0, loopback=-1;
static nanosecs_rel_t timeout = 0;
static struct can_frame frame;
static struct sockaddr_can to_addr;

static void cleanup(void)
{
    int ret;

    if (verbose)
        printf("Cleaning up...\n");

    usleep(100000);

    if (s >= 0) {
        ret = close(s);
        s = -1;
        if (ret) {
            fprintf(stderr, "close: %s\n",
                    strerror(-ret));
            exit(EXIT_SUCCESS);
        }
    }

    if (verbose)
        printf("Signal %d received\n",
                sig);

    cleanup();
    exit(0);
}

static void rt_task(void)
{
    int i, j, ret;

    for (i = 0; i < loops; i++) {
        rt_task_sleep(rt_timer_ns2ticks(delay));

        if (count)
            memcpy(&frame.data[0], &i, sizeof(i));

        /* Note: sendto avoids the definiton of a receive filter list */
        if (use_send)
            ret = send(s, (void *)&frame, sizeof(can_frame_t), 0);
        else
            ret = sendto(s, (void *)&frame, sizeof(can_frame_t), 0,
                         (struct sockaddr *)&to_addr, sizeof(to_addr));

        if (ret < 0) {
            switch (ret) {
            case -ETIMEDOUT:
                if (verbose)
                    printf("send(to): timed out\n");
                break;
            case -EBADF:
                if (verbose)
                    printf("send(to): aborted because socket was closed\n");
                break;
            }
        }
    }
}
```

default:
    fprintf(stderr, "send: \n", strerror(-ret));
    break;
}
i = loops; /* abort */
break;
}
if (verbose && (i % print) == 0) {
    if (frame.can_id & CAN_EFF_FLAG)
        printf("\x00\x00\x00\x00", frame.can_id & CAN_EFF_MASK);
    else
        printf("\x00\x00\x00\x00", frame.can_id & CAN_SFF_MASK);
    printf(" \[%d\]", frame.can_dlc);
    for (j = 0; j < frame.can_dlc; j++) {
        printf(" %02x", frame.data[j]);
    }
    printf("\n");
}
}
}
int main(int argc, char **argv)
{
    int i, opt, ret;
    struct can_ifreq ifr;
    char name[32];

    struct option long_options[] = {
        { "help", no_argument, 0, 'h' },
        { "identifier", required_argument, 0, 'i' },
        { "rtr", no_argument, 0, 'r' },
        { "extended", no_argument, 0, 'e' },
        { "verbose", no_argument, 0, 'v' },
        { "count", no_argument, 0, 'c' },
        { "print", required_argument, 0, 'p' },
        { "loop", required_argument, 0, 'l' },
        { "delay", required_argument, 0, 'd' },
        { "send", no_argument, 0, 's' },
        { "timeout", required_argument, 0, 't' },
        { "loopback", required_argument, 0, 'L' },
        { 0, 0, 0, 0 },
    };
    signal(SIGTERM, cleanup_and_exit);
    signal(SIGINT, cleanup_and_exit);
    frame.can_id = 1;
    while ((opt = getopt_long(argc, argv, "hvi:l:red:t:cp:sL:",
        long_options, NULL)) != -1) {
        switch (opt) {
            case 'h':
                print_usage(argv[0]);
                exit(0);
            case 'p':
                print = strtoul(optarg, NULL, 0);
                break;
            case 'v':
                verbose = 1;
                break;
            case 'c':
                count = 1;
                break;
            case 'l':
                loops = strtoul(optarg, NULL, 0);
                break;
            case 'i':
                frame.can_id = strtoul(optarg, NULL, 0);
                break;
            case 'r':
                rtr = 1;
                break;
            case 'e':
                extended = 1;
                break;
            case 'd':
                delay = strtoul(optarg, NULL, 0) * 1000000LL;
                break;
            case 's':
        }
    }

Generated by Doxygen
use_send = 1;
break;

case 't':
    timeout = strtoul(optarg, NULL, 0) * 1000000LL;
    break;

case 'L':
    loopback = strtoul(optarg, NULL, 0);
    break;

default:
    fprintf(stderr, "Unknown option X%c\n", opt);
    break;
}
}
if (optind == argc) {
    print_usage(argv[0]);
    exit(0);
}
if (argv[optind] == NULL) {
    fprintf(stderr, "No Interface supplied\n");
    exit(-1);
}
if (verbose) {
    printf("interface %s\n", argv[optind]);
}
ret = socket(PF_CAN, SOCK_RAW, CAN_RAW);
if (ret < 0) {
    fprintf(stderr, "socket: %s\n", strerror(-ret));
    return -1;
}
s = ret;
if (loopback >= 0) {
    ret = setsockopt(s, SOL_CAN_RAW, CAN_RAW_LOOPBACK,
                     &loopback, sizeof(loopback));
    if (ret < 0) {
        fprintf(stderr, "setsockopt: %s\n", strerror(-ret));
        goto failure;
    }
    if (verbose)
        printf("Using loopback=%d\n", loopback);
}
strncpy(ifr.ifr_name, argv[optind], IFNAMSIZ);
if (verbose) {
    printf("s=%d, ifr_name=%s\n", s, ifr.ifr_name);
}
ret = ioctl(s, SIOCGIFINDEX, &ifr);
if (ret < 0) {
    fprintf(stderr, "ioctl: %s\n", strerror(-ret));
    goto failure;
}
memset(&to_addr, 0, sizeof(to_addr));
to_addr.can_ifindex = ifr.ifr_ifindex;
to_addr.can_family = AF_CAN;
if (use_send) {
    /* Suppress definition of a default receive filter list */
    ret = setsockopt(s, SOL_CAN_RAW, CAN_RAW_FILTER, NULL, 0);
    if (ret < 0) {
        fprintf(stderr, "setsockopt: %s\n", strerror(-ret));
        goto failure;
    }
    ret = bind(s, (struct sockaddr *)&to_addr, sizeof(to_addr));
    if (ret < 0) {
        fprintf(stderr, "bind: %s\n", strerror(-ret));
        goto failure;
    }
}
if (count)
    frame.can_dlc = sizeof(int);
else {
    for (i = optind + 1; i < argc; i++) {
        frame.data[dlc] = strtoul(argv[i], NULL, 0);
        dlc++;
        if (dlc == 8)
            break;
    }
    frame.can_dlc = dlc;
if (rtr)
    frame.can_id |= CAN_RTR_FLAG;

if (extended)
    frame.can_id |= CAN_EFF_FLAG;

if (timeout) {
    if (verbose)
        printf("Timeout: %lld ns\n", (long long)timeout);
    ret = ioctl(s, RTCAN_RTIOC_SND_TIMEOUT, &timeout);
    if (ret) {
        fprintf(stderr, "ioctl SND_TIMEOUT: %s\n", strerror(-ret));
        goto failure;
    }

    snprintf(name, sizeof(name), "rtcansend-%d", getpid());
    ret = rt_task_shadow(&rt_task_desc, name, 1, 0);
    if (ret) {
        fprintf(stderr, "rt_task_shadow: %s\n", strerror(-ret));
        goto failure;
    }
}

rt_task();
cleanup();
return 0;

failure:
    cleanup();
    return -1;

9.10  xddp-echo.c

/ *
 * Copyright (C) 2009 Philippe Gerum <rpm@xenomai.org>
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2 of the License, or (at your option) any later version.
 * This library 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
 * Lesser General Public License for more details.
 *------------------------------------------------------------------
 * You should have received a copy of the GNU Lesser General Public
 * License along with this library; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.
 *------------------------------------------------------------------
 * XDDP-based RT/NRT threads communication demo.
 *------------------------------------------------------------------
 * Real-time Xenomai threads and regular Linux threads may want to
 * exchange data in a way that does not require the former to leave
 * the real-time domain (i.e. secondary mode). Message pipes - as
 * implemented by the RTDM-based XDDP protocol - are provided for this
 * purpose.
 *------------------------------------------------------------------
 * On the Linux domain side, pseudo-device files named /dev/rtp<minor>
 * give regular POSIX threads access to non real-time communication
 * endpoints, via the standard character-based I/O interface. On the
 * Xenomai domain side, sockets may be bound to XDDP ports, which act
 * as proxies to send and receive data to/from the associated
 * pseudo-device files. Ports and pseudo-device minor numbers are
 * paired, meaning that e.g. port 7 will proxy the traffic for
 * /dev/rtp7. Therefore, port numbers may range from 0 to
 * CONFIG_XENO_OPT_PIPE_NRDEV - 1.
 *------------------------------------------------------------------
 * All data sent through a bound/connected XDDP socket via sendto(2) or
 * write(2) will be passed to the peer endpoint in the Linux domain,
 * and made available for reading via the standard read(2) system
 * call. Conversely, all data sent using write(2) through the non
 * real-time endpoint will be conveyed to the real-time socket
 * endpoint, and made available to the recvfrom(2) or read(2) system
 * calls.
 *------------------------------------------------------------------
 * Both threads can use the bi-directional data path to send and
 *...
* receive datagrams in a FIFO manner, as illustrated by the simple
* echoing process implemented by this program.
* 
* realtime_thread-----------------------------+----->
* => get socket
* => bind socket to port 0
* => write traffic to NRT domain via sendto()
* => read traffic from NRT domain via recvfrom() <-> ---
* regular_thread-----------------------------+----->
* => open /dev/rtp0
* => read traffic from RT domain via read()
* => echo traffic back to RT domain via write()<--++
*/

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <string.h>
#include <malloc.h>
#include <pthread.h>
#include <fcntl.h>
#include <errno.h>
#include <rtdm/ipc.h>

pthread_t rt, nrt;
#define XDDP_PORT 0 /* [0..CONFIG-XENO_OPT_PIPE_NRDEV - 1] */

static const char *msg[] = {
    "Surfing With The Alien",
    "Lords of Karma",
    "Banana Mango",
    "Psycho Monkey",
    "Luminous Flesh Giants",
    "Moroccan Sunset",
    "Satch Boogie",
    "Flying In A Blue Dream",
    "Ride",
    "Summer Song",
    "Speed Of Light",
    "Crystal Planet",
    "Raspberry Jam Delta-V",
    "Champagne?",
    "Clouds Race Across The Sky",
    "Engines Of Creation"
};

static void fail(const char *reason)
{
    perror(reason);
    exit(EXIT_FAILURE);
}

static void *realtime_thread(void *arg)
{
    struct sockaddr_ipc saddr;
    int ret, s, n = 0, len;
    struct timespec ts;
    size_t poolsz;
    char buf[128];

    /*
    * Get a datagram socket to bind to the RT endpoint. Each
    * endpoint is represented by a port number within the XDDP
    * protocol namespace.
