Xenomai RTDM skin API Reference Manual

2.3.50

Generated by Doxygen 1.4.6

Thu Feb 22 20:33:30 2007

Contents

Chapter 1

Xenomai RTDM skin API Module Index

1.1 Xenomai RTDM skin API Modules

Here is a list of all modules:

Chapter 2

Xenomai RTDM skin API Hierarchical Index

2.1 Xenomai RTDM skin API Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Xenomai RTDM skin API Data Structure Index

3.1 Xenomai RTDM skin API Data Structures

Here are the data structures with brief descriptions:

Chapter 4

Xenomai RTDM skin API File Index

4.1 Xenomai RTDM skin API File List

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

Chapter 5

Xenomai RTDM skin API Module Documentation

5.1 CAN Devices

Collaboration diagram for CAN Devices:

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)
Specific return values:

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

Close
Blocking calls to any of the Send or Receive functions will be unblocked when the socket is closed and return with an error.
Environments: non-RT (RT optional)
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_PROTO_RAW)
  - Option CAN_RAW_FILTER : CAN filter list
  - Option CAN_RAW_ERR_FILTER : CAN error mask
  - Option CAN_RAW_LOOPBACK : CAN TX loopback to local sockets

Environments: non-RT (RT optional)
Specific return values: see links to options above.
Recv, Recvfrom, Recvmsg
These functions receive CAN messages from a socket. Only one message per call can be received, so only one buffer with the correct length must be passed. For `SOCK_RAW`, this is the size of struct can_frame.
Unlike a call to one of the Send functions, a Recv function will not return with an error if an interface is down (due to bus-off or setting of stop mode) or in sleep mode. Moreover, in such a case there may still be some CAN messages in the socket buffer which could be read out successfully.
It is possible to receive a high precision timestamp with every CAN message. The condition is a former instruction to the socket via RTCAN_RTIOC_TAKE_TIMESTAMP. The timestamp will be copied to the `msg_control` buffer of `struct msghdr` if it points to a valid memory location with size of nanosecs_abs_t. If this is a NULL pointer the timestamp will be discarded silently.
Note: A `msg_controllen` of `0` upon completion of the function call indicates that no timestamp is available for that message.
Supported Flags [in]:

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

Supported Flags [out]: none
Environments: RT (non-RT optional)
Specific return values:

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

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

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

Environments: RT (non-RT optional)
Specific return values:

- Non-negative value equal to given buffer size (Indicating the successful completion of the function call. See also note.)
- -EOPNOTSUPP (MSG_OOB flag is not supported.)
- -EINVAL (Unsupported flag detected *or:* Invalid length of socket address *or:* Invalid address family *or:* Data length code of CAN frame not between 0 and 15 *or:* CAN standard frame has got an ID not between 0 and 2031)
- -EMSGSIZE (Zero or more than one buffer passed or invalid size of buffer)
- -EFAULT (It was not possible to access user space memory area at one of the specified addresses.)
- -ENXIO (Invalid CAN interface index - `0` is not allowed here - or socket not bound or rather bound to all interfaces.)
- -ENETDOWN (Controller is bus-off or in stopped state.)
- -ECOMM (Controller is sleeping)

- -EAGAIN (Cannot transmit without blocking but a non-blocking call was requested.)
- -EINTR (Operation was interrupted explicitly or by signal)
- -EBADF (Socket was closed.)
- -ETIMEDOUT (Timeout)

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

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.

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.

CAN controller modes

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

- #define CAN_CTRLMODE_LISTENONLY 0x1

  Listen-Only mode.

- #define CAN_CTRLMODE_LOOPBACK 0x2

  Loopback mode.

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.

IOCTLs

CAN device IOCTLs

- #define SIOCGIFINDEX _IOWR(RTIOC_TYPE_CAN, 0x00, struct ifreq)

  Get CAN interface index by name.

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

  Set baud rate.

- #define SIOCGCANBAUDRATE _IOWR(RTIOC_TYPE_CAN, 0x02, struct ifreq)

  Get baud rate.

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

  Set custom bit time parameter.

- #define SIOCGCANCUSTOMBITTIME _IOWR(RTIOC_TYPE_CAN, 0x04, struct ifreq)

  Get custum bit-time parameters.

- #define SIOCSCANMODE _IOW(RTIOC_TYPE_CAN, 0x05, struct ifreq)

  Set operation mode of CAN controller.

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

  Get current state of CAN controller.

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

  Set special controller modes.

- #define SIOCGCANCTRLMODE _IOWR(RTIOC_TYPE_CAN, 0x08, struct ifreq)

  Get special controller modes.

- #define RTCAN_RTIOC_TAKE_TIMESTAMP _IOW(RTIOC_TYPE_CAN, 0x09, int)

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

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

  Specify a reception timeout for a socket.

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

  Specify a transmission timeout for a socket.

Error mask

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

- #define CAN_ERR_TX_TIMEOUT 0x00000001U

  TX timeout (netdevice driver).

- #define CAN_ERR_LOSTARB 0x00000002U

  Lost arbitration (see data[0]).

- #define CAN_ERR_CRTL 0x00000004U

  Controller problems (see data[1]).

- #define CAN_ERR_PROT 0x00000008U

  Protocol violations (see data[2], data[3]).

- #define CAN_ERR_TRX 0x00000010U

  Transceiver status (see data[4]).

- #define CAN_ERR_ACK 0x00000020U

  Received no ACK on transmission.

- #define CAN_ERR_BUSOFF 0x00000040U

  Bus off.

- #define CAN_ERR_BUSERROR 0x00000080U

  Bus error (may flood!).

- #define CAN_ERR_RESTARTED 0x00000100U

  Controller restarted.

- #define CAN_ERR_MASK 0x1FFFFFFFU

  Omit EFF, RTR, ERR flags.

Arbitration lost error

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

- #define CAN_ERR_LOSTARB_UNSPEC 0x00

  unspecified else bit number in bitstream

Controller problems

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

- #define CAN_ERR_CRTL_UNSPEC 0x00

  unspecified

- #define CAN_ERR_CRTL_RX_OVERFLOW 0x01

  RX buffer overflow.

- #define CAN_ERR_CRTL_TX_OVERFLOW 0x02

  TX buffer overflow.

