# Contents

1 Deprecated List ............................................ 1

2 Module Index .............................................. 3
   2.1 Modules ............................................... 3

3 Data Structure Index ...................................... 5
   3.1 Data Structures ....................................... 5

4 File Index ................................................ 7
   4.1 File List ............................................... 7

5 Module Documentation ...................................... 9
   5.1 CAN Devices ........................................... 9
      5.1.1 Detailed Description ............................. 15
      5.1.2 Macro Definition Documentation ................ 17
         5.1.2.1 CAN_CTRLMODE_3_SAMPLES .................. 17
         5.1.2.2 CAN_CTRLMODE_LISTENONLY ................ 18
         5.1.2.3 CAN_CTRLMODE_LOOPBACK ................... 18
         5.1.2.4 CAN_ERR_LOSTARB_UNSPEC .................. 18
         5.1.2.5 CAN_RAW_ERR_FILTER ......................... 18
         5.1.2.6 CAN_RAW_FILTER ................................ 19
         5.1.2.7 CAN_RAW_LOOPBACK ........................... 19
         5.1.2.8 CAN_RAW_RECV_OWN_MSGS .................... 20
         5.1.2.9 RTCAN_RTIOC_RCV_TIMEOUT ................... 20
         5.1.2.10 RTCAN_RTIOC_SND_TIMEOUT .................. 20
         5.1.2.11 RTCAN_RTIOC_TAKE_TIMESTAMP ............... 21
         5.1.2.12 SIOCGCANBAUDRATE ......................... 22
         5.1.2.13 SIOCGCANCTRLMODE ......................... 22
         5.1.2.14 SIOCGCANCUSTOMBITTIME .................... 23
         5.1.2.15 SIOCGCANSTATE .............................. 23
         5.1.2.16 SIOCGIFINDEX ............................... 24
         5.1.2.17 SIOCSCANBAUDRATE ......................... 24
         5.1.2.18 SIOCSCANCTRLMODE ......................... 25
5.5.2.6 RTSER_RTIOC_SET_CONTROL ........................................... 69
5.5.2.7 RTSER_RTIOC_WAIT_EVENT ........................................... 69

5.6 Testing Devices .............................................................. 71
5.6.1 Detailed Description ....................................................... 72

5.7 Inter-Driver API .............................................................. 73
5.7.1 Detailed Description ....................................................... 74
5.7.2 Function Documentation .................................................. 74
  5.7.2.1 rtdm_accept ............................................................... 74
  5.7.2.2 rtdm_bind ................................................................. 74
  5.7.2.3 rtdm_close ............................................................... 74
  5.7.2.4 rtdm_connect ............................................................. 74
  5.7.2.5 rtdm_context_get ....................................................... 75
  5.7.2.6 rtdm_context_lock ..................................................... 76
  5.7.2.7 rtdm_context_put ....................................................... 76
  5.7.2.8 rtdm_context_unlock ................................................ 77
  5.7.2.9 rtdm_getpeername ..................................................... 77
  5.7.2.10 rtdm_getsockname ................................................... 78
  5.7.2.11 rtdm_getsockopt ...................................................... 78
  5.7.2.12 rtdm_ioctl ............................................................ 78
  5.7.2.13 rtdm_listen ........................................................... 78
  5.7.2.14 rtdm_open ............................................................. 78
  5.7.2.15 rtdm_read ............................................................ 78
  5.7.2.16 rtdm_recv ............................................................ 79
  5.7.2.17 rtdm_recvfrom ....................................................... 79
  5.7.2.18 rtdm_recvmsg ......................................................... 79
  5.7.2.19 rtdm_select_bind ................................................... 79
  5.7.2.20 rtdm_send ............................................................ 80
  5.7.2.21 rtdm_sendmsg ......................................................... 80
  5.7.2.22 rtdm_sendsdto ......................................................... 80
  5.7.2.23 rtdm_setsockopt ...................................................... 81
  5.7.2.24 rtdm_shutdown ....................................................... 81
  5.7.2.25 rtdm_socket ........................................................ 81
  5.7.2.26 rtdm_write .......................................................... 81

5.8 Device Registration Services ............................................. 82
5.8.1 Detailed Description ..................................................... 84
5.8.2 Macro Definition Documentation ..................................... 84
  5.8.2.1 RTDM_CLOSING ......................................................... 84
  5.8.2.2 RTDM_CREATED_IN_NRT ............................................. 84
  5.8.2.3 RTDM_DEVICE_TYPE_MASK .......................................... 84
  5.8.2.4 RTDM_EXCLUSIVE ..................................................... 84
5.8.2.5 RTDM_NAMED_DEVICE ........................................ 84
5.8.2.6 RTDM_PROTOCOLDEVICE ................................... 84

5.8.3 Typedef Documentation ......................................... 84
5.8.3.1 rtdm_close_handler_t ....................................... 84
5.8.3.2 rtdm_ioctl_handler_t ....................................... 85
5.8.3.3 rtdm_open_handler_t ....................................... 85
5.8.3.4 rtdm_read_handler_t ....................................... 85
5.8.3.5 rtdm_recvmsg_handler_t .................................... 86
5.8.3.6 rtdm_select_bind_handler_t ............................... 86
5.8.3.7 rtdm_sendmsg_handler_t .................................... 87
5.8.3.8 rtdm_socket_handler_t ..................................... 87
5.8.3.9 rtdm_write_handler_t ..................................... 88

5.8.4 Function Documentation ....................................... 89
5.8.4.1 rtdm_context_to_private .................................. 89
5.8.4.2 rtdm_dev_register ......................................... 89
5.8.4.3 rtdm_dev_unregister ........................................ 90
5.8.4.4 rtdm_private_to_context .................................. 90

5.9 Driver Development API .......................................... 91
5.9.1 Detailed Description .......................................... 91

5.10 Clock Services .................................................. 92
5.10.1 Detailed Description .......................................... 92
5.10.2 Function Documentation ...................................... 92
5.10.2.1 rtdm_clock_read ........................................... 92
5.10.2.2 rtdm_clock_read_monotonic ................................ 92

5.11 Task Services ................................................... 94
5.11.1 Detailed Description .......................................... 95
5.11.2 Typedef Documentation ....................................... 95
5.11.2.1 rtdm_task_proc_t ......................................... 95
5.11.3 Function Documentation ...................................... 95
5.11.3.1 rtdm_task_busy_sleep ..................................... 95
5.11.3.2 rtdm_task_current ......................................... 95
5.11.3.3 rtdm_task_destroy ......................................... 96
5.11.3.4 rtdm_task_init ............................................ 97
5.11.3.5 rtdm_task_join_nrt ........................................ 97
5.11.3.6 rtdm_task_set_period ..................................... 98
5.11.3.7 rtdm_task_set_priority ................................... 98
5.11.3.8 rtdm_task_sleep .......................................... 99
5.11.3.9 rtdm_task_sleep_abs ...................................... 99
5.11.3.10 rtdm_task_sleep_until ................................... 100
5.11.3.11 rtdm_task_unblock ...................................... 100
5.15.3.1 rtdm_nrtsig_destroy ........................................... 133
5.15.3.2 rtdm_nrtsig_init .............................................. 134
5.15.3.3 rtdm_nrtsig_pend .............................................. 134

5.16 Utility Services .................................................... 135

5.16.1 Detailed Description ............................................ 135

5.16.2 Function Documentation ........................................ 135
  5.16.2.1 rtdm_copy_from_user ...................................... 135
  5.16.2.2 rtdm_copy_to_user ........................................ 136
  5.16.2.3 rtdm_free .................................................. 137
  5.16.2.4 rtdm_in_rt_context ....................................... 137
  5.16.2.5 rtdm_iomap_to_user ....................................... 137
  5.16.2.6 rtdm_malloc ............................................... 138
  5.16.2.7 rtdm_mmap_to_user ....................................... 139
  5.16.2.8 rtdm_munmap .............................................. 140
  5.16.2.9 rtdm_printk ............................................... 140
  5.16.2.10 rtdm_printk_ratelimited ................................ 141
  5.16.2.11 rtdm_ratelimit .......................................... 141
  5.16.2.12 rtdm_read_user_ok ..................................... 142
  5.16.2.13 rtdm_rt_capable ........................................ 142
  5.16.2.14 rtdm_rw_user_ok ....................................... 143
  5.16.2.15 rtdm_safe_copy_from_user ................................ 143
  5.16.2.16 rtdm_safe_copy_to_user ................................ 144
  5.16.2.17 rtdm_strncpy_from_user ................................ 145

5.16.2.18 rtdm_strncpy_to_user .................................... 146

5.17 Device Profiles .................................................. 146

5.17.1 Detailed Description .......................................... 147

5.17.2 Macro Definition Documentation ............................... 147
  5.17.2.1 RTIOC_DEVICE_INFO ...................................... 147
  5.17.2.2 RTIOC_PURGE .............................................. 147

6 Data Structure Documentation ........................................ 149

6.1 can_bittime Struct Reference .................................... 149
  6.1.1 Detailed Description ......................................... 149

6.2 can_bittime_btr Struct Reference ................................ 149
  6.2.1 Detailed Description ......................................... 150

6.3 can_bittime_std Struct Reference ................................ 150
  6.3.1 Detailed Description ......................................... 150

6.4 can_filter Struct Reference ...................................... 150
  6.4.1 Detailed Description ......................................... 151

6.4.2 Field Documentation ............................................. 151
  6.4.2.1 can_id ..................................................... 151

Generated on Thu Jul 7 2016 13:25:19 for Xenomai RTDM skin API by Doxygen
Chapter 1

Deprecated List

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

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

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

Global `rtdm_task_sleep_until (nanosecs_abs_t wakeup_time)`
   Use `rtdm_task_sleep_abs` instead!
Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

- Real-Time Driver Model .................................................. 30
- User API ............................................................................ 32
- Driver Development API .................................................... 91
  - Inter-Driver API .............................................................. 73
  - Device Registration Services ....................................... 82
    - Synchronisation Services ........................................... 106
  - Clock Services ............................................................. 92
  - Task Services ............................................................... 94
  - Timer Services .............................................................. 102
  - Synchronisation Services ............................................. 106
  - Interrupt Management Services ................................. 129
  - Non-Real-Time Signalling Services ........................... 133
  - Utility Services ............................................................. 135
- Device Profiles ................................................................. 146
  - CAN Devices ................................................................. 9
  - Real-time IPC protocols ............................................ 45
  - Serial Devices ............................................................... 61
  - Testing Devices ............................................................ 71
Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

- **can_bittime**: Custom CAN bit-time definition ........................................ 149
- **can_bittime_btr**: Hardware-specific BTR bit-times ................................... 149
- **can_bittime_std**: Standard bit-time parameters according to Bosch ............ 150
- **can_filter**: Filter for reception of CAN messages ................................... 150
- **can_frame**: Raw CAN frame ................................................................. 151
- **rtdm_dev_context**: Device context ......................................................... 152
- **rtdm_device**: RTDM device ................................................................. 153
- **rtdm_device_info**: Device information .................................................. 155
- **rtdm_operations**: Device operations ...................................................... 155
- **rtipc_port_label**: Port label information structure .................................. 156
- **rtser_config**: Serial device configuration .............................................. 157
- **rtser_event**: Additional information about serial device events .................. 158
- **rtser_status**: Serial device status ......................................................... 158
- **sockaddr_can**: Socket address structure for the CAN address family .......... 159
- **sockaddr_ipc**: Socket address structure for the RTIPC address family ......... 159
Chapter 4

File Index

4.1 File List

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

- include/rtdm/rtcan.h
  Real-Time Driver Model for RT-Socket-CAN, CAN device profile header ........ 161
- include/rtdm/rtdm.h
  Real-Time Driver Model for Xenomai, user API header .......................... 167
- include/rtdm/rtdm_driver.h
  Real-Time Driver Model for Xenomai, driver API header ......................... 169
- include/rtdm/rtipc.h
  This file is part of the Xenomai project ........................................... 174
- include/rtdm/rtserial.h
  Real-Time Driver Model for Xenomai, serial device profile header ........... 176
- include/rtdm/rttesting.h
  Real-Time Driver Model for Xenomai, testing device profile header .......... 180
- include/rtdm/syscall.h ................................................................. ??
- ksrc/skins/rtdm/core.c
  Real-Time Driver Model for Xenomai, device operation multiplexing ........... 186
- ksrc/skins/rtdm/device.c
  Real-Time Driver Model for Xenomai, device management ......................... 181
- ksrc/skins/rtdm/drvlib.c
  Real-Time Driver Model for Xenomai, driver library ............................ 182
- ksrc/skins/rtdm/internal.h ............................................................ ??
- ksrc/skins/rtdm/module.c
  Real-Time Driver Model for Xenomai ............................................... 186
Chapter 5

Module Documentation

5.1 CAN Devices

Files

- file rtcan.h
  Real-Time Driver Model for RT-Socket-CAN, CAN device profile header.

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.

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_ERROR_WARNING = 1, CAN_STATE_ERROR_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

- **#define SIOCGIFINDEX defined_by_kernel_header_file**
  
  Get CAN interface index by name.

- **#define SIOCSCANBAUDRATE _IOW(RTIOC_TYPE_CAN, 0x01, struct ifreq)**
  
  Set baud rate.

- **#define SIOCGCANBAUDRATE _IOWR(RTIOC_TYPE_CAN, 0x02, struct ifreq)**
  
  Get baud rate.

- **#define SIOCSCANCUSTOMBITTIME _IOW(RTIOC_TYPE_CAN, 0x03, struct ifreq)**
  
  Set custom bit time parameter.

- **#define SIOCGCANCUSTOMBITTIME _IOWR(RTIOC_TYPE_CAN, 0x04, struct ifreq)**
  
  Get custom bit-time parameters.

- **#define SIOCSCANMODE _IOW(RTIOC_TYPE_CAN, 0x05, struct ifreq)**
  
  Set operation mode of CAN controller.

- **#define SIOCGCANSTATE _IOW(RTIOC_TYPE_CAN, 0x06, struct ifreq)**
  
  Get current state of CAN controller.

- **#define SIOCSCANCTRLMODE _IOW(RTIOC_TYPE_CAN, 0x07, struct ifreq)**
  
  Set special controller modes.

- **#define SIOCGCANCTRLMODE _IOWR(RTIOC_TYPE_CAN, 0x08, struct ifreq)**
  
  Get special controller modes.

- **#define RTCAN_RTIOC_TAKE_TIMESTAMP _IOW(RTIOC_TYPE_CAN, 0x09, int)**
  
  Enable or disable storing a high precision timestamp upon reception of a CAN frame.

- **#define RTCAN_RTIOC_RCV_TIMEOUT _IOW(RTIOC_TYPE_CAN, 0x0A, nanosecs_rel_t)**
  
  Specify a reception timeout for a socket.

- **#define RTCAN_RTIOC_SND_TIMEOUT _IOW(RTIOC_TYPE_CAN, 0x0B, nanosecs_rel_t)**
  
  Specify a transmission timeout for a socket.

### Error mask

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

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

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

**Note:** Bus error interrupts (``CAN_ERR_BUSERROR``) are enabled when an application is calling a `Recv` function on a socket listening on bus errors (using `CAN_RAW_ERR_FILTER`). After one bus error has 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**
  
  Generated on Thu Jul 7 2016 13:25:19 for Xenomai RTDM skin API by Doxygen
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.
5.1 CAN Devices

- `#define CAN_ERR_PROT_LOC_ACK_DEL 0x1B`  
  ACK delimiter.
- `#define CAN_ERR_PROT_LOC_EOF 0x1A`  
  end of frame
- `#define CAN_ERR_PROT_LOC_INTERM 0x12`  
  intermission
- `#define CAN_ERR_TRX_UNSPEC 0x00`  
  0000 0000
- `#define CAN_ERR_TRX_CANH_NO_WIRE 0x04`  
  0000 0100
- `#define CAN_ERR_TRX_CANH_SHORT_TO_BAT 0x05`  
  0000 0101
- `#define CAN_ERR_TRX_CANH_SHORT_TO_VCC 0x06`  
  0000 0110
- `#define CAN_ERR_TRX_CANH_SHORT_TO_GND 0x07`  
  0000 0111
- `#define CAN_ERR_TRX_CANL_NO_WIRE 0x40`  
  0100 0000
- `#define CAN_ERR_TRX_CANL_SHORT_TO_BAT 0x50`  
  0101 0000
- `#define CAN_ERR_TRX_CANL_SHORT_TO_VCC 0x60`  
  0110 0000
- `#define CAN_ERR_TRX_CANL_SHORT_TO_GND 0x70`  
  0111 0000
- `#define CAN_ERR_TRX_CANL_SHORT_TO_CANH 0x80`  
  1000 0000

5.1.1 Detailed Description

This is the common interface a RTDM-compliant CAN device has to provide. Feel free to report bugs and comments on this profile to the "Socketcan" mailing list (Socketcan-core@lists.berlios.de) or directly to the authors (wg@grandegger.com or Sebastian.Smolorz@stud.uni-hannover.de).

Profile Revision: 2

Device Characteristics

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

Supported Operations

**Socket**

- Environments: non-RT (RT optional, deprecated)

Specific return values:

- `-EPROTONOTSUPPORT` (Protocol is not supported by the driver. See CAN protocols for possible protocols.)

**Close**

- Blocking calls to any of the Send or Receive functions will be unblocked when the socket is closed and return with an error.
- Environments: non-RT (RT optional, deprecated)

Specific return values: none
**IOCTL**

Mandatory Environments: see **below**
Specific return values: see **below**

**Bind**

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

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

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

Environments: non-RT (RT optional)

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

**Setsockopt, Getsockopt**

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

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

Environments: non-RT (RT optional)

Specific return values: see links to options above.

**Recv, Recvfrom, Recvmsg**

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

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

It is possible to receive a high precision timestamp with every CAN message. The condition is a former instruction to the socket via **RTCAN_RTIOC_TAKE_TIMESTAMP**. The timestamp will be copied to the `msg_control` buffer of struct `user_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_control` of `0` upon completion of the function call indicates that no timestamp is available for that message.

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

Supported Flags [out]: none

Environments: RT (non-RT optional)

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

**Send, Sendto, Sendmsg**
These functions send out CAN messages. Only one message per call can be transmitted, so only one buffer with the correct length must be passed. For SOCK_RAW, this is the size of struct can_frame. The following only applies to SOCK_RAW: If a socket address of struct sockaddr_can is given, only can_ifindex is used. It is also possible to omit the socket address. Then the interface the socket is bound to will be used for sending messages.
If an interface goes down (due to bus-off or setting of stop mode) all senders that were blocked on this interface will be woken up.

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

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

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

### 5.1.2 Macro Definition Documentation

#### 5.1.2.1 #define CAN_CTRLMODE_3_SAMPLES 0x4

Triple sampling mode
In this mode the CAN controller uses Triple sampling.
5.1.2.2  

#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:
   rtcanelog.c.

5.1.2.3  

#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:
   rtcanelog.c.

5.1.2.4  

#define CAN_ERR_LOSTARB_UNSPEC 0x00

unspecified
else bit number in bitstream

5.1.2.5  

#define CAN_RAW_ERR_FILTER 0x2

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

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>

Environments: non-RT (RT optional)

Specific return values:

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

Examples:
   rtcanelog.c.
5.1.2.6  #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 <code>can_filter</code>.</td>
</tr>
<tr>
<td>in</td>
<td>optlen</td>
<td>Size of filter list: <code>count * sizeof(struct can_filter)</code>.</td>
</tr>
</tbody>
</table>

Environments: non-RT (RT optional)

Specific return values:

- -EFAULT (It was not possible to access user space memory area at the specified address.)
- -ENOMEM (Not enough memory to fulfill the operation)
- -EINVAL (Invalid length "optlen")
- -ENOSPC (No space to store filter list, check RT-Socket-CAN kernel parameters)

Examples:

- `rtcan_rtt.c`, `rtcanrecv.c`, and `rtcansend.c`.

5.1.2.7  #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: <code>sizeof(int)</code>.</td>
</tr>
</tbody>
</table>

Environments: non-RT (RT optional)

Specific return values:

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

  rtcansend.c.

5.1.2.8  
#define CAN_RAW_RECV_OWN_MSGS 0x4

CAN receive own messages.
Not supported by RT-Socket-CAN, but defined for compatibility with Socket-CAN.

5.1.2.9  
#define RTCAN_RTIOC_RCV_TIMEOUT _IOW(RTIOC_TYPE_CAN, 0x0A, nanosecs_rel_t)

Specify a reception timeout for a socket.
Defines a timeout for all receive operations via a socket which will take effect when one of the
 receive functions is called without the MSG_DONTWAIT flag set.
The default value for a newly created socket is an infinite timeout.

Note

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

Parameters

| in | arg | Pointer to nanosecs_rel_t variable. The value is interpreted as relative
     |     | timeout in nanoseconds in case of a positive value. See Timeouts for
     |     | special timeouts. |

Returns

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

Environments:

  This service can be called from:

   • Kernel module initialization/cleanup code
   • Kernel-based task
   • User-space task (RT, non-RT)

  Rescheduling: never.

Examples:

  rtcanrecv.c.

5.1.2.10  
#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>
</table>

Pointer to `nanosecs_rel_t` variable. The value is interpreted as relative timeout in nanoseconds in case of a positive value. See `Timeouts` for special timeouts.

Returns

0 on success, otherwise:

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

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

Examples:

`rtcansend.c`.

5.1.2.11

#define RTCAN_RTILOC_TAKE_TIMESTAMP _IOW(RTILOC_TYPE_CAN, 0x09, int)

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

A newly created socket takes no timestamps by default.

Parameters

| in  | arg |

int variable, see `Timestamp switches`

Returns

0 on success.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)
Note
Activating taking timestamps only has an effect on newly received CAN messages from the bus. Frames that already are in the socket buffer do not have timestamps if it was deactivated before. See Receive for more details.

Rescheduling: never.

Examples:

rtcanrecv.c.

5.1.2.12  #define SIOCGCANBAUDRATE _IOWR(RTIOC_TYPE_CAN, 0x02, struct 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 ifreq from linux/if.h). ifr_name must hold a valid CAN interface name, ifr_ifru will be filled with an instance of can_baudrate_t.</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.

Environments:
This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.1.2.13  #define SIOCGCANCTRLMODE _IOWR(RTIOC_TYPE_CAN, 0x08, struct 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 ifreq from linux/if.h). ifr_name must hold a valid CAN interface name, ifr_ifru must be filled with an instance of can_ctrlmode_t.</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.
Environments:
This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: possible.

5.1.2.14  
#define SIOCGCANCUSTOMBITTIME _IOWR(RTIOC_TYPE_CAN, 0x04, struct 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 ifreq from linux/if.h). ifr_name must hold a valid CAN interface name, ifr_ifru will be filled with an instance of struct can_bittime.</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.

Environments:
This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.1.2.15  
#define SIOCGCANSTATE _IOWR(RTIOC_TYPE_CAN, 0x06, struct ifreq)

Get current state of CAN controller.

States are divided into main states and additional error indicators. A CAN controller is always in exactly one main state. CAN bus errors are registered by the CAN hardware and collected by the driver. There is one error indicator (bit) per error type. If this IOCTL is triggered the error types which 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 ifreq from linux/if.h). ifr_name must hold a valid CAN interface name, ifr_ifru will be filled with an instance of struct can_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.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: possible.

5.1.2.16  #define SIOCGIFINDEX defined_by_kernel_header_file

Get CAN interface index by name.

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 ifreq from linux/if.h). If ifr_name holds a valid CAN interface name ifr_ifindex will be filled with the corresponding interface index.</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.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

Examples:

rtcan_rtt.c, rtcanconfig.c, rtcanrecv.c, and rtcansend.c.

5.1.2.17  #define SIOCSCANBAUDRATE _IOW(RTIOC_TYPE_CAN, 0x01, struct ifreq)

Set baud rate.

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

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

Returns

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

Environments:

This service can be called from:
- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Note

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

Rescheduling: possible.

Examples:

rtnconfig.c

5.1.2.18 #define SIOCSCANCTRLMODE _IOW(RTIOC_TYPE_CAN, 0x07, struct ifreq)

Set special controller modes.

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

Parameters

| in | arg | Pointer to interface request structure buffer (struct ifreq from linux/if.h). ifr_name must hold a valid CAN interface name, ifr_ifru must be filled with an instance of can_ctrlmode_t. |

Returns

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

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Note
Setting special controller modes is a configuration task. It should be done deliberately or otherwise CAN messages will likely be lost.

Rescheduling: possible.

Examples:

rtcancanconfig.c.

5.1.2.19 #define SIOCSCANCUSTOMBITTIME _IOW(RTIOC_TYPE_CAN, 0x03, struct ifreq)

Set custom bit time parameter.

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

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>arg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pointer to interface request structure buffer (struct ifreq from linux/ff.h). ifr_name must hold a valid CAN interface name, ifr_ifru must 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.
- -ENOMEM: No device with specified name exists.
- -EINVAL: No valid baud rate, see can_baudrate_t.
- -EAGAIN: Request could not be successfully fulfilled. Try again.

Environments:
This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

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

Rescheduling: possible.

Examples:

rtcancanconfig.c.
5.1.2.20  
define SIOCSCANMODE _IOW(RTIOC_TYPE_CAN, 0x05, struct 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></td>
<td>Pointer to interface request structure buffer (struct ifreq from linux/if.h). ifr_name must hold a valid CAN interface name, ifr_ifru must be filled with an instance of can_mode_t.</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

Environments:
This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Note
Setting a CAN controller into normal operation after a bus-off can take some time (128 occurrences of 11 consecutive recessive bits). In such a case, although this IOCTL will return immediately with success and SIOCGCANSTATE will report CAN_STATE_ACTIVE, bus-off recovery may still be in progress.
If a controller is bus-off, setting it into stop mode will return no error but the controller remains bus-off.

Rescheduling: possible.

Examples:

```
rtcanconfig.c
```

5.1.2.21  
define SOL_CAN_RAW 103

CAN socket levels.
Used for Sockopts for the particular protocols.

Examples:

```
rtcan_rtt.c, rtcanrecv.c, and rtcansend.c
```
5.1.3 Typedef Documentation

5.1.3.1 typedef struct can_filter can_filter_t

Filter for reception of CAN messages.
This filter works as follows: A received CAN ID is AND'ed bitwise with \texttt{can\_mask} and then compared to \texttt{can\_id}. This also includes the \texttt{CAN\_EFF\_FLAG} and \texttt{CAN\_RTR\_FLAG} of \texttt{CAN\_xxx\_FLAG}. If this comparison is true, the message will be received by the socket. The logic can be inverted with the \texttt{can\_id} flag \texttt{CAN\_INV\_FILTER}:

\begin{verbatim}
  * 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;
  * }
\end{verbatim}

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

5.1.3.2 typedef struct can_frame can_frame_t

Raw CAN frame.
Central structure for receiving and sending CAN frames.

Examples:
\texttt{rtcanrecv.c}.

5.1.4 Enumeration Type Documentation

5.1.4.1 enum CAN\_BITTIME\_TYPE

Supported CAN bit-time types.

Enumerator
\begin{verbatim}
  CAN_BITTIME_STD  Standard bit-time definition according to Bosch.
  CAN_BITTIME_BTR  Hardware-specific BTR bit-time definition.
\end{verbatim}

5.1.4.2 enum CAN\_MODE

Enumerator
\begin{verbatim}
  CAN_MODE_STOP    Set controller in Stop mode (no reception / transmission possible)
  CAN_MODE_START   Set controller into normal operation.
                   Coming from stopped mode or bus off, the controller begins with no errors in CAN\_STATE\_ACTIVE.
  CAN_MODE_SLEEP   Set controller into Sleep mode.
                   This is only possible if the controller is not stopped or bus-off.
                   Notice that sleep mode will only be entered when there is no bus activity. If the controller
detects bus activity while "sleeping" it will go into operating mode again.
                   To actively leave sleep mode again trigger CAN\_MODE\_START.
\end{verbatim}
5.1.4.3  enum CAN_STATE

Enumerator

CAN_STATE_ERROR_ACTIVE  CAN controller is error active.
CAN_STATE_ERROR_WARNING  CAN controller is error active, warning level is reached.
CAN_STATE_ERROR_PASSIVE  CAN controller is error passive.
CAN_STATE_BUS_OFF       CAN controller went into Bus Off.
CAN_STATE_SCANNING_BAUDRATE  CAN controller is scanning to get the baudrate.
CAN_STATE_STOPPED       CAN controller is in stopped mode.
CAN_STATE_SLEEPING     CAN controller is in Sleep mode.
5.2 Real-Time Driver Model

Modules

- User API
- Driver Development API
- Device Profiles

Typedefs

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

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

API Versioning

- `#define RTDM_API_VER 8`
  
  Common user and driver API version.

- `#define RTDM_API_MIN_COMPAT_VER 6`
  
  Minimum API revision compatible with the current release.

RTDM_TIMEOUT_xxx

Special timeout values

- `#define RTDM_TIMEOUT_INFINITE 0`
  
  Block forever.

- `#define RTDM_TIMEOUT_NONE (-1)`
  
  Any negative timeout means non-blocking.

5.2.1 Detailed Description

The Real-Time Driver Model (RTDM) provides a unified interface to both users and developers of real-time device drivers. Specifically, it addresses the constraints of mixed RT/non-RT systems like Xenomai. RTDM conforms to POSIX semantics (IEEE Std 1003.1) where available and applicable.

**API Revision: 8**

5.2.2 Macro Definition Documentation

5.2.2.1 `#define RTDM_TIMEOUT_INFINITE 0`

Block forever.

5.2.2.2 `#define RTDM_TIMEOUT_NONE (-1)`

Any negative timeout means non-blocking.
5.2.3 Typedef Documentation

5.2.3.1 typedef uint64_t **nanosecs_abs_t**

RTDM type for representing absolute dates.
Its base type is a 64 bit unsigned integer. The unit is 1 nanosecond.