    */
    s = socket(AF_RTIPC, SOCK_DGRAM, IPCPROTO_XDDP);
    if (s < 0) {
        perror("socket");
        exit(EXIT_FAILURE);
    }

    /*
    * Set a local 16k pool for the RT endpoint. Memory needed to
    * convey datagrams will be pulled from this pool, instead of
    * Xenomai’s system pool.
    */
    poolsz = 16384; /* bytes */
    ret = setsockopt(s, SOL_XDDP, XDDP_POOLSZ,
                     &poolsz, sizeof(poolsz));
    if (ret)
        fail("setsockopt");

    /*
    * Bind the socket to the port, to setup a proxy to channel
    * traffic to/from the Linux domain.
    */

    return NULL;
}
* saddr.sipc_port specifies the port number to use.

memset(&saddr, 0, sizeof(saddr));
saddr.sipc_family = AF_RTIPC;
saddr.sipc_port = XDDP_PORT;
ret = bind(s, (struct sockaddr *)&saddr, sizeof(saddr));
if (ret)
  fail("bind");

for (;;) {
  len = strlen(msg[n]);
  /* Send a datagram to the NRT endpoint via the proxy.
   * We may pass a NULL destination address, since a
   * bound socket is assigned a default destination
   * address matching the binding address (unless
   * connect(2) was issued before bind(2), in which case
   * the former would prevail).
   */
  ret = sendto(s, msg[n], len, 0, NULL, 0);
  if (ret != len)
    fail("sendto");
  printf("%s: sent %d bytes, %.*s
" __FUNCTION__, ret, ret, msg[n]);

  /* Read back packets echoed by the regular thread */
  ret = recvfrom(s, buf, sizeof(buf), 0, NULL, 0);
  if (ret <= 0)
    fail("recvfrom");
  printf(" => %.*s echoed by peer\n", ret, buf);

  n = (n + 1) % (sizeof(msg) / sizeof(msg[0]));
  /* We run in full real-time mode (i.e. primary mode),
   * so we have to let the system breathe between two
   * iterations.
   */
  ts.tv_sec = 0;
  ts.tv_nsec = 500000000; /* 500 ms */
  clock_nanosleep(CLOCK_REALTIME, 0, &ts, NULL);
}

return NULL;
}

static void *regular_thread(void *arg)
{
  char buf[128], *devname;
  int fd, ret;

  if (asprintf(&devname, "/dev/rtp%d", XDDP_PORT) < 0)
    fail("asprintf");
  fd = open(devname, O_RDWR);
  free(devname);
  if (fd < 0)
    fail("open");

  for (;;) {
    /* Get the next message from realtime_thread. */
    ret = read(fd, buf, sizeof(buf));
    if (ret <= 0)
      fail("read");
    /* Echo the message back to realtime_thread. */
    ret = write(fd, buf, ret);
    if (ret <= 0)
      fail("write");
  }

  return NULL;
}

int main(int argc, char **argv)
{
  struct sched_param rtparam = { .sched_priority = 42 };
  pthread_attr_t rattr, regattr;
  sigset_t set;
  int sig;

  sigemptyset(&set);
  sigaddset(&set, SIGINT);
  sigaddset(&set, SIGTERM);
  sigaddset(&set, SIGSTOP);

  /* Add the regular thread to the scheduler. */
  rattr.sched_priority = 42;
  rattr.attos = &rtparam;
  pthread_create(&rattr, NULL, regular_thread, NULL);

  /* Create a signal handler that blocks the main thread. */
  sigprocmask(SIG_BLOCK, &set, NULL);
  signal(SIGINT, SIG_IGN);
  signal(SIGTERM, SIG_IGN);
  signal(SIGSTOP, SIG_IGN);

  /* Start the main thread. */
  pthread_join(&rattr, NULL);
  return 0;
}
pthread_sigmask(SIG_BLOCK, &set, NULL);

pthread_attr_init(&rtattr);
pthread_attr_setdetachstate(&rtattr, PTHREAD_CREATE_JOINABLE);
pthread_attr_setschedpolicy(&rtattr, SCHED_FIFO);
pthread_attr_setschedparam(&rtattr, &rtparam);
errno = pthread_create(&rt, &rtattr, &realtime_thread, NULL);
if (errno)
  fail("pthread_create");

pthread_attr_init(&regattr);
pthread_attr_setdetachstate(&regattr, PTHREAD_CREATE_JOINABLE);
pthread_attr_setschedpolicy(&regattr, SCHED_OTHER);
errno = pthread_create(&nrt, &regattr, &regular_thread, NULL);
if (errno)
  fail("pthread_create");
sigwait(&set, &sig);

pthread_cancel(rt);
pthread_cancel(nrt);
pthread_join(rt, NULL);
pthread_join(nrt, NULL);
return 0;
}

/*
 * Copyright (C) 2009 Philippe Gerum <rpm@xenomai.org>.
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2 of the License, or (at your option) any later version.
 * This library 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
 * Lesser General Public License for more details.
 * You should have received a copy of the GNU Lesser General Public
 * License along with this library; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.
 */

/*
 * XDDP-based RT/NRT threads communication demo.
 * Real-time Xenomai threads and regular Linux threads may want to
 * exchange data in a way that does not require the former to leave
 * the real-time domain (i.e. secondary mode). Message pipes - as
 * implemented by the RTDM-based XDDP protocol - are provided for this
 * purpose.
 * On the Linux domain side, pseudo-device files named /dev/rtp<minor>
 * give regular POSIX threads access to non real-time communication
 * endpoints, via the standard character-based I/O interface. On the
 * Xenomai domain side, sockets may be bound to XDDP ports, which act
 * as proxies to send and receive data to/from the associated
 * pseudo-device files. Ports and pseudo-device minor numbers are
 * paired, meaning that e.g. port 7 will proxy the traffic for
 * /dev/rtp7. Therefore, port numbers may range from 0 to
 * CONFIG_XENO_OPT_PIPE_NRDEV - 1.
 * All data sent through a bound/connected XDDP socket via sendto(2) or
 * write(2) will be passed to the peer endpoint in the Linux domain,
 * and made available for reading via the standard read(2) system
 * call. Conversely, all data sent using write(2) through the non
 * real-time endpoint will be conveyed to the real-time socket
 * endpoint, and made available to the recvfrom(2) or read(2) system
 * calls.
 * ASCII labels can be attached to bound ports, in order to connect
 * sockets to them in a more descriptive way than using plain numeric
 * port values.
 * The example code below illustrates the following process:
 */
/* realtime_thread1--------------------------->----------+
  * => get socket 
  * => bind socket to port "xddp-demo" 
  * => read traffic from NRT domain via recvfrom()   <------+
  * 
  * realtime_thread2-----------------------------| 
  * => get socket 
  * => connect socket to port "xddp-demo" 
  * => write traffic to NRT domain via sendto()      v 
  * 
  * regular_thread-----------------------------| 
  * => open /proc/xenomai/registry/rtipc/xddp/xddp-demo | 
  * => read traffic from RT domain via read()      | 
  * => mirror traffic to RT domain via write()      +--+
 */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <string.h>
#include <malloc.h>
#include <pthread.h>
#include <fcntl.h>
#include <errno.h>
#include <rtdm/ipc.h>

pthread_t rt1, rt2, nrt;
#define XDDP_PORT_LABEL "xddp-demo"

static const char *msg[] = {
    "Surfing With The Alien",
    "Lords of Karma",
    "Banana Mango",
    "Psycho Monkey",
    "Luminous Flesh Giants",
    "Moroccan Sunset",
    "Satch Boogie",
    "Flying In A Blue Dream",
    "Ride",
    "Summer Song",
    "Speed Of Light",
    "Crystal Planet",
    "Raspberry Jam Delta-V",
    "Champagne?",
    "Clouds Race Across The Sky",
    "Engines Of Creation"
};

static void fail(const char *reason)
{
    perror(reason);
    exit(EXIT_FAILURE);
}

static void *realtime_thread1(void *arg)
{
    struct rtipc_port_label plabel;
    struct sockaddr_ipc saddr;
    char buf[128];
    int ret, s;

    /* Get a datagram socket to bind to the RT endpoint. Each 
     * endpoint is represented by a port number within the XDDP 
     * protocol namespace. 
     */
    s = socket(AF_RTIPC, SOCK_DGRAM, IPCPROTO_XDDP);
    if (s < 0) {
        perror("socket");
        exit(EXIT_FAILURE);
    }

    /* Set a port label. This name will be registered when 
     * binding, in addition to the port number (if given). 
     */
    strcpy(plabel.label, XDDP_PORT_LABEL);
    ret = setsockopt(s, SOL_XDDP, XDDP_LABEL, 
                     &plabel, sizeof(plabel));
    if (ret) fail("setsockopt");

    /* Bind the socket to the port, to setup a proxy to channel 
     * traffic to/from the Linux domain. Assign that port a label, 
     * so that peers may use a descriptive information to locate
it. For instance, the pseudo-device matching our RT endpoint will appear as
/proc/xenomai/registry/rtipc/xdpp/<XDDP_PORT_LABEL> in the Linux domain, once the socket is bound.

saddr.sipc_port specifies the port number to use. If -1 is passed, the XDDP driver will auto-select an idle port.

memset(&saddr, 0, sizeof(saddr));
saddr.sipc_family = AF_RTIPC;
saddr.sipc_port = -1;
ret = bind(s, (struct sockaddr *)&saddr, sizeof(saddr));
if (ret)
    fail("bind");

for (;;) {
    /* Get packets relayed by the regular thread */
    ret = recvfrom(s, buf, sizeof(buf), 0, NULL, 0);
    if (ret <= 0)
        fail("recvfrom");
    printf("%s: \%s\" relayed by peer\n", __FUNCTION__, ret, buf);
}
return NULL;

static void *realtime_thread2(void *arg)
{
    struct rtipc_port_label plabel;
    struct sockaddr_ipc saddr;
    int ret, s, n = 0, len;
    struct timespec ts;
    struct timeval tv;
    socklen_t addrlen;

    s = socket(AF_RTIPC, SOCK_DGRAM, IPCPROTO_XDDP);
    if (s < 0) {
        perror("socket");
        exit(EXIT_FAILURE);
    }

    /* Set the socket timeout: it will apply when attempting to connect to a labeled port, and to recvfrom() calls. The following setup tells the XDDP driver to wait for at most one second until a socket is bound to a port using the same label, or return with a timeout error. */
    tv.tv_sec = 1;
    tv.tv_usec = 0;
    ret = setsockopt(s, SOL_SOCKET, SO_RCVTIMEO, &tv, sizeof(tv));
    if (ret)
        fail("setsockopt");

    /* Set a port label. This name will be used to find the peer when connecting, instead of the port number. */
    strcpy(plabel.label, XDDP_PORT_LABEL);
    ret = setsockopt(s, SOL_XDDP, XDDP_LABEL, &plabel, sizeof(plabel));
    if (ret)
        fail("setsockopt");

    /* Set a port address. This address will be used to find the peer when connecting, instead of the port number. */
    ret = setsockopt(s, SOL_XDP, XDP_PORT_LABEL, &label, sizeof(label));
    if (ret)
        fail("setsockopt");

    memset(&saddr, 0, sizeof(saddr));
    saddr.sipc_family = AF_RTIPC;
saddr.sipc_port = -1; /* Tell XDDP to search by label. */
    ret = connect(s, (struct sockaddr *)&saddr, sizeof(saddr));
    if (ret)
        fail("connect");

    /* We succeeded in making the port our default destination address by using its label, but we don’t know its actual port number yet. Use getpeername() to retrieve it. */
    addrlen = sizeof(saddr);
    ret = getpeername(s, (struct sockaddr *)&saddr, &addrlen);
    if (ret || addrlen != sizeof(saddr))
        fail("getpeername");
    printf("%s: NRT peer is reading from /dev/rtp%d\n", __FUNCTION__, saddr.sipc_port);
    for (;;) {
len = strlen(msg[n]);
/*
 * Send a datagram to the NRT endpoint via the proxy.