- #define CAN_ERR_CRTL_RX_WARNING 0x04

  reached warning level for RX errors

- #define CAN_ERR_CRTL_TX_WARNING 0x08

  reached warning level for TX errors

- #define CAN_ERR_CRTL_RX_PASSIVE 0x10

  reached passive level for RX errors

- #define CAN_ERR_CRTL_TX_PASSIVE 0x20

  reached passive level for TX errors

Protocol error type

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

- #define CAN_ERR_PROT_UNSPEC 0x00

  unspecified

- #define CAN_ERR_PROT_BIT 0x01

  single bit error

- #define CAN_ERR_PROT_FORM 0x02

  frame format error

- #define CAN_ERR_PROT_STUFF 0x04

  bit stuffing error

- #define CAN_ERR_PROT_BIT0 0x08

  unable to send dominant bit

- #define CAN_ERR_PROT_BIT1 0x10

  unable to send recessive bit

- #define CAN_ERR_PROT_OVERLOAD 0x20

  bus overload

- #define CAN_ERR_PROT_ACTIVE 0x40

  active error announcement

- #define CAN_ERR_PROT_TX 0x80

  error occured on transmission

Protocol error location

Error in the data[3] field of struct can_frame.

- #define CAN_ERR_PROT_LOC_UNSPEC 0x00

  unspecified

- #define CAN_ERR_PROT_LOC_SOF 0x03

  start of frame

- #define CAN_ERR_PROT_LOC_ID28_21 0x02

  ID bits 28 - 21 (SFF: 10 - 3).

- #define CAN_ERR_PROT_LOC_ID20_18 0x06

  ID bits 20 - 18 (SFF: 2 - 0).

- #define CAN_ERR_PROT_LOC_SRTR 0x04

  substitute RTR (SFF: RTR)

- #define CAN_ERR_PROT_LOC_IDE 0x05

  identifier extension

- #define CAN_ERR_PROT_LOC_ID17_13 0x07

  ID bits 17-13.

- #define CAN_ERR_PROT_LOC_ID12_05 0x0F

  ID bits 12-5.

- #define CAN_ERR_PROT_LOC_ID04_00 0x0E

  ID bits 4-0.

- #define CAN_ERR_PROT_LOC_RTR 0x0C

  RTR.

- #define CAN_ERR_PROT_LOC_RES1 0x0D

  reserved bit 1

- #define CAN_ERR_PROT_LOC_RES0 0x09

  reserved bit 0

- #define CAN_ERR_PROT_LOC_DLC 0x0B

  data length code

- #define CAN_ERR_PROT_LOC_DATA 0x0A

  data section

- #define CAN_ERR_PROT_LOC_CRC_SEQ 0x08

  CRC sequence.

- #define CAN_ERR_PROT_LOC_CRC_DEL 0x18

CRC delimiter.

- #define CAN_ERR_PROT_LOC_ACK 0x19

  ACK slot.

- #define CAN_ERR_PROT_LOC_ACK_DEL 0x1B

  ACK delimiter.

- #define CAN_ERR_PROT_LOC_EOF 0x1A

  end of frame

- #define CAN_ERR_PROT_LOC_INTERM 0x12

  intermission

Protocol error location

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

- #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

CAN protocols

Possible protocols for PF_CAN protocol family

- enum CAN_PROTO { CAN_PROTO_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 states

States a CAN controller can be in.

- enum CAN_STATE {

  CAN_STATE_ACTIVE = 0, CAN_STATE_BUS_WARNING, CAN_STATE_BUS_PASSIVE, CAN_STATE_BUS_OFF,

  CAN_STATE_SCANNING_BAUDRATE, CAN_STATE_STOPPED, CAN_STATE_SLEEPING }

Defines

- #define AF_CAN 29

  CAN address family.

- #define PF_CAN AF_CAN

  CAN protocol family.

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 can_filter can_filter_t

  Filter for reception of CAN messages.

- typedef can_frame can_frame_t

  Raw CAN frame.

Enumerations

- enum CAN_BITTIME_TYPE { CAN_BITTIME_STD, CAN_BITTIME_BTR }

  Supported CAN bit-time types.

5.1.2 Define Documentation

5.1.2.1 #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 decided if error frames are send to this socket in case of error condidtions. The error frames are marked with the CAN_ERR_FLAG of CAN_xxx_FLAG and must be handled by the application properly. A detailed description of the error can be found in the `can_id` and the `data` fields of struct can_frame (see Errors for futher details).

Parameters:

← ***level*** **SOL_CAN_RAW**

← ***optname*** **CAN_RAW_ERR_FILTER**

← ***optval*** Pointer to error mask of type can_err_mask_t.

← ***optlen*** Size of error mask: sizeof(can_err_mask_t).

Environments: non-RT (RT optional)

Specific return values:

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

Examples:

rtcanrecv.c.

5.1.2.2 #define CAN_RAW_FILTER 0x1

CAN filter definition.

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

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

Parameters:

← *level* **SOL_CAN_RAW**

← *optname* **CAN_RAW_FILTER**

← *optval* Pointer to array of struct can_filter.

← *optlen* Size of filter list: count * sizeof(struct can_filter).

Environments: non-RT (RT optional)

Specific return values:

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

Examples:

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

5.1.2.3 #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:

← *level* **SOL_CAN_RAW**

← *optname* **CAN_RAW_LOOPBACK**

← *optval* Pointer to integer value.

← *optlen* Size of int: sizeof(int).

Environments: non-RT (RT optional)

Specific return values:

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

- -EOPNOTSUPP (not supported, check RT-Socket-CAN kernel parameters).

Examples:
rtcansend.c.

5.1.2.4 #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:
← *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.

5.1.2.5 #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:

← *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:

rtcansend.c.

5.1.2.6 #define RTCAN_RTIOC_TAKE_TIMESTAMP _IOW(RTIOC_TYPE_CAN, 0x09, int)

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

A newly created socket takes no timestamps by default.

Parameters:

← *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.

5.1.2.7 #define SIOCGCANBAUDRATE _IOWR(RTIOC_TYPE_CAN, 0x02, struct ifreq)

Get baud rate.

Parameters:

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

Returns:

0 on success, otherwise:

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

Environments:

This service can be called from:

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

Rescheduling: never.

5.1.2.8 #define SIOCGCANCTRLMODE _IOWR(RTIOC_TYPE_CAN, 0x08, struct ifreq)

Get special controller modes.

Parameters:

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

Returns:

0 on success, otherwise:

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

Environments:

This service can be called from:

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

Rescheduling: possible.

5.1.2.9 #define SIOCGCANCUSTOMBITTIME _IOWR(RTIOC_TYPE_CAN, 0x04, struct ifreq)

Get custum bit-time parameters.

Parameters:

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

Returns:

0 on success, otherwise:

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

Environments:

This service can be called from:

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

Rescheduling: never.

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

Get current state of CAN controller.

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

Parameters:

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

Returns:

0 on success, otherwise:

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

Environments:

This service can be called from:

- Kernel module initialization/cleanup code

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

Rescheduling: possible.

5.1.2.11 #define SIOCGIFINDEX _IOWR(RTIOC_TYPE_CAN, 0x00, struct ifreq)

Get CAN interface index by name.