Examples:
- rtcanrecv.c

5.2.3.2 typedef int64_t **nanosecs_rel_t**

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

Examples:
- rtcanrecv.c
5.3 User API

Files

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

Functions

- int rt_dev_open (const char *path, int oflag,...)
  *Open a device.*
- int rt_dev_socket (int protocol_family, int socket_type, int protocol)
  *Create a socket.*
- int rt_dev_close (int fd)
  *Close a device or socket.*
- int rt_dev_ioctl (int fd, int request,...)
  *Issue an IOCTL.*
- ssize_t rt_dev_read (int fd, void *buf, size_t nbyte)
  *Read from device.*
- ssize_t rt_dev_write (int fd, const void *buf, size_t nbyte)
  *Write to device.*
- ssize_t rt_dev_recvmsg (int fd, struct user_msghdr *msg, int flags)
  *Receive message from socket.*
- ssize_t rt_dev_recvfrom (int fd, void *buf, size_t len, int flags, struct sockaddr *from, socklen_t *fromlen)
  *Receive message from socket.*
- ssize_t rt_dev_sendmsg (int fd, const struct user_msghdr *msg, int flags)
  *Transmit message to socket.*
- ssize_t rt_dev_sendto (int fd, const void *buf, size_t len, int flags, const struct sockaddr *to, socklen_t tolen)
  *Transmit message to socket.*
- ssize_t rt_dev_send (int fd, const void *buf, size_t len, int flags)
  *Transmit message to socket.*
- int rt_dev_bind (int fd, const struct sockaddr *my_addr, socklen_t addrlen)
  *Bind to local address.*
- int rt_dev_connect (int fd, const struct sockaddr *serv_addr, socklen_t addrlen)
  *Connect to remote address.*
- int rt_dev_listen (int fd, int backlog)
  *Listen for incoming connection requests.*
- int rt_dev_accept (int fd, struct sockaddr *addr, socklen_t *addrlen)
  *Accept a connection requests.*
- int rt_dev_shutdown (int fd, int how)
  *Shut down parts of a connection.*
- int rt_dev_getsockopt (int fd, int level, int optname, void *optval, socklen_t *optlen)
  *Get socket option.*
- int rt_dev_setsockopt (int fd, int level, int optname, const void *optval, socklen_t optlen)
  *Set socket option.*
- int rt_dev_getsockname (int fd, struct sockaddr *name, socklen_t *namelen)
  *Get local socket address.*
- int rt_dev_getpeername (int fd, struct sockaddr *name, socklen_t *namelen)
  *Get socket destination address.*
5.3.1 Detailed Description

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

5.3.2 Function Documentation

5.3.2.1 int rt_dev_accept ( int fd, struct sockaddr *addr, socklen_t *addrlen )

Accept a connection requests.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>File descriptor as returned by rt_dev_socket()</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>Buffer for remote address</td>
</tr>
<tr>
<td>in,out</td>
<td>Address buffer size</td>
</tr>
</tbody>
</table>

Returns

0 on success, otherwise negative error code

Environments:

Depends on driver implementation, see Device Profiles.
Rescheduling: possible.

See Also


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

Bind to local address.

Parameters

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

Returns

0 on success, otherwise negative error code

Environments:

Depends on driver implementation, see Device Profiles.
Rescheduling: possible.

See Also


Examples:
rtnsent.c, and rtcansend.c.
5.3.2.3   int rt_dev_close ( int fd )

Close a device or socket.
5.3 User API

### rt_dev_close()

**Parameters**

| in | fd | File descriptor as returned by rt_dev_open() or rt_dev_socket() |

**Returns**

0 on success, otherwise a negative error code.

**Note**

If the matching rt_dev_open() or rt_dev_socket() call took place in non-real-time context, rt_dev_close() must be issued within non-real-time as well. Otherwise, the call will fail.

**Environments:**

Depends on driver implementation, see Device Profiles.

**Rescheduling:** possible.

**See Also**


---

5.3.2.4 rt_dev_connect()

**Parameters**

| in | fd | File descriptor as returned by rt_dev_socket() |
| in | serv_addr | Address buffer |
| in | addrlen | Address buffer size |

**Returns**

0 on success, otherwise negative error code

**Environments:**

Depends on driver implementation, see Device Profiles.

**Rescheduling:** possible.

**See Also**

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

---

5.3.2.5 rt_dev_getpeername()

**Parameters**

| in | fd | File descriptor as returned by rt_dev_socket() |

Get socket destination address.
<table>
<thead>
<tr>
<th>out</th>
<th>name</th>
<th>Address buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>in, out</td>
<td>namelen</td>
<td>Address buffer size</td>
</tr>
</tbody>
</table>

Returns
0 on success, otherwise negative error code

Environments:
Depends on driver implementation, see Device Profiles.
Rescheduling: possible.

See Also

5.3.2.6 int rt_dev_getsockname ( int fd, struct sockaddr * name, socklen_t * namelen )

Get local socket address.
Parameters
<table>
<thead>
<tr>
<th>in</th>
<th>fd</th>
<th>File descriptor as returned by rt_dev_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

Environments:
Depends on driver implementation, see Device Profiles.
Rescheduling: possible.

See Also

5.3.2.7 int rt_dev_getsockopt ( int fd, int level, int optname, void * optval, socklen_t * optlen )

Get socket option.
Parameters
<table>
<thead>
<tr>
<th>in</th>
<th>fd</th>
<th>File descriptor as returned by rt_dev_socket()</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>

Generated on Thu Jul 7 2016 13:25:19 for Xenomai RTDM skin API by Doxygen
5.3.2.8  int rt_dev_ioctl ( int fd, int request, ... )

Issue an IOCTL.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>request</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>File descriptor as returned by rt_dev_open() or rt_dev_socket()</td>
</tr>
<tr>
<td>in</td>
<td>IOCTL code</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<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

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

See Also


5.3.2.9  int rt_dev_listen ( int fd, int backlog )

Listen for incomming connection requests.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>fd</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>File descriptor as returned by rt_dev_socket()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>in</th>
<th>backlog</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>Maximum queue length</td>
</tr>
</tbody>
</table>

Returns

0 on success, otherwise negative error code

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

See Also

5.3.2.10  int rt_dev_open ( const char * path, int oflag, ... )

Open a device.
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.

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

See Also


5.3.2.11 ssize_t rt_dev_read ( int fd, void * buf, size_t nbyte )

Read from device.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>fd</th>
<th>File descriptor as returned by rt_dev_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.

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

See Also


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

Receive message from socket.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>fd</th>
<th>File descriptor as returned by rt_dev_socket()</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

Environments:
Depends on driver implementation, see Device Profiles.
Rescheduling: possible.

See Also

5.3.2.13 ssize_t rt_dev_recvfrom ( int fd, void *buf, size_t len, int flags, struct sockaddr *from, socklen_t *fromlen )

Receive message from socket.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>fd</th>
<th>File descriptor as returned by rt_dev_socket()</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

Environments:
Depends on driver implementation, see Device Profiles.
Rescheduling: possible.

See Also

References rt_dev_recvfrom().
Referenced by rt_dev_recvfrom().

5.3.2.14 ssize_t rt_dev_recvmsg ( int fd, struct user_msghdr *msg, int flags )

Receive message from socket.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>fd</th>
<th>File descriptor as returned by rt_dev_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

Environments:
Depends on driver implementation, see Device Profiles.
Rescheduling: possible.

See Also

Examples:
rtcanrecv.c.

5.3.2.15 ssize_t rt_dev_send ( int fd, const void * buf, size_t len, int flags )
Transmit message to socket.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>fd</th>
<th>File descriptor as returned by rt_dev_socket()</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>
</tbody>
</table>

Returns
Number of bytes sent, otherwise negative error code

Environments:
Depends on driver implementation, see Device Profiles.
Rescheduling: possible.

See Also

Examples:
rtcansend.c.

5.3.2.16 ssize_t rt_dev_sendmsg ( int fd, const struct user _msghdr * msg, int flags )
Transmit message to socket.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>fd</th>
<th>File descriptor as returned by rt_dev_socket()</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>msg</td>
<td>Message descriptor</td>
</tr>
</tbody>
</table>
5.3.2.17 ssize_t rt_dev_sendto ( int fd, const void *buf, size_t len, int flags, const struct sockaddr *to, socklen_t tolen )

Transmit message to socket.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>fd</th>
<th>File descriptor as returned by rt_dev_socket()</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>

Returns

Number of bytes sent, otherwise negative error code

Environments:
Depends on driver implementation, see Device Profiles.
Rescheduling: possible.

See Also

Examples:
rtcansend.c.
### rt_dev_socket

Create a socket.

#### Parameters

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

#### Returns

0 on success, otherwise negative error code

#### Environments:

Depends on driver implementation, see [Device Profiles](#).

Rescheduling: possible.

#### See Also

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

#### Examples:

rtcanrecv.c, and rtcansend.c.

---

### rt_dev_shutdown

Shut down parts of a connection.

#### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>fd</th>
<th>File descriptor as returned by rt_dev_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

#### Environments:

Depends on driver implementation, see [Device Profiles](#).

Rescheduling: possible.

#### See Also

`shutdown()` in IEEE Std 1003.1, [http://www.opengroup.org/onlinepubs/009695399](http://www.opengroup.org/onlinepubs/009695399)
Returns
Positive file descriptor value on success, otherwise a negative error code.

Environments:
Depends on driver implementation, see Device Profiles.
Rescheduling: possible.

See Also

5.3.2.21 ssize_t rt_dev_write ( int fd, const void * buf, size_t nbyte )

Write to device.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>fd</th>
<th>File descriptor as returned by rt_dev_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

Environments:
Depends on driver implementation, see Device Profiles.
Rescheduling: possible.

See Also
5.4 Real-time IPC protocols

Profile Revision: 1

Files

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

Data Structures

- struct rtpc_port_label
  
  Port label information structure.

- struct sockaddr_ipc
  
  Socket address structure for the RTIPC address family.

Typedefs

- typedef int16_t rtpc_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 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 user_msghdr *msg, int flags)
  
  Send a message on a RTIPC socket.

- ssize_t recvmsg__AF_RTIPC (int sockfd, struct user_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.

5.4.1 Detailed Description

**Profile Revision:** 1

Device Characteristics

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

5.4.2 Macro Definition Documentation

5.4.2.1 `#define BUFP_BUFSZ 2`

**BUFP buffer size configuration.**

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

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

**Note**

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

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>level</th>
<th>SOL_BUFP</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>optname</td>
<td>BUFP_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 buffer 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
Module Documentation

Examples:

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

5.4.2.2 #define BUFP_LABEL 1

BUFP label assignment.

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

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

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

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>level</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>optname</td>
</tr>
<tr>
<td>in</td>
<td>optarg</td>
</tr>
<tr>
<td>in</td>
<td>optlen</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.

5.4.2.3 #define IDDP_LABEL 1

IDDP label assignment.

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

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

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

<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)

#### Calling context:

RT/non-RT

#### Examples:

iddp-label.c.

### 5.4.2.4 #define IDDP_POOLSZ 2

IDDP local pool size configuration.

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

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

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

#### Note

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

### Parameters

<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)
5.4.2.5  #define SO_RCVTIMEO defined_by_kernel_header_file

All RTIPC protocols support the standard SO_RCVTIMEO socket option, from the SOL_SOCKET level.

See Also


Examples:

 xddp-label.c.

5.4.2.6  #define SO_SNDTIMEO defined_by_kernel_header_file

IPCPROTO_IDDP and IPCPROTO_BUFP protocols support the standard SO_SNDTIMEO socket option, from the SOL_SOCKET level.

See Also


Examples:

 xddp-label.c.

5.4.2.7  #define XDDP_BUFSZ 3

XDDP streaming buffer size configuration.

In addition to sending datagrams, real-time threads may stream data in a byte-oriented mode through the port as well. This increases the bandwidth and reduces the overhead, when the overall data to send to the Linux domain is collected by bits, and keeping the message boundaries is not required.

This feature is enabled when a non-zero buffer size is set for the socket. In that case, the real-time data accumulates into the streaming buffer when MSG_MORE is passed to any of the send functions, until:

- the receiver from the Linux domain wakes up and consumes it,
- a different source port attempts to send data to the same destination port,
- MSG_MORE is absent from the send flags,
- the buffer is full,

whichever comes first.

Setting *optval* to zero disables the streaming buffer, in which case all sendings are conveyed in separate datagrams, regardless of MSG_MORE.

Note

only a single streaming buffer exists per socket. When this buffer is full, the real-time data stops accumulating and sending operations resume in mere datagram mode. Accumulation may happen again after some or all data in the streaming buffer is consumed from the Linux domain endpoint.

The streaming buffer size may be adjusted multiple times during the socket lifetime; the latest configuration change will take effect when the accumulation resumes after the previous buffer was flushed.
Parameters

<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:

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

Calling context:

RT/non-RT

Examples:

`xddp-stream.c`

5.4.2.8  #define XDDP_EVTDOWN 3

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

5.4.2.9  #define XDDP_EVTIN 1

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

5.4.2.10 #define XDDP_EVTNOBUF 4

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

5.4.2.11 #define XDDP_EVTOUT 2

Monitor reads from the non real-time endpoint.
XDDP_EVTOUT is sent when the non real-time endpoint successfully reads a complete message (i.e. via /dev/rtpN). The argument is the size of the outgoing message.
#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>in</th>
<th>level</th>
<th>SOL_XDDP</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>optname</td>
<td>XDDP_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 invalid)

Calling context:

RT/non-RT

Examples:

xddp-label.c.

#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>
</table>

Generated on Thu Jul 7 2016 13:25:19 for Xenomai RTDM skin API by Doxygen
5.4 Real-time IPC protocols

<table>
<thead>
<tr>
<th>in</th>
<th>optname</th>
<th>XDDP_MONITOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>optval</td>
<td>Pointer to a pointer to function of type int ( unsigned_t)(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(unsigned_t)</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

5.4.2.14 #define XDDP_POOLSZ 2

XDDP local pool size configuration.

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

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

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

Note

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

Parameters

<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)
-EINVAL (optlen invalid or *optval is zero)

Calling context:

RT/non-RT

Examples:

xddp-echo.c.
5.4.3 Enumeration Type Documentation

5.4.3.1 anonymous enum

Enumerator

**IPCPROTO_IPC**  Default protocol (IDDP)

**IPCPROTO_XDDP**  Cross-domain datagram protocol (RT <-> non-RT). Real-time Xenomai threads and regular Linux threads may want to exchange data in a way that does not require the former to leave the real-time domain (i.e. primary mode). The RTDM-based XDDP protocol is available for this purpose.

On the Linux domain side, pseudo-device files named /dev/rtp<minor> give regular POSIX threads access to non real-time communication endpoints, via the standard character-based I/O interface. On the Xenomai domain side, sockets may be bound to XDDP ports, which act as proxies to send and receive data to/from the associated pseudo-device files. Ports and pseudo-device minor numbers are paired, meaning that e.g. 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.

**IPCPROTO_IDDP**  Intra-domain datagram protocol (RT <-> RT). The RTDM-based IDDP protocol enables real-time threads to exchange datagrams within the Xenomai domain, via socket endpoints.

**IPCPROTO_BUFP**  Buffer protocol (RT <-> RT, byte-oriented). The RTDM-based BUFP protocol implements a lightweight, byte-oriented, one-way Producer-Consumer data path. All messages written are buffered into a single memory area in strict FIFO order, until read by the consumer.

This protocol always prevents short writes, and only allows short reads when a potential deadlock situation arises (i.e. readers and writers waiting for each other indefinitely).

5.4.4 Function Documentation

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

Bind a RTIPC socket to a port.

Bind the socket to a destination port.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>addr</th>
<th>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:</th>
</tr>
</thead>
</table>

- IPCPROTO_XDDP

This action creates an endpoint for channelling traffic between the Xenomai and Linux domains.

sipc_family must be AF_RTIPC, sipc_port is either -1, or a valid free port number between 0 and CONFIG_XENO_OPT_PIPE_NRDEV-1.

If sipc_port is -1, a free port will be assigned automatically.

Upon success, the pseudo-device /dev/rtpN will be reserved for this communication channel, where N is the assigned port number. The non real-time side shall open this device to exchange data over the bound socket.
If a label was assigned (see XDDP_LABEL) prior to binding the socket to a port, a registry link referring to the created pseudo-device will be automatically set up as /proc/xenomai/registry/rtipc/xddp/label, where label is the label string passed to setsockopt() for the XDDP_LABEL option.

- **IPCPROTO_IDDP**
  
  This action creates an endpoint for exchanging datagrams within the Xenomai domain.

  sipc_family must be AF_RTIPC, sipc_port is either -1, or a valid free port number between 0 and CONFIG_XENO_OPT_IDDP_NRPORT-1.

  If sipc_port is -1, a free port will be assigned automatically. The real-time peer shall connect to the same port for exchanging data over the bound socket.

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

- **IPCPROTO_BUFP**
  
  This action creates an endpoint for a one-way byte stream within the Xenomai domain.

  sipc_family must be AF_RTIPC, sipc_port is either -1, or a valid free port number between 0 and CONFIG_XENO_OPT_BUFP_NRPORT-1.

  If sipc_port is -1, an available port will be assigned automatically. The real-time peer shall connect to the same port for exchanging data over the bound socket.

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

Returns

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

- -EFAULT (Invalid data address given)
- -ENOMEM (Not enough memory)
- -EINVAL (Invalid parameter)
- -EADDRINUSE (Socket already bound to a port, or no port available)

Calling context:

- non-RT

5.4.4.2  int close__AF_RTIPC ( int sockfd )

Close a RTIPC socket descriptor.

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

Returns

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

Calling context:

- non-RT
5.4.4.3  int connect__AF_RTIPC ( int sockfd, const struct sockaddr_ipc * addr, socklen_t addrlen )

Initiate a connection on a RTIPC socket.
Parameters

| in    | addr | The address to connect the socket to (see struct sockaddr_ipc). |

- If sipc_port is a valid port for the protocol, it is used verbatim and the connection succeeds immediately, regardless of whether the destination is bound at the time of the call.
- If sipc_port is -1 and a label was assigned to the socket, connect() blocks for the requested amount of time (see SO_RCVTIMEO) until a socket is bound to the same label via bind(2) (see XDDP_LABEL, IDDP_LABEL, BUFP_LABEL), in which case a connection is established between both endpoints.
- If sipc_port is -1 and no label was assigned to the socket, the default destination address is cleared, meaning that any subsequent write to the socket will return -EDESTADDRREQ, until a valid destination address is set via connect(2) or bind(2).

Returns

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

Calling context:

RT/non-RT

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

Get socket peer.

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

Returns

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

Calling context:

RT/non-RT

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

Get socket name.

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

Returns

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

Calling context:

RT/non-RT
5.4.4.6 int getsockopt__AF_RTIPC (int sockfd, int level, int optname, void *optval, socklen_t *optlen)

Get options on RTIPC sockets.

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

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

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

Calling context:
RT/non-RT

5.4.4.7 ssize_t recvmsg__AF_RTIPC (int sockfd, struct user_msghdr *msg, int flags)

Receive a message from a RTIPC socket.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>flags</th>
<th>Operation flags:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>MSG_DONTWAIT Non-blocking I/O operation. The caller will not be blocked whenever no message is immediately available for receipt at the time of the call, but will rather return with -EWOULDBLOCK.</td>
</tr>
</tbody>
</table>

Note

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

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

Calling context:
RT

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

Send a message on a RTIPC socket.
Parameters

\[ \begin{array}{|c|c|c|} \hline
\text{in} & \text{flags} & \text{Operation flags:} \\
\hline
\end{array} \]

- MSG_OOB Send out-of-band message. For all RTIPC protocols except IPCPROTO_BUFP, sending out-of-band data actually means pushing them to the head of the receiving queue, so that the reader will always receive them before normal messages. IPCPROTO_BUFP does not support out-of-band sending.

- MSG_DONTWAIT Non-blocking I/O operation. The caller will not be blocked whenever the message cannot be sent immediately at the time of the call (e.g., memory shortage), but will rather return with -EWOULDBLOCK. Unlike other RTIPC protocols, IPCPROTO_XDDP accepts but never considers MSG_DONTWAIT since writing to a real-time XDDP endpoint is inherently a non-blocking operation.

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

Note

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

Returns

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

Calling context:

RT

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

Set options on RTIPC sockets.

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

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

Returns

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

Calling context:

non-RT

5.4.4.10 int socket__AF_RTIPC ( int domain = AF_RTIPC, int type = SOCK_DGRAM, int protocol )

Create an endpoint for communication in the AF_RTIPC domain.
### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Any of <strong>IPPROTO_XDDP</strong>, <strong>IPPROTO_IDDP</strong>, or <strong>IPPROTO_BUFP</strong>. <strong>IPPROTO_IPC</strong> is also valid, and refers to the default RTIPC protocol, namely <strong>IPPROTO_IDDP</strong>.</td>
</tr>
</tbody>
</table>

### Returns

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

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

### Calling context:

- non-RT
5.5 Serial Devices

Files

- file rtserial.h
  
  Real-Time Driver Model for Xenomai, serial device profile header.

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_EMPTY 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

RTSER_BREAK_xxx

Break control

- 
  typedef struct rtser_config rtser_config_t
  Serial device configuration.
- 
  typedef struct rtser_status rtser_status_t
  Serial device status.
- 
  typedef struct rtser_event rtser_event_t
  Additional information about serial device events.
- 
  \#define RTSER_BREAK_CLR 0x00
  Serial device configuration.
- 
  \#define RTSER_BREAK_SET 0x01
  Serial device configuration.
- 
  \#define RTIOC_TYPE_SERIAL RTDM_CLASS_SERIAL
  Serial device configuration.

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.

5.5.1 Detailed Description

This is the common interface a RTDM-compliant serial device has to provide. Feel free to com-
ment 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 Name: "rtser<N>", N >= 0
  Device Class: RTDM_CLASS_SERIAL

Supported Operations

Open
Environments: non-RT (RT optional, deprecated)
Specific return values: none

Close
Environments: non-RT (RT optional, deprecated)
Specific return values: none

IOCTL
Mandatory Environments: see below
Specific return values: see below

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

Write
Environments: RT (non-RT optional)
Specific return values:
  • -ETIMEDOUT
  • -EINTR (interrupted explicitly or by signal)
  • -EAGAIN (no data written in non-blocking mode)
  • -EBADF (device has been closed while writing)
5.5.2 Macro Definition Documentation

5.5.2.1 #define RTSER_RTIOC_BREAK_CTL _IOR(RTIOC_TYPE_SERIAL, 0x06, int)

Set or clear break on UART output line.
Parameters

| in  | arg                  | RTSER_BREAK_SET or RTSER_BREAK_CLR (int) |

Returns

0 on success, otherwise negative error code

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Note

A set break condition may also be cleared on UART line reconfiguration.

Rescheduling: never.

5.5.2.2 #define RTSER_RTIOC_GET_CONFIG _IOR(RTIOC_TYPE_SERIAL, 0x00, struct rtser_config)

Get serial device configuration.

Parameters

| out | arg                  | Pointer to configuration buffer (struct rtser_config) |

Returns

0 on success, otherwise negative error code

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.5.2.3 #define RTSER_RTIOC_GET_CONTROL _IOR(RTIOC_TYPE_SERIAL, 0x03, int)

Get serial device’s modem control register.
Parameters

| out | arg | Pointer to variable receiving the content (int, see RTSER_MCR_xxx) |

Returns

0 on success, otherwise negative error code

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.5.2.4 #define RTSER_RTIOC_GET_STATUS _IOR(RTIOC_TYPE_SERIAL, 0x02, struct rtser_status)

Get serial device status.

Parameters

| out | arg | Pointer to status buffer (struct rtser_status) |

Returns

0 on success, otherwise negative error code

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Note

The error states RTSER_LSR_OVERRUN_ERR, RTSER_LSR_PARITY_ERR, RTSER_LSR_FRAMING_ERR, and RTSER_SOFT_OVERRUN_ERR that may have 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.

Rescheduling: never.

5.5.2.5 #define RTSER_RTIOC_SET_CONFIG _IOW(RTIOC_TYPE_SERIAL, 0x01, struct rtser_config)

Set serial device configuration.
Parameters

| in | arg | Pointer to configuration buffer (struct rtser_config) |

Returns

0 on success, otherwise:

- -EPERM is returned if the caller’s context is invalid, see note below.
- -ENOMEM is returned if a new history buffer for timestamps cannot be allocated.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Note

If rtser_config contains a valid timestamp_history and the addressed device has been opened in non-real-time context, this IOCTL must be issued in non-real-time context as well. Otherwise, this command will fail.

Rescheduling: never.

Examples:

cross-link.c.

5.5.2.6  #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

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.5.2.7  #define RTSER_RTIOC_WAIT_EVENT _IOR(RTIOC_TYPE_SERIAL, 0x05, struct rtser_event)

Wait on serial device events according to previously set mask.
Parameters

| out | arg | Pointer to event information buffer (struct rtser_event) |

Returns

0 on success, otherwise:

- -EBUSY is returned if another task is already waiting on events of this device.
- -EBADF is returned if the file descriptor is invalid or the device has just been closed.

Environments:

This service can be called from:

- Kernel-based task
- User-space task (RT)

Rescheduling: possible.

Examples:

cross-link.c.
5.6 Testing Devices

Files

- file `rttesting.h`

  Real-Time Driver Model for Xenomai, testing device profile header.

Sub-Classes of RTDM_CLASS_TESTING

- `#define RTDM_SUBCLASS_TIMERBENCH 0`
  subclass name: "timerbench"
- `#define RTDM_SUBCLASS_IRQBENCH 1`
  subclass name: "irqbench"
- `#define RTDM_SUBCLASS_SWITCHTEST 2`
  subclass name: "switchtest"
- `#define RTDM_SUBCLASS_RTDMTEST 3`
  subclass name: "rtdm"

IOCTLs

Testing device IOCTLs

- `#define RTTST_RTIOC_INTERM_BENCH_RES _IOWR(RTIOC_TYPE_TESTING, 0x00, struct rtst_interm_bench_res)`
- `#define RTTST_RTIOC_TMBENCH_START _IOW(RTIOC_TYPE_TESTING, 0x10, struct rttst_tmbench_config)`
- `#define RTTST_RTIOC_TMBENCH_STOP _IOWR(RTIOC_TYPE_TESTING, 0x11, struct rttst_overall_bench_res)`
- `#define RTTST_RTIOC_IRQBENCH_START _IO(RTIOC_TYPE_TESTING, 0x20, struct rttst_irqbench_config)`
- `#define RTTST_RTIOC_IRQBENCH_STOP _IO(RTIOC_TYPE_TESTING, 0x21)`
- `#define RTTST_RTIOC_IRQBENCH_GET_STATS _IOR(RTIOC_TYPE_TESTING, 0x22, struct rttst_irqbench_stats)`
- `#define RTTST_RTIOC_IRQBENCH_WAIT_IRQ _IO(RTIOC_TYPE_TESTING, 0x23)`
- `#define RTTST_RTIOC_IRQBENCH_REPLY_IRQ _IO(RTIOC_TYPE_TESTING, 0x24)`
- `#define RTTST_RTIOC_SWTEST_SET_TASKS_COUNT _IOW(RTIOC_TYPE_TESTING, 0x30, unsigned long)`
- `#define RTTST_RTIOC_SWTEST_SET_CPU _IOW(RTIOC_TYPE_TESTING, 0x31, unsigned long)`
- `#define RTTST_RTIOC_SWTEST_REGISTER_UTASK _IOW(RTIOC_TYPE_TESTING, 0x32, struct rttst_swtest_task)`
- `#define RTTST_RTIOC_SWTEST_CREATE_KTASK _IOWR(RTIOC_TYPE_TESTING, 0x33, struct rttst_swtest_task)`
- `#define RTTST_RTIOC_SWTEST_PEND _IOR(RTIOC_TYPE_TESTING, 0x34, struct rttst_swtest_task)`
- `#define RTTST_RTIOC_SWTEST_SWITCH_TO _IOR(RTIOC_TYPE_TESTING, 0x35, struct rttst_swtest_dir)`
- `#define RTTST_RTIOC_SWTEST_GET_SWITCHES_COUNT _IOR(RTIOC_TYPE_TESTING, 0x36, unsigned long)`
- `#define RTTST_RTIOC_SWTEST_GET_LAST_ERROR _IOR(RTIOC_TYPE_TESTING, 0x37, struct rttst_swtest_error)`
- `#define RTTST_RTIOC_SWTEST_SET_PAUSE _IOW(RTIOC_TYPE_TESTING, 0x38, unsigned long)`
- `#define RTTST_RTIOC_RTDM_DEFER_CLOSE _IOW(RTIOC_TYPE_TESTING, 0x40, unsigned long)`
5.6.1 Detailed Description

This group of devices is intended to provide in-kernel testing results. Feel free to comment on this profile via the Xenomai mailing list (xenomai@xenomai.org) or directly to the author (jan.kiszka@web.de).

Profile Revision: 2

Device Characteristics

- **Device Flags**: RTDM_NAMED_DEVICE
- **Device Name**: "rttest[-<subclass>]<N>", N >= 0, optional subclass name to simplify device discovery
- **Device Class**: RTDM_CLASS_TESTING

Supported Operations

- **Open**
  Environments: non-RT (RT optional, deprecated)
  Specific return values: none

- **Close**
  Environments: non-RT (RT optional, deprecated)
  Specific return values: none

- **IOCTL**
  Mandatory Environments: see TSTIOCTLs below
  Specific return values: see TSTIOCTLs below
5.7 Inter-Driver API

Functions

- **void rtdm_context_get (int fd)**
  Retrieve and lock a device context.

- **int rtdm_select_bind (int fd, rtdm_selector_t *selector, enum rtdm_selecttype type, unsigned fd_index)**
  Bind a selector to specified event types of a given file descriptor.

- **void rtdm_context_lock (struct rtdm_dev_context *context)**
  Increment context reference counter.

- **void rtdm_context_unlock (struct rtdm_dev_context *context)**
  Decrement context reference counter.

- **void rtdm_context_put (struct rtdm_dev_context *context)**
  Release a device context obtained via `rtdm_context_get()`

- **int rtdm_open (const char *path, int oflag,...)**
  Open a device.

- **int rtdm_socket (int protocol_family, int socket_type, int protocol)**
  Create a socket.

- **int rtdm_close (int fd)**
  Close a device or socket.

- **int rtdm_ioctl (int fd, int request,...)**
  Issue an IOCTL.

- **ssize_t rtdm_read (int fd, void *buf, size_t nbyte)**
  Read from device.

- **ssize_t rtdm_write (int fd, const void *buf, size_t nbyte)**
  Write to device.

- **ssize_t rtdm_recvmsg (int fd, struct user_msghdr *msg, int flags)**
  Receive message from socket.

- **ssize_t rtdm_recvfrom (int fd, void *buf, size_t len, int flags, struct sockaddr *from, socklen_t *fromlen)**
  Receive message from socket.

- **ssize_t rtdm_recv (int fd, void *buf, size_t len, int flags)**
  Receive message from socket.

- **ssize_t rtdm_sendmsg (int fd, const struct user_msghdr *msg, int flags)**
  Transmit message to socket.

- **ssize_t rtdm_sendto (int fd, const void *buf, size_t len, int flags, const struct sockaddr *to, socklen_t tolen)**
  Transmit message to socket.

- **ssize_t rtdm_send (int fd, const void *buf, size_t len, int flags)**
  Transmit message to socket.

- **int rtdm_bind (int fd, const struct sockaddr *my_addr, socklen_t addrlen)**
  Bind to local address.

- **int rtdm_connect (int fd, const struct sockaddr *serv_addr, socklen_t addrlen)**
  Connect to remote address.

- **int rtdm_listen (int fd, int backlog)**
  Listen for incoming connection requests.

- **int rtdm_accept (int fd, struct sockaddr *addr, socklen_t *addrlen)**
  Accept a connection requests.