 * We may pass a NULL destination address, since the
 * socket was successfully assigned the proper default
 * address via connect(2).
 */
ret = sendto(s, msg[n], len, 0, NULL, 0);
if (ret != len)
  fail("sendto");
printf("%s: sent %d bytes, \"%.*s\"\n",
    __FUNCTION__, ret, ret, msg[n]);
n = (n + 1) % (sizeof(msg) / sizeof(msg[0]));
/*
 * We run in full real-time mode (i.e. primary mode),
 * so we have to let the system breathe between two
 * iterations.
 */
ts.tv_sec = 0;
ts.tv_nsec = 500000000; /* 500 ms */
clock_nanosleep(CLOCK_REALTIME, 0, &ts, NULL);
}
return NULL;
}
static void *regular_thread(void *arg)
{
  char buf[128], *devname;
  int fd, ret;
  if (asprintf(&devname,
    "/proc/xenomai/registry/rtipc/xddp/%s",
    XDDP_PORT_LABEL) < 0)
    fail("asprintf");
  fd = open(devname, O_RDWR);
  free(devname);
  if (fd < 0)
    fail("open");
  for (;;) {
    /* Get the next message from realtime_thread2. */
    ret = read(fd, buf, sizeof(buf));
    if (ret <= 0)
      fail("read");
    /* Relay the message to realtime_thread1. */
    ret = write(fd, buf, ret);
    if (ret <= 0)
      fail("write");
  }
  return NULL;
}
int main(int argc, char **argv)
{
  struct sched_param rtparam = { .sched_priority = 42 }; 
  pthread_attr_t rattr, regattr;
  sigset_t set;
  int sig;
  sigemptyset(&set);
  sigaddset(&set, SIGINT);
  sigaddset(&set, SIGTERM);
  sigaddset(&set, SIGHUP);
  pthread_sigmask(SIG_BLOCK, &set, NULL);
  pthread_attr_init(&rattr);
  pthread_attr_init(&regattr);
  pthread_attr_setdetachstate(&rattr, PTHREAD_CREATE_JOINABLE);
  pthread_attr_setdetachstate(&regattr, PTHREAD_CREATE_DETACHED);
  pthread_attr_setschedpolicy(&rattr, SCHED_FIFO);
  pthread_attr_setschedparam(&rattr, &rtparam);
  /* Both real-time threads have the same attribute set. */
  errno = pthread_create(&rt1, &rattr, &realtime_thread1, NULL);
  if (errno)
    fail("pthread_create");
  errno = pthread_create(&rt2, &rattr, &realtime_thread2, NULL);
  if (errno)
    fail("pthread_create");
}
```c
pthread_attr_init(&regattr);
pthread_attr_setdetachstate(&regattr, PTHREAD_CREATE_JOINABLE);
pthread_attr_setinheritsched(&regattr, PTHREAD_EXPLICIT_SCHED);
pthread_attr_setschedpolicy(&regattr, SCHED_OTHER);
errno = pthread_create(&nrt, &regattr, &regular_thread, NULL);
if (errno)
    fail("pthread_create");
sigwait(&set, &sig);
pthread_cancel(rt1);
pthread_cancel(rt2);
pthread_cancel(nrt);
pthread_join(rt1, NULL);
pthread_join(rt2, NULL);
pthread_join(nrt, NULL);
return 0;
}
```

/*
 * Copyright (C) 2009 Philippe Gerum <rpm@xenomai.org>.
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2 of the License, or (at your option) any later version.
 * This library 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
 * Lesser General Public License for more details.
 * You should have received a copy of the GNU Lesser General Public
 * License along with this library; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.
 *
 * XDDP-based RT/NRT threads communication demo.
 * Real-time Xenomai threads and regular Linux threads may want to
 * exchange data in a way that does not require the former to leave
 * the real-time domain (i.e. secondary mode). Message pipes - as
 * implemented by the RTDM-based XDDP protocol - are provided for this
 * purpose.
 * On the Linux domain side, pseudo-device files named /dev/rtp<minor>
 * give regular POSIX threads access to non real-time communication
 * endpoints, via the standard character-based I/O interface. On the
 * Xenomai domain side, sockets may be bound to XDDP ports, which act
 * as proxies to send and receive data to/from the associated
 * pseudo-device files. Ports and pseudo-device minor numbers are
 * paired, meaning that e.g. port 7 will proxy the traffic for
 * /dev/rtp7. Therefore, port numbers may range from 0 to
 * CONFIG_XENO_OPT_PIPE_NRDEV - 1.
 * All data sent through a bound/connected XDDP socket via sendto(2) or
 * write(2) will be passed to the peer endpoint in the Linux domain,
 * and made available for reading via the standard read(2) system
 * call. Conversely, all data sent using write(2) through the mon
 * real-time endpoint will be conveyed to the real-time socket
 * endpoint, and made available to the recvfrom(2) or read(2) system
 * calls.
 * In addition to sending datagrams, real-time threads may stream data
 * in a byte-oriented mode through the proxy as well. This increases
 * the bandwidth and reduces the overhead, when a lot of data has to
 * flow down to the Linux domain, if keeping the message boundaries is
 * not required. The example code below illustrates such use.
 *
 * realtime_thread-------------------------------------->----------+
 * => get socket | v
 * => bind socket to port 0
 * => write scattered traffic to NRT domain via sendto() |
 * => read traffic from NRT domain via recvfrom() <>-<>-
 * regular_thread------------------------------------------+
 * => open /dev/rtp0
 * => read traffic from RT domain via read() 
*/

Generated by Doxygen
Example Documentation

* => echo traffic back to RT domain via write()  

*/

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <string.h>
#include <malloc.h>
#include <pthread.h>
#include <fcntl.h>
#include <errno.h>
#include <rtdm/ipc.h>

pthread_t rt, nrt;
#define XDDP_PORT 0 /* [0..CONFIG-XENO_OPT_PIPE_NRDEV - 1] */

static const char *msg[] = {
    "Surfing With The Alien",
    "Lords of Karma",
    "Banana Mango",
    "Psycho Monkey",
    "Luminous Flesh Giants",
    "Moroccan Sunset",
    "Satch Boogie",
    "Flying In A Blue Dream",
    "Ride",
    "Summer Song",
    "Speed Of Light",
    "Crystal Planet",
    "Raspberry Jam Delta-V",
    "Champagne?",
    "Clouds Race Across The Sky",
    "Engines Of Creation"
};

static void fail(const char *reason)
{
    perror(reason);
    exit(EXIT_FAILURE);
}

static void *realtime_thread(void *arg)
{
    struct sockaddr_ipc saddr;
    int ret, s, n = 0, len, b;
    struct timespec ts;
    size_t streamsz;
    char buf[128];

    /* Get a datagram socket to bind to the RT endpoint. Each
    * endpoint is represented by a port number within the XDDP
    * protocol namespace.
    */
    s = socket(AF_RTIPC, SOCK_DGRAM, IPCPROTO_XDDP);
    if (s < 0) {
        perror("socket");
        exit(EXIT_FAILURE);
    }

    /* Tell the XDDP driver that we will use the streaming
    * capabilities on this socket. To this end, we have to
    * specify the size of the streaming buffer, as a count of
    * bytes. The real-time output will be buffered up to that
    * amount, and sent as a single datagram to the RTT endpoint
    * when fully gathered, or when another source port attempts
    * to send data to the same endpoint. Passing a null size
    * would disable streaming.
    */
    streamsz = 1284; /* bytes */
    ret = setsockopt(s, SOL_XDDP, XDP_BUFSIZE,
                     streamsz, sizeof(streamsz));
    if (ret)
        fail("setsockopt");

    /* Bind the socket to the port, to setup a proxy to channel
    * traffic to/from the Linux domain.
    * saddr.sipc_port specifies the port number to use.
    */
    memset(&saddr, 0, sizeof(saddr));
    saddr.sipc_port = XDDP_PORT;
    ret = bind(s, (struct sockaddr *)&saddr, sizeof(saddr));
    if (ret)
fail("bind");

for (;;) {
    len = strlen(msg[n]);
    /*
    * Send a datagram to the NRT endpoint via the proxy.
    * The output is artificially scattered in separate
    * one-byte sendings, to illustrate the use of
    * MSG_MORE.
    */
    for (b = 0; b < len; b++) {
        ret = sendto(s, msg[n] + b, 1, MSG_MORE, NULL, 0);
        if (ret != 1)
            fail("sendto");
    }
    printf("%s: sent (scattered) %d-bytes message,\n", __FUNCTION__, len, len, msg[n]);
    /* Read back packets echoed by the regular thread */
    ret = recvfrom(s, buf, sizeof(buf), 0, NULL, 0);
    if (ret <= 0)
        fail("recvfrom");
    printf(" => \%.*s\ echoed by peer\n", ret, buf);
    n = (n + 1) % (sizeof(msg) / sizeof(msg[0]));
    /* We run in full real-time mode (i.e. primary mode),
    * so we have to let the system breathe between two
    * iterations.