Parameters:

↔ *arg* Pointer to interface request structure buffer (`struct ifreq` from linux/if.h). If `ifr_name` holds a valid CAN interface name `ifr_ifindex` will be filled with the corresponding interface index.

Returns:

0 on success, otherwise:

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

Environments:

This service can be called from:

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

Rescheduling: never.

Examples:

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

5.1.2.12 #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:

← *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:

rtcanconfig.c.

5.1.2.13 #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:

← *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 specia controlelr modes is a configuration task. It should be done deliberately or otherwise CAN messages will likely be lost.

Rescheduling: possible.

Examples:

rtcanconfig.c.

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

Set custom bit time parameter.

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

Parameters:

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

Returns:

0 on success, otherwise:

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

Environments:

This service can be called from:

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

Note:

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

Rescheduling: possible.

Examples:

rtcanconfig.c.

5.1.2.15 #define SIOCSCANMODE _IOW(RTIOC_TYPE_CAN, 0x05, struct ifreq)

Set operation mode of CAN controller.

See CAN controller modes for available modes.

Parameters:

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

Returns:

0 on success, otherwise:

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

Environments:

This service can be called from:

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

Note:

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

Rescheduling: possible.

Examples:

rtcanconfig.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 `can_mask` and then compared to `can_id`. This also includes the CAN_EFF_FLAG and CAN_RTR_FLAG of CAN_xxx_FLAG. If this comparison is true, the message will be received by the socket. The logic can be inverted with the `can_id` flag CAN_INV_FILTER :

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

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

5.1.3.2 typedef struct can_frame can_frame_t

Raw CAN frame.

Central structure for receiving and sending CAN frames.

5.1.4 Enumeration Type Documentation

5.1.4.1 enum CAN_BITTIME_TYPE

Supported CAN bit-time types.

Enumerator:

CAN_BITTIME_STD Standard bit-time definition according to Bosch.

CAN_BITTIME_BTR Hardware-specific BTR bit-time definition.

5.1.4.2 enum CAN_MODE

Enumerator:

CAN_MODE_STOP Set controller in Stop mode (no reception / transmission possible).

CAN_MODE_START Set controller into normal operation.

Coming from stopped mode or bus off, the controller begins with no errors in CAN_STATE_ACTIVE.

CAN_MODE_SLEEP Set controller into Sleep mode.

This is only possible if the controller is not stopped or bus-off.

Notice that sleep mode will only be entered when there is no bus activity. If the controller detects bus activity while "sleeping" it will go into operating mode again.

To actively leave sleep mode again trigger `CAN_MODE_START`.

5.1.4.3 enum CAN_PROTO

Enumerator:

CAN_PROTO_RAW Raw protocol of `PF_CAN`, applicable to socket type `SOCK_RAW`.

5.1.4.4 enum CAN_STATE

Enumerator:

CAN_STATE_ACTIVE CAN controller is error active.

CAN_STATE_BUS_WARNING CAN controller is error active, warning level is reached.

CAN_STATE_BUS_PASSIVE CAN controller is error passive.

CAN_STATE_BUS_OFF CAN controller went into Bus Off.

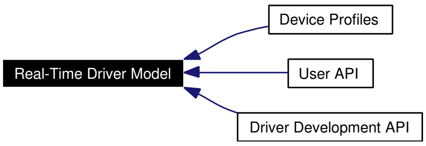
CAN_STATE_SCANNING_BAUDRATE CAN controller is scanning to get the baudrate.

CAN_STATE_STOPPED CAN controller is in stopped mode.

CAN_STATE_SLEEPING CAN controller is in Sleep mode.

5.2 Real-Time Driver Model

Collaboration diagram for Real-Time Driver Model:



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: 6

Modules

- User API
- Driver Development API
- Device Profiles

API Versioning

- #define RTDM_API_VER 6

  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.

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.

5.2.2 Typedef Documentation

5.2.2.1 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.

5.2.2.2 typedef int64_t nanosecs_rel_t

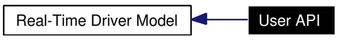
RTDM type for representing relative intervals.

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, and rtcansend.c.

5.3 User API

Collaboration diagram for User API:



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.

Files

- file rtdm.h

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

Functions

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

  Open a device.

- int rt_dev_socket (int protocol_family, int socket_type, int protocol)

  Create a socket.

- int rt_dev_close (int fd)

  Close a device or socket.

- int rt_dev_ioctl (int fd, int request,...)

  Issue an IOCTL.

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

  Read from device.

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

  Write to device.

- ssize_t rt_dev_recvmsg (int fd, struct msghdr *msg, int flags)

  Receive message from socket.

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

  Receive message from socket.

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

  Receive message from socket.

- ssize_t rt_dev_sendmsg (int fd, const struct msghdr ∗msg, int flags)

  Transmit message to socket.

- ssize_t rt_dev_sendto (int fd, const void ∗buf, size_t len, int flags, const struct sockaddr ∗to, socklen_t tolen)

  Transmit message to socket.

- ssize_t rt_dev_send (int fd, const void ∗buf, size_t len, int flags)

  Transmit message to socket.

- int rt_dev_bind (int fd, const struct sockaddr ∗my_addr, socklen_t addrlen)

  Bind to local address.

- int rt_dev_connect (int fd, const struct sockaddr ∗serv_addr, socklen_t addrlen)

  Connect to remote address.

- int rt_dev_listen (int fd, int backlog)

  Listen for incomming connection requests.

- int rt_dev_accept (int fd, struct sockaddr ∗addr, socklen_t ∗addrlen)

  Accept a connection requests.

- int rt_dev_shutdown (int fd, int how)

  Shut down parts of a connection.

- int rt_dev_getsockopt (int fd, int level, int optname, void ∗optval, socklen_t ∗optlen)

  Get socket option.

- int rt_dev_setsockopt (int fd, int level, int optname, const void ∗optval, socklen_t optlen)

  Set socket option.

- int rt_dev_getsockname (int fd, struct sockaddr ∗name, socklen_t ∗namelen)

  Get local socket address.

- int rt_dev_getpeername (int fd, struct sockaddr ∗name, socklen_t ∗namelen)

  Get socket destination address.

5.3.2 Function Documentation

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

Accept a connection requests.

Parameters:

← *fd* File descriptor as returned by rt_dev_socket()

→ ***addr*** Buffer for remote address

↔ ***addrlen*** Address buffer size

Returns:

0 on success, otherwise negative error code

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

See also:

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

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

Bind to local address.

Parameters:

← ***fd*** File descriptor as returned by rt_dev_socket()

← ***my_addr*** Address buffer

← ***addrlen*** Address buffer size

Returns:

0 on success, otherwise negative error code

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

See also:

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

5.3.2.3 int rt_dev_close (int *fd*)

Close a device or socket.