- **int rtdm_shutdown (int fd, int how)**
  Shut down parts of a connection.
• int rtdm_getsockopt (int fd, int level, int optname, void *optval, socklen_t *optlen)
  Get socket option.
• int rtdm_setsockopt (int fd, int level, int optname, const void *optval, socklen_t optlen)
  Set socket option.
• int rtdm_getsockname (int fd, struct sockaddr *name, socklen_t *namelen)
  Get local socket address.
• int rtdm_getpeername (int fd, struct sockaddr *name, socklen_t *namelen)
  Get socket destination address.

5.7.1 Detailed Description

5.7.2 Function Documentation

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

Accept a connection requests.

Refer to rt_dev_accept() for parameters and return values

Environments:
Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

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

Bind to local address.

Refer to rt_dev_bind() for parameters and return values

Environments:
Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

5.7.2.3 int rtdm_close ( int fd )

Close a device or socket.

Refer to rt_dev_close() for parameters and return values

Environments:
Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

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

Connect to remote address.

Refer to rt_dev_connect() for parameters and return values

Environments:
Depends on driver implementation, see Device Profiles.

Rescheduling: possible.
5.7.2.5  struct **rtdm_dev_context** rtmd_context_get ( int fd )

Retrieve and lock a device context.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>fd</th>
<th>File descriptor</th>
</tr>
</thead>
</table>

Returns

Pointer to associated device context, or NULL on error

Note

The device context has to be unlocked using `rtdm_context_put()` when it is no longer referenced.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

References `rtdm_dev_context::close_lock_count`, and `rtdm_dev_context::fd`.

Referenced by `rtdm_select_bind()`.

5.7.2.6 void rtdm_context_lock ( struct rtdm_dev_context * context )

Increment context reference counter.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>context</th>
<th>Device context</th>
</tr>
</thead>
</table>

Note

`rtdm_context_get()` automatically increments the lock counter. You only need to call this function in special scenarios, e.g. when keeping additional references to the context structure that have different lifetimes. Only use `rtdm_context_lock()` on contexts that are currently locked via an earlier `rtdm_context_get() / rtdm_context_lock()` or while running a device operation handler.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.7.2.7 void rtdm_context_put ( struct rtdm_dev_context * context )

Release a device context obtained via `rtdm_context_get()`
5.7 Inter-Driver API

5.7.8 void rtdm_context_unlock ( struct rtdm_dev_context *context )

Decrement context reference counter.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>context</th>
<th>Device context</th>
</tr>
</thead>
</table>

Note

Every call to rtdm_context_locked() must be matched by a rtdm_context_unlock() invocation.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

Referenced by rtdm_select_bind().

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

Get socket destination address.

Refer to rt_dev_getpeername() for parameters and return values

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

Generated on Thu Jul 7 2016 13:25:19 for Xenomai RTDM skin API by Doxygen
5.7.2.10 int rtdm_getsockname ( int fd, struct sockaddr * name, socklen_t * namelen )

Get local socket address.
Refer to rt_dev_getsockname() for parameters and return values
Environments:
Depends on driver implementation, see Device Profiles.
Rescheduling: possible.

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

Get socket option.
Refer to rt_dev_getsockopt() for parameters and return values
Environments:
Depends on driver implementation, see Device Profiles.
Rescheduling: possible.

5.7.2.12 int rtdm_ioctl ( int fd, int request, ... )

Issue an IOCTL.
Refer to rt_dev_ioctl() for parameters and return values
Environments:
Depends on driver implementation, see Device Profiles.
Rescheduling: possible.

5.7.2.13 int rtdm_listen ( int fd, int backlog )

Listen for incomming connection requests.
Refer to rt_dev_listen() for parameters and return values
Environments:
Depends on driver implementation, see Device Profiles.
Rescheduling: possible.

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

Open a device.
Refer to rt_dev_open() for parameters and return values
Environments:
Depends on driver implementation, see Device Profiles.
Rescheduling: possible.

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

Read from device.
5.7.2.16 ssize_t rtdm_recv (int fd, void *buf, size_t len, int flags)

Receive message from socket.
Refer to rt_dev_recv() for parameters and return values
Environments:
Depends on driver implementation, see Device Profiles.
Rescheduling: possible.

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

Receive message from socket.
Refer to rt_dev_recvfrom() for parameters and return values
Environments:
Depends on driver implementation, see Device Profiles.
Rescheduling: possible.

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

Receive message from socket.
Refer to rt_dev_recvmsg() for parameters and return values
Environments:
Depends on driver implementation, see Device Profiles.
Rescheduling: possible.

5.7.2.19 int rtdm_select_bind (int fd, rtdm_selector_t *selector, enum rtdm_selecttype type, unsigned fd_index)

Bind a selector to specified event types of a given file descriptor.
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>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in, out</td>
<td>selector</td>
</tr>
<tr>
<td>in</td>
<td>type</td>
</tr>
<tr>
<td>in</td>
<td>fd_index</td>
</tr>
</tbody>
</table>
Returns

0 on success, otherwise:

- -EBADF is returned if the file descriptor \textit{fd} cannot be resolved.
- -EINVAL is returned if \textit{type} or \textit{fd\_index} are invalid.

Environments:
This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.
References \texttt{rtdm\_dev\_context::ops}, \texttt{rtdm\_context\_get()}, \texttt{rtdm\_context\_unlock()}, and \texttt{rtdm\_operations-\_\_select\_bind}.

5.7.2.20 \texttt{ssize\_t rtdm\_send ( int fd, const void \* buf, size\_t len, int flags )}

Transmit message to socket.
Refer to \texttt{rt\_dev\_send()} for parameters and return values

Environments:
Depends on driver implementation, see \textbf{Device Profiles}.
Rescheduling: possible.

5.7.2.21 \texttt{ssize\_t rtdm\_sendmsg ( int fd, const struct user\_msghdr \* msg, int flags )}

Transmit message to socket.
Refer to \texttt{rt\_dev\_sendmsg()} for parameters and return values

Environments:
Depends on driver implementation, see \textbf{Device Profiles}.
Rescheduling: possible.

5.7.2.22 \texttt{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 \texttt{rt\_dev\_sendto()} for parameters and return values

Environments:
Depends on driver implementation, see \textbf{Device Profiles}.
Rescheduling: possible.
5.7.2.23 int rtdm_setsockopt ( int fd, int level, int optname, const void∗ optval, socklen_t optlen )

Set socket option.
Refer to rt_dev_setsockopt() for parameters and return values
Environments:
Depends on driver implementation, see Device Profiles.
Rescheduling: possible.

5.7.2.24 int rtdm_shutdown ( int fd, int how )

Shut down parts of a connection.
Refer to rt_dev_shutdown() for parameters and return values
Environments:
Depends on driver implementation, see Device Profiles.
Rescheduling: possible.

5.7.2.25 int rtdm_socket ( int protocol_family, int socket_type, int protocol )

Create a socket.
Refer to rt_dev_socket() for parameters and return values
Environments:
Depends on driver implementation, see Device Profiles.
Rescheduling: possible.

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

Write to device.
Refer to rt_dev_write() for parameters and return values
Environments:
Depends on driver implementation, see Device Profiles.
Rescheduling: possible.
5.8 Device Registration Services

Modules

- Synchronisation Services

Data Structures

- struct rtdm_operations
  
  Device operations.
- struct rtdm_dev_context
  
  Device context.
- struct rtdm_device
  
  RTDM device.

Functions

- static void * rtdm_context_to_private (struct rtdm_dev_context *context)
  
  Locate the driver private area associated to a device context structure.
- static struct rtdm_dev_context * rtdm_private_to_context (void *dev_private)
  
  Locate a device context structure from its driver private area.
- int rtdm_dev_register (struct rtdm_device *device)
  
  Register a RTDM device.
- int rtdm_dev_unregister (struct rtdm_device *device, unsigned int poll_delay)
  
  Unregisters a RTDM device.

Device Flags

Static flags describing a RTDM device

- #define RTDM_EXCLUSIVE 0x0001
  
  If set, only a single instance of the device can be requested by an application.
- #define RTDM_NAMED_DEVICE 0x0010
  
  If set, the device is addressed via a clear-text name.
- #define RTDM_PROTOCOL_DEVICE 0x0020
  
  If set, the device is addressed via a combination of protocol ID and socket type.
- #define RTDM_DEVICE_TYPE_MASK 0x00F0
  
  Mask selecting the device type.

Context Flags

Dynamic flags describing the state of an open RTDM device (bit numbers)

- #define RTDM_CREATED_IN_NRT 0
  
  Set by RTDM if the device instance was created in non-real-time context.
- #define RTDM_CLOSING 1
  
  Set by RTDM when the device is being closed.
- #define RTDM_USER_CONTEXT_FLAG 8 /* first user-definable flag */
  
  Lowest bit number the driver developer can use freely.
Driver Versioning

Current revisions of RTDM structures, encoding of driver versions. See API Versioning for the interface revision.

- `#define RTDM_DEVICE_STRUCT_VER 5`
  
  Version of struct `rtdm_device`.
- `#define RTDM_CONTEXT_STRUCT_VER 3`
  
  Version of struct `rtdm_dev_context`.
- `#define RTDM_SECURE_DEVICE 0x80000000`
  
  Flag indicating a secure variant of RTDM (not supported here).
- `#define RTDM_DRIVER_VER (major, minor, patch) (((major & 0xFF) << 16) | ((minor & 0xFF) << 8) | (patch & 0xFF))`
  
  Version code constructor for driver revisions.
- `#define RTDM_DRIVER_MAJOR_VER (ver) (((ver) >> 16) & 0xFF)`
  
  Get major version number from driver revision code.
- `#define RTDM_DRIVER_MINOR_VER (ver) (((ver) >> 8) & 0xFF)`
  
  Get minor version number from driver revision code.
- `#define RTDM_DRIVER_PATCH_VER (ver) (ver) & 0xFF`
  
  Get patch version number from driver revision code.

Operation Handler Prototypes

- `typedef int (*rtdm_open_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info, int flags)`
  
  Named device open handler.
- `typedef int (*rtdm_socket_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info, int protocol)`
  
  Socket creation handler for protocol devices.
- `typedef int (*rtdm_close_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info)`
  
  Close handler.
- `typedef int (*rtdm_ioctl_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info, unsigned int request, void __user *arg)`
  
  IOCTL handler.
- `typedef int (*rtdm_select_bind_handler_t)(struct rtdm_dev_context *context, rtdm_selector_t *selector, enum rtdm_selecttype type, unsigned fd_index)`
  
  Select binding handler.
- `typedef ssize_t (*rtdm_read_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info, void *buf, size_t nbyte)`
  
  Read handler.
- `typedef ssize_t (*rtdm_write_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info, const void *buf, size_t nbyte)`
  
  Write handler.
- `typedef ssize_t (*rtdm_recvmsg_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info, struct user_msghdr *msg, int flags)`
  
  Receive message handler.
- `typedef ssize_t (*rtdm_sendmsg_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info, const struct user_msghdr *msg, int flags)`
  
  Transmit message handler.
5.8.1 Detailed Description

5.8.2 Macro Definition Documentation

5.8.2.1 `#define RTDM_CLOSING 1`

Set by RTDM when the device is being closed.

5.8.2.2 `#define RTDM_CREATED_IN_NRT 0`

Set by RTDM if the device instance was created in non-real-time context.

5.8.2.3 `#define RTDM_DEVICE_TYPE_MASK 0x00F0`

Mask selecting the device type.

Referenced by `rtdm_dev_register()`, and `rtdm_dev_unregister()`.

5.8.2.4 `#define RTDM_EXCLUSIVE 0x0001`

If set, only a single instance of the device can be requested by an application.

Referenced by `rtdm_dev_register()`.

5.8.2.5 `#define RTDM_NAMED_DEVICE 0x0010`

If set, the device is addressed via a clear-text name.

Referenced by `rtdm_dev_register()`, and `rtdm_dev_unregister()`.

5.8.2.6 `#define RTDM_PROTOCOL_DEVICE 0x0020`

If set, the device is addressed via a combination of protocol ID and socket type.

Referenced by `rtdm_dev_register()`.

5.8.3 Typedef Documentation

5.8.3.1 `typedef int(* rtdm_close_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info)`

Close handler.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>context</th>
<th>Context structure associated with opened device instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>user_info</td>
<td>Opaque pointer to information about user mode caller, NULL if kernel mode or deferred user mode call</td>
</tr>
</tbody>
</table>

Returns

0 on success. On failure return either -ENOSYS, to request that this handler be called again from the opposite realtime/non-realtime context, -EAGAIN to request a recall after a grace period, or a valid negative error code according to IEEE Std 1003.1.
Drivers must be prepared for that case that the close handler is invoked more than once per open context (even if the handler already completed an earlier run successfully). The driver has to avoid releasing resources twice as well as returning false errors on successive close invocations.

See Also


5.8.3.2 typedef int(∗ rtdm_ioctl_handler_t)(struct rtdm_dev_context ∗context, rtdm_user_info_t ∗user_info, unsigned int request, void __user ∗arg)

IOCTL handler.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>context</th>
<th>Context structure associated with opened device instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>user_info</td>
<td>Opaque pointer to information about user mode caller, NULL if kernel mode call</td>
</tr>
<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


5.8.3.3 typedef int(∗ rtdm_open_handler_t)(struct rtdm_dev_context ∗context, rtdm_user_info_t ∗user_info, int oflag)

Named device open handler.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>context</th>
<th>Context structure associated with opened device instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>user_info</td>
<td>Opaque pointer to information about user mode caller, NULL if kernel mode call</td>
</tr>
<tr>
<td>in</td>
<td>oflag</td>
<td>Open flags as passed by the user</td>
</tr>
</tbody>
</table>

Returns

0 on success. On failure return either -ENOSYS, to request that this handler be called again from the opposite realtime/non-realtime context, or another negative error code.

See Also


5.8.3.4 typedef ssize_t(∗ rtdm_read_handler_t)(struct rtdm_dev_context ∗context, rtdm_user_info_t ∗user_info, void ∗buf, size_t nbyte)

Read handler.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>context</th>
<th>Context structure associated with opened device instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>user_info</td>
<td>Opaque pointer to information about user mode caller, NULL if kernel mode call</td>
</tr>
<tr>
<td>out</td>
<td>buf</td>
<td>Input buffer as passed by the user</td>
</tr>
<tr>
<td>in</td>
<td>nbyte</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


5.8.3.5 typedef ssize_t(* rtdm_recvmsg_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info, struct user_msghdr *msg, int flags)

Receive message handler.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>context</th>
<th>Context structure associated with opened device instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>user_info</td>
<td>Opaque pointer to information about user mode caller, NULL if kernel mode call</td>
</tr>
<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


5.8.3.6 typedef int(* rtdm_select_bind_handler_t)(struct rtdm_dev_context *context, rtdm_selector_t *selector, enum rtdm_selecttype type, unsigned fd_index)

Select binding handler.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>context</th>
<th>Context structure associated with opened device instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>in,out</td>
<td>selector</td>
<td>Object that shall be bound to the given event</td>
</tr>
</tbody>
</table>
### 5.8 Device Registration Services

<table>
<thead>
<tr>
<th>in</th>
<th>type</th>
<th>Event type the selector is interested in</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>fd_index</td>
<td>Opaque value, to be passed to rtdm_event_select_bind or rtdm_sem_select_bind unmodified</td>
</tr>
</tbody>
</table>

Returns

0 on success. On failure return either -ENOSYS, to request that this handler be called again from the opposite realtime/non-realtime context, or another negative error code.

#### 5.8.3.7 typedef ssize_t(* rtdm_sendmsg_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info, const struct user_msghdr *msg, int flags)

Transmit message handler.

<table>
<thead>
<tr>
<th>in</th>
<th>context</th>
<th>Context structure associated with opened device instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>user_info</td>
<td>Opaque pointer to information about user mode caller, NULL if kernel mode call</td>
</tr>
<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

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

#### 5.8.3.8 typedef int(* rtdm_socket_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info, int protocol)

Socket creation handler for protocol devices.

<table>
<thead>
<tr>
<th>in</th>
<th>context</th>
<th>Context structure associated with opened device instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>user_info</td>
<td>Opaque pointer to information about user mode caller, NULL if kernel mode call</td>
</tr>
<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 return either -ENOSYS, to request that this handler be called again from the opposite realtime/non-realtime context, or another negative error code.

See Also

socket() in IEEE Std 1003.1, [http://www.opengroup.org/onlinepubs/009695399](http://www.opengroup.org/onlinepubs/009695399)
5.8.3.9 typedef ssize_t(rtmm_write_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info, const void *buf, size_t nbyte)

Write handler.
5.8 Device Registration Services

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>context</td>
<td>Context structure associated with opened device instance</td>
</tr>
<tr>
<td>in</td>
<td>user_info</td>
<td>Opaque pointer to information about user mode caller, NULL if kernel mode call</td>
</tr>
<tr>
<td>in</td>
<td>buf</td>
<td>Output buffer as passed by the user</td>
</tr>
<tr>
<td>in</td>
<td>nbyte</td>
<td>Number of bytes the user requests to write</td>
</tr>
</tbody>
</table>

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


5.8.4 Function Documentation

5.8.4.1 static void* rtdm_context_to_private ( struct rtdm_dev_context * context ) [inline], [static]

Locate the driver private area associated to a device context structure.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>context</td>
<td>Context structure associated with opened device instance</td>
</tr>
</tbody>
</table>

Returns

The address of the private driver area associated to context.

References rtdm_dev_context::dev_private.

5.8.4.2 int rtdm_dev_register ( struct rtdm_device * device )

Register a RTDM device.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>device</td>
<td>Pointer to structure describing the new device.</td>
</tr>
</tbody>
</table>

Returns

0 is returned upon success. Otherwise:

- -EINVAL is returned if the device structure contains invalid entries. Check kernel log in this case.
- -ENOMEM is returned if the context for an exclusive device cannot be allocated.
- -EEXIST is returned if the specified device name of protocol ID is already in use.
- -EAGAIN is returned if some /proc entry cannot be created.

Environments:

This service can be called from:
Kernel module initialization/cleanup code

Rescheduling: never.

References rtdm_operations::close_nrt, rtdm_operations::close_rt, rtdm_device::context_size, rtdm_device::device_class, rtdm_device::device_flags, rtdm_device::device_name, rtdm_device::device_subclass, rtdm_device::driver_version, rtdm_device::open_rt, rtdm_device::ops, rtdm_device::proc_name, rtdm_device::profile_version, rtdm_device::protocol_family, rtdm_device::reserved, RTDM_DEVICE_STRUCTURE_VER, RTDM_DEVICE_TYPE_MASK, RTDM_EXCLUSIVE, RTDM_NAMED_DEVICE, RTDM_PROTOCOL_DEVICE, rtdm_operations::select_bind, rtdm_device::socket_rt, rtdm_device::socket_type, and rtdm_device::struct_version.

5.8.4.3 int rtdm_dev_unregister ( struct rtdm_device * device, unsigned int poll_delay )

Unregisters a RTDM device.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>device</th>
<th>Pointer to structure describing the device to be unregistered.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>poll_delay</td>
<td>Polling delay in milliseconds to check repeatedly for open instances of device, or 0 for non-blocking mode.</td>
</tr>
</tbody>
</table>

Returns

0 is returned upon success. Otherwise:

- -ENODEV is returned if the device was not registered.
- -EAGAIN is returned if the device is busy with open instances and 0 has been passed for poll_delay.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code

Rescheduling: never.

References rtdm_device::device_flags, rtdm_device::device_name, rtdm_device::protocol_family, rtdm_device::reserved, RTDM_DEVICE_TYPE_MASK, RTDM_NAMED_DEVICE, and rtdm_device::socket_type.

5.8.4.4 static struct rtdm_dev_context* rtdm_private_to_context ( void * dev_private ) [static]

Locate a device context structure from its driver private area.

Parameters

| in | dev_private | Address of a private context area |

Returns

The address of the device context structure defining dev_private.
5.9 Driver Development API

Modules

- Inter-Driver API
- Device Registration Services
- Clock Services
- Task Services
- Timer Services
- Synchronisation Services
- Interrupt Management Services
- Non-Real-Time Signalling Services
- Utility Services

Files

- file rtdm_driver.h

  Real-Time Driver Model for Xenomai, driver API header.

5.9.1 Detailed Description

This is the lower interface of RTDM provided to device drivers, currently limited to kernel-space. Real-time drivers should only use functions of this interface in order to remain portable.
5.10 Clock Services

Functions

- nanosecs_abs_t rtdm_clock_read (void)
  Get system time.
- nanosecs_abs_t rtdm_clock_read_monotonic (void)
  Get monotonic time.

5.10.1 Detailed Description

5.10.2 Function Documentation

5.10.2.1 nanosecs_abs_t rtdm_clock_read ( void )

Get system time.

Returns
- The system time in nanoseconds is returned

Note
- The resolution of this service depends on the system timer. In particular, if the system timer is running in periodic mode, the return value will be limited to multiples of the timer tick period. The system timer may have to be started to obtain valid results. Whether this happens automatically (as on Xenomai) or is controlled by the application depends on the RTDM host environment.

Environments:
- This service can be called from:
  - Kernel module initialization/cleanup code
  - Interrupt service routine
  - Kernel-based task
  - User-space task (RT, non-RT)

Rescheduling: never.

Referenced by rtdm_ratelimit().

5.10.2.2 nanosecs_abs_t rtdm_clock_read_monotonic ( void )

Get monotonic time.

Returns
- The monotonic time in nanoseconds is returned
Note

The resolution of this service depends on the system timer. In particular, if the system timer is running in periodic mode, the return value will be limited to multiples of the timer tick period. The system timer may have to be started to obtain valid results. Whether this happens automatically (as on Xenomai) or is controlled by the application depends on the RTDM host environment.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.
5.11 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.
- void rtdm_task_set_priority (rtdm_task_t *task, int priority)
  
  Adjust real-time task priority.
- int rtdm_task_set_period (rtdm_task_t *task, nanosecs_rel_t period)
  
  Adjust real-time task period.
- int rtdm_task_wait_period (void)
  
  Wait on next real-time task period.
- int rtdm_task_unblock (rtdm_task_t *task)
  
  Activate a blocked real-time task.
- rtdm_task_t * rtdm_task_current (void)
  
  Get current real-time task.
- int rtdm_task_sleep (nanosecs_rel_t delay)
  
  Sleep a specified amount of time.
- int rtdm_task_sleep_until (nanosecs_abs_t wakeup_time)
  
  Sleep until a specified absolute time.
- int rtdm_task_sleep_abs (nanosecs_abs_t wakeup_time, enum rtdm_timer_mode mode)
  
  Sleep until a specified absolute time.
- void rtdm_task_join_nrt (rtdm_task_t *task, unsigned int poll_delay)
  
  Wait on a real-time task to terminate.
- void rtdm_task_busy_sleep (nanosecs_rel_t delay)
  
  Busy-wait a specified amount of time.

Task Priority Range

Maximum and minimum task priorities

- #define RTDM_TASK_LOWEST_PRIORITY XNSCHED_LOW_PRIO
- #define RTDM_TASK_HIGHEST_PRIORITY XNSCHED_HIGH_PRIO

Task Priority Modification

Raise or lower task priorities by one level

- #define RTDM_TASK_RAISE_PRIORITY (+1)
- #define RTDM_TASK_LOWER_PRIORITY (-1)
5.11.1 Detailed Description

5.11.2 Typedef Documentation

5.11.2.1 typedef void(__rtdm_task_proc_t)(void *arg)

Real-time task procedure.
Parameters

| in, out | arg | argument as passed to rtdm_task_init() |

5.11.3 Function Documentation

5.11.3.1 void rtdm_task_busy_sleep ( nanosecs_rel_t delay )

Busy-wait a specified amount of time.
Parameters

| in | delay | Delay in nanoseconds. Note that a zero delay does not have the meaning of RTDM_TIMEOUT_INFINITE here. |

Note

The caller must not be migratable to different CPUs while executing this service. Otherwise, the actual delay will be undefined.

Environments:
This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine (should be avoided or kept short)
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never (except due to external interruptions).

5.11.3.2 rtdm_task_t* rtdm_task_current ( void )

Get current real-time task.

Returns

- Pointer to task handle

Environments:
This service can be called from:

- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.
5.11.3.3 void rtdm_task_destroy ( rtdm_task_t * task )

Destroy a real-time task.
5.11 Task Services

Parameters

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in,out task</td>
<td>Task handle as returned by <code>rtdm_task_init()</code></td>
</tr>
</tbody>
</table>

Note

Passing the same task handle to RTDM services after the completion of this function is not allowed.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.11.3.4 int rtdm_task_init ( rtdm_task_t *task, const char *name, rtdm_task_proc_t task_proc, void *arg, int priority, nanosecs_rel_t period )

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_nrt()` was invoked.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in, out task</td>
<td>Task handle</td>
</tr>
<tr>
<td>in name</td>
<td>Optional task name</td>
</tr>
<tr>
<td>in task_proc</td>
<td>Procedure to be executed by the task</td>
</tr>
<tr>
<td>in arg</td>
<td>Custom argument passed to <code>task_proc()</code> on entry</td>
</tr>
<tr>
<td>in priority</td>
<td>Priority of the task, see also <code>Task Priority Range</code></td>
</tr>
<tr>
<td>in period</td>
<td>Period in nanoseconds of a cyclic task, 0 for non-cyclic mode</td>
</tr>
</tbody>
</table>

Returns

0 on success, otherwise negative error code

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: possible.

5.11.3.5 void rtdm_task_join_nrt ( rtdm_task_t *task, unsigned int poll_delay )

Wait on a real-time task to terminate.
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task</td>
<td>Task handle as returned by <code>rtdm_task_init()</code></td>
</tr>
<tr>
<td>poll_delay</td>
<td>Delay in milliseconds between periodic tests for the state of the real-time task. This parameter is ignored if the termination is internally realised without polling.</td>
</tr>
</tbody>
</table>

**Note**

Passing the same task handle to RTDM services after the completion of this function is not allowed. This service does not trigger the termination of the targeted task. The user has to take of this, otherwise `rtdm_task_join_nrt()` will never return.

**Environments:**

This service can be called from:

- Kernel module initialization/cleanup code
- User-space task (non-RT)

Rescheduling: possible.

### 5.11.3.6 int rtdm_task_set_period (rtdm_task_t *task, nanosecs_rel_t period)

Adjust real-time task period.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task</td>
<td>Task handle as returned by <code>rtdm_task_init()</code></td>
</tr>
<tr>
<td>period</td>
<td>New period in nanoseconds of a cyclic task, 0 for non-cyclic mode</td>
</tr>
</tbody>
</table>

**Environments:**

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: possible.

### 5.11.3.7 void rtdm_task_set_priority (rtdm_task_t *task, int priority)

Adjust real-time task priority.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task</td>
<td>Task handle as returned by <code>rtdm_task_init()</code></td>
</tr>
<tr>
<td>priority</td>
<td>New priority of the task, see also Task Priority Range</td>
</tr>
</tbody>
</table>

**Environments:**

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
• Kernel-based task
• User-space task (RT, non-RT)

Rescheduling: possible.

### 5.11.3.8 int rtdm_task_sleep (nanosecs_rel_t delay)

Sleep a specified amount of time.

**Parameters**

| in  | delay | Delay in nanoseconds, see RTDM_TIMEOUT_xxx for special values. |

**Returns**

- 0 on success, otherwise:
  - -EINTR is returned if calling task has been unblock by a signal or explicitly via rtdm_task_unblock().
  - -EPERM may be returned if an illegal invocation environment is detected.

**Environments:**

This service can be called from:

- Kernel-based task
- User-space task (RT)

Rescheduling: always.

### 5.11.3.9 int rtdm_task_sleep_abs (nanosecs_abs_t wakeup_time, enum rtdm_timer_mode mode)

Sleep until a specified absolute time.

**Parameters**

| in  | wakeup_time | Absolute timeout in nanoseconds |
| in  | mode | Selects the timer mode, see RTDM_TIMERMODE_xxx for details |

**Returns**

- 0 on success, otherwise:
  - -EINTR is returned if calling task has been unblock by a signal or explicitly via rtdm_task_unblock().
  - -EPERM may be returned if an illegal invocation environment is detected.
  - -EINVAL is returned if an invalid parameter was passed.

**Environments:**

This service can be called from:

- Kernel-based task
- User-space task (RT)

Rescheduling: always, unless the specified time already passed.
5.11.3.10  int rtdm_task_sleep_until ( nanosecs_abs_t wakeup_time )

Sleep until a specified absolute time.

**Deprecated** Use rtdm_task_sleep_abs instead!

Parameters

| in   | wakeup_time | Absolute timeout in nanoseconds |

Returns

0 on success, otherwise:

- -EINTR is returned if calling task has been unblock by a signal or explicitly via rtdm_task_unblock().
- -EPERM may be returned if an illegal invocation environment is detected.

Environments:

This service can be called from:

- Kernel-based task
- User-space task (RT)

Rescheduling: always, unless the specified time already passed.

5.11.3.11  int rtdm_task_unblock ( rtdm_task_t * task )

Activate a blocked real-time task.

Returns

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

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: possible.

5.11.3.12  int rtdm_task_wait_period ( void )

Wait on next real-time task period.
Returns

0 on success, otherwise:

- -EINVAL is returned if calling task is not in periodic mode.
- -ETIMEDOUT is returned if a timer overrun occurred, which indicates that a previous release point has been missed by the calling task.

Environments:

This service can be called from:

- Kernel-based task
- User-space task (RT)

Rescheduling: always, unless a timer overrun occurred.
5.12  Timer Services

Typedefs

- typedef void(* rtdm_timer_handler_t)(rtdm_timer_t *timer)
  
  *Timer handler.*

Functions

- int rtdm_timer_init (rtdm_timer_t *timer, rtdm_timer_handler_t handler, const char *name)
  
  *Initialise a timer.*

- void rtdm_timer_destroy (rtdm_timer_t *timer)
  
  *Destroy a timer.*

- int rtdm_timer_start (rtdm_timer_t *timer, nanosecs_abs_t expiry, nanosecs_rel_t interval, enum rtdm_timer_mode mode)
  
  *Start a timer.*

- void rtdm_timer_stop (rtdm_timer_t *timer)
  
  *Stop a timer.*

- int rtdm_timer_start_in_handler (rtdm_timer_t *timer, nanosecs_abs_t expiry, nanosecs_rel_t interval, enum rtdm_timer_mode mode)
  
  *Start a timer from inside a timer handler.*

- void rtdm_timer_stop_in_handler (rtdm_timer_t *timer)
  
  *Stop a timer from inside a timer handler.*

RTDM_TIMERMODE_xxx

Timer operation modes

- enum rtdm_timer_mode {
  RTDM_TIMERMODE_RELATIVE = XN_RELATIVE,
  RTDM_TIMERMODE_ABSOLUTE = XN_ABSOLUTE,
  RTDM_TIMERMODE_REALTIME = XN_REALTIME
}

5.12.1  Detailed Description

5.12.2  Typedef Documentation

5.12.2.1  typedef void(* rtdm_timer_handler_t)(rtdm_timer_t *timer)

*Timer handler.*

Parameters

| in   | timer | Timer handle as returned by rtdm_timer_init() |

5.12.3  Enumeration Type Documentation

5.12.3.1  enum rtdm_timer_mode

**Enumerator**

- RTDM_TIMERMODE_RELATIVE  Monotonic timer with relative timeout.
- RTDM_TIMERMODE_ABSOLUTE  Monotonic timer with absolute timeout.
- RTDM_TIMERMODE_REALTIME  Adjustable timer with absolute timeout.
5.12.4 Function Documentation

5.12.4.1 void rtdm_timer_destroy ( rtdm_timer_t * timer )

Destroy a timer.