    */
    ts.tv_sec = 0;
    ts.tv_nsec = 500000000; /* 500 ms */
    clock_nanosleep(CLOCK_REALTIME, 0, &ts, NULL);
}

return NULL;

static void *regular_thread(void *arg)
{
    char buf[128], *devname;
    int fd, ret;
    if (asprintf(&devname, "/dev/rtp%d", XDDP_PORT) < 0)
        fail("asprintf");
    fd = open(devname, O_RDWR);
    free(devname);
    if (fd < 0)
        fail("open");

    for (;;) {
        /* Get the next message from realtime_thread. */
        ret = read(fd, buf, sizeof(buf));
        if (ret <= 0)
            fail("read");

        /* Echo the message back to realtime_thread. */
        ret = write(fd, buf, ret);
        if (ret <= 0)
            fail("write");
    }

    return NULL;
}

int main(int argc, char **argv)
{
    struct sched_param rtparam = { .sched_priority = 42 };  
    pthread_attr_t rattr, regattr;
    sigset_t set;
    int sig;
    sigemptyset(&set);
    sigaddset(&set, SIGINT);
    sigaddset(&set, SIGTERM);
    sigaddset(&set, SIGSTOP);
    pthread_sigmask(SIG_BLOCK, &set, NULL);

    pthread_attr_init(&rattr);
    pthread_attr_setdetachstate(&rattr, PTHREAD_CREATE_JOINABLE);
    pthread_attr_setschedpolicy(&rattr, SCHED_FIFO);
    pthread_attr_setschedparam(&rattr, &rtparam);

    Generated by Doxygen
errno = pthread_create(&rt, &rtattr, &realtime_thread, NULL);
if (errno)
    fail("pthread_create");

pthread_attr_init(&regattr);
pthread_attr_setdetachstate(&regattr, PTHREAD_CREATE_JOINABLE);
pthread_attr_setschedpolicy(&regattr, SCHED_OTHER);
errno = pthread_create(&nrt, &regattr, &regular_thread, NULL);
if (errno)
    fail("pthread_create");

sigwait(&set, &sig);
pthread_cancel(rt);
pthread_cancel(nrt);
pthread_join(rt, NULL);
pthread_join(nrt, NULL);

return 0;
}
Index

__xntimer_migrate
   Timer services, 272

A4L_RNG_FACTOR
   uapi/analogy.h, 681
a4l_add_subd
   Subdevice management services, 294
a4l_alloc_subd
   Subdevice management services, 294
a4l_async_read
   Asynchronous acquisition API, 399
a4l_async_write
   Asynchronous acquisition API, 400
a4l_buf_commit_absget
   Buffer management services, 297
a4l_buf_commit_absput
   Buffer management services, 298
a4l_buf_commit_get
   Buffer management services, 298
a4l_buf_commit_put
   Buffer management services, 299
a4l_buf_count
   Buffer management services, 299
a4l_buf_evt
   Buffer management services, 301
a4l_buf_get
   Buffer management services, 301
a4l_buf_prepare_absget
   Buffer management services, 302
a4l_buf_prepare_absput
   Buffer management services, 302
a4l_buf_prepare_get
   Buffer management services, 303
a4l_buf_prepare_put
   Buffer management services, 304
a4l_buf_put
   Buffer management services, 304
a4l_channel, 563
   flags, 563
   nb_bits, 563
a4l_channels_desc, 564
   chans, 564
   length, 564
   mode, 565
a4l_close
   Descriptor API, 406
a4l_cmd_desc, 565
   data_len, 566
   idx_subd, 566
a4l_config_subd
   Subdevice management services, 295
   Synchronous acquisition API, 425
a4l_dcaltoraw
   Software calibration API, 401
a4l_descriptor, 566
   board_name, 567
   driver_name, 567
   fd, 567
   idx_read_subd, 567
   idx_write_subd, 567
   magic, 568
   nb_subd, 568
   sbdata, 568
   sbsize, 568
a4l_driver, 568
a4l_dtoraw
   Range / conversion API, 413
a4l_fill_desc
   Descriptor API, 407
a4l_find_range
   Range / conversion API, 414
a4l_free_irq
   Interrupt management services, 306
a4l_ftoraw
   Range / conversion API, 415
a4l_get_bufsize
   Asynchronous acquisition API, 393
a4l_get_chan
   Buffer management services, 305
a4l_get_chinfo
   Descriptor API, 407
a4l_get_cmd
   Buffer management services, 305
a4l_get_irq
   Interrupt management services, 307
a4l_get_rnginfo
   Descriptor API, 408
a4l_get_softcal_converter
   Software calibration API, 402
a4l_get_subd
   Subdevice management services, 295
a4l_get_subdinfo
   Descriptor API, 408
a4l_get_time
   Misc services, 309
a4l_instruction, 569
   idx_subd, 570
a4l_instruction_list, 570
a4l_mark_bufrw
   Asynchronous acquisition API, 393
a4l_math_mean
   Math API, 410
a4l_math_polyfit
   Math API, 411
a4l_math_stdddev
   Math API, 411
a4l_math_stdddev_of_mean
   Math API, 412
a4l_mmap
   Asynchronous acquisition API, 394
a4l_open
   Descriptor API, 409
a4l_poll
   Asynchronous acquisition API, 395
a4l_range, 571
   flags, 571
   max, 571
   min, 571
a4l_rawtod
   Range / conversion API, 415
a4l_rawtodcal
   Software calibration API, 402
a4l_rawtof
   Range / conversion API, 416
a4l_roughtoul
   Range / conversion API, 416
a4l_set_bufsize
   Asynchronous acquisition API, 396
a4l_sizeof_chan
   Range / conversion API, 417
a4l_sizeof_subd
   Range / conversion API, 417
a4l_snd_cancel
   Asynchronous acquisition API, 396
a4l_snd_command
   Asynchronous acquisition API, 397
a4l_snd_insn
   Synchronous acquisition API, 422
a4l_snd_insnlist
   Synchronous acquisition API, 422
a4l_subdevice, 572
a4l_sync_dio
   Synchronous acquisition API, 426
a4l_sync_read
   Synchronous acquisition API, 426
a4l_sync_write
   Synchronous acquisition API, 427
a4l_sys_attach
   Attach / detach Syscall API, 434
a4l_sys_bufcfg
   Attach / detach Syscall API, 435
a4l_sys_close
   Basic Syscall API, 431
a4l_sys_desc
   Descriptor Syscall API, 404
a4l_sys_detach
   Attach / detach Syscall API, 435
a4l_sys_open
   Basic Syscall API, 431
a4l_sys_read
   Basic Syscall API, 432
a4l_sys_write
   Basic Syscall API, 432
a4l_ultoraw
   Range / conversion API, 418
a4l_unregister_drv
   Driver management services, 291
addr
   udd_memregion, 607
Alarm services, 437
rt_alarm_create, 437
rt_alarm_delete, 438
rt_alarm_inquire, 439
rt_alarm_start, 439
rt_alarm_stop, 440
Alchemy API, 479
Analogy framework, 288
Analogy user API, 429
Asynchronous acquisition API, 391, 399
a4l_async_read, 399
a4l_async_write, 400
a4l_get_bufsize, 393
a4l_mark_bufrw, 393
a4l_mmap, 394
a4l_poll, 395
a4l_set_bufsize, 396
a4l_snd_cancel, 396
a4l_snd_command, 397
Asynchronous Procedure Calls, 93
xnapc_alloc, 93
xnapc_free, 94
xnapc_schedule, 94
atomic_t, 573
Attach / detach Syscall API, 434
a4l_sys_attach, 434
a4l_sys_bufcfg, 435
a4l_sys_detach, 435
B_PRIO
Buffer services, 443
BUFP_BUFSZ
Real-time IPC, 76
BUFP_LABEL
Real-time IPC, 77
base_minor
rtدم_driver, 592
Basic Syscall API, 431
a4l_sys_close, 431
a4l_sys_open, 431
a4l_sys_read, 432
a4l_sys_write, 432

---

Generated by Doxygen
begin
  xnvfile_regular_ops, 614
  xnvfile_snapshot_ops, 622
Big dual kernel lock, 24
cobalt_atomic_enter, 24
cobalt_atomic_leave, 25
RTDM_EXECUTE_ATOMICALLY, 25
bind __AF_RTIPC
  Real-time IPC, 85
board_name
  a4l_descriptor, 567
Buffer descriptor, 97
  xnbufd_copy_from_kmem, 99
  xnbufd_copy_to_kmem, 100
  xnbufd_invalidate, 101
  xnbufd_map_kread, 101
  xnbufd_map_kwrite, 102
  xnbufd_map_uread, 102
  xnbufd_map_uwrite, 103
  xnbufd_reset, 103
  xnbufd_unmap_kread, 104
  xnbufd_unmap_kwrite, 104
  xnbufd_unmap_uread, 105
  xnbufd_unmap_uwrite, 105
Buffer management services, 296
  a4l_buf_commit_absget, 297
  a4l_buf_commit_absput, 298
  a4l_buf_commit_get, 298
  a4l_buf_commit_put, 299
  a4l_buf_count, 299
  a4l_buf_evt, 301
  a4l_buf_get, 301
  a4l_buf_prepare_absget, 302
  a4l_buf_prepare_absput, 302
  a4l_buf_prepare_get, 303
  a4l_buf_prepare_put, 304
  a4l_buf_put, 304
  a4l_get_chan, 305
  a4l_get_cmd, 305
Buffer services, 442
  B_PRIO, 443
  rt_buffer_bind, 443
  rt_buffer_clear, 444
  rt_buffer_create, 444
  rt_buffer_delete, 445
  rt_buffer_inquire, 446
  rt_buffer_read, 447
  rt_buffer_read_timed, 447
  rt_buffer_read_until, 448
  rt_buffer_unbind, 449
  rt_buffer_write, 449
  rt_buffer_write_timed, 450
  rt_buffer_write_until, 451
CAN Devices, 43
  CAN_BITTIME_TYPE, 65
  CAN_CTRLMODE_3_SAMPLES, 53
  CAN_CTRLMODE_LISTENONLY, 54
  CAN_CTRLMODE_LOOPBACK, 54
  CAN_ERR_LOSTARB_UNSPEC, 54
  CAN_MODE, 65
  CAN_RAW_ERR_FILTER, 54
  CAN_RAW_FILTER, 55
  CAN_RAW_LOOPBACK, 56
  CAN_RAW_RECV_OWN_MSGS, 56
  CAN_STATE, 66
can_filter_t, 64
can_frame_t, 65
  RTCAN_RTIOC_RCV_TIMEOUT, 56
  RTCAN_RTIOC_SND_TIMEOUT, 57
  RTCAN_RTIOC_TAKE_TIMESTAMP, 58
  SIOCGCANBAUDRATE, 58
  SIOCGCANCTRLMODE, 59
  SIOCGCANCUSTOMBITTIME, 59
  SIOCGCANSTATE, 60
  SIOCGINDEX, 61
  SIOCSCANBAUDRATE, 61
  SIOCSCANCTRLMODE, 62
  SIOCSCANCUSTOMBITTIME, 63