Parameters:

← ***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.

Killing a real-time task that is blocked on some device operation can lead to stalled file descriptors. To avoid such scenarios, always close the device before explicitely terminating any real-time task which may use it. To cleanup a stalled file descriptor, send its number to the `open_fildes` /proc entry, e.g. via

```
#> echo 3 > /proc/xenomai/rtdm/open_fildes
```

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

See also:

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

5.3.2.4 int rt_dev_connect (int *fd*, const struct sockaddr * *serv_addr*, socklen_t *addrlen*)

Connect to remote address.

Parameters:

← ***fd*** File descriptor as returned by rt_dev_socket()

← ***serv_addr*** Address buffer

← ***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 int rt_dev_getpeername (int *fd*, struct sockaddr * *name*, socklen_t * *namelen*)

Get socket destination address.

Parameters:

← ***fd*** File descriptor as returned by rt_dev_socket()

→ ***name*** Address buffer

↔ ***namelen*** Address buffer size

Returns:

0 on success, otherwise negative error code

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

See also:

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

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

Get local socket address.

Parameters:

← *fd* File descriptor as returned by rt_dev_socket()

→ *name* Address buffer

↔ *namelen* Address buffer size

Returns:

0 on success, otherwise negative error code

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

See also:

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

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

Get socket option.

Parameters:

← *fd* File descriptor as returned by rt_dev_socket()

← *level* Addressed stack level

← *optname* Option name ID

→ *optval* Value buffer

↔ *optlen* Value buffer size

Returns:

0 on success, otherwise negative error code

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

See also:

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

5.3.2.8 int rt_dev_ioctl (int *fd*, int *request*, ...)

Issue an IOCTL.

Parameters:

← *fd* File descriptor as returned by rt_dev_open() or rt_dev_socket()

← *request* IOCTL code
... Optional third argument, depending on IOCTL function (`void` * or `unsigned long`)

Returns:
Positiv value on success, otherwise negative error code

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

See also:
`ioctl()` in IEEE Std 1003.1, `http://www.opengroup.org/onlinepubs/009695399`

5.3.2.9 int rt_dev_listen (int *fd*, int *backlog*)

Listen for incomming connection requests.

Parameters:
← *fd* File descriptor as returned by rt_dev_socket()
← *backlog* Maximum queue length

Returns:
0 on success, otherwise negative error code

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

See also:
`lsiten()` in IEEE Std 1003.1, `http://www.opengroup.org/onlinepubs/009695399`

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

Open a device.

Parameters:
← *path* Device name
← *oflag* Open flags
... Further parameters will be ignored.

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

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

See also:
`open()` in IEEE Std 1003.1, `http://www.opengroup.org/onlinepubs/009695399`

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

Read from device.

Parameters:

← *fd* File descriptor as returned by rt_dev_open()

→ *buf* Input buffer

← *nbyte* Number of bytes to read

Returns:

Number of bytes read, otherwise negative error code

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

See also:

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

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

Receive message from socket.

Parameters:

← *fd* File descriptor as returned by rt_dev_socket()

→ *buf* Message buffer

← *len* Message buffer size

← *flags* Message flags

Returns:

Number of bytes received, otherwise negative error code

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

See also:

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

5.3.2.13 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:

← *fd* File descriptor as returned by rt_dev_socket()

→ *buf* Message buffer

← *len* Message buffer size

← *flags* Message flags

→ *from* Buffer for message sender address

↔ *fromlen* Address buffer size

Returns:

Number of bytes received, otherwise negative error code

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

See also:

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

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

Receive message from socket.

Parameters:

← *fd* File descriptor as returned by rt_dev_socket()

↔ *msg* Message descriptor

← *flags* Message flags

Returns:

Number of bytes received, otherwise negative error code

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

See also:

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

5.3.2.15 ssize_t rt_dev_send (int *fd*, const void ∗ *buf*, size_t *len*, int *flags*)

Transmit message to socket.

Parameters:

← *fd* File descriptor as returned by rt_dev_socket()

← *buf* Message buffer

← *len* Message buffer size

← *flags* Message flags

Returns:

Number of bytes sent, otherwise negative error code

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

See also:

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

5.3.2.16 ssize_t rt_dev_sendmsg (int *fd*, const struct msghdr * *msg*, int *flags*)

Transmit message to socket.

Parameters:

← *fd* File descriptor as returned by rt_dev_socket()

← *msg* Message descriptor

← *flags* Message flags

Returns:

Number of bytes sent, otherwise negative error code

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

See also:

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

5.3.2.17 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:

← *fd* File descriptor as returned by rt_dev_socket()

← *buf* Message buffer

← *len* Message buffer size

← *flags* Message flags

← *to* Buffer for message destination address

← *tolen* Address buffer size

Returns:

Number of bytes sent, otherwise negative error code

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

See also:

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

5.3.2.18 int rt_dev_setsockopt (int *fd*, int *level*, int *optname*, const void ∗ *optval*, socklen_t *optlen*)

Set socket option.

Parameters:

← *fd* File descriptor as returned by rt_dev_socket()

← *level* Addressed stack level

← *optname* Option name ID

← *optval* Value buffer

← *optlen* Value buffer size

Returns:

0 on success, otherwise negative error code

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

See also:

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

5.3.2.19 int rt_dev_shutdown (int *fd*, int *how*)

Shut down parts of a connection.

Parameters:

← *fd* File descriptor as returned by rt_dev_socket()

← *how* Specifies the part to be shut down (`SHUT_xxx`)

Returns:

0 on success, otherwise negative error code

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

See also:

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

5.3.2.20 int rt_dev_socket (int *protocol_family*, int *socket_type*, int *protocol*)

Create a socket.

Parameters:

← ***protocol_family*** Protocol family (`PF_xxx`)

← ***socket_type*** Socket type (`SOCK_xxx`)

← ***protocol*** Protocol ID, 0 for default

Returns:

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

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

See also:

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

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

Write to device.

Parameters:

← ***fd*** File descriptor as returned by rt_dev_open()

← ***buf*** Output buffer

← ***nbyte*** Number of bytes to write

Returns:

Number of bytes written, otherwise negative error code

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

See also:

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

5.4 Serial Devices

Collaboration diagram for Serial Devices:

5.4.1 Detailed Description

This is the common interface a RTDM-compliant serial device has to provide. Feel free to comment on this profile via the Xenomai mailing list (`Xenomai-core@gna.org`) or directly to the author (`jan.kiszka@web.de`).

Profile Revision: 1

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

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.