Parameters

| in, out | timer   | Timer handle as returned by rtdm_timer_init() |

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.12.4.2 int rtdm_timer_init ( rtdm_timer_t * timer, rtdm_timer_handler_t handler, const char * name )

Initialise a timer.

Parameters

| in, out | timer   | Timer handle                        |
| in      | handler | Handler to be called on timer expiry |
| in      | name    | Optional timer name                  |

Returns

0 on success, otherwise negative error code

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.12.4.3 int rtdm_timer_start ( rtdm_timer_t * timer, nanosecs_abs_t expiry, nanosecs_rel_t interval, enum rtdm_timer_mode mode )

Start a timer.

Parameters
 Returns

0 on success, otherwise:

- `-ETIMEDOUT` is returned if `expiry` describes an absolute date in the past.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.12.4.5  void rtdm_timer_stop ( rtdm_timer_t * timer )

Stop a timer.

Parameters

<table>
<thead>
<tr>
<th>in, out</th>
<th>timer</th>
<th>Timer handle as returned by <code>rtdm_timer_init()</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>expiry</td>
<td>Firing time of the timer, <code>mode</code> 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 <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.

Environments:

This service can be called from:

- Timer handler

Rescheduling: never.
Parameters

| in, out | timer | Timer handle as returned by rtdm_timer_init() |

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.12.4.6 void rtdm_timer_stop_in_handler ( rtdm_timer_t * timer )

Stop a timer from inside a timer handler.

Parameters

| in, out | timer | Timer handle as returned by rtdm_timer_init() |

Environments:

This service can be called from:

- Timer handler

Rescheduling: never.
5.13 Synchronisation Services

Functions

- int rtdm_select_bind (int fd, rtdm_selector_t *selector, enum rtdm_selecttype type, unsigned fd_index)
  
  *Bind a selector to specified event types of a given file descriptor.*

RTDM_SELECTTYPE_xxx

Event types select can bind to

- enum rtdm_selecttype { RTDM_SELECTTYPE_READ = XNSELECT_READ, RTDM_SELECTTYPE_WRITE = XNSELECT_WRITE, RTDM_SELECTTYPE_EXCEPT = XNSELECT_EXCEPT }

Global Lock across Scheduler Invocation

- #define RTDM_EXECUTE_ATOMICALLY (code_block)
  
  *Execute code block atomically.*

Spinlock with Preemption Deactivation

- typedef rthal_spinlock_t rtdm_lock_t
  
  *Lock variable.*

- typedef unsigned long rtdm_lockctx_t
  
  *Variable to save the context while holding a lock.*

- #define RTDM_LOCK_UNLOCKED RTHAL_SPIN_LOCK_UNLOCKED
  
  *Static lock initialisation.*

- #define rtdm_lock_init(lock) rthal_spin_lock_init(lock)
  
  *Dynamic lock initialisation.*

- #define rtdm_lock_get(lock) rthal_spin_lock(lock)
  
  *Acquire lock from non-preemptible contexts.*

- #define rtdm_lock_put(lock)
  
  *Release lock without preemption restoration.*

- #define rtdm_lock_get_irqsave(lock, context)
  
  *Acquire lock and disable preemption.*

- #define rtdm_lock_put_irqrestore(lock, context)
  
  *Release lock and restore preemption state.*

- #define rtdm_lock_irqsave(context) rthal_local_irq_save(context)
  
  *Disable preemption locally.*

- #define rtdm_lock_irqrestore(context) rthal_local_irq_restore(context)
  
  *Restore preemption state.*

Timeout Sequence Management

- void rtdm_toseq_init (rtdm_toseq_t *timeout_seq, nanosecs_rel_t timeout)
  
  *Initialise a timeout sequence.*

- EXPORT_SYMBOL_GPL (rtdm_toseq_init)
  
  *Initialise a timeout sequence.*

Generated on Thu Jul 7 2016 13:25:19 for Xenomai RTDM skin API by Doxygen
Event Services

- **void rtdm_event_init** (rtdm_event_t *event, unsigned long pending)
  
  Initialises an event.

- **EXPORT_SYMBOL_GPL (rtdm_event_init)**
  
  Initialises an event.

- **void rtdm_event_destroy** (rtdm_event_t *event)
  
  Destroys an event.

- **void rtdm_event_pulse** (rtdm_event_t *event)
  
  Signals an event occurrence to currently listening waiters.

- **void rtdm_event_signal** (rtdm_event_t *event)
  
  Signals an event occurrence.

- **EXPORT_SYMBOL_GPL (rtdm_event_signal)**
  
  Initialises an event.

- **void rtdm_event_wait** (rtdm_event_t *event)
  
  Waits on event occurrence.

- **EXPORT_SYMBOL_GPL (rtdm_event_wait)**
  
  Initialises an event.

- **int rtdm_event_timedwait** (rtdm_event_t *event, nanosecs_rel_t timeout, rtdm_toseq_t *timeout_seq)
  
  Waits on event occurrence with timeout.

- **EXPORT_SYMBOL_GPL (rtdm_event_timedwait)**
  
  Initialises an event.

- **void rtdm_event_clear** (rtdm_event_t *event)
  
  Clears event state.

- **EXPORT_SYMBOL_GPL (rtdm_event_clear)**
  
  Initialises an event.

- **void rtdm_event_select_bind** (rtdm_event_t *event, rtdm_selector_t *selector, enum rtdm_selecttype type, unsigned fd_index)
  
  Binds a selector to an event.

- **EXPORT_SYMBOL_GPL (rtdm_event_select_bind)**
  
  Initialises an event.

Semaphore Services

- **void rtdm_sem_init** (rtdm_sem_t *sem, unsigned long value)
  
  Initialises a semaphore.

- **EXPORT_SYMBOL_GPL (rtdm_sem_init)**
  
  Initialises a semaphore.

- **void rtdm_sem_destroy** (rtdm_sem_t *sem)
  
  Destroys a semaphore.

- **int rtdm_sem_down** (rtdm_sem_t *sem)
  
  Decrements a semaphore.

- **EXPORT_SYMBOL_GPL (rtdm_sem_down)**
  
  Initialises a semaphore.

- **int rtdm_sem_timeddown** (rtdm_sem_t *sem, nanosecs_rel_t timeout, rtdm_toseq_t *timeout_seq)
  
  Decrements a semaphore with timeout.

- **EXPORT_SYMBOL_GPL (rtdm_sem_timeddown)**
  
  Initialises a semaphore.

- **void rtdm_sem_up** (rtdm_sem_t *sem)
  
  Increments a semaphore.
• **EXPORT_SYMBOL_GPL** *(rtmsem_up)*  
  *Initialise a semaphore.*

• int **rtmsem_select_bind** *(rtmsem_t *sem, rtmselector_t *selector, enum rtmselecttype type, unsigned fd_index)*  
  *Bind a selector to a semaphore.*

• **EXPORT_SYMBOL_GPL** *(rtmsem_select_bind)*  
  *Initialise a semaphore.*

**Mutex Services**

• void **rtmmutex_init** *(rtmmutex_t *mutex)*  
  *Initialise a mutex.*

• **EXPORT_SYMBOL_GPL** *(rtmmutex_init)*  
  *Initialise a mutex.*

• void **rtmmutex_destroy** *(rtmmutex_t *mutex)*  
  *Destroy a mutex.*

• void **rtmmutex_unlock** *(rtmmutex_t *mutex)*  
  *Release a mutex.*

• int **rtmmutex_lock** *(rtmmutex_t *mutex)*  
  *Request a mutex.*

• **EXPORT_SYMBOL_GPL** *(rtmmutex_lock)*  
  *Initialise a mutex.*

• int **rtmmutex_timedlock** *(rtmmutex_t *mutex, nanosecstel timeout, rtmtoseq_t *timeout_seq)*  
  *Request a mutex with timeout.*

• **EXPORT_SYMBOL_GPL** *(rtmmutex_timedlock)*  
  *Initialise a mutex.*

5.13.1  Detailed Description

5.13.2  Macro Definition Documentation

5.13.2.1  `#define RTDM_EXECUTE_ATOMICALLY( code_block )`

**Value:**

```
{
  <ENTER_ATOMIC_SECTION>
  code_block;
  <LEAVE_ATOMIC_SECTION>
}
```

Execute code block atomically.

Generally, it is illegal to suspend the current task by calling `rtm_task_sleep()`, `rtm_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.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: possible, depends on functions called within `code_block`.

5.13.2.2  

```c
#define rtdm_lock_get( lock ) rthal_spin_lock( lock )
```

Acquire lock from non-preemptible contexts.

Parameters

| lock | Address of lock variable |

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.13.2.3  

```c
#define rtdm_lock_get_irqsave( lock, context )
```

Value:

```c
  do { 
    rthal_spin_lock_irqsave(lock, context); |
    __xnpod_lock_sched(); |
  } while (!$)
```

Acquire lock and disable preemption.
5.13.2.4  #define rtdm_lock_init(lock) rthal_spin_lock_init(lock)

Dynamic lock initialisation.

Parameters

<table>
<thead>
<tr>
<th>lock</th>
<th>Address of lock variable</th>
</tr>
</thead>
</table>

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

Referenced by rtdm_ratelimit().

5.13.2.5  #define rtdm_lock_irqrestore(context) rthal_local_irq_restore(context)

Restore preemption state.

Parameters

<table>
<thead>
<tr>
<th>context</th>
<th>name of local variable which stored the context</th>
</tr>
</thead>
</table>

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: possible.

5.13.2.6  #define rtdm_lock_irqsave(context) rthal_local_irq_save(context)

Disable preemption locally.
5.13 Synchronisation Services

Parameters

| context | name of local variable to store the context in |

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.13.2.7  \#define rtdm_lock_put( lock )

Value:

```
  do {
      rthal_spin_unlock(lock);
      __xnpod_unlock_sched();
  } while (0)
```

Release lock without preemption restoration.

Parameters

| lock  | Address of lock variable |

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.13.2.8 \#define rtdm_lock_put_irqrestore( lock, context )

Value:

```
  do {
      rthal_spin_unlock(lock);
      __xnpod_unlock_sched();
      rthal_local_irq_restore(context);
  } while (0)
```

Release lock and restore preemption state.
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 which stored the context</td>
</tr>
</tbody>
</table>

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: possible.

Referenced by rtdm_ratelimt().

5.13.3 Enumeration Type Documentation

5.13.3.1 enum rtdm_selecttype

Enumerator

- RTDM_SELECTTYPE_READ Select input data availability events.
- RTDM_SELECTTYPE_WRITE Select output buffer availability events.
- RTDM_SELECTTYPE_EXCEPT Select exceptional events.

5.13.4 Function Documentation

5.13.4.1 EXPORT_SYMBOL_GPL ( rtdm_toseq_init )

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>in_out</th>
<th>timeout_seq</th>
<th>Timeout sequence handle</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>
</tbody>
</table>

Application Scenario:

```c
int device_service_routine(...) {
    rtdm_toseq_t timeout_seq;
    ...
    rtdm_toseq_init(&timeout_seq, timeout);
    ...
    while (received < requested) {
        ret = rtdm_event_timedwait(data_available, timeout, &timeout_seq);
        if (ret < 0) // including -ETIMEOUT
            break;
        // receive some data
        ...
    }
    ...
}```
Using a timeout sequence in such a scenario avoids that the user-provided relative timeout is restarted on every call to `rtdm_event_timedwait()`, potentially causing an overall delay that is larger than specified by timeout. Moreover, all functions supporting timeout sequences also interpret special timeout values (infinite and non-blocking), disburdening the driver developer from handling them separately.

Environments:
This service can be called from:

- Kernel-based task
- User-space task (RT)

Rescheduling: never.

5.13.4.2 EXPORT_SYMBOL_GPL ( rtdm_event_init )

Initialise an event.
Parameters

<table>
<thead>
<tr>
<th>in,out</th>
<th>event</th>
<th>Event handle</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>pending</td>
<td>Non-zero if event shall be initialised as set, 0 otherwise</td>
</tr>
</tbody>
</table>

Environments:
This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.13.4.3 EXPORT_SYMBOL_GPL ( rtdm_event_signal )

Initialise an event.
Parameters

<table>
<thead>
<tr>
<th>in,out</th>
<th>event</th>
<th>Event handle</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>pending</td>
<td>Non-zero if event shall be initialised as set, 0 otherwise</td>
</tr>
</tbody>
</table>

Environments:
This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.13.4.4 EXPORT_SYMBOL_GPL ( rtdm_event_wait )

Initialise an event.
Parameters

<table>
<thead>
<tr>
<th></th>
<th>event</th>
<th>Event handle</th>
</tr>
</thead>
<tbody>
<tr>
<td>in,out</td>
<td></td>
<td>Event handle</td>
</tr>
<tr>
<td>in</td>
<td>pending</td>
<td>Non-zero if event shall be initialised as set, 0 otherwise</td>
</tr>
</tbody>
</table>

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.13.4.5  EXPORT_SYMBOL_GPL ( rtdm_event_timedwait )

Initialise an event.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>event</th>
<th>Event handle</th>
</tr>
</thead>
<tbody>
<tr>
<td>in,out</td>
<td></td>
<td>Event handle</td>
</tr>
<tr>
<td>in</td>
<td>pending</td>
<td>Non-zero if event shall be initialised as set, 0 otherwise</td>
</tr>
</tbody>
</table>

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.13.4.6  EXPORT_SYMBOL_GPL ( rtdm_event_clear )

Initialise an event.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>event</th>
<th>Event handle</th>
</tr>
</thead>
<tbody>
<tr>
<td>in,out</td>
<td></td>
<td>Event handle</td>
</tr>
<tr>
<td>in</td>
<td>pending</td>
<td>Non-zero if event shall be initialised as set, 0 otherwise</td>
</tr>
</tbody>
</table>

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.13.4.7  EXPORT_SYMBOL_GPL ( rtdm_event_select_bind )

Initialise an event.
5.13 Synchronisation Services

Parameters

<table>
<thead>
<tr>
<th>in,out</th>
<th>event</th>
<th>Event handle</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>pending</td>
<td>Non-zero if event shall be initialised as set, 0 otherwise</td>
</tr>
</tbody>
</table>

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.13.4.8  EXPORT_SYMBOL_GPL ( rtdm_sem_init )

Initialise a semaphore.

Parameters

<table>
<thead>
<tr>
<th>in,out</th>
<th>sem</th>
<th>Semaphore handle</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>value</td>
<td>Initial value of the semaphore</td>
</tr>
</tbody>
</table>

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.13.4.9  EXPORT_SYMBOL_GPL ( rtdm_sem_down )

Initialise a semaphore.

Parameters

<table>
<thead>
<tr>
<th>in,out</th>
<th>sem</th>
<th>Semaphore handle</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>value</td>
<td>Initial value of the semaphore</td>
</tr>
</tbody>
</table>

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.13.4.10  EXPORT_SYMBOL_GPL ( rtdm_sem_timeddown )

Initialise a semaphore.
### 5.13.4.11 **EXPORT_SYMBOL_GPL ( rtdm_sem_up )**

Initialise a semaphore.

**Parameters**

<table>
<thead>
<tr>
<th>in, out</th>
<th>sem</th>
<th>Semaphore handle</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>value</td>
<td>Initial value of the semaphore</td>
</tr>
</tbody>
</table>

**Environments:**

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

### 5.13.4.12 **EXPORT_SYMBOL_GPL ( rtdm_sem_select_bind )**

Initialise a semaphore.

**Parameters**

<table>
<thead>
<tr>
<th>in, out</th>
<th>sem</th>
<th>Semaphore handle</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>value</td>
<td>Initial value of the semaphore</td>
</tr>
</tbody>
</table>

**Environments:**

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

### 5.13.4.13 **EXPORT_SYMBOL_GPL ( rtdm_mutex_init )**

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 |

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.13.4.14  EXPORT_SYMBOL_GPL ( rtdm_mutex_lock )

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 |

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.13.4.15  EXPORT_SYMBOL_GPL ( rtdm_mutex_timedlock )

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 |

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.13.4.16  void rtdm_event_clear ( rtdm_event_t * event )

Clear event state.
Parameters

| in, out | event | Event handle as returned by \texttt{rtdm\_event\_init()} |

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.13.4.17 \texttt{void rtdm\_event\_destroy ( rtdm\_event\_t * event )}

Destroy an event.

Parameters

| in, out | event | Event handle as returned by \texttt{rtdm\_event\_init()} |

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: possible.

5.13.4.18 \texttt{void rtdm\_event\_init ( rtdm\_event\_t * event, unsigned long pending )}

Initialise an event.

Parameters

| in, out | event | Event handle as returned by \texttt{rtdm\_event\_init()} |
| in      | pending | Non-zero if event shall be initialised as set, 0 otherwise |

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.13.4.19 \texttt{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 \texttt{rtdm\_event\_wait()} or \texttt{rtdm\_event\_timedwait()} will therefore be blocked first.
5.13 Synchronisation Services

Parameters

| in,out | event | Event handle as returned by rtdm_event_init() |

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: possible.

5.13.4.20 int rtdm_event_select_bind ( rtdm_event_t * event, rtdm_selector_t * selector, enum rtdm_selecttype type, unsigned fd_index )

Bind a selector to an event.

This 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.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.13.4.21 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>
<th>Event handle as returned by rtdm_event_init()</th>
</tr>
</thead>
</table>

environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task (RT, non-RT)

rescheduling: possible.

5.13.4.22 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.
- EINVAL is returned if calling task has been unblock by a signal or explicitly via rtdm_task_unblock().
- EIDRM is returned if event has been destroyed.
- EPERM may be returned if an illegal invocation environment is detected.
- EWOULDBLOCK is returned if a negative timeout (i.e., non-blocking operation) has been specified.

environments:

This service can be called from:

- Kernel-based task
- User-space task (RT)

rescheduling: possible.

Referenced by rtdm_event_wait().
5.13.4.23  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.
## 5.13.4.24 void rtdm_mutex_destroy ( rtdm_mutex_t *mutex )

Destroy a mutex.

### Parameters

| in, out | mutex | Mutex handle as returned by rtdm_mutex_init() |

### Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

### Rescheduling:

Possible.

### References

rtdm_event_timedwait().

## 5.13.4.25 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 |

### Environments:

This service can be called from:

- Kernel module initialization/cleanup code
5.13.4.26 int rtdm_mutex_lock ( rtdm_mutex_t *mutex )

Request a mutex.
This is the light-weight version of rtdm_mutex_timedlock(), implying an infinite timeout.

Parameters

| in, out | mutex | Mutex handle as returned by rtdm_mutex_init() |

Returns

0 on success, otherwise:

- -EIDRM is returned if mutex has been destroyed.
- -EPERM may be returned if an illegal invocation environment is detected.

Environments:
This service can be called from:

- Kernel-based task
- User-space task (RT)

Rescheduling: possible.

References rtdm_mutex_timedlock().

5.13.4.27 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 if the request has not been satisfied within the specified amount of time.
- EWOULDBLOCK is returned if \textit{timeout} is negative and the semaphore value is currently not positive.
- EIDRM is returned if \textit{mutex} has been destroyed.
- EPERM \textit{may} be returned if an illegal invocation environment is detected.

Environments:
This service can be called from:

- Kernel-based task
- User-space task (RT)

Rescheduling: possible.

Referenced by \textit{rtdm_mutex_lock}().

5.13.4.28 \textbf{void rtdm_mutex_unlock ( rtdm_mutex_t \ast mutex )}

Release a mutex.
This function releases the given mutex, waking up a potential waiter which was blocked upon \textit{rtdm_mutex_lock()} or \textit{rtdm_mutex_timedlock()}.

Parameters

| in, out | \textit{mutex} | Mutex handle as returned by \textit{rtdm_mutex_init()} |

Environments:
This service can be called from:

- Kernel-based task
- User-space task (RT)

Rescheduling: possible.

5.13.4.29 \textbf{int rtdm_select_bind ( int \textit{fd}, rtdm_selector_t \ast \textit{selector}, enum \textit{rtdm_selecttype} \textit{type}, unsigned \textit{fd_index})}

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>in</th>
<th>\textit{fd}</th>
<th>File descriptor to bind to</th>
</tr>
</thead>
<tbody>
<tr>
<td>in, out</td>
<td>\textit{selector}</td>
<td>Selector object that shall be bound to the given event</td>
</tr>
<tr>
<td>in</td>
<td>\textit{type}</td>
<td>Event type the caller is interested in</td>
</tr>
<tr>
<td>in</td>
<td>\textit{fd_index}</td>
<td>Index in the file descriptor set of the caller</td>
</tr>
</tbody>
</table>

Returns

0 on success, otherwise:

- EBADF is returned if the file descriptor \textit{fd} cannot be resolved.
- EINVAL is returned if \textit{type} or \textit{fd_index} are invalid.
Environments:
This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

References rtdm_dev_context::ops, rtdm_context_get(), rtdm_context_unlock(), and rtdm_operations-::select_bind.

5.13.4.30  void rtdm_sem_destroy ( rtdm_sem_t *sem )

Destroy a semaphore.

Parameters

| in_out | sem | Semaphore handle as returned by rtdm_sem_init() |

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: possible.

5.13.4.31  int rtdm_sem_down ( rtdm_sem_t *sem )

Decrement a semaphore.

This is the light-weight version of rtdm_sem_timeddown(), implying an infinite timeout.

Parameters

| in,out | sem | Semaphore handle as returned by rtdm_sem_init() |

Returns

0 on success, otherwise:

- -EINTR is returned if calling task has been unblock by a signal or explicitly via rtdm_task_unblock().
- -EIDRM is returned if sem has been destroyed.
- -EPERM may be returned if an illegal invocation environment is detected.

Environments:

This service can be called from:

- Kernel-based task
- User-space task (RT)

Rescheduling: possible.

References rtdm_sem_timeddown().
5.13.4.32 void rtdm_sem_init ( rtdm_sem_t *sem, unsigned long value )

Initialise a semaphore.

Parameters

<table>
<thead>
<tr>
<th>in,out</th>
<th>sem</th>
<th>Semaphore handle</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>value</td>
<td>Initial value of the semaphore</td>
</tr>
</tbody>
</table>

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.13.4.33 int rtdm_sem_select_bind ( rtdm_sem_t *sem, rtdm_selector_t *selector, enum rtdm_selecttype type, unsigned fd_index )

Bind a selector to a semaphore.

This function 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.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.13.4.34 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.
5.13 Synchronization Services

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in/out</td>
<td>sem</td>
</tr>
<tr>
<td>in</td>
<td>timeout</td>
</tr>
<tr>
<td>in/out</td>
<td>timeout_seq</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.
- EINTR is returned if calling task has been unblock by a signal or explicitly via rtdm_task_unblock().
- EIDRM is returned if sem has been destroyed.
- EPERM may be returned if an illegal invocation environment is detected.

Environments:

This service can be called from:

- Kernel-based task
- User-space task (RT)

Rescheduling: possible.

Referenced by rtdm_sem_down().

5.13.4.35 void rtdm_sem_up ( rtdm_sem_t *sem )

Increment a semaphore.

This function increments the given semaphore's value, waking up a potential waiter which was blocked upon rtdm_sem_down().

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in/out</td>
<td>sem</td>
</tr>
</tbody>
</table>

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: possible.
5.13.4.36  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>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in,out timeout_seq</td>
<td>Timeout sequence handle</td>
</tr>
<tr>
<td>in timeout</td>
<td>Relative timeout in nanoseconds, see RTDM_TIMEOUT_xxx for special values</td>
</tr>
</tbody>
</table>

Application Scenario:

```c
int device_service_routine(...) {
    rtdm_toseq_t timeout_seq;
    ...
    rtdm_toseq_init(&timeout_seq, timeout);
    ...
    while (received < requested) {
        ret = rtdm_event_timedwait(&data_available, timeout, &timeout_seq);
        if (ret < 0) // including -ETIMEDOUT
            break;
        // receive some data
        ...
    }
    ...
```

Using a timeout sequence in such a scenario avoids that the user-provided relative timeout is restarted on every call to rtdm_event_timedwait(), potentially causing an overall delay that is larger than specified by timeout. Moreover, all functions supporting timeout sequences also interpret special timeout values (infinite and non-blocking), disburdening the driver developer from handling them separately.

Environments:

This service can be called from:

- Kernel-based task
- User-space task (RT)

Rescheduling: never.
5.14 Interrupt Management Services

Macros

- 
  
  ```c
  #define rtdm_irq_get_arg(irq_handle, type) (((type *)irq_handle)->cookie)
  
  Retrieve IRQ handler argument.
  ```

Typedefs

- 
  
  ```c
  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

- 
  
  ```c
  #define RTDM_IRQTYPE_SHARED XN_ISR_SHARED
  
  Enable IRQ-sharing with other real-time drivers.
  ```

- 
  
  ```c
  #define RTDM_IRQTYPE_EDGE XN_ISR_EDGE
  
  Mark IRQ as edge-triggered, relevant for correct handling of shared edge-triggered IRQs.
  ```

RTDM_IRQ_xxx

Return flags of interrupt handlers

- 
  
  ```c
  #define RTDM_IRQ_NONE XN_ISR_NONE
  
  Unhandled interrupt.
  ```

- 
  
  ```c
  #define RTDM_IRQ_HANDLED XN_ISR_HANDLED
  
  Denote handled interrupt.
  ```

5.14.1 Detailed Description

5.14.2 Macro Definition Documentation

5.14.2.1 

```c
#define rtdm_irq_get_arg( irq_handle, type ) ((type *)irq_handle->cookie)
```

Retrieve IRQ handler argument.
### Module Documentation

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>irq_handle</td>
<td>IRQ handle</td>
</tr>
<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`.

#### Environments:

This service can be called from:

- Interrupt service routine

Rescheduling: never.

### 5.14.3 Typedef Documentation

5.14.3.1 typedef int(* rtdm_irq_handler_t)(rtdm_irq_t *irq_handle)

Interrupt handler.

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>irq_handle</td>
<td>IRQ handle as returned by <code>rtdm_irq_request()</code></td>
</tr>
</tbody>
</table>

#### Returns

0 or a combination of `RTDM_IRQ_xxx` flags

### 5.14.4 Function Documentation

5.14.4.1 int rtdm_irq_disable ( rtdm_irq_t *irq_handle )

Disable interrupt line.

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>irq_handle</td>
<td>IRQ handle as returned by <code>rtdm_irq_request()</code></td>
</tr>
</tbody>
</table>

#### 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, certain IRQ types may not allow the invocation over RT and interrupt contexts. The caller is responsible for excluding such conflicts.

#### Environments:

This service can be called from:
5.14 Interrupt Management Services

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.14.4.2 int rtdm_irq_enable ( rtdm_irq_t * irq_handle )

Enable interrupt line.

Parameters

| in, out | irq_handle | IRQ handle as returned by rtdm_irq_request() |

Returns

0 on success, otherwise negative error code

Note

This service is for exceptional use only. Drivers should always prefer interrupt masking at device level (via corresponding control registers etc.) over masking at line level. Keep in mind that the latter is incompatible with IRQ line sharing and can also be more costly as interrupt controller access requires broader synchronization. Also, certain IRQ types may not allow the invocation over RT and interrupt contexts. The caller is responsible for excluding such conflicts.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: possible.

5.14.4.3 int rtdm_irq_free ( rtdm_irq_t * irq_handle )

Release an interrupt handler.

Parameters

| in, out | irq_handle | IRQ handle as returned by rtdm_irq_request() |

Returns

0 on success, otherwise negative error code
The caller is responsible for shutting down the IRQ source at device level before invoking this service. In turn, rtdm_irq_free ensures that any pending event on the given IRQ line is fully processed on return from this service.

Environments:
This service can be called from:

- Kernel module initialization/cleanup code
- User-space task (non-RT)

Rescheduling: never.

5.14.4.4 int rtdm_irq_request ( rtdm_irq_t *irq_handle, unsigned int irq_no, rtdm_irq_handler_t handler, unsigned long flags, const char *device_name, void *arg )

Register an interrupt handler.
This function registers the provided handler with an IRQ line and enables the line.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in,out</td>
<td>irq_handle</td>
<td>IRQ handle</td>
</tr>
<tr>
<td>in</td>
<td>irq_no</td>
<td>Line number of the addressed IRQ</td>
</tr>
<tr>
<td>in</td>
<td>handler</td>
<td>Interrupt handler</td>
</tr>
<tr>
<td>in</td>
<td>flags</td>
<td>Registration flags, see RTDM_IRQTYPE_... for details</td>
</tr>
<tr>
<td>in</td>
<td>device_name</td>
<td>Device name to show up in real-time IRQ lists</td>
</tr>
<tr>
<td>in</td>
<td>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.

Environments:
This service can be called from:

- Kernel module initialization/cleanup code
- User-space task (non-RT)

Rescheduling: never.
5.15 Non-Real-Time Signalling Services

Typedefs

- typedef void\((rtdm_nrtsig_handler_t)(rtdm_nrtsig_t\ nrt\_sig, \ void *\arg)\)
  
  Non-real-time signal handler.

Functions

- int rtdm_nrtsig_init (rtdm_nrtsig_t *nrt\_sig, rtdm_nrtsig_handler_t\ handler, void *arg)
  
  Register a non-real-time signal handler.

- void rtdm_nrtsig_destroy (rtdm_nrtsig_t *nrt\_sig)
  
  Release a non-realtime signal handler.

- void rtdm_nrtsig_pend (rtdm_nrtsig_t *nrt\_sig)
  
  Trigger non-real-time signal.

5.15.1 Detailed Description

These services provide a mechanism to request the execution of a specified handler in non-real-time context. The triggering can safely be performed in real-time context without suffering from unknown delays. The handler execution will be deferred until the next time the real-time subsystem releases the CPU to the non-real-time part.

5.15.2 Typedef Documentation

5.15.2.1 typedef void\((rtdm_nrtsig_handler_t)(rtdm_nrtsig_t\ nrt\_sig, \ void *\arg)\)

Non-real-time signal handler.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>nrt_sig</th>
<th>Signal handle as returned by rtdm_nrtsig_init()</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>arg</td>
<td>Argument as passed to rtdm_nrtsig_init()</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.