  SIOCSCANMODE, 63
  SOL_CAN_RAW, 64
  CAN_BITTIME_TYPE
    CAN Devices, 65
  CAN_CTRLMODE_3_SAMPLES
    CAN Devices, 53
  CAN_CTRLMODE_LISTENONLY
    CAN Devices, 54
  CAN_CTRLMODE_LOOPBACK
    CAN Devices, 54
  CAN_ERR_LOSTARB_UNSPEC
    CAN Devices, 54
  CAN_MODE
    CAN Devices, 65
  CAN_RAW_ERR_FILTER
    CAN Devices, 54
  CAN_RAW_FILTER
    CAN Devices, 55
  CAN_RAW_LOOPBACK
    CAN Devices, 56
  CAN_RAW_RECV_OWN_MSGS
    CAN Devices, 56
  CAN_STATE
    CAN Devices, 66
  COMPAT___rt_alarm_create
    Transition Kit, 553
  COMPAT___rt_event_clear
    Transition Kit, 554
  COMPAT___rt_event_create
    Transition Kit, 555
  COMPAT___rt_event_signal
    Transition Kit, 556
  COMPAT___rt_pipe_create
    Transition Kit, 556
  COMPAT___rt_task_create
    Transition Kit, 558
  COMPAT___rt_task_set_periodic
    Transition Kit, 559
can_bittime, 574
can_bittime_btr, 575
can_bittime_std, 575
can_filter, 576
can_id, 576
can_mask, 577
can_filter_t
CAN Devices, 64
can_frame, 577
can_id, 578
can_frame_t
CAN Devices, 65
can_id
can_filter, 576
can_frame, 578
can_ifindex
sockaddr_can, 601
can_ifreq, 578
can_mask
can_filter, 577
Channels and ranges, 21
chans
   a4l_channels_desc, 564
Clock Services, 169
   rtdm_clock_read, 169
   rtdm_clock_read_monotonic, 169
Clock services, 107
   xnclock_adjust, 107
   xnclock_deregister, 108
   xnclock_register, 108
   xnclock_tick, 109
clock_getres
   Clocks and timers, 311
clock_gettime
   Clocks and timers, 312
clock_nanosleep
   Clocks and timers, 312
clock_settime
   Clocks and timers, 313
Clocks and timers, 310
   clock_getres, 311
   clock_gettime, 312
   clock_nanosleep, 312
   clock_settime, 313
   nanosleep, 314
   timer_create, 315
   timer_delete, 316
   timer_getoverrun, 317
   timer_gettime, 317
   timer_settime, 318
close
   rtdm_fd_ops, 594
   udd_device, 603
close__AF_RTIPC
   Real-time IPC, 87
Cobalt, 115
Cobalt kernel, 116
cobalt_atomic_enter
   Big dual kernel lock, 24
cobalt_atomic_leave
   Big dual kernel lock, 25
cobalt_vroot
   Virtual file services, 287
Condition variable services, 453
   rt_cond_bind, 454
   rt_cond_broadcast, 455
   rt_cond_create, 455
   rt_cond_delete, 456
   rt_cond_inquire, 457
   rt_cond_signal, 457
   rt_cond_unbind, 458
   rt_cond_wait, 458
   rt_cond_wait_timed, 459
   rt_cond_wait_until, 460
Condition variables, 320
   pthread_cond_broadcast, 321
   pthread_cond_destroy, 322
   pthread_cond_init, 322
   pthread_cond_signal, 323
   pthread_cond_timedwait, 324
   pthread_cond_wait, 324
   pthread_condattr_destroy, 325
   pthread_condattr_getclock, 326
   pthread_condattr_getpshared, 327
   pthread_condattr_init, 327
   pthread_condattr_setclock, 328
   pthread_condattr_setpshared, 329
connect__AF_RTIPC
   Real-time IPC, 87
context_size
   rtdm_driver, 592
cpu
   xnsched, 610
curr
   xnsched, 610
data_len
   a4l_cmd_desc, 566
databuf
   xnvfile_snapshot_iterator, 621
date
   rt_timer_info, 586
Debugging services, 110
Descriptor API, 406
   a4l_close, 406
   a4l_fill_desc, 407
   a4l_get_chinfo, 407
   a4l_get_rnginfo, 408
   a4l_get_subdinfo, 408
   a4l_open, 409
Descriptor Syscall API, 404
   a4l_sys_desc, 404
device
   rtdm_dev_context, 587
Device Profiles, 155
   RTIOC_DEVICE_INFO, 157
   RTIOC_PURGE, 157

Generated by Doxygen
Device Registration Services, 158
  RTDM_DEVICE_TYPE_MASK, 159
  RTDM_EXCLUSIVE, 159
  RTDM_FIXED_MINOR, 159
  RTDM_MAX_MINOR, 160
  RTDM_NAMED_DEVICE, 160
  RTDM_PROTOCOL_DEVICE, 160
  rtdm_close_handler, 160
  rtdm_dev_register, 161
  rtdm_dev_unregister, 161
  rtdm_drv_set_sysclass, 162
  rtdm_get_unmapped_area_handler, 162
  rtdm_ioctl_handler, 163
  rtdm_mmap_handler, 164
  rtdm_open_handler, 164
  rtdm_read_handler, 165
  rtdm_recvmsg_handler, 165
  rtdm_select_handler, 166
  rtdm_sendmsg_handler, 167
  rtdm_socket_handler, 167
  rtdm_write_handler, 168
  device_count
    rtdm_driver, 592
  device_flags
    rtdm_driver, 592
    udd_device, 603
  device_subclass
    udd_device, 604
driver
  rtdm_device, 589
Driver API, 289
Driver management services, 290
  a4l_register_drv, 290
  a4l_unregister_drv, 291
Driver programming interface, 136
Driver to driver services, 137
  rtdm_accept, 138
  rtdm_bind, 139
  rtdm_close, 140
  rtdm_connect, 140
  rtdm_getpeername, 141
  rtdm_getsockname, 142
  rtdm_getsockopt, 142
  rtdm_ioctl, 143
  rtdm_listen, 144
  rtdm_open, 145
  rtdm_read, 145
  rtdm_recv, 146
  rtdm_recvfrom, 147
  rtdm_recvmmsg, 148
  rtdm_send, 148
  rtdm_sendmsg, 149
  rtdm_sendto, 150
  rtdm_setsockopt, 151
  rtdm_shutdown, 152
  rtdm_socket, 152
  rtdm_write, 153
driver.h
  RTDM_CLASS_MAGIC, 638
  RTDM_PROFILE_INFO, 638
  rtdm_fd_device, 639
  rtdm_fd_is_user, 639
  rtdm_fd_to_private, 640
  rtdm_private_to_fd, 640
driver_name
  a4l_descriptor, 567
Dynamic memory allocation services, 111
  xnheap_alloc, 112
  xnheap_destroy, 112
  xnheap_free, 113
  xnheap_init, 113
  xnheap_set_name, 114

EV_ANY
  Event flag group services, 462
EV_PRIO
  Event flag group services, 462
end
  xnviile_regular_ops, 615
  xnviile_snapshot_ops, 623
endfn
  xnviile_snapshot_iterator, 621
Event flag group services, 461
  EV_ANY, 462
  EV_PRIO, 462
  rt_event_bind, 462
  rt_event_clear, 463
  rt_event_create, 464
  rt_event_delete, 465
  rt_event_inquire, 465
  rt_event_signal, 466
  rt_event_unbind, 466
  rt_event_wait, 468
  rt_event_wait_timed, 468
  rt_event_wait_until, 470
Event Services, 201
  rtdm_event_clear, 201
  rtdm_event_destroy, 202
  rtdm_event_init, 202
  rtdm_event_pULSE, 203
  rtdm_event_select, 203
  rtdm_event_signal, 204
  rtdm_event_timedwait, 204
  rtdm_event_wait, 205

fd
  a4l_descriptor, 567
fd.h
  rtdm_fd_get, 651
  rtdm_fd_lock, 652
  rtdm_fd_put, 653
  rtdm_fd_select, 653
  rtdmFd_unlock, 654
flags
  a4l_channel, 563
  a4l_range, 571
get
  xnvfile_lock_ops, 612
get_unmapped_area
  rtdm_fd_ops, 594
getpeername__AF_RTIPC
  Real-time IPC, 88
getsockname__AF_RTIPC
  Real-time IPC, 88
getssockopt__AF_RTIPC
  Real-time IPC, 89
H_PRIO
  Heap management services, 472
Heap management services, 471
  H_PRIO, 472
    rt_heap_alloc, 472
    rt_heap_alloc_timed, 472
    rt_heap_alloc_until, 473
    rt_heap_bind, 474
    rt_heap_create, 475
    rt_heap_delete, 476
    rt_heap_free, 476
    rt_heap_inquire, 477
    rt_heap_unbind, 478
heapsize
  RT_HEAP_INFO, 582
htimer
  xnsched, 610
IDDP_LABEL
  Real-time IPC, 78
IDDP_POOLSZ
  Real-time IPC, 79
idx_read_subd
  a4l_descriptor, 567
idx_subd
  a4l_cmd_desc, 566
  a4l_instruction, 570
idx_write_subd
  a4l_descriptor, 567
In-kernel arithmetics, 96
  xnarch_generic_full_divmod64, 96
include/cobalt/kernel/rtdm/analogy/channel_.--range.h, 629
include/cobalt/kernel/rtdm/analogy/driver.h, 632
include/cobalt/kernel/rtdm/analogy/subdevice.h, 641
include/cobalt/kernel/rtdm/driver.h, 633
include/cobalt/kernel/rtdm/fd.h, 650
include/cobalt/kernel/rtdm/udd.h, 672
include/rtdm/analogy.h, 675
include/rtdm/can.h, 642
include/rtdm/rtdm.h, 658
include/rtdm/rtdm/analogynivel.h, 660
include/rtdm/rtdm/testing.h, 669
include/rtdm/uapi/analogy.h, 676
include/rtdm/uapi/can.h, 643
include/rtdm/uapi/ipc.h, 654
include/rtdm/uapi/rtdm.h, 658
include/rtdm/uapi/serial.h, 661
include/rtdm/uapi/testing.h, 670
include/rtdm/uapi/udd.h, 674
inesting
  xnsched, 610
interrupt
  udd_device, 604
Interrupt Management services, 119
  xnintr_affinity, 119
  xnintr_attach, 120
  xnintr_destroy, 120
  xnintr_detach, 121
  xnintr_disable, 121
  xnintr_enable, 122
  xnintr_init, 122
Interrupt Management Services, 215
  rtdm_irq_disable, 218
  rtdm_irq_enable, 218
  rtdm_irq_free, 219
  rtdm_irq_get_arg, 216
  rtdm_irq_handler_t, 216
  rtdm_irq_request, 219
Interrupt management services, 306
  a4l_free_irq, 306
  a4l_get_irq, 307
  a4l_request_irq, 307
ioctl
  udd_device, 604
ioctl_nrt
  rtdm_fd_ops, 594
ioctl_rt
  rtdm_fd_ops, 594
irq
  udd_device, 605
label
  rtdm_device, 589
  rtipc_port_label, 598
len
  udd_memregion, 607
length
  a4l_channels_desc, 564
Level 0 API, 430
Level 1 API, 419
Level 2 API, 424
lflags
  xnsched, 611
lib/analogy/async.c, 683
lib/analogy/calibration.c, 684
lib/analogy/calibration.h, 681
lib/analogy/descriptor.c, 685
lib/analogy/internal.h, 682
lib/analogy/range.c, 686