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
- #define RTSER_1_5_STOPB 0x01

  valid only in combination with 5 data bits

- #define **RTSER_2_STOPB** 0x01
- #define **RTSER_DEF_STOPB** RTSER_1_STOPB

RTSER_xxx_HAND

Handshake mechanisms

- #define **RTSER_NO_HAND** 0x00
- #define **RTSER_RTSCTS_HAND** 0x01
- #define **RTSER_DEF_HAND** RTSER_NO_HAND

RTSER_FIFO_xxx

Reception FIFO interrupt threshold

- #define **RTSER_FIFO_DEPTH_1** 0x00
- #define **RTSER_FIFO_DEPTH_4** 0x40
- #define **RTSER_FIFO_DEPTH_8** 0x80
- #define **RTSER_FIFO_DEPTH_14** 0xC0
- #define **RTSER_DEF_FIFO_DEPTH** RTSER_FIFO_DEPTH_1

RTSER_TIMEOUT_xxx

Special timeout values, see also RTDM_TIMEOUT_xxx

- #define **RTSER_TIMEOUT_INFINITE** RTDM_TIMEOUT_INFINITE
- #define **RTSER_TIMEOUT_NONE** RTDM_TIMEOUT_NONE
- #define **RTSER_DEF_TIMEOUT** RTDM_TIMEOUT_INFINITE

RTSER_xxx_TIMESTAMP_HISTORY

Timestamp history control

- #define **RTSER_RX_TIMESTAMP_HISTORY** 0x01
- #define **RTSER_DEF_TIMESTAMP_HISTORY** 0x00

RTSER_EVENT_xxx

Events bits

- #define **RTSER_EVENT_RXPEND** 0x01
- #define **RTSER_EVENT_ERRPEND** 0x02
- #define **RTSER_EVENT_MODEMHI** 0x04
- #define **RTSER_EVENT_MODEMLO** 0x08
- #define **RTSER_DEF_EVENT_MASK** 0x00

RTSER_SET_xxx

Configuration mask bits

- #define **RTSER_SET_BAUD** 0x0001
- #define **RTSER_SET_PARITY** 0x0002
- #define **RTSER_SET_DATA_BITS** 0x0004
- #define **RTSER_SET_STOP_BITS** 0x0008
- #define **RTSER_SET_HANDSHAKE** 0x0010
- #define **RTSER_SET_FIFO_DEPTH** 0x0020
- #define **RTSER_SET_TIMEOUT_RX** 0x0100
- #define **RTSER_SET_TIMEOUT_TX** 0x0200
- #define **RTSER_SET_TIMEOUT_EVENT** 0x0400
- #define **RTSER_SET_TIMESTAMP_HISTORY** 0x0800
- #define **RTSER_SET_EVENT_MASK** 0x1000

RTSER_LSR_xxx

Line status bits

- #define **RTSER_LSR_DATA** 0x01
- #define **RTSER_LSR_OVERRUN_ERR** 0x02
- #define **RTSER_LSR_PARITY_ERR** 0x04
- #define **RTSER_LSR_FRAMING_ERR** 0x08
- #define **RTSER_LSR_BREAK_IND** 0x10
- #define **RTSER_LSR_THR_EMTPY** 0x20
- #define **RTSER_LSR_TRANSM_EMPTY** 0x40
- #define **RTSER_LSR_FIFO_ERR** 0x80
- #define **RTSER_SOFT_OVERRUN_ERR** 0x0100

RTSER_MSR_xxx

Modem status bits

- #define **RTSER_MSR_DCTS** 0x01
- #define **RTSER_MSR_DDSR** 0x02
- #define **RTSER_MSR_TERI** 0x04
- #define **RTSER_MSR_DDCD** 0x08
- #define **RTSER_MSR_CTS** 0x10
- #define **RTSER_MSR_DSR** 0x20
- #define **RTSER_MSR_RI** 0x40
- #define **RTSER_MSR_DCD** 0x80

RTSER_MCR_xxx

Modem control bits

- #define **RTSER_MCR_DTR** 0x01
- #define **RTSER_MCR_RTS** 0x02
- #define **RTSER_MCR_OUT1** 0x04
- #define **RTSER_MCR_OUT2** 0x08
- #define **RTSER_MCR_LOOP** 0x10

IOCTLs

Serial device IOCTLs

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

  Get serial device configuration.

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

  Set serial device configuration.

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

  Get serial device status.

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

  Get serial device's modem contol register.

- #define RTSER_RTIOC_SET_CONTROL _IOW(RTIOC_TYPE_SERIAL, 0x04, int)

  Set serial device's modem contol register.

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

  Wait on serial device events according to previously set mask.

Typedefs

- typedef rtser_config rtser_config_t

  Serial device configuration.

- typedef rtser_status rtser_status_t

  Serial device status.

- typedef rtser_event rtser_event_t

  Additional information about serial device events.

5.4.2 Define Documentation

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

Get serial device configuration.

Parameters:

→ *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.

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

Get serial device's modem contol register.

Parameters:

→ *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.

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

Get serial device status.

Parameters:

→ *arg* Pointer to status buffer (struct rtser_status)

Returns:

0 on success, otherwise negative error code

Environments:

This service can be called from:

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

Note:

The error states `RTSER_LSR_OVERRUN_ERR`, `RTSER_LSR_PARITY_ERR`, `RTSER_LSR_FRAMING_ERR`, and `RTSER_SOFT_OVERRUN_ERR` that may have occured during previous read accesses to the device will be saved for being reported via this IOCTL. Upon return from `RTSER_RTIOC_GET_STATUS`, the saved state will be cleared.

Rescheduling: never.

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

Set serial device configuration.

Parameters:

← *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.

5.4.2.5 #define RTSER_RTIOC_SET_CONTROL _IOW(RTIOC_TYPE_SERIAL, 0x04, int)

Set serial device's modem contol register.

Parameters:
← *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.

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

Wait on serial device events according to previously set mask.

Parameters:
→ *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.