5.15.3 Function Documentation

5.15.3.1 void rtdm_nrtsig_destroy ( rtdm_nrtsig_t *\ nrt\_sig )

Release a non-realtime signal handler.

Parameters

| in,out | nrt\_sig | Signal handle |

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
• Kernel-based task
• User-space task (RT, non-RT)

Rescheduling: never.

5.15.3.2 int rtdm_nrtsig_init ( rtdm_nrtsig_t * nrt_sig, rtdm_nrtsig_handler_t handler, void * arg )

Register a non-real-time signal handler.

Parameters

<table>
<thead>
<tr>
<th>in,out</th>
<th>nrt_sig</th>
<th>Signal handle</th>
</tr>
</thead>
<tbody>
<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 handler() on each invocation</td>
</tr>
</tbody>
</table>

Returns

0 on success, otherwise:

- EAGAIN is returned if no free signal slot is available.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.15.3.3 void rtdm_nrtsig_pend ( rtdm_nrtsig_t * nrt_sig )

Trigger non-real-time signal.

Parameters

| in,out | nrt_sig | Signal handle |

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never in real-time context, possible in non-real-time environments.
5.16 Utility Services

Functions

- **int rtdm_mmap_to_user**(rtdm_user_info_t *user_info, void *src_addr, size_t len, int prot, void **pptr, struct vm_operations_struct *vm_ops, void *vm_private_data)
  
  Map a kernel memory range into the address space of the user.

- **int rtdm_iomap_to_user**(rtdm_user_info_t *user_info, phys_addr_t src_addr, size_t len, int prot, void **pptr, struct vm_operations_struct *vm_ops, void *vm_private_data)
  
  Map an I/O memory range into the address space of the user.

- **int rtdm_munmap**(rtdm_user_info_t *user_info, void *ptr, size_t len)
  
  Unmap a user memory range.

- **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 in real-time context.

- **void rtdm_free**(void *ptr)
  
  Release real-time memory block.

- **int rtdm_read_user_ok**(rtdm_user_info_t *user_info, const void __user *ptr, size_t size)
  
  Check if read access to user-space memory block is safe.

- **int rtdm_rw_user_ok**(rtdm_user_info_t *user_info, const void __user *ptr, size_t size)
  
  Check if read/write access to user-space memory block is safe.

- **int rtdm_copy_from_user**(rtdm_user_info_t *user_info, void *dst, const void __user *src, size_t size)
  
  Copy user-space memory block to specified buffer.

- **int rtdm_safe_copy_from_user**(rtdm_user_info_t *user_info, void *dst, const void __user *src, size_t size)
  
  Check if read access to user-space memory block and copy it to specified buffer.

- **int rtdm_copy_to_user**(rtdm_user_info_t *user_info, void __user *dst, const void *src, size_t size)
  
  Copy specified buffer to user-space memory block.

- **int rtdm_safe_copy_to_user**(rtdm_user_info_t *user_info, void __user *dst, const void *src, size_t size)
  
  Check if read/write access to user-space memory block is safe and copy specified buffer to it.

- **int rtdm_strncpy_from_user**(rtdm_user_info_t *user_info, char *dst, const char __user *src, size_t count)
  
  Copy user-space string to specified buffer.

- **int rtdm_in_rt_context**(void)
  
  Test if running in a real-time task.

- **int rtdm_rt_capable**(rtdm_user_info_t *user_info)
  
  Test if the caller is capable of running in real-time context.

5.16.1 Detailed Description

5.16.2 Function Documentation

5.16.2.1 **int rtdm_copy_from_user**( rtdm_user_info_t *user_info, void *dst, const void __user *src, size_t size )

Copy user-space memory block to specified buffer.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>user_info</th>
<th>User information pointer 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.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.16.2.2 int rtdm_copy_to_user ( rtdm_user_info_t * user_info, void __user * dst, const void * src, size_t size )

Copy specified buffer to user-space memory block.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>user_info</th>
<th>User information pointer 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.

Environments:

This service can be called from:
5.16.2.3  void rtdm_free ( void *ptr )

Release real-time memory block.

Parameters

| in | ptr | Pointer to memory block as returned by rtdm_malloc() |

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine (consider the overhead!)
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.16.2.4  int rtdm_in_rt_context ( void )

Test if running in a real-time task.

Returns

Non-zero is returned if the caller resides in real-time context, 0 otherwise.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.16.2.5  int rtdm_iomap_to_user ( rtdm_user_info_t *user_info, phys_addr_t src_addr, size_t len, int prot, void **pptr, struct vm_operations_struct *vm_ops, void *vm_private_data )

Map an I/O memory range into the address space of the user.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>user_info</th>
<th>User information pointer as passed to the invoked device operation handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>src_addr</td>
<td>physical I/O 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</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 vma_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 vma_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

RTDM supports two models for unmapping the user memory range again. One is explicit unmapping via rtdm_munmap(), either performed when the user requests it via an IOCTL etc. or when the related device is closed. The other is automatic unmapping, triggered by the user invoking standard munmap() or by the termination of the related process. To track release of the mapping and therefore relinquishment of the referenced physical memory, the caller of rtdm_iomap_to_user() can pass a vm_operations_struct on invocation, defining a close handler for the vm_area. See Linux documentation (e.g. Linux Device Drivers book) on virtual memory management for details.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- User-space task (non-RT)

Rescheduling: possible.

5.16.2.6 void* rtdm_malloc ( size_t size )

Allocate memory block in real-time context.

Parameters
5.16 Utility Services

5.16.2.7 int rtdm_mmap_to_user ( rtdm_user_info_t *user_info, void *src_addr, size_t len, int prot, void **pptr, struct vm_operations_struct *vm_ops, void *vm_private_data )

Map a kernel memory range into the address space of the user.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>user_info</td>
<td>User information pointer 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</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 vma_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 vma_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.
This service only works on memory regions allocated via kmalloc() or vmalloc(). To map physical I/O memory to user-space use rtdm_iomap_to_user() instead.

RTDM supports two models for unmapping the user memory range again. One is explicit unmapping via rtdm_munmap(), either performed when the user requests it via an IOCTL etc. or when the related device is closed. The other is automatic unmapping, triggered by the user invoking standard munmap() or by the termination of the related process. To track release of the mapping and therefore relinquishment of the referenced physical memory, the caller of rtdm_mmap_to_user() can pass a vm_operations_struct on invocation, defining a close handler for the vm_area. See Linux documentation (e.g. Linux Device Drivers book) on virtual memory management for details.

Environments:
This service can be called from:

- Kernel module initialization/cleanup code
- User-space task (non-RT)

Rescheduling: possible.

5.16.2.8 int rtdm_munmap ( rtdm_user_info_t *user_info, void *ptr, size_t len )

Unmap a user memory range.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>user_info</th>
<th>User information pointer as passed to rtdm_mmap_to_user() when requesting to map the memory range</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>ptr</td>
<td>User address or the memory range</td>
</tr>
<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.

Environments:
This service can be called from:

- Kernel module initialization/cleanup code
- User-space task (non-RT)

Rescheduling: possible.

5.16.2.9 void rtdm_printk ( const char *format, ... )

Real-time safe message printing on kernel console.
5.16 Utility Services

### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>format</th>
<th>Format string (conforming standard <code>printf()</code>)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Arguments referred by <code>format</code></td>
</tr>
</tbody>
</table>

### Returns

On success, this service returns the number of characters printed. Otherwise, a negative error code is returned.

### Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine (consider the overhead!)
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never in real-time context, possible in non-real-time environments.

5.16.2.10 `void rtdm_printk_ratelimited ( const char * format, ... )`

Real-time safe rate-limited message printing on kernel console.

### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>format</th>
<th>Format string (conforming standard <code>printf()</code>)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Arguments referred by <code>format</code></td>
</tr>
</tbody>
</table>

### Returns

On success, this service returns the number of characters printed. Otherwise, a negative error code is returned.

### Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Interrupt service routine (consider the overhead!)
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never in real-time context, possible in non-real-time environments.

5.16.2.11 `int rtdm_ratelimit ( struct rtdm_ratelimit_state * rs, const char * func )`

Enforces a rate limit.

This function enforces a rate limit: not more than `->burst` callbacks in every `->interval`. 
Parameters

<table>
<thead>
<tr>
<th>in, out</th>
<th>rtdm_ratelimit_state</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>name</td>
<td>of calling function</td>
</tr>
</tbody>
</table>

Returns

0 means callback will be suppressed and 1 means go ahead and do it.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: possible.

References rtdm_clock_read(), rtdm_lock_get_irqsave, and rtdm_lock_put_irqrestore.

5.16.2.12 int rtdm_read_user_ok ( rtdm_user_info_t *user_info, const void __user *ptr, size_t size )

Check if read access to user-space memory block is safe.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>user_info</th>
<th>User information pointer 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.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.16.2.13 int rtdm_rt_capable ( rtdm_user_info_t *user_info )

Test if the caller is capable of running in real-time context.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>user_info</th>
<th>User information pointer as passed to the invoked device operation handler</th>
</tr>
</thead>
</table>

Returns

Non-zero is returned if the caller is able to execute in real-time context (independent of its current execution mode), 0 otherwise.

Note

This function can be used by drivers that provide different implementations for the same service depending on the execution mode of the caller. If a caller requests such a service in non-real-time context but is capable of running in real-time as well, it might be appropriate for the driver to reject the request via -ENOSYS so that RTDM can switch the caller and restart the request in real-time context.

Environments:

This service can be called from:

- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.16.2.14 int rtdm_rw_user_ok ( rtdm_user_info_t * user_info, const void __user * ptr, size_t size )

Check if read/write access to user-space memory block is safe.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>user_info</th>
<th>User information pointer 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.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.16.2.15 int rtdm_safe_copy_from_user ( rtdm_user_info_t * user_info, void * dst, const void __user * src, size_t size )

Check if read access to user-space memory block and copy it to specified buffer.

---

Generated on Thu Jul 7 2016 13:25:19 for Xenomai RTDM skin API by Doxygen
### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th><strong>user_info</strong></th>
<th>User information pointer as passed to the invoked device operation handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><strong>dst</strong></td>
<td>Destination buffer address</td>
</tr>
<tr>
<td>in</td>
<td><strong>src</strong></td>
<td>Address of the user-space memory block</td>
</tr>
<tr>
<td>in</td>
<td><strong>size</strong></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.

### Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

### Rescheduling: never.

#### 5.16.2.16 int rtdm_safe_copy_to_user ( rtdm_user_info_t * user_info, void __user * dst, const void * src, size_t size )

Check if read/write access to user-space memory block is safe and copy specified buffer to it.

### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th><strong>user_info</strong></th>
<th>User information pointer as passed to the invoked device operation handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><strong>dst</strong></td>
<td>Address of the user-space memory block</td>
</tr>
<tr>
<td>in</td>
<td><strong>src</strong></td>
<td>Source buffer address</td>
</tr>
<tr>
<td>in</td>
<td><strong>size</strong></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_rwlock_user_ok and rtdm_copy_to_user.

### Environments:

This service can be called from:

- Kernel module initialization/cleanup code
5.16 Utility Services

- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.16.2.17 int rtdm_strncpy_from_user ( rtdm_user_info_t *user_info, char *dst, const char __user *src, size_t count )

Copy user-space string to specified buffer.

Parameters

| in | user_info | User information pointer as passed to the invoked device operation handler |
| in | dst       | Destination buffer address                                               |
| in | src       | Address of the user-space string                                          |
| in | count     | Maximum number of bytes to copy, including the trailing '0'               |

Returns

Length of the string on success (not including the trailing '0'), otherwise:

-EFAULT is returned if an invalid memory area was accessed.

Note

This service already includes a check of the source address, calling rtdm_read_user_ok() for src explicitly is not required.

Environments:

This service can be called from:

- Kernel module initialization/cleanup code
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.
5.17 Device Profiles

Modules

- CAN Devices
- Real-time IPC protocols

*Profile Revision:* 1

- Serial Devices
- Testing Devices

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_EXPERIMENTAL 224`
- `#define RTDM_CLASS_MAX 255`

Device Naming

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

- `#define RTDM_MAX_DEVNAME_LEN 31`

RTDM_PURGE_xxx BUFFER

Flags selecting buffers to be purged

- `#define RTDM_PURGE_RX_BUFFER 0x0001`
- `#define RTDM_PURGE_TX_BUFFER 0x0002`
Common IOCTLs

The following IOCTLs are common to all device profiles.

- `#define RTIOC_DEVICE_INFO _IOR(RTIOC_TYPE_COMMON, 0x00, struct rtdm_device_info)`
  *Retrieves information about a device or socket.*

- `#define RTIOC_PURGE _IOW(RTIOC_TYPE_COMMON, 0x10, int)`
  *Purges internal device or socket buffers.*

### 5.17.1 Detailed Description

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

### 5.17.2 Macro Definition Documentation

#### 5.17.2.1 `#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></td>
<td>Pointer to information buffer (struct rtdm_device_info)</td>
</tr>
</tbody>
</table>

#### 5.17.2.2 `#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></td>
<td>Purge mask, see RTDM_PURGE_xxx_BUFFER</td>
</tr>
</tbody>
</table>
Chapter 6

Data Structure Documentation

6.1 can_bittime Struct Reference

Custom CAN bit-time definition.
#include <rtcan.h>

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.

6.1.1 Detailed Description

Custom CAN bit-time definition.

Examples:
  
  rtcanconfig.c.

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

- include/rtdm/rtcan.h

6.2 can_bittime_btr Struct Reference

Hardware-specific BTR bit-times.
#include <rtcan.h>

Data Fields

- **uint8_t btr0**
Bus timing register 0.
- `uint8_t btr1`

Bus timing register 1.

6.2.1 Detailed Description

Hardware-specific BTR bit-times.

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

- `include/rtdm/rtcan.h`

6.3 `can_bittime_std` Struct Reference

Standard bit-time parameters according to Bosch.

```c
#include <rtcan.h>
```

**Data Fields**

- `uint32_t brp`
  
  *Baud rate prescaler.*

- `uint8_t prop_seg`
  
  *from 1 to 8*

- `uint8_t phase_seg1`
  
  *from 1 to 8*

- `uint8_t phase_seg2`
  
  *from 1 to 8*

- `uint8_t sjw:7`
  
  *from 1 to 4*

- `uint8_t sam:1`
  
  *1 - enable triple sampling*

6.3.1 Detailed Description

Standard bit-time parameters according to Bosch.

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

- `include/rtdm/rtcan.h`

6.4 `can_filter` Struct Reference

Filter for reception of CAN messages.

```c
#include <rtcan.h>
```
Data Fields

- `uint32_t can_id`
  CAN ID which must match with incoming IDs after passing the mask.
- `uint32_t can_mask`
  Mask which is applied to incoming IDs.

### 6.4.1 Detailed Description

Filter for reception of CAN messages.

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

```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:

- `rtcan_rtt.c`, and `rtcanrecv.c`.

### 6.4.2 Field Documentation

#### 6.4.2.1 `uint32_t can_filter::can_id`

CAN ID which must match with incoming IDs after passing the mask.

The filter logic can be inverted with the flag `CAN_INV_FILTER`.

Examples:

- `rtcanrecv.c`.

#### 6.4.2.2 `uint32_t can_filter::can_mask`

Mask which is applied to incoming IDs.

See `CAN ID masks` if exactly one CAN ID should come through.

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

- `include/rtdm/rtcan.h`

### 6.5 `can_frame` Struct Reference

Raw CAN frame.

```c
#include <rtcan.h>
```
Public Member Functions

- uint8_t data[8] __attribute__((aligned(8)))

  Payload data bytes.

Data Fields

- can_id_t can_id

  CAN ID of the frame.

- uint8_t can_dlc

  Size of the payload in bytes.

6.5.1 Detailed Description

Raw CAN frame.

Central structure for receiving and sending CAN frames.

Examples:

  rtcan_rtt.c, rtcanrecv.c, and rtcansend.c.

6.5.2 Field Documentation

6.5.2.1 can_id_t can_frame::can_id

CAN ID of the frame.

See CAN ID flags for special bits.

Examples:

  rtcan_rtt.c.

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

- include/rtdm/rtcan.h

6.6 rtdm_dev_context Struct Reference

Device context.

#include <rtmd_driver.h>

Data Fields

- unsigned long context_flags

  Context flags, see Context Flags for details.

- int fd

  Associated file descriptor.

- atomic_t close_lock_count

  Lock counter of context, held while structure is referenced by an operation handler.
• struct rtdm_operations * ops
  Set of active device operation handlers.
• struct rtdm_device * device
  Reference to owning device.
• struct rtdm_devctx_reserved reserved
  Data stored by RTDM inside a device context (internal use only)
• char dev_private [0]
  Begin of driver defined context data structure.

6.6.1 Detailed Description

Device context.

A device context structure is associated with every open device instance. RTDM takes care of its creation and destruction and passes it to the operation handlers when being invoked.

Drivers can attach arbitrary data immediately after the official structure. The size of this data is provided via rtdm_device.context_size during device registration.

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

• include/rtdm/rtdm_driver.h

6.7 rtdm_device Struct Reference

RTDM device.

#include <rtdm_driver.h>

Data Fields

• int struct_version
  Revision number of this structure, see Driver Versioning defines.
• int device_flags
  Device flags, see Device Flags for details.
• size_t context_size
  Size of driver defined appendix to struct rtdm_dev_context.
• char device_name [RTDM_MAX_DEVNAME_LEN+1]
  Named device identification (orthogonal to Linux device name space)
• int protocol_family
  Protocol device identification: protocol family (PF_xxx)
• int socket_type
  Protocol device identification: socket type (SOCK_xxx)
• rtdm_open_handler_t open_rt
  Named device instance creation for real-time contexts, optional (but deprecated) if open_nrt is non-NULL, ignored for protocol devices.
• rtdm_open_handler_t open_nrt
  Named device instance creation for non-real-time contexts, optional if open_rt is non-NULL, ignored for protocol devices.
• rtdm_socket_handler_t socket_rt
  Protocol socket creation for real-time contexts, optional (but deprecated) if socket_nrt is non-NULL, ignored for named devices.
- **rtdm_socket_handler_t** socket_nrt
  
  Protocol socket creation for non-real-time contexts, optional if socket_rt is non-NULL, ignored for named devices.

- **struct rtdm_operations** ops
  
  Default operations on newly opened device instance.

- **int** device_class
  
  Device class ID, see **RTDM_CLASS_xxx**.

- **int** device_sub_class
  
  Device sub-class, see **RTDM_SUBCLASS_xxx** definition in the **Device Profiles**.

- **int** profile_version
  
  Supported device profile version.

- **const char ∗** driver_name
  
  Informational driver name (reported via /proc)

- **int** driver_version
  
  Driver version, see **Driver Versioning** defines.

- **const char ∗** peripheral_name
  
  Informational peripheral name the device is attached to (reported via /proc)

- **const char ∗** provider_name
  
  Informational driver provider name (reported via /proc)

- **const char ∗** proc_name
  
  Name of /proc entry for the device, must not be NULL.

- **int** device_id
  
  Driver definable device ID.

- **void ∗** device_data
  
  Driver definable device data.

- **struct rtdm_dev_reserved** reserved
  
  Data stored by RTDM inside a registered device (internal use only)

### 6.7.1 Detailed Description

RTDM device.

This structure specifies a RTDM device. As some fields, especially the reserved area, will be modified by RTDM during runtime, the structure must not reside in write-protected memory.

### 6.7.2 Field Documentation

#### 6.7.2.1 **rtdm_open_handler_t** rtdm_device::open Rt

Named device instance creation for real-time contexts, optional (but deprecated) if open_nrt is non-NULL, ignored for protocol devices.

**Deprecated** Only use non-real-time open handler in new drivers.

Referenced by rtdm_dev_register().
6.7.2.2  rtdm_socket_handler_t rtdm_device::socket_{rt}

Protocol socket creation for real-time contexts, optional (but deprecated) if socket_nrt is non-NULL, ignored for named devices.

**Deprecated** Only use non-real-time socket creation handler in new drivers.

Referenced by rtdm_dev_register().

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

- include/rtdm/rtdm_driver.h

6.8  rtdm_device_info Struct Reference

Device information.

```c
#include <rtdm.h>
```

**Data Fields**

- `int device_flags`
  
  Device flags, see *Device Flags* for details.

- `int device_class`
  
  Device class ID, see *RTDM_CLASS_xxx*.

- `int device_sub_class`
  
  Device sub-class, either *RTDM_SUBCLASS_GENERIC* or a *RTDM_SUBCLASS_xxx* definition of the related *Device Profile*.

- `int profile_version`
  
  Supported device profile version.

6.8.1  Detailed Description

Device information.

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

- include/rtdm/rtdm.h

6.9  rtdm_operations Struct Reference

Device operations.

```c
#include <rtdm_driver.h>
```

**Data Fields**

**Common Operations**

- `rtdm_close_handler_t close_{rt}
  
  Close handler for real-time contexts (optional, deprecated)

- `rtdm_close_handler_t close_nrt`
Close handler for non-real-time contexts (required)
- rtdm_ioctl_handler_t ioctl_nrt
  IOCTL from non-real-time context (optional)
- rtdm_ioctl_handler_t ioctl_rtnrt
  IOCTL from real-time context (optional)
- rtdm_select_bind_handler_t select_bind
  Select binding handler for any context (optional)

Stream-Oriented Device Operations
- rtdm_read_handler_t read_rt
  Read handler for real-time context (optional)
- rtdm_read_handler_t read_nrt
  Read handler for non-real-time context (optional)
- rtdm_write_handler_t write_rt
  Write handler for real-time context (optional)
- rtdm_write_handler_t write_nrt
  Write handler for non-real-time context (optional)

Message-Oriented Device Operations
- rtdm_recvmsg_handler_t recvmsg_rt
  Receive message handler for real-time context (optional)
- rtdm_recvmsg_handler_t recvmsg_nrt
  Receive message handler for non-real-time context (optional)
- rtdm_sendmsg_handler_t sendmsg_rt
  Transmit message handler for real-time context (optional)
- rtdm_sendmsg_handler_t sendmsg_nrt
  Transmit message handler for non-real-time context (optional)

6.9.1 Detailed Description
Device operations.

6.9.2 Field Documentation

6.9.2.1 rtdm_close_handler_t rtdm_operations::close_rt
Close handler for real-time contexts (optional, deprecated)

Deprecated Only use non-real-time close handler in new drivers.

Referenced by rtdm_dev_register().
The documentation for this struct was generated from the following file:
- include/rtdm/rtdm_driver.h

6.10 rtipc_port_label Struct Reference
Port label information structure.
#include <rtipc.h>
Data Fields

- char `label` [XOBJECT_NAME_LEN]
  Port label string, null-terminated.

6.10.1 Detailed Description

Port label information structure.

Examples:

  `bufp-label.c`, `iddp-label.c`, and `xxdp-label.c`.

6.10.2 Field Documentation

6.10.2.1 char `rtipc_port_label::label` [XOBJECT_NAME_LEN]

Port label string, null-terminated.

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

- include/rtdm/rtipc.h

6.11 `rtser_config` Struct Reference

Serial device configuration.

```c
#include <rtserial.h>
```

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
6.11.1 Detailed Description

Serial device configuration.

Examples:

    cross-link.c.

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

- include/rtdm/rtserial.h

6.12 rtser_event Struct Reference

Additional information about serial device events.

#include <rtserial.h>

Data Fields

- int events
  signalled events, see RTSER_EVENT_xxx
- int rx_pending
  number of pending input characters
- nanosecs_abs_t last_timestamp
  last interrupt timestamp
- nanosecs_abs_t rxpend_timestamp
  reception timestamp of oldest character in input queue

6.12.1 Detailed Description

Additional information about serial device events.

Examples:

    cross-link.c.

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

- include/rtdm/rtserial.h

6.13 rtser_status Struct Reference

Serial device status.

#include <rtserial.h>
Data Fields

- int line_status
  line status register, see RTSER_LSR_XXX
- int modem_status
  modem status register, see RTSER_MSR_XXX

6.13.1 Detailed Description

Serial device status.

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

- include/rtdm/rtserial.h

6.14 sockaddr_can Struct Reference

Socket address structure for the CAN address family.

#include <rtcan.h>

Data Fields

- sa_family_t can_family
  CAN address family, must be AF_CAN.
- int can_ifindex
  Interface index of CAN controller.

6.14.1 Detailed Description

Socket address structure for the CAN address family.

Examples:

- rtcan_rtt.c, rtcanrecv.c, and rtcansend.c.

6.14.2 Field Documentation

6.14.2.1 int sockaddr_can::can_ifindex

Interface index of CAN controller.

See SIOCGIFINDEX.

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

- include/rtdm/rtcan.h

6.15 sockaddr_ipc Struct Reference

Socket address structure for the RTIPC address family.

#include <rtipc.h>
Data Fields

- `sa_family_t sipc_family`
  RTIPC address family, must be AF_RTIPC.
- `rtipc_port_t sipc_port`
  Port number.

6.15.1 Detailed Description

Socket address structure for the RTIPC address family.

Examples:

  bufp-label.c, bufp-readwrite.c, iddp-label.c, iddp-sendrecv.c, xddp-echo.c, xddp-label.c, and xddp-stream.c.

6.15.2 Field Documentation

6.15.2.1 `rtipc_port_t sockaddr_ipc::sipc_port`

Port number.

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

- include/rtdm/rtipc.h
Chapter 7

File Documentation

7.1 include/rtdm/rtcan.h File Reference

Real-Time Driver Model for RT-Socket-CAN, CAN device profile header.

```c
#include <linux/net.h>
#include <linux/socket.h>
#include <linux/if.h>
#include <rtdm/rtdm.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.

Macros

- `#define AF_CAN 29`
  
  CAN address family.
- `#define PF_CAN AF_CAN`
  
  CAN protocol family.
- `#define SOL_CAN_RAW 103`
  
  CAN socket levels.

CAN ID masks

*Bit masks for masking CAN IDs*
• #define CAN_EFF_MASK 0x1FFFFFFF
  Bit mask for extended CAN IDs.
• #define CAN_SFF_MASK 0x000007FF
  Bit mask for standard CAN IDs.

CAN ID flags
Flags within a CAN ID indicating special CAN frame attributes

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

Particular CAN protocols
Possible protocols for the PF_CAN protocol family
Currently only the RAW protocol is supported.

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

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

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

• #define CAN_CTRLMODE_LISTENONLY 0x1
• #define CAN_CTRLMODE_LOOPBACK 0x2
• #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

- `#define SIOCGIFINDEX defined_by_kernel_header_file`
  Get CAN interface index by name.
- `#define SIOCSBAUDRATE _IOW(RTIOC_TYPE_CAN, 0x01, struct ifreq)`
  Set baud rate.
- `#define SIOCGBTBAUDRATE _IOWR(RTIOC_TYPE_CAN, 0x02, struct ifreq)`
  Get baud rate.
- `#define SIOCSANCUSTOMBITTIME _IOW(RTIOC_TYPE_CAN, 0x03, struct ifreq)`
  Set custom bit time parameter.
- `#define SIOCGANCUSTOMBITTIME _IOWR(RTIOC_TYPE_CAN, 0x04, struct ifreq)`
  Get custom bit-time parameters.
- `#define SIOCSCANMODE _IOW(RTIOC_TYPE_CAN, 0x05, struct ifreq)`
  Set operation mode of CAN controller.
- `#define SIOCGCANSTATE _IOWR(RTIOC_TYPE_CAN, 0x06, struct ifreq)`
  Get current state of CAN controller.
- `#define SIOCSANCCTRLMODE _IOW(RTIOC_TYPE_CAN, 0x07, struct ifreq)`
  Set special controller modes.
- `#define SIOCGANCCTRLMODE _IOWR(RTIOC_TYPE_CAN, 0x08, struct ifreq)`
  Get special controller modes.
- `#define RTCAN_RTIOC_TAKE_TIMESTAMP _IOW(RTIOC_TYPE_CAN, 0x09, int)`
  Enable or disable storing a high precision timestamp upon reception of a CAN frame.
- `#define RTCAN_RTIOC_RCV_TIMEOUT _IOW(RTIOC_TYPE_CAN, 0x0A, nanosecs_rel_t)`
  Specify a reception timeout for a socket.
- `#define RTCAN_RTIOC_SND_TIMEOUT _IOW(RTIOC_TYPE_CAN, 0x0B, nanosecs_rel_t)`
  Specify a transmission timeout for a socket.

Error mask

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

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

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

**Note:** Bus error interrupts (`CAN_ERR_BUSERROR`) are enabled when an application is calling a `Recv` function on a socket listening on bus errors (using `CAN_RAW_ERR_FILTER`). After one bus error has 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
#define CAN_ERR_PROT_LOC_SOF 0x03
  start of frame
#define CAN_ERR_PROT_LOC_ID28_21 0x02
  ID bits 28 - 21 (SFF: 10 - 3)
#define CAN_ERR_PROT_LOC_ID20_18 0x06
  ID bits 20 - 18 (SFF: 2 - 0)
#define CAN_ERR_PROT_LOC_SRTR 0x04
  substitute RTR (SFF: RTR)
#define CAN_ERR_PROT_LOC_IDE 0x05
  identifier extension
#define CAN_ERR_PROT_LOC_ID17_13 0x07
  ID bits 17-13.
#define CAN_ERR_PROT_LOC_ID12_05 0x0F
  ID bits 12-5.
#define CAN_ERR_PROT_LOC_ID04_00 0x0E
  ID bits 4-0.
#define CAN_ERR_PROT_LOC_RTR 0x0C
  RTR.
#define CAN_ERR_PROT_LOC_RES1 0x0D
  reserved bit 1
#define CAN_ERR_PROT_LOC_RES0 0x09
  reserved bit 0
#define CAN_ERR_PROT_LOC_DLC 0x0B
  data length code
#define CAN_ERR_PROT_LOC_DATA 0x0A
  data section
#define CAN_ERR_PROT_LOC_CRC_SEQ 0x08
  CRC sequence.
#define CAN_ERR_PROT_LOC_CRC_DEL 0x18
  CRC delimiter.
#define CAN_ERR_PROT_LOC_ACK 0x19
  ACK slot.
#define CAN_ERR_PROT_LOC_ACK_DEL 0x1B
  ACK delimiter.
#define CAN_ERR_PROT_LOC_EOF 0x1A
  end of frame
#define CAN_ERR_PROT_LOC_INTERM 0x12
  intermission
#define CAN_ERR_TRX_UNSPEC 0x00
  0000 0000
#define CAN_ERR_TRX_CANH_NO_WIRE 0x04
  0000 0100
#define CAN_ERR_TRX_CANH_SHORT_TO_BAT 0x05
  0000 0101
#define CAN_ERR_TRX_CANH_SHORT_TO_VCC 0x06
  0000 0110
#define CAN_ERR_TRX_CANH_SHORT_TO_GND 0x07
  0000 0111
#define CAN_ERR_TRX_CANL_NO_WIRE 0x40
  0100 0000
#define CAN_ERR_TRX_CANL_SHORT_TO_BAT 0x50
  0101 0000
#define CAN_ERR_TRX_CANL_SHORT_TO_VCC 0x60
  0110 0000
#define CAN_ERR_TRX_CANL_SHORT_TO_GND 0x70
  0111 0000
#define CAN_ERR_TRX_CANL_SHORT_TO_CANH 0x80
  1000 0000
Typedefs

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

Enumerations

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

CAN operation modes

Modes into which CAN controllers can be set

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

CAN controller states

States a CAN controller can be in.