lib/analogy/root_leaf.h, 687
lib/analogy/sync.c, 688
lib/analogy/sys.c, 689
Lightweight key-to-object mapping service, 127
  xnmap_create, 127
  xnmap_delete, 128
xnmap_enter, 129
xnmap_fetch, 129
xnmap_fetch_nocheck, 130
xnmap_remove, 130

Locking services, 125
splexit, 125
splhigh, 126
splitext, 126

macb_dma_desc, 579
macb_tx_skb, 579

magic
a4l_descriptor, 568

Math API, 410
a4l_math_mean, 410
a4l_math_polyfit, 411
a4l_math_stddev, 411
a4l_math_stddev_of_mean, 412

max
a4l_range, 571

mem_regions
udd_device, 605

Message pipe services, 488
P_MINOR_AUTO, 489
P_URGENT, 489
rt_pipe_bind, 489
rt_pipe_create, 490
rt_pipe_delete, 491
rt_pipe_read, 492
rt_pipe_read_timed, 492
rt_pipe_read_until, 493
rt_pipe_stream, 494
rt_pipe_unbind, 495
rt_pipe_write, 495

Message queue services, 497
Q_PRIQ, 498
rt_queue_alloc, 498
rt_queue_bind, 499
rt_queue_create, 500
rt_queue_delete, 501
rt_queue_flush, 501
rt_queue_free, 502
rt_queue_inquire, 503
rt_queue_read, 503
rt_queue_read_timed, 504
rt_queue_read_until, 505
rt_queue_receive, 505
rt_queue_receive_timed, 506
rt_queue_receive_until, 507
rt_queue_send, 507
rt_queue_unbind, 508

Message queues, 333
mq_close, 334
mq_getattr, 334
mq_notify, 335
mq_open, 336
mq_receive, 337
mq_send, 338
mq_setattr, 339
mq_timedreceive, 340
mq_timedsend, 341
mq_unlink, 342

min
a4l_range, 571

minor
rtdm_device, 589

Misc services, 309
a4l_get_time, 309

mmap
rtdm_fd_ops, 594
udd_device, 605

mode
a4l_channels_desc, 565

mq_close
Message queues, 334
mq_getattr
Message queues, 334
mq_notify
Message queues, 335
mq_open
Message queues, 336
mq_receive
Message queues, 337
mq_send
Message queues, 338
mq_setattr
Message queues, 339
mq_timedreceive
Message queues, 340
mq_timedsend
Message queues, 341
mq_unlink
Message queues, 342

Mutex services, 211, 481
rt_mutex_acquire, 482
rt_mutex_acquire_timed, 482
rt_mutex_acquire_until, 483
rt_mutex_bind, 484
rt_mutex_create, 484
rt_mutex_delete, 485
rt_mutex_inquire, 486
rt_mutex_release, 486
rt_mutex_unlock, 487
rt_mutex_unbind, 487
rtdm_mutex_destroy, 211
rtdm_mutex_init, 212
rtdm_mutex_lock, 212
rtdm_mutex_timedlock, 213
rtdm_mutex_unlock, 213

Mutual exclusion, 344
pthread_mutex_destroy, 345
pthread_mutex_init, 346
pthread_mutex_lock, 346
pthread_mutex_timedlock, 347
pthread_mutex_trylock, 348
pthread_mutex_unlock, 349
pthread_mutexattr_destroy, 350
pthread_mutexattr_getprotocol, 350

Generated by Doxygen
pthread_mutexattr_getpshared, 351
pthread_mutexattr_gettype, 352
pthread_mutexattr_init, 353
pthread_mutexattr_setprotocol, 353
pthread_mutexattr_setpshared, 354
pthread_mutexattr_settype, 355

nanosecs_abs_t
  RTDM, 68
nanosecs_rel_t
  RTDM, 69
nanosleep
  Clocks and timers, 314
nb_bits
  a4l_channel, 563
  a4l_descriptor, 568
next
  xnvfile_regular_ops, 615
  xnvfile_snapshot_ops, 623
Non-Real-Time Signalling Services, 221
  rtdm_nrtsig_destroy, 222
  rtdm_nrtsig_handler_t, 221
  rtdm_nrtsig_init, 222
  rtdm_nrtsig_pend, 223
  rtdm_schedule_nrt_work, 223
nrdata
  xnvfile_snapshot_iterator, 621
open
  rtdm_fd_ops, 595
  udd_device, 605
owner
  RT_MUTEX_INFO, 583
P_MINOR_AUTO
  Message pipe services, 489
P_URGENT
  Message pipe services, 489
POSIX interface, 331
pSOS® emulator, 552
pid
  udd_signotify, 609
pos
  xnvfile_regular_iterator, 613
private
  xnvfile_regular_iterator, 613
  xnvfile_snapshot_iterator, 621
Process scheduling, 357
  sched_get_priority_max, 358
  sched_get_priority_max_ex, 358
  sched_get_priority_min, 359
  sched_get_priority_min_ex, 360
  sched_getconfig_np, 360
  sched_getscheduler, 361
  sched_getscheduler_ex, 362
  sched_getconfig_np, 363
  sched_setscheduler, 365
  sched_setscheduler_ex, 366
  sched_yield, 367
profile_info
  rtdm_driver, 592
program_htick_shot
  Timer services, 272
pthread_cond_broadcast
  Condition variables, 321
pthread_cond_destroy
  Condition variables, 322
pthread_cond_init
  Condition variables, 322
pthread_cond_signal
  Condition variables, 323
pthread_cond_timedwait
  Condition variables, 324
pthread_cond_wait
  Condition variables, 324
pthread_condattr_destroy
  Condition variables, 325
pthread_condattr_getclock
  Condition variables, 326
pthread_condattr_getpshared
  Condition variables, 327
pthread_condattr_init
  Condition variables, 327
pthread_condattr_setclock
  Condition variables, 328
pthread_condattr_setpshared
  Condition variables, 329
pthread_create
  Thread management, 376
pthread_getschedparam
  Scheduling management, 382
pthread_getschedparam_ex
  Scheduling management, 383
pthread_join
  Thread management, 377
pthread_kill
  Thread management, 378
pthread_make_periodic_np
  Transition Kit, 560
pthread_mutex_destroy
  Mutual exclusion, 345
pthread_mutex_init
  Mutual exclusion, 346
pthread_mutex_lock
  Mutual exclusion, 346
pthread_mutex_timedlock
  Mutual exclusion, 347
pthread_mutextrylock
  Mutual exclusion, 348
pthread_mutex_unlock
  Mutual exclusion, 349
pthread_mutexattr_destroy
  Mutual exclusion, 350
pthread_mutexattr_getprotocol
  Mutual exclusion, 350
pthread_mutexattr_setpshared
  Mutual exclusion, 350
Mutual exclusion, 351
pthread_mutexattr_gettype
    Mutual exclusion, 352
pthread_mutexattr_init
    Mutual exclusion, 353
pthread_mutexattr_setprotocol
    Mutual exclusion, 354
pthread_mutexattr_setpshared
    Mutual exclusion, 355
pthread_mutexattr_settype
    Mutual exclusion, 355
pthread_setmode_np
    Thread management, 379
pthread_setname_np
    Thread management, 380
pthread_setschedparam
    Scheduling management, 384
pthread_setschedparam_ex
    Scheduling management, 385
pthread_wait_np
    Transition Kit, 561
pthread_yield
    Scheduling management, 386
put
    xvfile_lock_ops, 612
Q_PRIO
    Message queue services, 498
RT_ALARM_INFO, 580
RT_BUFFER_INFO, 580
RT_COND_INFO, 581
RT_EVENT_INFO, 581
RT_HEAP_INFO, 582
    heapsize, 582
    usablemem, 582
    usedmem, 583
RT_MUTEX_INFO, 583
    owner, 583
RT_QUEUE_INFO, 584
RT_SEM_INFO, 584
RT_TASK_INFO, 585
RT_TIMER_INFO
    Timer management services, 547
RTCAN_RTIOC_RCV_TIMEOUT
    CAN Devices, 56
RTCAN_RTIOC_SND_TIMEOUT
    CAN Devices, 57
RTCAN_RTIOC_TAKE_TIMESTAMP
    CAN Devices, 58
RTDM_User_API, 70
RTDM_CLASS_MAGIC
    driver.h, 638
RTDM_FIXED_MINOR
    Device Registration Services, 159
RTDM_MAX_MINOR
    Device Registration Services, 160
RTDM_NAMED_DEVICE
    Device Registration Services, 160
RTDM_PROFILE_INFO
    driver.h, 638
RTDM_PROTOCOL_DEVICE
    Device Registration Services, 160
RTDM_TIMEOUT_INFINITE
    RTDM, 68
RTDM_TIMEOUT_NONE
    RTDM, 68
RTDM
    nanosecs_abs_t, 68
    nanosecs_rel_t, 69
    RTDM_TIMEOUT_INFINITE, 68
    RTDM_TIMEOUT_NONE, 68
RTIOC_DEVICE_INFO
    Device Profiles, 157
RTIOC_PURGE
    Device Profiles, 157
RTSER_RTIOC_BREAK_CTL
    rtdm/uapi/serial.h, 666
RTSER_RTIOC_GET_CONFIG
    rtdm/uapi/serial.h, 666
RTSER_RTIOC_GET_CONTROL
    rtdm/uapi/serial.h, 666
RTSER_RTIOC_GET_STATUS
    rtdm/uapi/serial.h, 667
RTSER_RTIOC_SET_CONFIG
    rtdm/uapi/serial.h, 667
RTSER_RTIOC_SET_CONTROL
    rtdm/uapi/serial.h, 668
RTSER_RTIOC_SET_TIMEOUT
    rtdm/uapi/serial.h, 668
RTSER_RTIOC_WAIT_EVENT
    rtdm/uapi/serial.h, 668
Range / conversion API, 413
    a4l_dtoraw, 413
    a4l_find_range, 414
    a4l_ftoraw, 415
    a4l_rawtod, 415
    a4l_rawtof, 416
    a4l_rawtoul, 416
    a4l_sizeof_chan, 417
    a4l_sizeof_subd, 417
    a4l_ultoraw, 418
    read_nrt
    rtdm_fd_ops, 595
read_rt
    rtdm_fd_ops, 595
Real-time IPC, 74
    BUFP_BUFSIZE, 76
    BUFP_LABEL, 77
    bind__AF_RTIPC, 85
    close__AF_RTIPC, 87
    connect__AF_RTIPC, 87
    getpeername__AF_RTIPC, 88
INDEX 741

Event flag group services, 470
rt_heap_alloc
Heap management services, 472
rt_heap_alloc_timed
Heap management services, 472
rt_heap_alloc_until
Heap management services, 473
rt_heap_bind
Heap management services, 474
rt_heap_create
Heap management services, 475
rt_heap_delete
Heap management services, 476
rt_heap_free
Heap management services, 476
rt_heap_inquire
Heap management services, 477
rt_heap_unbind
Heap management services, 478
rt_mutex_acquire
Mutex services, 482
rt_mutex_acquire_timed
Mutex services, 482
rt_mutex_acquire_until
Mutex services, 483
rt_mutex_bind
Mutex services, 484
rt_mutex_create
 Mutex services, 484
rt_mutex_delete