5.5 Testing Devices

Collaboration diagram for Testing Devices:

5.5.1 Detailed Description

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

Profile Revision: 1

Device Characteristics
Device Flags: `RTDM_NAMED_DEVICE`
Device Name: `"rttest<N>"`, N >= 0
Device Class: `RTDM_CLASS_TESTING`

Supported Operations
Open
Environments: non-RT (RT optional)
Specific return values: none
Close
Environments: non-RT (RT optional)
Specific return values: none
IOCTL
Mandatory Environments: see IOCTLs below
Specific return values: see IOCTLs below

Files

- file rttesting.h

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

IOCTLs

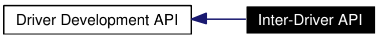
Testing device IOCTLs

- #define **RTTST_RTIOC_INTERM_BENCH_RES** _IOWR(RTIOC_TYPE_TESTING, 0x00, struct rttst_interm_bench_res)
- #define **RTTST_RTIOC_TMBENCH_START** _IOW(RTIOC_TYPE_TESTING, 0x10, struct rttst_tmbench_config)
- #define **RTTST_RTIOC_TMBENCH_STOP** _IOWR(RTIOC_TYPE_TESTING, 0x11, struct rttst_overall_bench_res)
- #define **RTTST_RTIOC_IRQBENCH_START** _IOW(RTIOC_TYPE_TESTING, 0x20, struct rttst_irqbench_config)
- #define **RTTST_RTIOC_IRQBENCH_STOP** _IO(RTIOC_TYPE_TESTING, 0x21)

- #define **RTTST_RTIOC_IRQBENCH_GET_STATS** _IOR(RTIOC_TYPE_TESTING, 0x22, struct rttst_irqbench_stats)
- #define **RTTST_RTIOC_IRQBENCH_WAIT_IRQ** _IO(RTIOC_TYPE_TESTING, 0x23)
- #define **RTTST_RTIOC_IRQBENCH_REPLY_IRQ** _IO(RTIOC_TYPE_TESTING, 0x24)
- #define **RTTST_RTIOC_SWTEST_SET_TASKS_COUNT** _IOW(RTIOC_TYPE_TESTING, 0x30, unsigned long)
- #define **RTTST_RTIOC_SWTEST_SET_CPU** _IOW(RTIOC_TYPE_TESTING, 0x31, unsigned long)
- #define **RTTST_RTIOC_SWTEST_REGISTER_UTASK** _IOW(RTIOC_TYPE_TESTING, 0x32, struct rttst_swtest_task)
- #define **RTTST_RTIOC_SWTEST_CREATE_KTASK** _IOWR(RTIOC_TYPE_TESTING, 0x33, struct rttst_swtest_task)
- #define **RTTST_RTIOC_SWTEST_PEND** _IOR(RTIOC_TYPE_TESTING, 0x34, struct rttst_swtest_task)
- #define **RTTST_RTIOC_SWTEST_SWITCH_TO** _IOR(RTIOC_TYPE_TESTING, 0x35, struct rttst_swtest_dir)
- #define **RTTST_RTIOC_SWTEST_GET_SWITCHES_COUNT** _IOR(RTIOC_TYPE_TESTING, 0x36, unsigned long)
- #define **RTTST_RTIOC_SWTEST_GET_LAST_ERROR** _IOR(RTIOC_TYPE_TESTING, 0x37, struct rttst_swtest_error)

5.6 Inter-Driver API

Collaboration diagram for Inter-Driver API:



Functions

- rtdm_dev_context ∗ rtdm_context_get (int fd)

  Resolve file descriptor to device context.

- 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.

- int rtdm_open (const char ∗path, int oflag,...)

  Open a device.

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

  Create a socket.

- int rtdm_close (int fd)

  Close a device or socket.

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

  Issue an IOCTL.

- ssize_t rtdm_read (int fd, void ∗buf, size_t nbyte)

  Read from device.

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

  Write to device.

- ssize_t rtdm_recvmsg (int fd, struct msghdr ∗msg, int flags)

  Receive message from socket.

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

  Receive message from socket.

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

  Receive message from socket.

- ssize_t rtdm_sendmsg (int fd, const struct msghdr ∗msg, int flags)

  Transmit message to socket.

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

  Transmit message to socket.

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

  Transmit message to socket.

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

  Bind to local address.

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

  Connect to remote address.

- int rtdm_listen (int fd, int backlog)

  Listen for incomming connection requests.

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

  Accept a connection requests.

- int rtdm_shutdown (int fd, int how)

  Shut down parts of a connection.

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

  Get socket option.

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

  Set socket option.

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

  Get local socket address.

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

  Get socket destination address.

5.6.1 Function Documentation

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

Accept a connection requests.

Refer to rt_dev_accept() for parameters and return values

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

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

Bind to local address.

Refer to rt_dev_bind() for parameters and return values

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

5.6.1.3 int rtdm_close (int *fd*)

Close a device or socket.

Refer to rt_dev_close() for parameters and return values

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

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

Connect to remote address.

Refer to rt_dev_connect() for parameters and return values

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

5.6.1.5 struct rtdm_dev_context* rtdm_context_get (int *fd*)

Resolve file descriptor to device context.

Parameters:

← *fd* File descriptor

Returns:

Pointer to associated device context, or NULL on error

Note:

The device context has to be unlocked using rtdm_context_unlock() 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.

5.6.1.6 void rtdm_context_lock (struct rtdm_dev_context * *context*)

Increment context reference counter.

Parameters:

← *context* Device context

Note:

rtdm_context_get() automatically increments the lock counter. You only need to call this function in special scenrios.

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.

5.6.1.7 void rtdm_context_unlock (struct rtdm_dev_context * *context*)

Decrement context reference counter.

Parameters:

← *context* Device context

Note:

Every successful call to rtdm_context_get() 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.

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

Get socket destination address.

Refer to rt_dev_getpeername() for parameters and return values

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

5.6.1.9 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.

5.6.1.10 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.

5.6.1.11 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.

5.6.1.12 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.

5.6.1.13 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.

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

Read from device.

Refer to rt_dev_read() for parameters and return values

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

5.6.1.15 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.

5.6.1.16 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.

5.6.1.17 ssize_t rtdm_recvmsg (int *fd*, struct msghdr * *msg*, int *flags*)

Receive message from socket.

Refer to rt_dev_recvmsg() for parameters and return values

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

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

Transmit message to socket.

Refer to rt_dev_send() for parameters and return values

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

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

Transmit message to socket.

Refer to rt_dev_sendmsg() for parameters and return values

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

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

Transmit message to socket.

Refer to rt_dev_sendto() for parameters and return values

Environments:

Depends on driver implementation, see Device Profiles.

Rescheduling: possible.

5.6.1.21 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.

5.6.1.22 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.

5.6.1.23 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.

5.6.1.24 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.

5.7 Device Registration Services

Collaboration diagram for Device Registration Services:

Data Structures

- struct rtdm_operations

    Device operations.

- struct rtdm_dev_context

    Device context.

- struct rtdm_device

    RTDM device.

Device Flags

Static flags describing a RTDM device

- #define RTDM_EXCLUSIVE 0x0001

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

- #define RTDM_NAMED_DEVICE 0x0010

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

- #define RTDM_PROTOCOL_DEVICE 0x0020

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

- #define RTDM_DEVICE_TYPE_MASK 0x00F0

    Mask selecting the device type.

Context Flags

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

- #define RTDM_CREATED_IN_NRT 0

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

- #define RTDM_CLOSING 1

    Set by RTDM when the device is being closed.

- #define RTDM_USER_CONTEXT_FLAG 8

    Lowest bit number the driver developer can use freely.