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

7.1.1 Detailed Description

Real-Time Driver Model for RT-Socket-CAN, CAN device profile header.
This RTDM CAN device profile header is based on:
include/linux/can.h, include/linux/socket.h, net/can/pf_can.h in linux-can.patch, a CAN socket framework for Linux
Copyright (C) 2004, 2005, Robert Schwebel, Benedikt Spranger, Marc Kleine-Budde, Pengutronix
This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

7.2 include/rtdm/rtdm.h File Reference

Real-Time Driver Model for Xenomai, user API header.

#include <linux/types.h>
#include <linux/fcntl.h>
#include <linux/ioctl.h>
#include <linux/sched.h>
#include <linux/socket.h>
#include <asm/xenomai/wrappers.h>

Data Structures

- struct rtdm_device_info
  Device information.

Macros

API Versioning

- #define RTDM_API_VER 8
  Common user and driver API version.
- #define RTDM_API_MIN_COMPAT_VER 6
  Minimum API revision compatible with the current release.

RTDM_TIMEOUT_xxx

Special timeout values

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

**RTDM_CLASS_xxx**

Device classes

- `#define RTDM_CLASS_PARPORT 1`
- `#define RTDM_CLASS_SERIAL 2`
- `#define RTDM_CLASS_CAN 3`
- `#define RTDM_CLASS_NETWORK 4`
- `#define RTDM_CLASS_RTMAC 5`
- `#define RTDM_CLASS_TESTING 6`
- `#define RTDM_CLASS_RTPC 7`
- `#define RTDM_CLASS_EXPERIMENTAL 224`
- `#define RTDM_CLASS_MAX 255`

**Device Naming**

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

- `#define RTDM_MAX_DEVNAME_LEN 31`

**RTDM_PURGE_xxx_BUFFER**

Flags selecting buffers to be purged

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

**Common IOCTLs**

The following IOCTLs are common to all device profiles.

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

**Typedefs**

- `typedef uint64_t nanosecs_abs_t`
  RTDM type for representing absolute dates.
- `typedef int64_t nanosecs_rel_t`
  RTDM type for representing relative intervals.
- `typedef struct rtdm_device_info rtdm_device_info_t`
  Device information.

**7.2.1 Detailed Description**

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

Xenomai is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

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

7.3 include/rtdm/rtdm_driver.h File Reference

Real-Time Driver Model for Xenomai, driver API header.

```c
#include <asm/atomic.h>
#include <linux/list.h>
#include <nucleus/xenomai.h>
#include <nucleus/heap.h>
#include <nucleus/pod.h>
#include <nucleus/synch.h>
#include <nucleus/select.h>
#include <nucleus/vfile.h>
#include <rtdm/rtdm.h>
#include <nucleus/assert.h>
```

Data Structures

- `struct rtdm_operations`
  
  Device operations.

- `struct rtdm_dev_context`
  
  Device context.

- `struct rtdm_device`
  
  RTDM device.

Macros

- `#define rtdm_irq_get_arg(irq_handle, type) ((type *)irq_handle->cookie)`
  
  Retrieve IRQ handler argument.

Device Flags

Static flags describing a RTDM device

- `#define RTDM_EXCLUSIVE 0x0001`
  
  If set, only a single instance of the device can be requested by an application.

- `#define RTDM_NAMED_DEVICE 0x0010`
  
  If set, the device is addressed via a clear-text name.

- `#define RTDM_PROTOCOL_DEVICE 0x0020`
  
  If set, the device is addressed via a combination of protocol ID and socket type.
• #define RTDM_DEVICE_TYPE_MASK 0x00F0
  Mask selecting the device type.

Context Flags
Dynamic flags describing the state of an open RTDM device (bit numbers)

• #define RTDM_CREATED_IN_NRT 0
  Set by RTDM if the device instance was created in non-real-time context.
• #define RTDM_CLOSING 1
  Set by RTDM when the device is being closed.
• #define RTDM_USER_CONTEXT_FLAG 8
  /* first user-definable flag */
  Lowest bit number the driver developer can use freely.

Driver Versioning
Current revisions of RTDM structures, encoding of driver versions. See API Versioning for the interface revision.

• #define RTDM_DEVICE_STRUCT_VER 5
  Version of struct rtdm_device.
• #define RTDM_CONTEXT_STRUCT_VER 3
  Version of struct rtdm_dev_context.
• #define RTDM_SECURE_DEVICE 0x80000000
  Flag indicating a secure variant of RTDM (not supported here)
• #define RTDM_DRIVER_VER(major, minor, patch) (((major & 0xFF) << 16) | ((minor & 0xFF) << 8) | (patch & 0xFF))
  Version code constructor for driver revisions.
• #define RTDM_DRIVER_MAJOR_VER(ver) (((ver) >> 16) & 0xFF)
  Get major version number from driver revision code.
• #define RTDM_DRIVER_MINOR_VER(ver) (((ver) >> 8) & 0xFF)
  Get minor version number from driver revision code.
• #define RTDM_DRIVER_PATCH_VER(ver) ((ver) & 0xFF)
  Get patch version number from driver revision code.

Global Lock across Scheduler Invocation

• #define RTDM_EXECUTE_ATOMALLY(code_block)
  Execute code block atomically.

RTDM_IRQTYPE_xxx
Interrupt registrations flags

• #define RTDM_IRQTYPE_SHARED XN_ISR_SHARED
  Enable IRQ-sharing with other real-time drivers.
• #define RTDM_IRQTYPE_EDGE XN_ISR_EDGE
  Mark IRQ as edge-triggered, relevant for correct handling of shared edge-triggered IRQs.

RTDM_IRQ_xxx
Return flags of interrupt handlers

• #define RTDM_IRQ_NONE XN_ISR_NONE
  Unhandled interrupt.
• #define RTDM_IRQ_HANDLED XN_ISR_HANDLED
  Denote handled interrupt.

Task Priority Range
Maximum and minimum task priorities
```plaintext
• #define RTDM_TASK_LOWEST_PRIORITY XNSCHED_LOW_PRIO
• #define RTDM_TASK_HIGHEST_PRIORITY XNSCHED_HIGH_PRIO

Task Priority Modification
Raise or lower task priorities by one level
• #define RTDM_TASK_RAISE_PRIORITY (+1)
• #define RTDM_TASK_LOWER_PRIORITY (-1)

Typedefs
• typedef int(* rtdm_irq_handler_t)(rtdm_irq_t*irq_handle)
  Interrupt handler.
• typedef void(* rtdm_nrtsig_handler_t)(rtdm_nrtsig_t nrtsig, void*arg)
  Non-real-time signal handler.
• typedef void(* rtdm_timer_handler_t)(rtdm_timer_t*timer)
  Timer handler.
• typedef void(* rtdm_task_proc_t)(void*arg)
  Real-time task procedure.

Operation Handler Prototypes
• typedef int(* rtdm_open_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info, int oflag)
  Named device open handler.
• typedef int(* rtdm_socket_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info, int protocol)
  Socket creation handler for protocol devices.
• typedef int(* rtdm_close_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info)
  Close handler.
• typedef int(* rtdm_ioctl_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info, unsigned int request, void __user *arg)
  IOCTL handler.
• typedef ssize_t(* rtdm_select_bind_handler_t)(struct rtdm_dev_context *context, rtdm_selector_t *selector, enum rtdm_selecttype type, unsigned fd_index)
  Select binding handler.
• typedef ssize_t(* rtdm_read_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info, void *buf, size_t nbyte)
  Read handler.
• typedef ssize_t(* rtdm_write_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info, const void *buf, size_t nbyte)
  Write handler.
• typedef ssize_t(* rtdm_recvmsg_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info, struct user_msghdr *msg, int flags)
  Receive message handler.
• typedef ssize_t(* rtdm_sendmsg_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info, const struct user_msghdr *msg, int flags)
  Transmit message handler.

Enumerations

RTDM_SELECTTYPE_xxx
Event types select can bind to
```
enum rtdm_selecttype { RTDM_SELECTTYPE_READ = XNSELECT_READ, RTDM_SELECTTYPE_WRITE = XNSELECT_WRITE, RTDM_SELECTTYPE_EXCEPT = XNSELECT_EXCEPT }

RTDM_TIMERMODE_\_\_xxx

Timer operation modes

enum rtdm_timer_mode { RTDM_TIMERMODE_RELATIVE = XN_RELATIVE, RTDM_TIMERMODE_ABSOLUTE = XN_ABSOLUTE, RTDM_TIMERMODE_REALTIME = XN_REALTIME }

Functions

- static void * rtdm_context_to_private (struct rtdm_dev_context *context)
  Locate the driver private area associated to a device context structure.
- static struct rtdm_dev_context * rtdm_private_to_context (void *dev_private)
  Locate a device context structure from its driver private area.
- int rtdm_dev_register (struct rtdm_device *device)
  Register a RTDM device.
- int rtdm_dev_unregister (struct rtdm_device *device, unsigned int poll_delay)
  Unregisters a RTDM device.
- struct rtdm_dev_context * rtdm_context_get (int fd)
  Retrieve and lock a device context.
- int rtdm_select_bind (int fd, rtdm_selector_t *selector, enum rtdm_selecttype type, unsigned fd_index)
  Bind a selector to specified event types of a given file descriptor.
- int rtdm_irq_request (rtdm_irq_t *irq_handle, unsigned int irq_no, rtdm_irq_handler_t handler, unsigned long flags, const char *device_name, void *arg)
  Register an interrupt handler.
- void rtdm_timer_destroy (rtdm_timer_t *timer)
  Destroy a timer.
- int rtdm_timer_start (rtdm_timer_t *timer, nanosecs_abs_t expiry, nanosecs_rel_t interval, enum rtdm_timer_mode mode)
  Start a timer.
- void rtdm_timer_stop (rtdm_timer_t *timer)
  Stop a timer.
- int rtdm_task_init (rtdm_task_t *task, const char *name, rtdm_task_proc_t task_proc, void *arg, int priority, nanosecs_rel_t period)
  Initialise and start a real-time task.
- void rtdm_task_busy_sleep (nanosecs_rel_t delay)
  Busy-wait a specified amount of time.
- void rtdm_toseq_init (rtdm_toseq_t *timeout_seq, nanosecs_rel_t timeout)
  Initialise a timeout sequence.
- void rtdm_event_init (rtdm_event_t *event, unsigned long pending)
  Initialise an event.
- int rtdm_event_select_bind (rtdm_event_t *event, rtdm_selector_t *selector, enum rtdm_selecttype type, unsigned fd_index)
  Bind a selector to an event.
- int rtdm_event_wait (rtdm_event_t *event)
  Wait on event occurrence.
- int rtdm_event_timedwait (rtdm_event_t *event, nanosecs_rel_t timeout, rtdm_toseq_t *timeout_seq)
  Wait on event occurrence with timeout.
- void rtdm_event_signal (rtdm_event_t *event)
Signal an event occurrence.

- void rtdm_event_clear (rtdm_event_t *event)
  Clear event state.

- void rtdm_sem_init (rtdm_sem_t *sem, unsigned long value)
  Initialise a semaphore.

- int rtdm_sem_select_bind (rtdm_sem_t *sem, rtdm_selector_t *selector, enum rtdm_selecttype type, unsigned fd_index)
  Bind a selector to a semaphore.

- int rtdm_sem_down (rtdm_sem_t *sem)
  Decrement a semaphore.

- int rtdm_sem_timeddown (rtdm_sem_t *sem, nanosecs_rel_t timeout, rtdm_toseq_t *timeout_seq)
  Decrement a semaphore with timeout.

- void rtdm_sem_up (rtdm_sem_t *sem)
  Increment a semaphore.

- void rtdm_mutex_init (rtdm_mutex_t *mutex)
  Initialise a mutex.

- int rtdm_mutex_lock (rtdm_mutex_t *mutex)
  Request a mutex.

- int rtdm_mutex_timedlock (rtdm_mutex_t *mutex, nanosecs_rel_t timeout, rtdm_toseq_t *timeout_seq)
  Request a mutex with timeout.

- int rtdm_ratelimit (struct rtdm_ratelimit_state *rs, const char *func)
  Enforces a rate limit.

Spinlock with Preemption Deactivation

- #define RTDM_LOCK_UNLOCKED RTHAL_SPIN_LOCK_UNLOCKED
  Static lock initialisation.

- #define rtdm_lock_init(lock) rthal_spin_lock_init(lock)
  Dynamic lock initialisation.

- #define rtdm_lock_get(lock) rthal_spin_lock(lock)
  Acquire lock from non-preemptible contexts.

- #define rtdm_lock_put(lock)
  Release lock without preemption restoration.

- #define rtdm_lock_get_irqsave(lock, context)
  Acquire lock and disable preemption.

- #define rtdm_lock_put_irqrestore(lock, context)
  Release lock and restore preemption state.

- #define rtdm_lock_irqsave(context) rthal_local_irq_save(context)
  Disable preemption locally.

- #define rtdm_lock_irqrestore(context) rthal_local_irq_restore(context)
  Restore preemption state.

typedef rthal_spinlock_t rtdm_lock_t
  Lock variable.

typedef unsigned long rtdm_lockctx_t
  Variable to save the context while holding a lock.
7.3.1 Detailed Description

Real-Time Driver Model for Xenomai, driver API header.

Note

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

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

Xenomai is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

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

7.4 include/rtdm/rtipc.h File Reference

This file is part of the Xenomai project.

#include <linux/net.h>
#include <linux/socket.h>
#include <linux/if.h>
#include <nucleus/types.h>
#include <rtdm/rtdm.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 IPCPROTO_BUFP protocols support the standard SO_SNDTIMEO socket option, from the SOL_SOCKET level.
• #define SO_RCVTIMEO defined_by_kernel_header_file
  All RTIPC protocols support the standard SO_RCVTIMEO socket option, from the SOL_SOCKET level.

Typedefs
• typedef int16_t rtipc_port_t
  Port number type for the RTIPC address family.

Enumerations

RTIPC protocol list
  protocols for the PF_RTIPC protocol family
• enum { IPCPROTO_IPC = 0, IPCPROTO_XDDP = 1, IPCPROTO_IDDP = 2, IPCPROTO_BUFP = 3 }

Functions

Supported operations
  Standard socket operations supported by the RTIPC protocols.
• int socket_AF_RTIPC (int domain=AF_RTIPC, int type=SOCK_DGRAM, int protocol)
  Create an endpoint for communication in the AF_RTIPC domain.
• int close__AF_RTIPC (int sockfd)
  Close a RTIPC socket descriptor.
• int bind__AF_RTIPC (int sockfd, const struct sockaddr_ipc *addr, socklen_t addrlen)
  Bind a RTIPC socket to a port.
• int connect__AF_RTIPC (int sockfd, const struct sockaddr_ipc *addr, socklen_t addrlen)
  Initiate a connection on a RTIPC socket.
• int setsockopt__AF_RTIPC (int sockfd, int level, int optname, const void *optval, socklen_t optlen)
  Set options on RTIPC sockets.
• int getsockopt__AF_RTIPC (int sockfd, int level, int optname, void *optval, socklen_t *optlen)
  Get options on RTIPC sockets.
• ssize_t sendmsg__AF_RTIPC (int sockfd, const struct user_msghdr *msg, int flags)
  Send a message on a RTIPC socket.
• ssize_t recvmsg__AF_RTIPC (int sockfd, struct user_msghdr *msg, int flags)
  Receive a message from a RTIPC socket.
• int getsockname__AF_RTIPC (int sockfd, struct sockaddr_ipc *addr, socklen_t *addrlen)
  Get socket name.
• int getpeername__AF_RTIPC (int sockfd, struct sockaddr_ipc *addr, socklen_t *addrlen)
  Get socket peer.

7.4.1 Detailed Description

This file is part of the Xenomai project.

Note

Copyright (C) 2009 Philippe Gerum rpm@xenomai.org

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

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

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

7.5 include/rtdm/rtserial.h File Reference

Real-Time Driver Model for Xenomai, serial device profile header.

#include <rtdm/rtdm.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 mode control register.

#define RTSER_RTIOC_SET_CONTROL _IOW(RTIOC_TYPE_SERIAL, 0x04, int)
Set serial device's mode 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.

typedef struct rtser_config rtser_config_t
Serial device configuration.

typedef struct rtser_status rtser_status_t
Serial device status.

typedef struct rtser_event rtser_event_t
Additional information about serial device events.
7.5.1 Detailed Description

Real-Time Driver Model for Xenomai, serial device profile header.

Note

Copyright (C) 2005-2007 Jan Kiszka jan.kiszka@web.de

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

Xenomai is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

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

7.6 include/rtdm/rttesting.h File Reference

Real-Time Driver Model for Xenomai, testing device profile header.

#include <rtdm/rtdm.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 _IOWR(RTIOC_TYPE_TESTING, 0x00, struct rttst_interm_bench_res)
- #define RTTST_RTIOC_TMBENCH_START _IOW(RTIOC_TYPE_TESTING, 0x10, struct rttst_tmbench_config)
- #define RTTST_RTIOC_TMBENCH_STOP _IOWR(RTIOC_TYPE_TESTING, 0x11, struct rttst_overall_bench_res)
- #define RTTST_RTIOC_IRQBENCH_START _IOW(RTIOC_TYPE_TESTING, 0x20, struct rttst_irqbench_config)
- #define RTTST_RTIOC_IRQBENCH_STOP _IO(RTIOC_TYPE_TESTING, 0x21)
- #define RTTST_RTIOC_IRQBENCH_GET_STATS _IOR(RTIOC_TYPE_TESTING, 0x22, struct rttst_irqbench_stats)
- #define RTTST_RTIOC_IRQBENCH_WAIT_IRQ _IO(RTIOC_TYPE_TESTING, 0x23)
- #define RTTST_RTIOC_IRQBENCH_REPLY_IRQ _IO(RTIOC_TYPE_TESTING, 0x24)
- #define RTTST_RTIOC_SWTEST_SET_TASKS_COUNT _IOW(RTIOC_TYPE_TESTING, 0x30, unsigned long)
## RTTST RTIOC SWTEST Set CPU

```
define RTTST_RTIOC_SWTEST_SET_CPU   _IOW(RTIOC_TYPE_TESTING, 0x31, unsigned long)
```

## RTTST RTIOC SWTEST Register UTASK

```
define RTTST_RTIOC_SWTEST_REGISTER_UTASK   _IOW(RTIOC_TYPE_TESTING, 0x32, struct rttst_swtest_task)
```

## RTTST RTIOC SWTEST Create KTASK

```
define RTTST_RTIOC_SWTEST_CREATE_KTASK   _IOWR(RTIOC_TYPE_TESTING, 0x33, struct rttst_swtest_task)
```

## RTTST RTIOC SWTEST PEND

```
define RTTST_RTIOC_SWTEST_PEND   _IOR(RTIOC_TYPE_TESTING, 0x34, struct rttst_swtest_task)
```

## RTTST RTIOC SWTEST Switch To

```
define RTTST_RTIOC_SWTEST_SWITCH_TO   _IOR(RTIOC_TYPE_TESTING, 0x35, struct rttst_swtest_dir)
```

## RTTST RTIOC SWTEST Get Switches Count

```
define RTTST_RTIOC_SWTEST_GET_SWITCHES_COUNT   _IOR(RTIOC_TYPE_TESTING, 0x36, unsigned long)
```

## RTTST RTIOC SWTEST Get Last Error

```
define RTTST_RTIOC_SWTEST_GET_LAST_ERROR   _IOR(RTIOC_TYPE_TESTING, 0x37, struct rttst_swtest_error)
```

## RTTST RTIOC SWTEST Set Pause

```
define RTTST_RTIOC_SWTEST_SET_PAUSE   _IOW(RTIOC_TYPE_TESTING, 0x38, unsigned long)
```

## RTTST RTIOC RTDM Defers Close

```
define RTTST_RTIOC_RTDM_DEFER_CLOSE   _IOW(RTIOC_TYPE_TESTING, 0x40, unsigned long)
```

### 7.6.1 Detailed Description

Real-Time Driver Model for Xenomai, testing device profile header.

Note

Copyright (C) 2005 Jan Kiszka jan.kiszka@web.de

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

Xenomai is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

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

### 7.7 ksrc/skins/rtdm/device.c File Reference

Real-Time Driver Model for Xenomai, device management.

```
#include <linux/module.h>
#include <linux/delay.h>
#include "rtdm/internal.h"
```

### Functions

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

- int rtdm_dev_unregister (struct rtdm_device *device, unsigned int poll_delay)
  
  Unregisters a RTDM device.
7.7.1 Detailed Description

Real-Time Driver Model for Xenomai, device management.

Note
Copyright (C) 2005 Jan Kiszka jan.kiszka@web.de
Copyright (C) 2005 Joerg Langenberg joerg.langenberg@gmx.net

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

Xenomai is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even
the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

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

7.8 ksrc/skins/rtdm/drvlib.c File Reference

Real-Time Driver Model for Xenomai, driver library.

#include <linux/bitops.h>
#include <linux/delay.h>
#include <linux/mman.h>
#include <asm/page.h>
#include <asm/io.h>
#include <asm/pgtable.h>
#include <linux/highmem.h>
#include <linux/err.h>
#include <rtdm/rtdm_driver.h>

Functions

- **nanosecs_abs_t rtdm_clock_read (void)***
  
  Get system time.

- **nanosecs_abs_t rtdm_clock_read_monotonic (void)***
  
  Get monotonic time.

- **int rtdm_task_init (rtdm_task_t *task, const char *name, rtdm_task_proc_t task_proc, void *arg, int priority, nanosecs_rel_t period)**
  
  Initialise and start a real-time task.

- **void rtdm_task_destroy (rtdm_task_t *task)***
  
  Destroy a real-time task.

- **void rtdm_task_set_priority (rtdm_task_t *task, int priority)**
  
  Adjust real-time task priority.

- **int rtdm_task_set_period (rtdm_task_t *task, nanosecs_rel_t period)**
  
  Adjust real-time task period.

- **int rtdm_task_wait_period (void)***
  
  Wait on next real-time task period.

- **int rtdm_task_unblock (rtdm_task_t *task)**
  
  Activate a blocked real-time task.
- `rtdm_task_t *rtdm_task_current (void)`
  Get current real-time task.
- `int rtdm_task_sleep (nanosecs_rel_t delay)`
  Sleep a specified amount of time.
- `int rtdm_task_sleep_until (nanosecs_abs_t wakeup_time)`
  Sleep until a specified absolute time.
- `int rtdm_task_sleep_abs (nanosecs_abs_t wakeup_time, enum rtdm_timer_mode mode)`
  Sleep until a specified absolute time.
- `void rtdm_task_join_nrt (rtdm_task_t *task, unsigned int poll_delay)`
  Wait on a real-time task to terminate.
- `void rtdm_task_busy_sleep (nanosecs_rel_t delay)`
  Busy-wait a specified amount of time.
- `int rtdm_timer_init (rtdm_timer_t *timer, rtdm_timer_handler_t handler, const char *name)`
  Initialise a timer.
- `void rtdm_timer_destroy (rtdm_timer_t *timer)`
  Destroy a timer.
- `int rtdm_timer_start (rtdm_timer_t *timer, nanosecs_abs_t expiry, nanosecs_rel_t interval, enum rtdm_timer_mode mode)`
  Start a timer.
- `void rtdm_timer_stop (rtdm_timer_t *timer)`
  Stop a timer.
- `int rtdm_timer_start_in_handler (rtdm_timer_t *timer, nanosecs_abs_t expiry, nanosecs_rel_t interval, enum rtdm_timer_mode mode)`
  Start a timer from inside a timer handler.
- `void rtdm_timer_stop_in_handler (rtdm_timer_t *timer)`
  Stop a timer from inside a timer handler.
- `int rtdm_irq_request (rtdm_irq_t *irq_handle, unsigned int irq_no, rtdm_irq_handler_t handler, unsigned long flags, const char *device_name, void *arg)`
  Register an interrupt handler.
- `int rtdm_irq_free (rtdm_irq_t *irq_handle)`
  Release an interrupt handler.
- `int rtdm_irq_enable (rtdm_irq_t *irq_handle)`
  Enable interrupt line.
- `int rtdm_irq_disable (rtdm_irq_t *irq_handle)`
  Disable interrupt line.
- `int rtdm_nrtsig_init (rtdm_nrtsig_t *nrt_sig, rtdm_nrtsig_handler_t handler, void *arg)`
  Register a non-real-time signal handler.
- `void rtdm_nrtsig_destroy (rtdm_nrtsig_t *nrt_sig)`
  Release a non-realtime signal handler.
- `void rtdm_nrtsig_pend (rtdm_nrtsig_t *nrt_sig)`
  Trigger non-real-time signal.
- `int rtdm_mmap_to_user (rtdm_user_info_t *user_info, void *src_addr, size_t len, int prot, void **pptr, struct vm_operations_struct *vm_ops, void *vm_private_data)`
  Map a kernel memory range into the address space of the user.
- `int rtdm_iomap_to_user (rtdm_user_info_t *user_info, phys_addr_t src_addr, size_t len, int prot, void **pptr, struct vm_operations_struct *vm_ops, void *vm_private_data)`
  Map an I/O memory range into the address space of the user.
- `int rtdm_munmap (rtdm_user_info_t *user_info, void *ptr, size_t len)`
  Unmap a user memory range.
- `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 in real-time context.

- **void rtdm_free** (void *ptr)
  
  Release real-time memory block.

- **int rtdm_read_user_ok** (rtdm_user_info_t *user_info, const void __user *ptr, size_t size)
  
  Check if read access to user-space memory block is safe.

- **int rtdm_rw_user_ok** (rtdm_user_info_t *user_info, const void __user *ptr, size_t size)
  
  Check if read/write access to user-space memory block is safe.

- **int rtdm_copy_from_user** (rtdm_user_info_t *user_info, void *dst, const void __user *src, size_t size)
  
  Copy user-space memory block to specified buffer.

- **int rtdm_safe_copy_from_user** (rtdm_user_info_t *user_info, void *dst, const void __user *src, size_t size)
  
  Check if read access to user-space memory block and copy it to specified buffer.

- **int rtdm_copy_to_user** (rtdm_user_info_t *user_info, void __user *dst, const void *src, size_t size)
  
  Copy specified buffer to user-space memory block.

- **int rtdm_safe_copy_to_user** (rtdm_user_info_t *user_info, void __user *dst, const void *src, size_t size)
  
  Check if read/write access to user-space memory block is safe and copy specified buffer to it.

- **int rtdm_strncpy_from_user** (rtdm_user_info_t *user_info, char *dst, const char __user *src, size_t count)
  
  Copy user-space string to specified buffer.

- **int rtdm_in_rt_context** (void)
  
  Test if running in a real-time task.

- **int rtdm_rt_capable** (rtdm_user_info_t *user_info)
  
  Test if the caller is capable of running in real-time context.

### Timeout Sequence Management

- **void rtdm_toseq_init** (rtdm_toseq_t *timeout_seq, nanosecs_rel_t timeout)
  
  Initialise a timeout sequence.

- **EXPORT_SYMBOL_GPL (rtdm_toseq_init)**
  
  Initialise a timeout sequence.

### Event Services

- **void rtdm_event_init** (rtdm_event_t *event, unsigned long pending)
  
  Initialise an event.

- **EXPORT_SYMBOL_GPL (rtdm_event_init)**
  
  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.

- **EXPORT_SYMBOL_GPL (rtdm_event_signal)**
  
  Initialise an event.

- **int rtdm_event_wait** (rtdm_event_t *event)
  
  Wait on event occurrence.
- **EXPORT_SYMBOL_GPL** (*rtdm_event_wait*)
  Initialise an event.

- **int** **rtdm_event_timedwait** (*rtdm_event_t* *event*, **nanosecs_rel_t** **timeout**, **rtdm_toseq_t** *timeout_seq*)
  Wait on event occurrence with timeout.

- **EXPORT_SYMBOL_GPL** (*rtdm_event_timedwait*)
  Initialise an event.

- **void** **rtdm_event_clear** (*rtdm_event_t* *event*)
  Clear event state.

- **EXPORT_SYMBOL_GPL** (*rtdm_event_clear*)
  Initialise an event.

- **int** **rtdm_event_select_bind** (*rtdm_event_t* *event*, *rtdm_selector_t* *selector*, **enum** **rtdm_selecttype** **type**, **unsigned** **fd_index**)
  Bind a selector to an event.

- **EXPORT_SYMBOL_GPL** (*rtdm_event_select_bind*)
  Initialise an event.

**Semaphore Services**

- **void** **rtdm_sem_init** (*rtdm_sem_t* *sem*, **unsigned long** value)
  Initialise a semaphore.

- **EXPORT_SYMBOL_GPL** (*rtdm_sem_init*)
  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.

- **EXPORT_SYMBOL_GPL** (*rtdm_sem_down*)
  Initialise a semaphore.

- **int** **rtdm_sem_timeddown** (*rtdm_sem_t* *sem*, **nanosecs_rel_t** **timeout**, **rtdm_toseq_t** *timeout_seq*)
  Decrement a semaphore with timeout.

- **EXPORT_SYMBOL_GPL** (*rtdm_sem_timeddown*)
  Initialise a semaphore.

- **void** **rtdm_sem_up** (*rtdm_sem_t* *sem*)
  Increment a semaphore.

- **EXPORT_SYMBOL_GPL** (*rtdm_sem_up*)
  Initialise a semaphore.

- **int** **rtdm_sem_select_bind** (*rtdm_sem_t* *sem*, *rtdm_selector_t* *selector*, **enum** **rtdm_selecttype** **type**, **unsigned** **fd_index**)
  Bind a selector to a semaphore.

- **EXPORT_SYMBOL_GPL** (*rtdm_sem_select_bind*)
  Initialise a semaphore.

**Mutex Services**

- **void** **rtdm_mutex_init** (*rtdm_mutex_t* *mutex*)
  Initialise a mutex.

- **EXPORT_SYMBOL_GPL** (*rtdm_mutex_init*)
  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.

- **EXPORT_SYMBOL_GPL** (*rtdm_mutex_lock*)
  Initialise a mutex.

- **int** **rtdm_mutex_timedlock** (*rtdm_mutex_t* *mutex*, **nanosecs_rel_t** **timeout**, **rtdm_toseq_t** *timeout_seq*)
  Request a mutex with timeout.