 Mutex services, 485
rt_mutex_inquire
 Mutex services, 486
rt_mutex_release
 Mutex services, 486
rt_mutex_unbind
 Mutex services, 487
rt_pipe_bind
Message pipe services, 489
rt_pipe_create
Message pipe services, 490
rt_pipe_delete
Message pipe services, 491
rt_pipe_read
Message pipe services, 492
rt_pipe_read_timed
Message pipe services, 492
rt_pipe_read_until
Message pipe services, 493
rt_pipe_stream
Message pipe services, 494
rt_pipe_unbind
Message pipe services, 495
rt_pipe_write
Message pipe services, 495
rt_queue_alloc
Message queue services, 498
rt_queue_bind
Message queue services, 499
rt_queue_create
Message queue services, 500
rt_queue_delete
Message queue services, 501
rt_queue_free
Message queue services, 501
rt_queue_read
Message queue services, 502
rt_queue_inquire
Message queue services, 503
rt_queue_unbind
Message queue services, 503
rt_queue_read_timed
Message queue services, 504
rt_queue_read_until
Message queue services, 505
rt_queue_receive
Message queue services, 505
rt_queue_receive_timed
Message queue services, 506
rt_queue_receive_until
Message queue services, 507
rt_queue_send
Message queue services, 507
rt_queue_unbind
Message queue services, 508
rt_sem_bind
Semaphore services, 511
rt_sem_broadcast
Semaphore services, 512
rt_sem_create
Semaphore services, 512
rt_sem_delete
Semaphore services, 513
rt_sem_inquire
Semaphore services, 515
rt_sem_p
Semaphore services, 515
rt_sem_p_timed
Semaphore services, 516
rt_sem_p_until
Semaphore services, 517
rt_sem_unbind
Semaphore services, 517
rt_sem_v
Semaphore services, 518
rt_task_bind
Task management services, 521
rt_task_create
Task management services, 522
rt_task_delete
Task management services, 523
rt_task_inquire
Task management services, 524
rt_task_join
Task management services, 525
rt_task_receive
rtdm_event_signal
    Event Services, 204
rtdm_event_timedwait
    Event Services, 204
rtdm_event_wait
    Event Services, 205
rtdm_fd_device
    driver.h, 639
rtdm_fd_get
    fd.h, 651
rtdm_fd_is_user
    driver.h, 639
rtdm_fd_lock
    fd.h, 652
rtdm_fd_ops, 593
    close, 594
get_unmapped_area, 594
ioctl_nrt, 594
ioctl_rt, 594
mmap, 594
open, 595
read_nrt, 595
read_rt, 595
recvmsg_nrt, 595
recvmsg_rt, 595
select, 595
sendmsg_nrt, 596
sendmsg_rt, 596
socket, 596
write_nrt, 596
write_rt, 596
rtdm_fd_put
    fd.h, 653
rtdm_fd_select
    fd.h, 653
rtdm_fd_to_private
    driver.h, 640
rtdm_fd_unlock
    fd.h, 654
rtdm_for_each_waiter
    Synchronisation Services, 189
rtdm_for_each_waiter_safe
    Synchronisation Services, 189
rtdm_free
    Utility Services, 226
rtdm_get_unmapped_area_handler
    Device Registration Services, 162
rtdm_getpeername
    Driver to driver services, 141
rtdm_getsockname
    Driver to driver services, 142
rtdm_getsockopt
    Driver to driver services, 142
rtdm_in_rt_context
    Utility Services, 227
rtdm_ioctl
    Driver to driver services, 143
rtdm_ioctl_handler
    Utility Services, 227
rtdm_ioctl_handler_t
    Utility Services, 227
rtdm_ioctl
    Interrupt Management Services, 218
rtdm_ioctl_handler
    Interrupt Management Services, 218
rtdm_ioctl_handler
    Interrupt Management Services, 219
rtdm_ioctl_handler
    Interrupt Management Services, 216
rtdm_ioctl_handler
    Interrupt Management Services, 216
rtdm_ioctl
    Interrupt Management Services, 219
rtdm_ioctl
    Driver to driver services, 144
rtdm_lock_get
    Spinlock with preemption deactivation, 29
rtdm_lock_get_irqsave
    Spinlock with preemption deactivation, 27
rtdm_lock_init
    Spinlock with preemption deactivation, 29
rtdm_lock_irqrestore
    Spinlock with preemption deactivation, 28
rtdm_lock_irqsave
    Spinlock with preemption deactivation, 28
rtdm_lock_put
    Spinlock with preemption deactivation, 29
rtdm_lock_put_irqrestore
    Spinlock with preemption deactivation, 30
rtdm_malloc
    Utility Services, 228
rtdm_mmap_handler
    Device Registration Services, 164
rtdm_mmap_iomem
    Utility Services, 228
rtdm_mmap_kmem
    Utility Services, 229
rtdm_mmap_to_user
    Utility Services, 230
rtdm_mmap_vmem
    Utility Services, 231
rtdm_munmap
    Utility Services, 231
rtdm_mutex_destroy
    Mutex services, 211
rtdm_mutex_init
    Mutex services, 212
rtdm_mutex_lock
    Mutex services, 212
rtdm_mutex_timedlock
    Mutex services, 213
rtdm_mutex_unlock
    Mutex services, 213
rtdm_nrtsig_destroy
    Non-Real-Time Signalling Services, 222
rtdm_nrtsig_handler_t
    Non-Real-Time Signalling Services, 222
INDEX

Timer Services, 183
rtdm_timer_handler_t
  Timer Services, 182
rtdm_timer_init
  Timer Services, 183
rtdm_timer_mode
  Timer Services, 183
rtdm_timer_start
  Timer Services, 184
rtdm_timer_start_in_handler
  Timer Services, 185
rtdm_timer_stop
  Timer Services, 185
rtdm_timer_stop_in_handler
  Timer Services, 186
rtdm_toseq_init
  Synchronisation Services, 193
rtdm_wait
  Synchronisation Services, 194
rtdm_wait_complete
  Task Services, 180
rtdm_wait_condition
  Synchronisation Services, 194
rtdm_wait_condition_locked
  Synchronisation Services, 195
rtdm_wait_is_completed
  Task Services, 180
rtdm_wait_locked
  Synchronisation Services, 196
rtdm_wait_prepare
  Task Services, 180
rtdm_waitqueue_broadcast
  Synchronisation Services, 196
rtdm_waitqueue_destroy
  Synchronisation Services, 197
rtdm_waitqueue_flush
  Synchronisation Services, 197
rtdm_waitqueue_init
  Synchronisation Services, 198
rtdm_waitqueue_lock
  Synchronisation Services, 198
rtdm_waitqueue сигнал
  Synchronisation Services, 199
rtdm_waitqueue_unlock
  Synchronisation Services, 199
rtdm_waitqueue_wakeup
  Synchronisation Services, 200
rtdm_write
  Driver to driver services, 153
rtdm_write_handler
  Device Registration Services, 168
rtipc_port_label, 597
  label, 598
rtser_config, 598
rtser_event, 599
rtser_status, 600
S_PRIORITY
  Semaphore services, 511
SCHED_QUOTA scheduling policy, 238
SIOCGCANBAUDRATE
  CAN Devices, 58
SIOCGCANCTRLMODE
  CAN Devices, 59
SIOCGCANCUSTOMBITTIMEMODE
  CAN Devices, 59
SIOCGCANSTATE
  CAN Devices, 60
SIOCGFINDEX
  CAN Devices, 61
SIOCSCANBAUDRATE
  CAN Devices, 61
SIOCSCANCTRLMODE
  CAN Devices, 62
SIOCSCANCUSTOMBITTIMEMODE
  CAN Devices, 63
SIOCSCANMODE
  CAN Devices, 63
SO_RCVTIMEO
  Real-time IPC, 79
SO_SNDTIMEO
  Real-time IPC, 80
S_CAN_RAW
  CAN Devices, 64
sdata
  a4l_descriptor, 568
sbsize
  a4l_descriptor, 568
sched_get_priority_max
  Process scheduling, 358
sched_get_priority_max_ex
  Process scheduling, 358
sched_get_priority_min
  Process scheduling, 359
sched_get_priority_min_ex
  Process scheduling, 360
sched_getconfig_np
  Process scheduling, 360
sched_getscheduler
  Process scheduling, 361
sched_getscheduler_ex
  Process scheduling, 362
sched_setconfig_np
  Process scheduling, 362
sched_setscheduler
  Process scheduling, 363
sched_setscheduler_ex
  Process scheduling, 365
sched_setconfig_np
  Process scheduling, 366
sched_yield
  Process scheduling, 367
Scheduling management, 382
pthread_getschedparam
  Process scheduling, 382
pthread_getschedparam_ex
  Process scheduling, 383
pthread_setscheduler
  Process scheduling, 384
pthread_setscheduler_ex
  Process scheduling, 385
pthread_yield
  Process scheduling, 386
select
sem_close
Semaphores, 368
sem_destroy
Semaphores, 369
sem_init
Semaphores, 370
sem_post
Semaphores, 371
sem_timedwait
Semaphores, 372
sem_trywait
Semaphores, 373
sem_unlink
Semaphores, 374
sem_wait
Semaphores, 375
Semaphore Services, 207
  rtdm_sem_destroy, 207
  rtdm_sem_down, 208
  rtdm_sem_init, 208
  rtdm_sem_select, 209
  rtdm_sem_timeddown, 209
  rtdm_sem_up, 210
Semaphore services, 510
  rt_sem_bind, 511
  rt_sem_broadcast, 512
  rt_sem_create, 512
  rt_sem_delete, 513
  rt_sem_inquire, 515
  rt_sem_p, 515
  rt_sem_p_timed, 516
  rt_sem_p_until, 517
  rt_sem_unbind, 517
  rt_sem_v, 518
S_PRIQ, 511
Semaphores, 368
  sem_close, 368
  sem_destroy, 369
  sem_init, 370
  sem_post, 371
  sem_timedwait, 372
  sem_trywait, 373
  sem_unlink, 374
  sem_wait, 375
sendmsg__AF_RTIPC
Real-time IPC, 90
sendmsg_nrt
rtdm_fd_ops, 596
sendmsg_rt
rtdm_fd_ops, 596
seq
  xnvfile_regular_iterator, 613
  xnvfile_snapshot_iterator, 621
Serial Devices, 71
setsockopt__AF_RTIPC
Real-time IPC, 91
show
  xnvfile_regular_ops, 616