Driver Versioning

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

- #define RTDM_DEVICE_STRUCT_VER 4

  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 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 ∗arg)

  IOCTL handler.

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

  Read handler.

- typedef ssize_t(* rtdm_write_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info, const void *buf, size_t nbyte)

  Write handler.

- typedef ssize_t(* rtdm_recvmsg_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info, struct msghdr *msg, int flags)

  Receive message handler.

- typedef ssize_t(* rtdm_sendmsg_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info, const struct msghdr *msg, int flags)

  Transmit message handler.

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.

5.7.1 Typedef Documentation

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

Close handler.

Parameters:

← *context* Context structure associated with opened device instance

← *user_info* Opaque pointer to information about user mode caller, NULL if kernel mode call

Returns:

0 on success, otherwise negative error code

See also:

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

5.7.1.2 typedef int(* rtdm_ioctl_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info, unsigned int request, void *arg)

IOCTL handler.

Parameters:

← *context* Context structure associated with opened device instance

← *user_info* Opaque pointer to information about user mode caller, NULL if kernel mode call

← *request* Request number as passed by the user

↔ *arg* Request argument as passed by the user

Returns:

Positiv value on success, otherwise negative error code

See also:

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

5.7.1.3 typedef int(* rtdm_open_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info, int oflag)

Named device open handler.

Parameters:

← *context* Context structure associated with opened device instance

← *user_info* Opaque pointer to information about user mode caller, NULL if kernel mode call

← *oflag* Open flags as passed by the user

Returns:

0 on success, otherwise negative error code

See also:

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

5.7.1.4 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:

← *context* Context structure associated with opened device instance

← *user_info* Opaque pointer to information about user mode caller, NULL if kernel mode call

→ *buf* Input buffer as passed by the user

← *nbyte* Number of bytes the user requests to read

Returns:

On success, the number of bytes read, otherwise negative error code

See also:

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

5.7.1.5 typedef ssize_t(* rtdm_recvmsg_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info, struct msghdr *msg, int flags)

Receive message handler.

Parameters:

← *context* Context structure associated with opened device instance

← *user_info* Opaque pointer to information about user mode caller, NULL if kernel mode call

↔ *msg* Message descriptor as passed by the user, automatically mirrored to safe kernel memory in case of user mode call

← *flags* Message flags as passed by the user

Returns:

On success, the number of bytes received, otherwise negative error code

See also:

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

5.7.1.6 typedef ssize_t(* rtdm_sendmsg_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info, const struct msghdr *msg, int flags)

Transmit message handler.

Parameters:

← *context* Context structure associated with opened device instance

← *user_info* Opaque pointer to information about user mode caller, NULL if kernel mode call

← *msg* Message descriptor as passed by the user, automatically mirrored to safe kernel memory in case of user mode call

← *flags* Message flags as passed by the user

Returns:

On success, the number of bytes transmitted, otherwise negative error code

See also:

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

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

Socket creation handler for protocol devices.

Parameters:

← *context* Context structure associated with opened device instance

← *user_info* Opaque pointer to information about user mode caller, NULL if kernel mode call

← *protocol* Protocol number as passed by the user

Returns:

0 on success, otherwise negative error code

See also:

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

5.7.1.8 typedef ssize_t(* rtdm_write_handler_t)(struct rtdm_dev_context *context, rtdm_user_info_t *user_info, const void *buf, size_t nbyte)

Write handler.

Parameters:

← *context* Context structure associated with opened device instance

← *user_info* Opaque pointer to information about user mode caller, NULL if kernel mode call

← *buf* Output buffer as passed by the user

← *nbyte* Number of bytes the user requests to write

Returns:

On success, the number of bytes written, otherwise negative error code

See also:

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

5.7.2 Function Documentation

5.7.2.1 int rtdm_dev_register (struct rtdm_device * *device*)

Register a RTDM device.

Parameters:

← *device* Pointer to structure describing the new device.

Returns:

0 is returned upon success. Otherwise:

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

Environments:

This service can be called from:

- Kernel module initialization/cleanup code

Rescheduling: never.

5.7.2.2 int rtdm_dev_unregister (struct rtdm_device ∗ *device*, unsigned int *poll_delay*)

Unregisters a RTDM device.

Parameters:

← ***device*** Pointer to structure describing the device to be unregistered.

← ***poll_delay*** Polling delay in milliseconds to check repeatedly for open instances of *device*, or 0 for non-blocking mode.

Returns:

0 is returned upon success. Otherwise:

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

Environments:

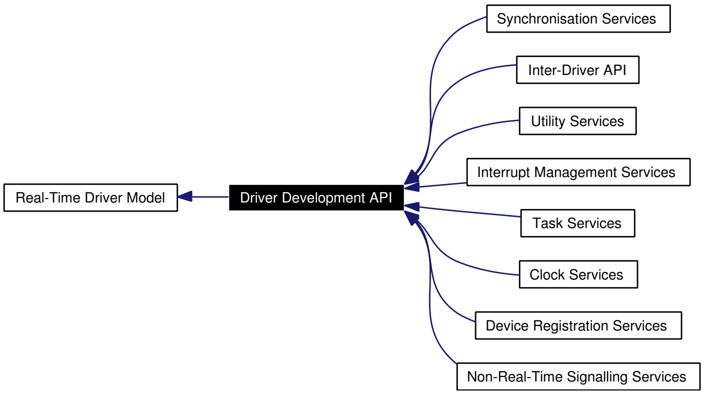
This service can be called from:

- Kernel module initialization/cleanup code

Rescheduling: never.

5.8 Driver Development API

Collaboration diagram for Driver Development API:



5.8.1 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.

Files

- file rtdm_driver.h

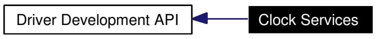
  Real-Time Driver Model for Xenomai, driver API header.

Modules

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

5.9 Clock Services

Collaboration diagram for Clock Services:



Functions

- nanosecs_abs_t rtdm_clock_read (void)

  Get system time.

5.9.1 Function Documentation

5.9.1.1 nanosecs_abs_t rtdm_clock_read (void)

Get system time.

Returns:

The system time in nanoseconds is returned

Note:

The resolution of this service depends on the system timer. In particular, if the system timer is running in periodic mode, the return value will be limited to multiples of the timer tick period.

The system timer may have to be started to obtain valid results. Whether this happens automatically (as on Xenomai) or is controlled by the application depends on the RTDM host environment.

Environments:

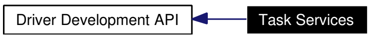
This service can be called from:

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

Rescheduling: never.