- **EXPORT_SYMBOL_GPL** (*rtdm_mutex_timedlock*)
  Initialise a mutex.
7.8.1 Detailed Description

Real-Time Driver Model for Xenomai, driver library.

Note

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

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

Xenomai is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

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

7.9 ksrc/skins/rtdm/module.c File Reference

Real-Time Driver Model for Xenomai.

#include <nucleus/pod.h>
#include <rtdm/syscall.h>
#include "rtdm/internal.h"

7.9.1 Detailed Description

Real-Time Driver Model for Xenomai.

Note

Copyright (C) 2005, 2006 Jan Kiszka jan.kiszka@web.de
Copyright (C) 2005 Joerg Langenberg joerg.langenberg@gmx.net

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

Xenomai is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

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

7.10 ksrc/skins/rtdm/core.c File Reference

Real-Time Driver Model for Xenomai, device operation multiplexing.
#include <nucleus/pod.h>
#include <nucleus/ppd.h>
#include <nucleus/heap.h>
#include <rtdm/syscall.h>
#include "rtdm/internal.h"
#include <linux/workqueue.h>

Functions

- struct rtdm_dev_context * rtdm_context_get (int fd)
  Retrieve and lock a device context.
- int rtdm_select_bind (int fd, rtdm_selector_t *selector, enum rtdm_selecttype type, unsigned fd_index)
  Bind a selector to specified event types of a given file descriptor.
- void rtdm_context_lock (struct rtdm_dev_context *context)
  Increment context reference counter.
- void rtdm_context_unlock (struct rtdm_dev_context *context)
  Decrement context reference counter.
- void rtdm_context_put (struct rtdm_dev_context *context)
  Release a device context obtained via rtdm_context_get()
- int rtdm_open (const char *path, int oflag,...)
  Open a device.
- int rtdm_socket (int protocol_family, int socket_type, int protocol)
  Create a socket.
- int rtdm_close (int fd)
  Close a device or socket.
- int rtdm_ioctl (int fd, int request,...)
  Issue an IOCTL.
- ssize_t rtdm_read (int fd, void *buf, size_t nbyte)
  Read from device.
- ssize_t rtdm_write (int fd, const void *buf, size_t nbyte)
  Write to device.
- ssize_t rtdm_recvmsg (int fd, struct user_msghdr *msg, int flags)
  Receive message from socket.
- ssize_t rtdm_recvfrom (int fd, void *buf, size_t len, int flags, struct sockaddr *from, socklen_t *fromlen)
  Receive message from socket.
- ssize_t rtdm_recv (int fd, void *buf, size_t len, int flags)
  Receive message from socket.
- ssize_t rtdm_sendmsg (int fd, const struct user_msghdr *msg, int flags)
  Transmit message to socket.
- ssize_t rtdm_sendto (int fd, const void *buf, size_t len, int flags, const struct sockaddr *to, socklen_t tolen)
  Transmit message to socket.
- ssize_t rtdm_send (int fd, const void *buf, size_t len, int flags)
  Transmit message to socket.
- int rtdm_bind (int fd, const struct sockaddr *my_addr, socklen_t addrlen)
  Bind to local address.
- int rtdm_connect (int fd, const struct sockaddr *serv_addr, socklen_t addrlen)
  Connect to remote address.
**rtdm_listen** (int fd, int backlog)

*Listen for incoming connection requests.*

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

*Accept a connection request.*

**rtdm_shutdown** (int fd, int how)

*Shut down parts of a connection.*

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

*Get socket option.*

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

*Set socket option.*

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

*Get local socket address.*

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

*Get socket destination address.*

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

*Open a device.*

**rt_dev_socket** (int protocol_family, int socket_type, int protocol)

*Create a socket.*

**rt_dev_close** (int fd)

*Close a device or socket.*

**rt_dev_ioctl** (int fd, int request, ...)

*Issue an IOCTL.*

**rt_dev_read** (int fd, void *buf, size_t nbyte)

*Read from device.*

**rt_dev_write** (int fd, const void *buf, size_t nbyte)

*Write to device.*

**rt_dev_recvmsg** (int fd, struct user_msghdr *msg, int flags)

*Receive message from socket.*

**rt_dev_recvfrom** (int fd, void *buf, size_t len, int flags, struct sockaddr *from, socklen_t *fromlen)

*Receive message from socket.*

**rt_dev_recv** (int fd, void *buf, size_t len, int flags)

*Receive message from socket.*

**rt_dev_sendmsg** (int fd, const struct user_msghdr *msg, int flags)

*Transmit message to socket.*

**rt_dev_sendto** (int fd, const void *buf, size_t len, int flags, const struct sockaddr *to, socklen_t tolen)

*Transmit message to socket.*

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

*Transmit message to socket.*

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

*Bind to local address.*

**rt_dev_connect** (int fd, const struct sockaddr *serv_addr, socklen_t addrlen)

*Connect to remote address.*

**rt_dev_listen** (int fd, int backlog)

*Listen for incoming connection requests.*

**rt_dev_accept** (int fd, struct sockaddr *addr, socklen_t *addrlen)

*Accept a connection request.*

**rt_dev_shutdown** (int fd, int how)

*Shut down parts of a connection.*

**rt_dev_getsockopt** (int fd, int level, int optname, void *optval, socklen_t *optlen)
7.10 File Reference

Get socket option.

- int rt_dev_setsockopt (int fd, int level, int optname, const void */optval, socklen_t *optlen)

Set socket option.

- int rt_dev_getsockopt (int fd, struct sockaddr */name, socklen_t *namelen)

Get local socket address.

- int rt_dev_getpeername (int fd, struct sockaddr */name, socklen_t */namelen)

Get socket destination address.

7.10.1 Detailed Description

Real-Time Driver Model for Xenomai, device operation multiplexing.

Note

Copyright (C) 2005 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.
Chapter 8

Example Documentation

8.1 bufp-label.c

/*
 * BUF-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).
 * See Makefile in this directory for build directives.
 */
#include <sys/mman.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <string.h>
#include <pthread.h>
#include <errno.h>
#include <rtdk.h>
#include <rtdm/rtipc.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_RTPC, 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, BUF_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_BUFP, 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_port specifies the port number to use. If -1 is passed, the BUFP driver will auto-select an idle port.
 */
saddr.sipc_family = AF_RTIPC;
saddr.sipc_port = -1;
ret = bind(s, (struct sockaddr *)&saddr,
            sizeof(saddr));
if (ret)
    fail("bind");

for (;;) {
    len = strlen(msg[n]);
    ret = write(s, msg[n], len);
    if (ret < 0)
        close(s);
    else
        rt_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_RTIPC, SOCK_DGRAM, IPCPROTO_BUFPP);  
    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 BUFFP does not try to register this label for the client port as well (like the server thread did).
    */
    strcpy(plabel.label, BUFP_PORT_LABEL);
    ret = setsockopt(s, SOL_BUFP, BUFP_LABEL,
                    &plabel, sizeof(plabel));
    if (ret)
        fail("setsockopt");

    memset(&svaddr, 0, sizeof(svaddr));
    svaddr.sipc_family = AF_RTIPC;
    svaddr.sipc_port = -1; /* Tell BUFPP to search by label. */
    ret = connect(s, (struct sockaddr *)&svaddr, sizeof(svaddr));
    if (ret)
        fail("connect");

    for (;;) {
        len = strlen(msg[n]);
        ret = write(s, msg[n], len);
    }
}
if (ret < 0) {
    close(s);
    fail("write");
}
rt_printf("%s: sent %d bytes, \%s\%n",
    __FUNCTION__, ret, ret, msg[n]);
/*
 * 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 cleanup_upon_sig(int sig)
{
    pthread_cancel(svtid);
pthread_cancel(cltid);
signal(sig, SIG_DFL);
pthread_join(svtid, NULL);
pthread_join(cltid, NULL);
}

int main(int argc, char **argv)
{
    struct sched_param svparam = {.sched_priority = 71};
    struct sched_param clparam = {.sched_priority = 70};
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);
ererrno = 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);
ererrno = pthread_create(&cltid, &clattr, &client, NULL);
    if (errno)
        fail("pthread_create");
sigsuspend(&oldmask);
    return 0;
}

8.2 bufp-readwrite.c

/*
 * 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).

See Makefile in this directory for build directives.

```
#include <sys/socket.h>
#include <sys/un.h>
#include <sys/mman.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <string.h>
#include <pthread.h>
#include <errno.h>
#include <rtdk.h>
#include <rtdm/rtipc.h>
```

```
#include <sys/sman.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <string.h>
#include <pthread.h>
#include <errno.h>
#include <rtdk.h>
#include <rtdm/rtipc.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");

    for (;;) {
        ret = read(s, buf, sizeof(buf));
        if (ret < 0) {
            close(s);
            fail("read");
        }

        rt_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_BUFPP);
    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");
        }
        rt_printf("%s: sent %d bytes, %.*s
", __FUNCTION__, ret, ret, msg[n]);
        n = (n + 1) % (sizeof(msg) / sizeof(msg[0]));
    }
    return NULL;
}

static void cleanup_upon_sig(int sig)
{
    pthread_cancel(svtid);
    pthread_cancel(cltid);
    signal(sig, SIG_DFL);
    pthread_join(svtid, NULL);
    pthread_join(cltid, 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 mask, oldmask;

    mlockall(MCL_CURRENT | MCL_FUTURE);
    sigemptyset(&mask);
    sigaddset(&mask, SIGINT);
    signal(SIGINT, cleanup_upon_sig);
    sigaddset(&mask, SIGTERM);
    signal(SIGTERM, cleanup_upon_sig);
    sigaddset(&mask, SIGHUP);
    signal(SIGHUP, cleanup_upon_sig);
    pthread_sigmask(SIG_BLOCK, &mask, &oldmask);

    /*
    * This is a real-time compatible printf() package from
    * Xenomai's RT Development Kit (RTDK), that does NOT cause
    * any transition to secondary mode.
    */
    rt_print_auto_init(1);

    pthread_attr_init(&svattr);
    pthread_attr_setdetachstate(&svattr, PTHREAD_CREATE_JOINABLE);
    pthread_attr_setschedpolicy(&svattr, PTHREAD_EXPLICIT_SCHED);
    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");

sigsuspend(&oldmask);

return 0;
}

8.3 cross-link.c

/*
 * cross-link.c
 *
 * Userspace test program (Xenomai native 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 <native/task.h>
#include <native/timer.h>
#include <rtdm/rtserial.h>

#define MAIN_PREFIX "main : 
#define WTASK_PREFIX "write_task: 
#define RTASK_PREFIX "read_task: 
#define WRITE_FILE "rtser0"
#define READ_FILE "rtser1"

int read_fd = -1;
int write_fd = -1;

#define STATE_FILE_OPENED 1
#define STATE_TASK_CREATED 2

unsigned int read_state = 0;
unsigned int write_state = 0;

/* --s-ms-us-ns */
RTIME write_task_period_ns = 100000000llu;
RT_TASK write_task;
RT_TASK read_task;

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,
};
```c
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 = rt_dev_close(fd);
        switch (err) {
            case -EAGAIN:
                printf(MAIN_PREFIX "%s -> EAGAIN (%d times)\n", name, i);
                rt_task_sleep(50000); /* wait 50us */
                break;
            case 0:
                printf(MAIN_PREFIX "%s -> closed\n", name);
                break;
            default:
                printf(MAIN_PREFIX "%s -> %s\n", name,
                    strerror(-err));
                break;
        }
    } while (err == -EAGAIN && i < 10);
    return err;
}

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;
    }
}

void catch_signal(int sig)
{
    cleanup_all();
    printf(MAIN_PREFIX "exit\n");
    return;
}

void write_task_proc(void *arg)
{
    int err;
    RTIME write_time;
    ssize_t sz = sizeof(RTIME);
    ssize_t 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));
            goto exit_write_task;
        }
    }
exit_write_task:
    return;
}
```
break;
}
write_time = rt_timer_read();
written = rt_dev_write(write_fd, &write_time, sz);
if (written < 0 ) {
    printf(WTASK_PREFIX "error on rt_dev_write, %s\n",
            strerror(-err));
    break;
} else if (written != sz) {
    printf(WTASK_PREFIX "only %d / %d 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;
}
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);
    ssize_t read = 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 = rt_dev_ioctl(read_fd, RTSER_RTIOC_WAIT_EVENT, &rx_event);
        if (err) {
            printf(RTASK_PREFIX "error on RTSER_RTIOC_WAIT_EVENT, %s\n",
                   strerror(-err));
            if (err == -ETIMEDOUT)
                continue;
            break;
        }
        irq_time = rx_event.rxpend_timestamp;
        read = rt_dev_read(read_fd, &write_time, sz);
        if (read == 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 (read < 0 ) {
            printf(RTASK_PREFIX "error on rt_dev_read, code %s\n",
                   strerror(-err));
            break;
        } else {
            printf(RTASKPREFIX "only %d / %d byte received \n", read, sz);
            break;
        }
    }
    if ((read_state & STATE_FILE_OPENED) &&
        close_file(read_fd, READ_FILE " (read)" ) == 0)
read_state &= ~STATE_FILE_OPENED;
}
int main(int argc, char* argv[])
{
    int err = 0;
}
signal(SIGTERM, catch_signal);
signal(SIGINT, catch_signal);

/* no memory-swapping for this program */
mlockall(MCL_CURRENT | MCL_FUTURE);

/* open rtser0 */
write_fd = rt_dev_open( WRITE_FILE, 0);
if (write_fd < 0) {
    printf(MAIN_PREFIX "can't open %s (write), %s\n", WRITE_FILE, strerror(-write_fd));
    goto error;
}
write_state |= STATE_FILE_OPENED;
printf(MAIN_PREFIX "write-file opened\n");

/* writing write-config */
err = rt_dev_ioctl(write_fd, RTSER_RTIOC_SET_CONFIG, &write_config);
if (err) {
    printf(MAIN_PREFIX "error while RTSER_RTIOC_SET_CONFIG, %s\n", strerror(-err));
    goto error;
}
printf(MAIN_PREFIX "write-config written\n");

/* open rtser1 */
read_fd = rt_dev_open( READ_FILE, 0 );
if (read_fd < 0) {
    printf(MAIN_PREFIX "can't open %s (read), %s\n", READ_FILE, strerror(-read_fd));
    goto error;
}
read_state |= STATE_FILE_OPENED;
printf(MAIN_PREFIX "read-file opened\n");

/* writing read-config */
err = rt_dev_ioctl(read_fd, RTSER_RTIOC_SET_CONFIG, &read_config);
if (err) {
    printf(MAIN_PREFIX "error while rt_dev_ioctl, %s\n", strerror(-err));
    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");

/* 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 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;
}

/* 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;
}
pause();
return 0;

error:
8.4 iddp-label.c

"/*
 * 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.
 *
 * See Makefile in this directory for build directives.
 */
#include <sys/mman.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <string.h>
#include <pthread.h>
#include <errno.h>
#include <rtdk.h>
#include <rtdm/rtipc.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;
    char buf[128];
    int ret, s;
    
    s = socket(AF_RTIPC, 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");
    }
    rt_printf("%s: received %d bytes, \%s\n", "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)
8.5 iddp-sendrecv.c

/*
 * IDDP-based client/server demo, using the sendto(2)/recvfrom(2)
 * system calls to exchange data over a socket.
*/

Generated on Thu Jul 7 2016 13:25:19 for Xenomai RTDM skin API by Doxygen
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.

See Makefile in this directory for build directives.

```
#include <sys/mman.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <string.h>
#include <pthread.h>
#include <errno.h>
#include <rtdk.h>
#include <rtdm/rtipc.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));
    *
    * 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");
    *     }
    */
```

Generated on Thu Jul 7 2016 13:25:19 for Xenomai RTDM skin API by Doxygen
rt_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 svaddr, 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");
    svaddr.sipc_family = AF_RTIPC;
    svaddr.sipc_port = IDDP_SRVPORT;
    for (; ; )
    {
        len = strlen(msg[n]);
        ret = sendto(s, msg[n], len, 0, (struct sockaddr *)&svaddr, sizeof(svaddr));
        if (ret < 0)
            close(s);
        rt_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 cleanup_upon_sig(int sig)
{
    pthread_cancel(svtid);
    pthread_cancel(cltid);
    signal(sig, SIG_DFL);
    pthread_join(svtid, NULL);
    pthread_join(cltid, 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 mask, oldmask;
    mlockall(MCL_CURRENT | MCL_FUTURE);
    sigemptyset(&mask);
    sigaddset(&mask, SIGINT);
    signal(SIGINT, cleanup_upon_sig);
    sigaddset(&mask, SIGTERM);
    signal(SIGTERM, cleanup_upon_sig);
    sigaddset(&mask, SIGHUP);
    signal(SIGHUP, cleanup_upon_sig);
    pthread_sigmask(SIG_BLOCK, &mask, &oldmask);
    
    /*
    * This is a real-time compatible printf() package from
    * Xenomai’s RT Development Kit (RTDK), that does NOT cause
    * any transition to secondary mode.
    */
    rt_print_auto_init(1);
    pthread_attr_init(&svattr);
    pthread_attr_setdetachstate(&svattr, PTHREAD_CREATE_JOINABLE);
    pthread_attr_setschedparam(&svattr, svparam);
    pthread_attr_setschedpolicy(&svattr, SCHED_FIFO);
    
    Generated on Thu Jul 7 2016 13:25:19 for Xenomai RTDM skin API by Doxygen
pthread_attr_setschedparam(&svattr, &svparam);
errno = pthread_create(&svtid, &svattr, &server, NULL);
if (errno)
    fail("pthread_create");

pthread_attr_init(&clattr);
pthread_attr_setschedparam(&clattr, &clparam);

pthread_attr_setschedpolicy(&clattr, SCHED_FIFO);

errno = pthread_create(&cltid, &clattr, &client, NULL);
if (errno)
    fail("pthread_create");
sigsuspend(&oldmask);

return 0;
}

8.6 rtcan_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 <mqueue.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 <netinet/in.h>
#include <net/if.h>
#include <linux/ioct1.h>
#include <sys/mman.h>

ifdef __XENO__
#include <rtdm/rtcan.h>
#else
#include <linux/can.h>
#include <linux/can/raw.h>
#endif

#define NSEC_PER_SEC 1000000000

Generated on Thu Jul 7 2016 13:25:19 for Xenomai RTDM skin API by Doxygen
static unsigned int cycle = 10000; /* 10 ms */
static canid_t can_id = 0x1;

static pthread_t txthread, rxthread;
static int txsock, rxsock;
static mqd_t mq;
static int txcount, rxcount;
static int overruns;
static int repeater;

struct rtt_stat {
    long long rtt;
    long long rtt_min;
    long long rtt_max;
    long long rtt_sum;
    long long rtt_sum_last;
    int counts_per_sec;
};

static void print_usage(char *prg) {
    fprintf(stderr,
            "Usage: %s [Options] <tx-can-interface> <rx-can-interface>
            Options:
            " -h, --help This help
            " -r, --repeater Repeater, send back received messages
            " -i, --id=ID CAN Identifier (default = 0x1)
            " -c, --cycle Cycle time in us (default = 10000us)
            ",
            prg);
}

void *transmitter(void *arg) {
    struct sched_param param = { .sched_priority = 80 };
    struct timespec next_period;
    struct timespec time;
    struct can_frame frame;
    long long *rtt_time = (long long *)&frame.data;
    /* Pre-fill CAN frame */
    frame.can_id = can_id;
    frame.can_dlc = sizeof(*rtt_time);
    #ifdef __XENO__
    pthread_set_name_np(pthread_self(), "rtcan_rtt_transmitter");
    #endif
    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);
        *rtt_time = (long long)time.tv_sec * NSEC_PER_SEC + time.tv_nsec;
        /* Transmit the message containing the local time */
        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++;
    }
}

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;
    struct rtt_stat rtt_stat = {0, 1000000000000000000LL, -1000000000000000000LL, 

Generated on Thu Jul 7 2016 13:25:19 for Xenomai RTDM skin API by Doxygen
#ifdef __XENO__
    pthread_set_name_np(pthread_self(), "rtcan_rtt_receiver");
#endif

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 == EBADF)
            printf("terminating receiver thread\n");
        else
            perror("recv failed");
        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 {
        clock_gettime(CLOCK_MONOTONIC, &time);
        if (rxcount > 0) {
            rtt_stat.rtt = ((long long)time.tv_sec * 1000000000LL +
                            time.tv_nsec - *rtt_time);
            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;
        }
    }
}

void catch_signal(int sig)
{
    mq_close(mq);
}

int main(int argc, char *argv[])
{
    struct sched_param param = { .sched_priority = 1 };  
    pthread_attr_t thattr;
    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 ifreq ifr;
    int ret, opt;

    struct option long_options[] = {
        ["id", required_argument, 0, "i"],
        ["cycle", required_argument, 0, "c"],
        ["repeater", no_argument, 0, "r"],
        ["help", no_argument, 0, "h"],
        [ 0, 0, 0, 0],
    };

    while (((opt = getopt_long(argc, argv, "hri:c:", long_options, NULL)) != -1) {
        switch (opt) {
        case 'c':
            cycle = atoi(optarg);
            break;
        case 'i':
            can_id = strtoul(optarg, NULL, 0);
            break;
        case 'r':
            break;
        }
repeater = 1;
break;

default:
fprintf(stderr, "Unknown option \%c\n", opt);
case 'h':
print_usage(argv[0]);
exit(-1);
}

printf("%d %d\n", optind, argc);
if (optind + 2 != argc) {
print_usage(argv[0]);
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\n", 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 = 0x3ff;
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\n", txsock, ifr.ifr_name);
  if (ioctl(txsock, SIOCGIFINDEX, &ifr) < 0) {
    perror("TX ioctl SIOCGIFINDEX failed");
    goto failure2;
  }

  /* Suppress definition of a default receive filter list */
  if (setsockopt(txsock, SOL_CAN_RAW, CAN_RAW_FILTER, NULL, 0) < 0) {
    perror("TX setsockopt CAN_RAW_FILTER 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);
mlockall(MCL_CURRENT|MCL_FUTURE);
printf("Round-Trip-Time test %s -> %s with CAN ID 0x%x\n", argv[optind], argv[optind + 1], can_id);
cycle; printf("Cycle time: %d us\n", cycle);
printf("All RTT timing figures are in us.\n\n\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 mqueue failed");
go failure2;
}
/* Create receiver RT-thread */
pthread_attr_init(&thattr);
pthread_attr_setdetachstate(&thattr, PTHREAD_CREATE_JOINABLE);
pthread_attr_setstacksize(&thattr, PTHREAD_STACK_MIN);
ret = pthread_create(&rxthread, &thattr, &receiver, NULL);
if (ret) {
    fprintf(stderr, "%s: pthread_create(receiver) failed\n", strerror(-ret));
go failure3;
}
if (!repeater) {
    /* Create transmitter RT-thread */
    ret = pthread_create(&txthread, &thattr, &transmitter, NULL);
    if (ret) {
        fprintf(stderr, "%s: pthread_create(transmitter) failed\n", strerror(-ret));
go failure4;
    }
    pthread_setschedparam(pthread_self(), SCHED_FIFO, &param);
}
if (repeater) {
    printf("Messages\n");
} else {
    printf("Messages RTT_last RTT_avg RTT_min RTT_max Overruns\n");
}
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("%8ld\n", rxcount);
    } else {
        rtt_avg = ((rtt_stat.rtt_sum - rtt_stat.rtt_sum_last) / rtt_stat.counts_per_sec);
        printf("%8ld %7ld %7ld %7ld %7ld %8d\n",
                (long)(rtt_avg / 1000), (long)(rtt_stat.rtt / 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! */
    while ((close(rxsock) < 0) && (errno == EAGAIN)) {
        printf("RX socket busy - waiting...\n");
sleep(1);
    }
    while ((close(txsock) < 0) && (errno == EAGAIN)) {
        printf("TX socket busy - waiting...\n");
sleep(1);
    }
}
pthread_join(txthread, NULL);
pthread_kill(rxthread, SIGHUP);
pthread_join(rxthread, NULL);

return 0;

failure4:
pthread_kill(rxthread, SIGHUP);
pthread_join(rxthread, NULL);

failure3:
mq_close(mq);
failure2:
close(txsock);
failure1:
close(rxsock);

return 1;

}
return CAN_MODE_START;
else if (strncmp(str, "down") || strncmp(str, "stop") )
  return CAN_MODE_STOP;
else if (strncmp(str, "sleep") )
  return CAN_MODE_SLEEP;
return -EINVAL;
}

int string_to_ctrlmode(char *str)
{
  if (strncmp(str, "listenonly") )
    return CAN_CTRLMODE_LISTENONLY;
  else if (strncmp(str, "loopback") )
    return CAN_CTRLMODE_LOOPBACK;
  else if (strncmp(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;
  int bittime_count = 0, bittime_data[6];
  struct ifreq ifr;
  can_baudrate_t *baudrate;
  can_ctrlmode_t *ctrlmode;
  can_mode_t *mode;
  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 || !strcmp(ptr, ':'))
          break;
        ptr = strchr(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;
default:
    fprintf(stderr, "Unknown option %c\n", opt);
    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;
    }
}

if (new_baudrate != -1) {
    baudrate = (can_baudrate_t *)&ifr.ifr_ifru;
    *baudrate = new_baudrate;
    ret = rt_dev_ioctl(can_fd, SIOCSCANBAUDRATE, &ifr);
    if (ret) {
        goto abort;
    }
}

if (bittime_count) {
    bittime = (struct can_bittime *)&ifr.ifr_ifru;
    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 = rt_dev_ioctl(can_fd, SIOCSCANCUSTOMBITTIME, &ifr);
if (ret) {
    goto abort;
}

Generated on Thu Jul 7 2016 13:25:19 for Xenomai RTDM skin API by Doxygen
if (set_ctrlmode != 0) {
    ctrlmode = (can_ctrlmode_t *)&ifr.ifr_ifru;
    *ctrlmode = new_ctrlmode;
    if (verbose)
        printf("ctrlmode: %#x\n", new_ctrlmode);
    if (ret) {
        goto abort;
    }
}

if (new_mode != -1) {
    mode = (can_mode_t *)&ifr.ifr_ifru;
    *mode = new_mode;
    ret = rt_dev_ioctl(can_fd, SIOCSCANMODE, &ifr);
    if (ret) {
        goto abort;
    }
}

rt_dev_close(can_fd);
return 0;

abort:
rt_dev_close(can_fd);
return ret;

8.8 rtcanrecv.c

/*
 * Program to receive CAN messages
 *
 * Copyright (C) 2006 Wolfgang Grandegger <wg@grandegger.com>
 *
 * 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 <time.h>
#include <errno.h>
#include <getopt.h>
#include <sys/mman.h>

#include <native/task.h>
#include <native/pipe.h>

#include <rtdm/rtdcan.h>

static void print_usage(char *prg)
{
    fprintf(stderr,
        "Usage: %s [-<can-interface>] [Options]\n" "Options:\n" " -f --filter=id:mask[: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);
}

extern int optind, opterr, optopt;

Generated on Thu Jul 7 2016 13:25:19 for Xenomai RTDM skin API by Doxygen
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;

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;
}

void cleanup(void)
{
    int ret;
    if (verbose)
        printf("Cleaning up...\n");
    if (s >= 0) {
        ret = rt_dev_close(s);
        s = -1;
        if (ret) {
            printf(stderr, "rt_dev_close: %s\n", strerror(-ret));
            exit(EXIT_SUCCESS);
        }
    }
}

void cleanup_and_exit(int sig)
{
    if (verbose)
        printf("Signal %d received\n", sig);
    cleanup();
    exit(0);
}

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 = rt_dev_recvmsg(s, &msg, 0);
        }
        else
            ret = rt_dev_recvfrom(s, (void *)&frame, sizeof(can_frame_t), 0,
                                 (struct sockaddr *)&addr, &addrlen);
        if (ret < 0) {
            switch (ret) {
            case -ETIMEDOUT:
                if (verbose)
                    printf("rt_dev_recv: timed out\n");
                continue;
            case -EBADF:
                if (verbose)
                    printf("rt_dev_recv: aborted because socket was closed\n");
                break;
            default:
                printf(stderr, "rt_dev_recv: %s\n", strerror(-ret));
            }
        }
        i = ret;
    }
if (print && (count % print) == 0) {
    printf("#%d: (%d) ", count, addr.can_ifindex);
    if (timestamp && msg.msg_controllen) {
        if (timestamp_rel) {
            printf("%lldns ", (long long)(timestamp - timestamp_prev));
            timestamp_prev = timestamp;
        } else
            printf("%lldns ", (long long)timestamp);
    }
    if (frame.can_id & CAN_ERR_FLAG)
        printf("!0x%08x! ", frame.can_id & CAN_ERR_MASK);
    else if (frame.can_id & CAN_EFF_FLAG)
        printf("<0x%08x> ", frame.can_id & CAN_EFF_MASK);
    else
        printf("<0x%03x> ", frame.can_id & CAN_SFF_MASK);
    printf(" [%d] ", frame.can_dlc);
    if (!(frame.can_id & CAN_RTR_FLAG))
        for (i = 0; i < frame.can_dlc; i++)
            printf(" %02x", frame.data[i]);
    if (frame.can_id & CAN_ERR_FLAG) {
        printf(" ERROR ");
        if (frame.can_id & CAN_ERR_BUSOFF)
            printf("bus-off");
        if (frame.can_id & CAN_ERR_CRTL)
            printf("controller problem");
    } else if (frame.can_id & CAN_RTR_FLAG)
        printf(" remote request");
    printf("\n");
    count++;
}

int main(int argc, char **argv)
{
    int opt, ret;
    u_int32_t id, mask;
    u_int32_t err_mask = 0;
    struct 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' },
        {"error", required_argument, 0, 'e' },
        {"timeout", required_argument, 0, 't' },
        {"timestamp", no_argument, 0, 'T' },
        {"timestamp-rel", no_argument, 0, 'R' },
        { 0, 0, 0, 0 },
    };

    mlockall(MCL_CURRENT | MCL_FUTURE);
    signal(SIGTERM, cleanup_and_exit);
    signal(SIGINT, cleanup_and_exit);

    while ((opt = getopt_long(argc, argv,"hve:f: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);

Generated on Thu Jul 7 2016 13:25:19 for Xenomai RTDM skin API by Doxygen
ptr = strchr(ptr, ':');
if (!ptr) {
    fprintf(stderr, "filter must be applied in the form id:mask[:id:mask]...
"
    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 = rt_dev_socket(PF_CAN, SOCK_RAW, CAN_RAW);
if (ret < 0) {
    fprintf(stderr, "rt_dev_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 = rt_dev_ioctl(s, SIOCGIFINDEX, &ifr);
    if (ret < 0) {
        fprintf(stderr, "rt_dev_ioctl GET_IFINDEX: %s\n", strerror(-ret));
        goto failure;
    }
    if (err_mask) {
        ret = rt_dev_setsockopt(s, SOL_CAN_RAW, CAN_RAW_ERR_FILTER,
            &err_mask, sizeof(err_mask));
        if (ret < 0) {
            fprintf(stderr, "rt_dev_setsockopt: %s\n", strerror(-ret));
            goto failure;
        }
        if (verbose)
            printf("Using err_mask=%#x\n", err_mask);
    }
    if (filter_count) {
        ret = rt_dev_setsockopt(s, SOL_CAN_RAW, CAN_RAW_FILTER,
            &recv_filter, filter_count * sizeof(struct can_filter));
        if (ret < 0) {
            fprintf(stderr, "rt_dev_setsockopt: %s\n", strerror(-ret));
            goto failure;
        }
    }
    recv_addr.can_family = AF_CAN;
    recv_addr.can_ifindex = ifr.ifr_ifindex;
    ret = rt_dev_bind(s, (struct sockaddr *)&recv_addr,
        sizeof(struct sockaddr_can));
    if (ret < 0) {
        fprintf(stderr, "rt_dev_bind: %s\n", strerror(-ret));
        goto failure;
    }
}
if (timeout) {
    if (verbose)
        printf("Timeout: %lld ns\n", (long long)timeout);
    ret = rt_dev_ioctl(s, RTCAN_RTIOC_RCV_TIMEOUT, &timeout);
    if (ret) {
        fprintf(stderr, "rt_dev_ioctl RCV_TIMEOUT: %s\n", strerror(-ret));
        goto failure;
    }
}

if (with_timestamp) {
    ret = rt_dev_ioctl(s, RTCAN_RTIOC_TAKE_TIMESTAMP, RTCAN_TAKE_TIMESTAMPS);
    if (ret) {
        fprintf(stderr, "rt_dev_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 */

failure:
    cleanup();
    return -1;
}

8.9 rtcansend.c

/*
 * Program to send CAN messages
 *
 * Copyright (C) 2006 Wolfgang Grandegger <wg@grandegger.com>
 *
 * 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 <time.h>
#include <errno.h>
#include <getopt.h>
#include <sys/mman.h>
#include <native/task.h>
#include <native/timer.h>
#include <native/pipe.h>
#include <rtdm/rtcan.h>

extern int optind, opterr, optopt;

static void print_usage(char *prg)
{
    fprintf(stderr, "Usage: %s <can-interface> [Options] <can-msg>\n", prg);
    fprintf(stderr, "<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[0-3]\n");
Example Documentation