  xnvfile_snapshot_ops, 625
sig
  udd_signotify, 609
sipc_port
  sockaddr_ipc, 602
Smookey API, 387
sockaddr_can, 600
  can_ifindex, 601
sockaddr_ipc, 601
  sipc_port, 602
socket
  rtdm_fd_ops, 596
socket__AF_RTIPC
Real-time IPC, 91
Software calibration API, 401
  a4l_dcaltoraw, 401
  a4l_get_softcal_converter, 402
  a4l_rtwdcal, 402
  a4l_read_calibration_file, 403
Spinlock with preemption deactivation, 27
  rtdm_lock_get, 29
  rtdm_lock_get_irqsave, 27
  rtdm_lock_init, 29
  rtdm_lock_irqrestore, 28
  rtdm_lock_irqsave, 28
  rtdm_lock_put, 29
  rtdm_lock_put_irqrestore, 30
splexit
  Locking services, 125
splhigh
  Locking services, 126
spltest
  Locking services, 126
status
  xnsched, 611
store
  xnvfile_regular_ops, 618
  xnvfile_snapshot_ops, 625
Subdevice management services, 292
  a4l_add_subd, 294
  a4l_alloc_subd, 294
  a4l_get_subd, 295
switch_htick_mode
  Timer services, 272
Synchronisation Services, 187
  rtdm_for_each_waiter, 189
  rtdm_for_each_waiter_safe, 189
  rtdm_selecttype, 188
  rtdm_timedwait, 190
  rtdm_timedwait_condition, 191
  rtdm_timedwait_condition_locked, 191
  rtdm_timedwait_locked, 192
  rtdm_toseq_init, 193
  rtdm_wait, 194
  rtdm_wait_condition, 194
  rtdm_wait_condition_locked, 195
  rtdm_wait_locked, 196
udd_unregister_device
User-space driver core, 38
usablemem
RT_HEAP_INFO, 582
usedmem
RT_HEAP_INFO, 583
User-space driver core, 31
UDD_IRQ_CUSTOM, 32
UDD_IRQ_NONE, 33
UDD_MEM_LOGICAL, 33
UDD_MEM_NONE, 33
UDD_MEM_PHYS, 33
UDD_MEM_VIRTUAL, 33
UDD_RTIOC_IRQDIS, 34
UDD_RTIOC_IRQEN, 34
UDD_RTIOC_IRQSIG, 34
udd_disable_irq, 35
udd_enable_irq, 35
udd_get_device, 36
udd_notify_event, 36
udd_register_device, 37
udd_unregister_device, 38
Utility Services, 224
rtdm_copy_from_user, 225
rtdm_copy_to_user, 225
rtdm_free, 226
rtdm_in_rt_context, 227
rtdm_iomap_to_user, 227
rtdm_malloc, 228
rtdm_mmap_iomem, 228
rtdm_mmap_kmem, 229
rtdm_mmap_to_user, 230
rtdm_mmap_vmem, 231
rtdm_munmap, 231
rtdm_printk, 232
rtdm_printk_ratelimited, 232
rtdm_ratelimit, 233
rtdm_read_user_ok, 234
rtdm_rt_capable, 234
rtdm_rwlock_user_ok, 235
rtdm_safe_copy_from_user, 235
rtdm_safe_copy_to_user, 236
rtdm_strncpy_from_user, 237

vfile
xnvfile_regular_iterator, 614
xnvfile_snapshot_iterator, 621
Virtual file services, 280
cobalt_vfroot, 287
xnvfile_destroy, 281
xnvfile_get_blob, 282
xnvfile_get_integer, 282
xnvfile_get_string, 283
xnvfile_init_dir, 284
xnvfile_init_link, 284
xnvfile_init_regular, 285
xnvfile_init_snapshot, 286
VxWorks® emulator, 551

write_nrt
rtdm_fd_ops, 596
write_rt
rtdm_fd_ops, 596
XDDP_BUFSZ
Real-time IPC, 80
XDDP_EVTDOWN
Real-time IPC, 81
XDDP_EVTIN
Real-time IPC, 81
XDDP_EVTNOBUF
Real-time IPC, 81
XDDP_EVTOUT
Real-time IPC, 82
XDDP_LABEL
Real-time IPC, 82
XDDP_MONITOR
Real-time IPC, 83
XDDP_POOLSZ
Real-time IPC, 83
XNHELD
Thread state flags, 40
XNMIGRATE
Thread state flags, 40
XNPEND
Thread state flags, 40
XNREADY
Thread state flags, 40
XNSUSP
Thread state flags, 41
XNTRAPLB
Thread state flags, 41
xnapc_alloc
Asynchronous Procedure Calls, 93
xnapc_free
Asynchronous Procedure Calls, 94
xnapc_schedule
Asynchronous Procedure Calls, 94
xnarch_generic_full_divmod64
In-kernel arithmetics, 96
xnbufd_copy_from_kmem
Buffer descriptor, 99
xnbufd_copy_to_kmem
Buffer descriptor, 100
xnbufd_invalidate
Buffer descriptor, 101
xnbufd_map_kread
Buffer descriptor, 101
xnbufd_map_kwrite
Buffer descriptor, 102
xnbufd_map_uread
Buffer descriptor, 102
xnbufd_map_uwrite
Buffer descriptor, 103
xnbufd_reset
Buffer descriptor, 103
xnbufd_unmap_kread
Buffer descriptor, 104
xnbufd_unmap_kwrite
Buffer descriptor, 104

xnbufd_unmap_uread
Buffer descriptor, 105

xnbufd_unmap_uwrite
Buffer descriptor, 105

xclock_adjust
Clock services, 107

xclock_deregister
Clock services, 108

xclock_register
Clock services, 108

xclock_tick
Clock services, 109

xnheap::xnbucket, 609

xnheap::xnheap_alloc
Dynamic memory allocation services, 112

xnheap_destroy
Dynamic memory allocation services, 112

xnheap_free
Dynamic memory allocation services, 113

xnheap_init
Dynamic memory allocation services, 113

xnheap_set_name
Dynamic memory allocation services, 114

xinintr_affinity
Interrupt management, 119

xinintr_attach
Interrupt management, 120

xinintr_destroy
Interrupt management, 120

xinintr_detach
Interrupt management, 121

xinintr_disable
Interrupt management, 121

xinintr_enable
Interrupt management, 122

xinintr_init
Interrupt management, 122

xnmnap_create
Lightweight key-to-object mapping service, 127

xnmnap_delete
Lightweight key-to-object mapping service, 128

xnmnap_enter
Lightweight key-to-object mapping service, 129

xnmnap_fetch
Lightweight key-to-object mapping service, 129

xnmnap_fetch_nocheck
Lightweight key-to-object mapping service, 130

xnmnap_remove
Lightweight key-to-object mapping service, 130

xnregistry_bind
Registry services, 132

xnregistry_enter
Registry services, 133

xnregistry_lookup
Registry services, 134

xnregistry_remove
Registry services, 135

xnregistry_unlink
Registry services, 135

xnsched, 610
cpu, 610
curr, 610
htimer, 610
inesting, 610
lflags, 611
resched, 611
rbitimer, 611
rt, 611
status, 611

xnsched_rotate
Thread scheduling control, 239

xnsched_run
Thread scheduling control, 240

xnselct
Synchronous I/O multiplexing, 242

xnselct_bind
Synchronous I/O multiplexing, 242

xnselct_destroy
Synchronous I/O multiplexing, 243

xnselct_init
Synchronous I/O multiplexing, 244

xnselct_signal
Synchronous I/O multiplexing, 244

xnselctor_destroy
Synchronous I/O multiplexing, 244

xnselctor_init
Synchronous I/O multiplexing, 245

xnsynch_acquire
Thread synchronization services, 246

xnsynch_destroy
Thread synchronization services, 247

xnsynch_flush
Thread synchronization services, 248

xnsynch_init
Thread synchronization services, 249

xnsynch_peek_pendq
Thread synchronization services, 250

xnsynch_release
Thread synchronization services, 250

xnsynch_sleep_on
Thread synchronization services, 251

xnsynch_try_acquire
Thread synchronization services, 252

xnsynch_wakeup_one_sleeper
Thread synchronization services, 252

xnsynch_wakeup_this_sleeper
Thread synchronization services, 253

xnthread_cancel

Generated by Doxygen
Thread services, 256
xnthread_current
Thread services, 257
xnthread_from_task
Thread services, 257
xnthread_harden
Thread services, 257
xnthread_init
Thread services, 258
xnthread_join
Thread services, 259
xnthread_map
Thread services, 260
xnthread_migrate
Thread services, 261
xnthread_relax
Thread services, 261
xnthread_resume
Thread services, 262
xnthread_set_mode
Thread services, 263
xnthread_set_periodic
Thread services, 264
xnthread_set_schedparam
Thread services, 265
xnthread_set_slice
Thread services, 266
xnthread_start
Thread services, 266
xnthread_suspend
Thread services, 267
xnthread_test_cancel
Thread services, 268
xnthread_unblock
Thread services, 269
xnthread_wait_period
Thread services, 269
xntimer_destroy
Timer services, 273
xntimer_get_date
Timer services, 274
xntimer_get_overruns
Timer services, 274
xntimer_get_timeout
Timer services, 275
xntimer_grab_hardware
Timer services, 275
xntimer_init
Timer services, 276
xntimer_interval
Timer services, 277
xntimer_release_hardware
Timer services, 277
xntimer_start
Timer services, 277
xntimer_stop
Timer services, 278
xnvfile_destroy
Virtual file services, 281
xnvfile_get_blob
Virtual file services, 282
xnvfile_get_integer
Virtual file services, 282
xnvfile_get_string
Virtual file services, 283
xnvfile_init_dir
Virtual file services, 284
xnvfile_init_link
Virtual file services, 284
xnvfile_init_regular
Virtual file services, 284
xnvfile_init_snapshot
Virtual file services, 285
xnvfile_init_lock_ops
Virtual file services, 286
xnvfile_regular_iterator
Virtual file services, 613
pos, 613
private, 613
seq, 613
vfile, 614
xnvfile_regular_ops
Virtual file services, 614
begin, 614
de, 615
next, 615
rewind, 616
show, 616
store, 618
xnvfile_rev_tag
Virtual file services, 619
rev, 619
xnvfile_snapshot
Virtual file services, 619
xnvfile_snapshot_iterator
Virtual file services, 620
databuf, 621
dfn, 621
ndata, 621
private, 621
seq, 621
vfile, 621
xnvfile_snapshot_ops
Virtual file services, 622
begin, 622
de, 623
next, 623
rewind, 624
show, 625
store, 625