```
-d, --delay=MS    delay in ms (default = 1ms)
-s, --send        use send instead of sendto
-t, --timeout=MS  timeout in ms
-l, --loopback=0|1 switch local loopback off or on
-v, --verbose     be verbose
-p, --print=MODULO print every MODULO message
-h, --help        this help

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;

void cleanup(void)
{
    int ret;
    if (verbose)
        printf("Cleaning up...
");

    usleep(100000);

    if (s >= 0) {
        ret = rt_dev_close(s);
        s = -1;
        if (ret) {
            fprintf(stderr, "rt_dev_close: %s\n", strerror(-ret));
            exit(EXIT_SUCCESS);
        }
    }
}

void cleanup_and_exit(int sig)
{
    if (verbose)
        printf("Signal %d received\n", sig);
    cleanup();
    exit(0);
}

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 definition of a receive filter list */
        if (use_send)
            ret = rt_dev_send(s, (void *)&frame, sizeof(can_frame_t), 0);
        else
            ret = rt_dev_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("rt_dev_send(to): timed out\n");
                break;
            case -EBADF:
                if (verbose)
                    printf("rt_dev_send(to): aborted because socket was closed\n");
                break;
            default:
                fprintf(stderr, "rt_dev_send: %s\n", strerror(-ret));
                break;
            }
            i = loops;    /* abort */
            break;
        }
    }
    if (verbose && (i % print) == 0) {
        if (frame.can_id & CAN_EFF_FLAG)
            printf("<0x%08x>\n", frame.can_id & CAN_EFF_MASK);
        else
            printf("<0x%03x>\n", frame.can_id & CAN_SFF_MASK);
        printf(" [%d]\n", frame.can_dlc);
        for (j = 0; j < frame.can_dlc; j++) {
            printf(" %02x\n", frame.data[j]);
        }
    }
}

} 
printf("\n");
}

int main(int argc, char **argv)
{
    int i, opt, ret;
    struct 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 },
    };

    mlockall(MCL_CURRENT | MCL_FUTURE);

    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);
                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':
                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 \%c\n", opt);
                break;
        }
    }

    return 0;
}
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 = rt_dev_socket(PF_CAN, SOCK_RAW, CAN_RAW);
    if (ret < 0) {
        fprintf(stderr, "rt_dev_socket: %s\n", strerror(-ret));
        return -1;
    }

    s = ret;

    if (loopback >= 0) {
        ret = rt_dev_setsockopt(s, SOL_CAN_RAW, CAN_RAW_LOOPBACK,
            &loopback, sizeof(loopback));
        if (ret < 0) {
            fprintf(stderr, "rt_dev_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 = rt_dev_ioctl(s, SIOCGIFINDEX, &ifr);
    if (ret < 0) {
        fprintf(stderr, "rt_dev_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 = rt_dev_setsockopt(s, SOL_CAN_RAW, CAN_RAW_FILTER, NULL, 0);
        if (ret < 0) {
            fprintf(stderr, "rt_dev_setsockopt: %s\n", strerror(-ret));
            goto failure;
        }

        ret = rt_dev_bind(s, (struct sockaddr *)&to_addr, sizeof(to_addr));
        if (ret < 0) {
            fprintf(stderr, "rt_dev_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] = strtol(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 = rt_dev_ioctl(s, RTCAN_RTIOC_SND_TIMEOUT, &timeout);
        if (ret) {
            fprintf(stderr, "rt_dev_ioctl SND_TIMEOUT: %s\n", strerror(-ret));
        }
    }

Generated on Thu Jul 7 2016 13:25:19 for Xenomai RTDM skin API by Doxygen
printf(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;
}

8.10 xddp-echo.c

/*
 * 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,
 * 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() +--+
 * See Makefile in this directory for build directives.
 * NOTE: XDDP is a replacement for the legacy RT_PIPE interface
 * available from the native skin until Xenomai 3.
 */
#include <sys/mman.h>
#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 <rtdk.h>
#include <rtipc.h>
#include <rtipc.rth>
#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?"
};

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.
     * 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");
        rt_printf("%s: sent %d bytes, \"%s\"\n",

__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");
rt_printf(" => "
        "%.*s" echoed by peer", 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, &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;
}

static void cleanup_upon_sig(int sig)
{
    pthread_cancel(rt);
    pthread_cancel(nrt);
    signal(sig, SIG_DFL);
    pthread_join(rt, NULL);
    pthread_join(nrt, NULL);
}

int main(int argc, char **argv)
{
    struct sched_param rtparam = { .sched_priority = 42 };;
    pthread_attr_t rtattr, regattr;
    sigset_t mask, oldmask;
    mlockall(MCL_CURRENT | MCL_FUTURE);
    sigemptyset(&mask);
    sigaddset(&mask, SIGINT);
    signal(SIGINT, cleanup_upon_sig);
    sigaddset(&mask, SIGTERM);
    signal(SIGTERM, cleanup_upon_sig);
    sigaddset(&mask, SIGHUP);
    signal(SIGHUP, cleanup_upon_sig);
    pthread_sigmask(SIG_BLOCK, &mask, &oldmask);
    /*
    * This is a real-time compatible printf() package from
    * Xenomai’s RT Development Kit (RTDK), that does NOT cause
    * any transition to secondary (i.e. non real-time) mode when
    * writing output.
    */
    rt_print_auto_init(1);
    pthread_attr_init(&rtattr);
    pthread_attr_setdetachstate(&rtattr, PTHREAD_CREATE_JOINABLE);
}

Generated on Thu Jul 7 2016 13:25:19 for Xenomai RTDM skin API by Doxygen
pthread_attr_setinheritsched(&rtattr, PTHREAD_EXPLICIT_SCHED);
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_setinheritsched(&regattr, PTHREAD_EXPLICIT_SCHED);
pthread_attr_setschedpolicy(&regattr, SCHED_OTHER);

erno = pthread_create(&nrt, &regattr, &regular_thread, NULL);
if (errno)
    fail("pthread_create");
sigsuspend(&oldmask);

return 0;

}  

8.11 xddp-label.c

/*
 * 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 node). 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()  +--+
 * |
 * See Makefile in this directory for build directives.
*
* NOTE: XDDP is a replacement for the legacy RT_PIPE interface
* available from the native skin until Xenomai 3.
*/
#include <sys/mman.h>
#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 <rtdk.h>
#include <rtdm/rtipc.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 *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/xddp/<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");
    }

    return;
}
```
rt_printf("%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");

    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");

    rt_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");

        rt_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, &ts, NULL);
}
return NULL;
}

static void *regular_thread(void *arg)
{
    char buf[128], *devname;
    int fd, ret;
    if (asprintf(&devname, "/proc/xenomai/registry/rtipc/xdpp/%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;
}

static void cleanup_upon_sig(int sig)
{
    pthread_cancel(rt1);
    pthread_cancel(rt2);
    pthread_cancel(nrt);
    signal(sig, SIG_DFL);
    pthread_join(rt1, NULL);
    pthread_join(rt2, NULL);
    pthread_join(nrt, NULL);
}

int main(int argc, char **argv)
{
    struct sched_param rtparam = { .sched_priority = 42 };
    pthread_attr_t rtattr, regattr;
    sigset_t mask, oldmask;
    mlockall(MCL_CURRENT | MCL_FUTURE);
    sigemptyset(&mask);
    sigaddset(&mask, SIGINT);
    signal(SIGINT, cleanup_upon_sig);
    sigaddset(&mask, SIGTERM);
    signal(SIGTERM, cleanup_upon_sig);
    sigaddset(&mask, SIGHUP);
    signal(SIGHUP, cleanup_upon_sig);
    pthread_sigmask(SIG_BLOCK, &mask, &oldmask);

    /*
    * This is a real-time compatible printf() package from
    * Xenomai’s RT Development Kit (RTDK), that does NOT cause
    * any transition to secondary (i.e. non real-time) mode when
    * writing output.
    */
    rt_print_auto_init(1);

    pthread_attr_init(&artattr);
    pthread_attr_setdetachstate(&artattr, PTHREAD_CREATE_JOINABLE);
    pthread_attr_setinheritsched(&artattr, PTHREAD_EXPLICIT_SCHED);
    pthread_attr_setschedpolicy(&artattr, SCHED_FIFO);
    pthread_attr_setschedparam(&artattr, &rtparam);

    /* Both real-time threads have the same attribute set. */
    errno = pthread_create(&rtl, &artattr, &realtime_thread1, NULL);
    if (errno)
        fail("pthread_create");
    errno = pthread_create(&rt2, &artattr, &realtime_thread2, NULL);
```
if (errno)
    fail("pthread_create");

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");
sigsuspend(&oldmask);
return 0;
}

8.12 xddp-stream.c

/*
 * 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.
 *
 * 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 |
 * => bind socket to port 0 v
 * => 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() | |
 * => echo traffic back to RT domain via write() +--+
 *
 * See Makefile in this directory for build directives.
 *
 * NOTE: XDDP is a replacement for the legacy RT_PIPE interface available from the native skin until Xenomai 3.
 */
#include <sys/mman.h>
#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 <rtdk.h>
#include <rtipc.h>

pthread_t rt, nrt;
/*
   * The following array contains the names of the songs that are going to be
   * played by the realtime_thread function. Each song is represented by a
   * message that is sent to the NRT endpoint via the proxy.
   * 
   * The output is artificially scattered in separate one-byte sendings, to
   * illustrate the use of MSG_MORE.
   */
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"
};
rt_printf("%s: sent (scattered) %d-bytes message, \"%.s\"\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");
rt_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, &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;
}

static void cleanup_upon_sig(int sig)
{
pthread_cancel(rt);
pthread_cancel(nrt);
signal(sig, SIG_DFL);
pthread_join(rt, NULL);
pthread_join(nrt, NULL);
}

int main(int argc, char **argv)
{
struct sched_param rtparam = { .sched_priority = 42 };
pthread_attr_t rtattr, regattr;
sigset_t mask, oldmask;
mlockall(MCL_CURRENT | MCL_FUTURE);
sigemptyset(&mask);
sigaddset(&mask, SIGINT);
signal(SIGINT, cleanup_upon_sig);
sigaddset(&mask, SIGTERM);
signal(SIGTERM, cleanup_upon_sig);
sigaddset(&mask, SIGHUP);
signal(SIGHUP, cleanup_upon_sig);
pthread_sigmask(SIG_BLOCK, &mask, &oldmask);

"/*
 * This is a real-time compatible printf() package from
 * Xenomai’s RT Development Kit (RTDK), that does NOT cause
 * any transition to secondary (i.e. non real-time) mode when
 * writing output.
 */
rt_print_auto_init(1);

Generated on Thu Jul 7 2016 13:25:19 for Xenomai RTDM skin API by Doxygen
pthread_attr_init(&rtattr);
pthread_attr_setdetachstate(&rtattr, PTHREAD_CREATE_JOINABLE);
pthread_attr_setinheritsched(&rtattr, PTHREAD_EXPLICIT_SCHED);
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_setinheritsched(&regattr, PTHREAD_EXPLICIT_SCHED);
pthread_attr_setschedpolicy(&regattr, SCHED_OTHER);
errno = pthread_create(&nrt, &regattr, &regular_thread, NULL);
if (errno)
    fail("pthread_create");
sigsuspend(&oldmask);

}
Index

BUFP_BUFSZ
Real-time IPC protocols, 47
BUFP_LABEL
Real-time IPC protocols, 48
bind__AF_RTIPC
Real-time IPC protocols, 54

CAN Devices, 9
CAN_BITTIME_BTR, 28
CAN_BITTIME_STD, 28
CAN_BITTIME_TYPE, 28
CAN_CTRLMODE_3_SAMPLES, 17
CAN_CTRLMODE_LISTENONLY, 17
CAN_CTRLMODE_LOOPBACK, 18
CAN_ERR_LOSTARB_UNSPEC, 18
CAN_MODE, 28
CAN_MODE_SLEEP, 28
CAN_MODE_START, 28
CAN_MODE_STOP, 28
CAN_RAW_ERR_FILTER, 28
CAN_RAW_FILTER, 28
CAN_RAW_LOOPBACK, 19
CAN_RAW_RECV_OWN_MSGS, 20
CAN_STATE, 28
CAN_STATE_BUS_OFF, 29
CAN_STATE_ERROR_ACTIVE, 29
CAN_STATE_ERROR_PASSIVE, 29
CAN_STATE_ERROR_WARNING, 29
CAN_STATE_SCANNING_BAUDRATE, 29
CAN_STATE_SLEEPING, 29
CAN_STATE_STOPPED, 29

can_filter_t, 28
can_frame_t, 28
RTCAN_RTOC_RCV_TIMEOUT, 20
RTCAN_RTOC_SND_TIMEOUT, 20
RTCAN_RTOC_TAKETIMEOUT, 21
SIOGC_CAN_BAUDRATE, 22
SIOGC_CAN_CTRLMODE, 22
SIOGC_CAN_CUSTOM_BITTIME, 23
SIOGC_CAN_STATE, 23
SIOGC_IFINDEX, 24
SIOSCAN_BAUDRATE, 24
SIOSCAN_CTRLMODE, 25
SIOSCAN_CUSTOM_BITTIME, 26
SIOSCAN_MODE, 26
SOL_CAN_RAW, 27

CAN_BITTIME_BTR
CAN Devices, 28
CAN_BITTIME_STD
CAN Devices, 28

CAN_BITTIME_TYPE
CAN Devices, 28
CAN_CTRLMODE_3_SAMPLES
CAN Devices, 17
CAN_CTRLMODE_LISTENONLY
CAN Devices, 17
CAN_CTRLMODE_LOOPBACK
CAN Devices, 18
CAN_ERR_LOSTARB_UNSPEC
CAN Devices, 18
CAN_MODE
CAN Devices, 28
CAN_MODE_SLEEP
CAN Devices, 28
CAN_MODE_START
CAN Devices, 28
CAN_MODE_STOP
CAN Devices, 28
CAN_RAW_ERR_FILTER
CAN Devices, 18
CAN_RAW_FILTER
CAN Devices, 18
CAN_RAW_LOOPBACK
CAN Devices, 19
CAN_RAW_RECV_OWN_MSGS
CAN Devices, 20
CAN_STATE
CAN Devices, 28
CAN_STATE_BUS_OFF
CAN Devices, 29
CAN_STATE_ERROR_ACTIVE
CAN Devices, 29
CAN_STATE_ERROR_PASSIVE
CAN Devices, 29
CAN_STATE_ERROR_WARNING
CAN Devices, 29
CAN_STATE_SCANNING_BAUDRATE
CAN Devices, 29
CAN_STATE_SLEEPING
CAN Devices, 29
CAN_STATE_STOPPED
CAN Devices, 29
can_bittime, 149
can_bittime_btr, 149
can_bittime_std, 150
can_filter, 150
can_id, 151
can_mask, 151
can_filter_t
can_frame, 151
  can_id, 152
  can_frame_t
    CAN Devices, 28
    can_id
      can_filter, 151
      can_frame, 152
    can_ifindex
      sockaddr_can, 159
    can_mask
      can_filter, 151
Clock Services, 92
  rtdm_clock_read, 92
  rtdm_clock_read_monotonic, 92
close__AF_RTIPC
  Real-time IPC protocols, 55
close_rt
  rtdm_operations, 156
connect__AF_RTIPC
  Real-time IPC protocols, 55
Device Profiles, 146
  RTIOC_DEVICE_INFO, 147
  RTIOC_PURGE, 147
Device Registration Services, 82
  RTDM_CLOSING, 84
  RTDM_CREATED_IN_NRT, 84
  RTDM_DEVICE_TYPE_MASK, 84
  RTDM_EXCLUSIVE, 84
  RTDM_NAMED_DEVICE, 84
  RTDM_PROTOCOL_DEVICE, 84
  rtdm_close_handler_t, 84
  rtdm_context_to_private, 89
  rtdm_dev_register, 89
  rtdm_dev_unregister, 90
  rtdm_ioctl_handler_t, 90
  rtdm_open_handler_t, 85
  rtdm_private_to_context, 90
  rtdm_read_handler_t, 85
  rtdm_recvmsg_handler_t, 86
  rtdm_select_bind_handler_t, 86
  rtdm_sendmsg_handler_t, 87
  rtdm_socket_handler_t, 87
  rtdm_write_handler_t, 87
Driver Development API, 91
EXPORT_SYMBOL_GPL
  Synchronisation Services, 112–117
getpeername__AF_RTIPC
  Real-time IPC protocols, 57
getsockname__AF_RTIPC
  Real-time IPC protocols, 57
getsockopt__AF_RTIPC
  Real-time IPC protocols, 57
IDDP_LABEL
  Real-time IPC protocols, 48
IDDP_POOLSZ
  Real-time IPC protocols, 49
IPCPROTO_BUFP
  Real-time IPC protocols, 54
IPCPROTO_IDDP
  Real-time IPC protocols, 54
IPCPROTO_IPC
  Real-time IPC protocols, 54
IPCPROTO_XDDP
  Real-time IPC protocols, 54
include/rtdm/rtcan.h, 161
include/rtdm/rtdm.h, 167
include/rtdm/rtdm_driver.h, 169
include/rtdm/rtipc.h, 174
include/rtdm/rtserial.h, 176
include/rtdm/rttesting.h, 180
Inter-Driver API, 73
  rtdm_accept, 74
  rtdm_bind, 74
  rtdm_close, 74
  rtdm_connect, 74
  rtdm_context_get, 74
  rtdm_context_lock, 76
  rtdm_context_put, 76
  rtdm_context_unlock, 77
  rtdm_getpeername, 77
  rtdm_getsockname, 77
  rtdm_getsockopt, 78
  rtdm_ioctl, 78
  rtdm_listen, 78
  rtdm_open, 78
  rtdm_read, 78
  rtdm_recv, 79
  rtdm_recvfrom, 79
  rtdm_recvmsg, 79
  rtdm_select_bind, 79
  rtdm_send, 80
  rtdm_sendmsg, 80
  rtdm_sendto, 80
  rtdm_setsockopt, 80
  rtdm_shutdown, 81
  rtdm_socket, 81
  rtdm_write, 81
Interrupt Management Services, 129
  rtdm_irq_disable, 130
  rtdm_irq_enable, 131
  rtdm_irq_free, 131
  rtdm_irq_get_arg, 129
  rtdm_irq_handler_t, 130
  rtdm_irq_request, 132
ksrc/skins/rtdm/core.c, 186
ksrc/skins/rtdm/device.c, 181
ksrc/skins/rtdm/drvlib.c, 182
ksrc/skins/rtdm/module.c, 186
label
  rtpc_port_label, 157
nanosecs_abs_t
Real-Time Driver Model, 31
nanosecs_rel_t
Real-Time Driver Model, 31
Non-Real-Time Signalling Services, 133
 rtdm_nrtsig_destroy, 133
 rtdm_nrtsig_handler_t, 133
 rtdm_nrtsig_init, 134
 rtdm_nrtsig_pend, 134
open_rt
 rtdm_device, 154
RTCAN_RTIIOC_RCV_TIMEOUT
CAN Devices, 20
RTCAN_RTIIOC_SND_TIMEOUT
CAN Devices, 20
RTCAN_RTIIOC_TAKE_TIMESTAMP
CAN Devices, 21
RTDM_CLOSING
Device Registration Services, 84
RTDM_CREATED_IN_NRT
Device Registration Services, 84
RTDM_DEVICE_TYPE_MASK
Device Registration Services, 84
RTDM_EXCLUSIVE
Device Registration Services, 84
RTDM_EXECUTE_ATOMICALLY
Synchronisation Services, 108
RTDM_NAMED_DEVICE
Device Registration Services, 84
RTDM_PROTOCOL_DEVICE
Device Registration Services, 84
RTDM_SELECTTYPE_EXCEPT
Synchronisation Services, 112
RTDM_SELECTTYPE_READ
Synchronisation Services, 112
RTDM_SELECTTYPE_WRITE
Synchronisation Services, 112
RTDM_TIMEOUT_INFINITE
Real-Time Driver Model, 30
RTDM_TIMEOUT_NONE
Real-Time Driver Model, 30
RTDM_TIMERMODE_ABSOLUTE
Timer Services, 102
RTDM_TIMERMODE_REALTIME
Timer Services, 102
RTDM_TIMERMODE_RELATIVE
Timer Services, 102
RTIOC_DEVICE_INFO
Device Profiles, 147
RTIOC_PURGE
Device Profiles, 147
RTSER_RTIIOC_BREAK_CTL
Serial Devices, 66
RTSER_RTIIOC_GET_CONFIG
Serial Devices, 67
RTSER_RTIIOC_GET_CONTROL
Serial Devices, 67
RTSER_RTIIOC_GET_STATUS
Serial Devices, 68
RTSER_RTIIOC_SET_CONFIG
Serial Devices, 68
RTSER_RTIIOC_SET_CONTROL
Serial Devices, 69
RTSER_RTIIOC_WAIT_EVENT
Serial Devices, 69
Real-Time Driver Model, 30
nanosecs_abs_t, 31
nanosecs_rel_t, 31
RTDM_TIMEOUT_INFINITE, 30
RTDM_TIMEOUT_NONE, 30
Real-time IPC protocols, 45
BUFP_BUFSZ, 47
BUFP_LABEL, 48
bind__AF_RTIPC, 54
close__AF_RTIPC, 55
connect__AF_RTIPC, 55
getpeername__AF_RTIPC, 57
getsockopt__AF_RTIPC, 57
recvmsg__AF_RTIPC, 58
rt_dev_accept
User API, 33
rt_dev_bind
User API, 33
rt_dev_close
User API, 33
rt_dev_connect
User API, 35
rt_dev_getpeername
User API, 35
rt_dev_getsockopt
User API, 36
rt_dev_getsockopt
rtdm_task_proc_t
  Task Services, 95

rtdm_task_set_period
  Task Services, 98

rtdm_task_set_priority
  Task Services, 98

rtdm_task_sleep
  Task Services, 99

rtdm_task_sleep_abs
  Task Services, 99

rtdm_task_sleep_until
  Task Services, 99

rtdm_task_unblock
  Task Services, 100

rtdm_timer_destroy
  Timer Services, 103

rtdm_timer_handler_t
  Timer Services, 102

rtdm_timer_init
  Timer Services, 103

rtdm_timer_mode
  Timer Services, 102

rtdm_timer_start
  Timer Services, 103

rtdm_timer_start_in_handler
  Timer Services, 104

rtdm_timer_stop
  Timer Services, 104

rtdm_timer_stop_in_handler
  Timer Services, 105

rtdm_toseq_init
  Synchronisation Services, 127

rtdm_write
  Inter-Drive API, 81

rtdm_write_handler_t
  Device Registration Services, 87

rtipc_port_label, 156
  label, 157

rtser_config, 156

rtser_event, 158

rtser_status, 158

SIOCGLOCK
  CAN Devices, 22

SIOCGLOCKCTRLMODE
  CAN Devices, 22

SIOCGLOCKCUSTOMBITTIME
  CAN Devices, 23

SIOCGLOCKSTATE
  CAN Devices, 23

SIOCGLOCKINDEX
  CAN Devices, 24

SIOCGLOCKCANBAUDRATE
  CAN Devices, 24

SIOCGLOCKCANCTRLMODE
  CAN Devices, 25

SIOCGLOCKCUSTOMBITTIME
  CAN Devices, 26

SIOCSCANTIMEOUTMODE
  CAN Devices, 26

SO_RCVTIMEO
  Real-time IPC protocols, 50

SO_SNDCOMPRESSOR
  Real-time IPC protocols, 50

SOL_CAN_RAW
  CAN Devices, 27

sendmsg__AF_RTIPC
  Real-time IPC protocols, 58

Serial Devices, 61
  RTSER_RTIOC_BREAK_CTL, 66
  RTSER_RTIOC_GET_CONFIG, 67
  RTSER_RTIOC_GET_CONTROL, 67
  RTSER_RTIOC_GET_STATUS, 68
  RTSER_RTIOC_SET_CONFIG, 68
  RTSER_RTIOC_SET_CONTROL, 69
  RTSER_RTIOC_WAIT_EVENT, 69

setsockopt__AF_RTIPC
  Real-time IPC protocols, 59

sipc_port
  sockaddr_ipc, 160

sockaddr_can, 159
  can_ifindex, 159

sockaddr_ipc, 159
  sipc_port, 160

socket__AF_RTIPC
  Real-time IPC protocols, 59

socket_rt
  rtdm_device, 154

Synchronisation Services, 106
  EXPORT_SYMBOL_GPL, 112–117
  RTDM_EXECUTE_ATOMICALLY, 108
  RTDM_SELECTTYPE_EXCEPT, 112
  RTDM_SELECTTYPE_READ, 112
  RTDM_SELECTTYPE_WRITE, 112
  rtdm_event_clear, 117
  rtdm_event_destroy, 118
  rtdm_event_init, 118
  rtdm_event_pulse, 118
  rtdm_event_select_bind, 119
  rtdm_event_signal, 119
  rtdm_event_timedwait, 120
  rtdm_event_wait, 120
  rtdm_lock_get, 109
  rtdm_lock_get_irqsave, 109
  rtdm_lock_init, 110
  rtdm_lock_irqrestore, 110
  rtdm_lock_irqsave, 110
  rtdm_lock_put, 111
  rtdm_lock_put_irqrestore, 111
  rtdm_mutex_destroy, 122
  rtdm_mutex_init, 122
  rtdm_mutex_lock, 123
  rtdm_mutex_timedlock, 123
  rtdm_mutex_unlock, 124
  rtdm_select_bind, 124
rtom_selecttype, 112
rtom_sem_destroy, 125
rtom_sem_down, 125
rtom_sem_init, 125
rtom_sem_select_bind, 126
rtom_sem_timeddown, 126
rtom_sem_up, 127
rtom_toseq_init, 127

Task Services, 94
rtom_task_busy_sleep, 95
rtom_task_current, 95
rtom_task_destroy, 95
rtom_task_join_nrt, 97
rtom_task_proc_t, 95
rtom_task_set_period, 98
rtom_task_set_priority, 98
rtom_task_sleep, 99
rtom_task_sleep_abs, 99
rtom_task_sleep_until, 99
rtom_task_unblock, 100
rtom_task_wait_period, 100

Testing Devices, 71
Timer Services, 102
RTDM_TIMERMODE_ABSOLUTE, 102
RTDM_TIMERMODE_REALTIME, 102
RTDM_TIMERMODE_RELATIVE, 102
rtom_timer_destroy, 103
rtom_timer_handler_t, 102
rtom_timer_init, 103
rtom_timer_mode, 102
rtom_timer_start, 103
rtom_timer_start_in_handler, 104
rtom_timer_stop, 104
rtom_timer_stop_in_handler, 105

Utility Services, 135
rtom_copy_from_user, 135
rtom_copy_to_user, 136
rtom_free, 137
rtom_in_rt_context, 137
rtom_iomap_to_user, 137
rtom_malloc, 138
rtom_mmap_to_user, 139
rtom_munmap, 140
rtom_printk, 140
rtom_printk_ratelimited, 141
rtom_ratelimit, 141
rtom_read_user_ok, 142
rtom_rt_capable, 142
rtom_rw_user_ok, 143
rtom_safe_copy_from_user, 143
rtom_safe_copy_to_user, 144
rtom_strncpy_from_user, 145

XDDP_BUFSZ
Real-time IPC protocols, 50
XDDP_EVTDOWN
Real-time IPC protocols, 51
XDDP_EVTIN
Real-time IPC protocols, 51
XDDP_EVTNOBUF
Real-time IPC protocols, 51
XDDP_EVTOUT
Real-time IPC protocols, 51
XDDP_LABEL
Real-time IPC protocols, 51
XDDP_MONITOR
Real-time IPC protocols, 52
XDDP_POOLSZ
Real-time IPC protocols, 53

User API, 32
rt_dev_accept, 33
rt_dev_bind, 33
rt_dev_close, 33
rt_dev_connect, 35
rt_dev_getpeername, 35
rt_dev_getsockname, 36
rt_dev_getsockopt, 36
rt_dev_ioctl, 37
rt_dev_listen, 37
rt_dev_open, 37
rt_dev_read, 39
rt_dev_recv, 39
rt_dev_recvfrom, 40
rt_dev_recvmsg, 40
rt_dev_send, 41
rt_dev_sendmsg, 41
rt_dev_sendto, 42
rt_dev_setsockopt, 42
rt_devShutdown, 43
rt_dev_socket, 43
rt_dev_write, 43