5.10 Task Services

Collaboration diagram for Task Services:



Task Priority Range

Maximum and minimum task priorities

- #define **RTDM_TASK_LOWEST_PRIORITY** XNCORE_LOW_PRIO
- #define **RTDM_TASK_HIGHEST_PRIORITY** XNCORE_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 void(* rtdm_task_proc_t)(void *arg)

  Real-time task procedure.

Functions

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

  Intialise and start a real-time task.

- void rtdm_task_destroy (rtdm_task_t *task)

  Destroy a real-time task.

- void rtdm_task_set_priority (rtdm_task_t *task, int priority)

  Adjust real-time task priority.

- int rtdm_task_set_period (rtdm_task_t *task, nanosecs_rel_t period)

  Adjust real-time task period.

- int rtdm_task_wait_period (void)

  Wait on next real-time task period.

- int rtdm_task_unblock (rtdm_task_t *task)

  Activate a blocked real-time task.

- rtdm_task_t * rtdm_task_current (void)

  Get current real-time task.

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

  Wait on a real-time task to terminate.

- 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.

- void rtdm_task_busy_sleep (nanosecs_rel_t delay)

  Busy-wait a specified amount of time.

5.10.1 Typedef Documentation

5.10.1.1 typedef void(* rtdm_task_proc_t)(void *arg)

Real-time task procedure.

Parameters:

↔ ***arg*** argument as passed to rtdm_task_init()

5.10.2 Function Documentation

5.10.2.1 void rtdm_task_busy_sleep (nanosecs_rel_t *delay*)

Busy-wait a specified amount of time.

Parameters:

← ***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).

5.10.2.2 rtdm_task_t* rtdm_task_current (void)

Get current real-time task.

Returns:

Pointer to task handle

Environments:

This service can be called from:

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

Rescheduling: never.

5.10.2.3 void rtdm_task_destroy (rtdm_task_t * *task*)

Destroy a real-time task.

Parameters:

↔ ***task*** Task handle as returned by rtdm_task_init()

Note:

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

Environments:

This service can be called from:

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

Rescheduling: never.

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

Intialise and start a real-time task.

After initialising a task, the task handle remains valid and can be passed to RTDM services until either rtdm_task_destroy() or rtdm_task_join_nrt() was invoked.

Parameters:

↔ ***task*** Task handle

← ***name*** Optional task name

← ***task_proc*** Procedure to be executed by the task

← *arg* Custom argument passed to `task_proc()` on entry

← *priority* Priority of the task, see also Task Priority Range

← *period* Period in nanosecons of a cyclic task, 0 for non-cyclic mode

Returns:

0 on success, otherwise negative error code

Environments:

This service can be called from:

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

Rescheduling: possible.

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

Wait on a real-time task to terminate.

Parameters:

↔ *task* Task handle as returned by rtdm_task_init()

← *poll_delay* Polling delay in milliseconds

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.

5.10.2.6 int rtdm_task_set_period (rtdm_task_t * *task*, nanosecs_rel_t *period*)

Adjust real-time task period.

Parameters:

↔ *task* Task handle as returned by rtdm_task_init()

← *period* New period in nanosecons of a cyclic task, 0 for non-cyclic mode

Environments:

This service can be called from:

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

Rescheduling: possible.

5.10.2.7 void rtdm_task_set_priority (rtdm_task_t * *task*, int *priority*)

Adjust real-time task priority.

Parameters:

↔ ***task*** Task handle as returned by rtdm_task_init()

← ***priority*** New priority of the task, see also Task Priority Range

Environments:

This service can be called from:

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

Rescheduling: possible.

5.10.2.8 int rtdm_task_sleep (nanosecs_rel_t *delay*)

Sleep a specified amount of time.

Parameters:

← ***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 explicitely 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.

5.10.2.9 int rtdm_task_sleep_until (nanosecs_abs_t *wakeup_time*)

Sleep until a specified absolute time.

Parameters:

← ***wakeup_time*** Absolute timeout in nanoseconds

Returns:

0 on success, otherwise:

- -EINTR is returned if calling task has been unblock by a signal or explicitely 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.

5.10.2.10 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.

5.10.2.11 int rtdm_task_wait_period (void)

Wait on next real-time task period.

Returns:

0 on success, otherwise:

- -EINVAL is returned if calling task is not in periodic mode.

- -ETIMEDOUT is returned if a timer overrun occurred, which indicates that a previous release point has been missed by the calling task.

Environments:

This service can be called from:

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

Rescheduling: always, unless a timer overrun occured.

5.11 Synchronisation Services

Collaboration diagram for Synchronisation Services:

Global Lock across Scheduler Invocation

- #define RTDM_EXECUTE_ATOMICALLY(code_block)

  Execute code block atomically.

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) rthal_spin_unlock(lock)

  Release lock without preemption restoration.

- #define rtdm_lock_get_irqsave(lock, context) rthal_spin_lock_irqsave(lock, context)

  Acquire lock and disable preemption.

- #define rtdm_lock_put_irqrestore(lock, context) rthal_spin_unlock_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.

Timeout Sequence Management

- void rtdm_toseq_init (rtdm_toseq_t *timeout_seq, nanosecs_rel_t timeout)

  Initialise a timeout sequence.

Event Services

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

  Initialise an event.

- void rtdm_event_destroy (rtdm_event_t *event)

  Destroy an event.

- void rtdm_event_pulse (rtdm_event_t *event)

  Signal an event occurrence to currently listening waiters.

- void rtdm_event_signal (rtdm_event_t *event)

  Signal an event occurrence.

- int rtdm_event_wait (rtdm_event_t *event)

  Wait on event occurrence.

- int rtdm_event_timedwait (rtdm_event_t *event, nanosecs_rel_t timeout, rtdm_toseq_t *timeout_seq)

  Wait on event occurrence with timeout.

- void rtdm_event_clear (rtdm_event_t *event)

  Clear event state.

Semaphore Services

- void rtdm_sem_init (rtdm_sem_t *sem, unsigned long value)

  Initialise a semaphore.

- void rtdm_sem_destroy (rtdm_sem_t *sem)

  Destroy a semaphore.

- int rtdm_sem_down (rtdm_sem_t *sem)

  Decrement a semaphore.

- int rtdm_sem_timeddown (rtdm_sem_t *sem, nanosecs_rel_t timeout, rtdm_toseq_t *timeout_seq)

  Decrement a semaphore with timeout.

- void rtdm_sem_up (rtdm_sem_t *sem)

  Increment a semaphore.

Mutex Services

- void rtdm_mutex_init (rtdm_mutex_t *mutex)

  Initialise a mutex.

- void rtdm_mutex_destroy (rtdm_mutex_t *mutex)

  Destroy a mutex.

- void rtdm_mutex_unlock (rtdm_mutex_t *mutex)

  Release a mutex.

- int rtdm_mutex_lock (rtdm_mutex_t *mutex)

  Request a mutex.

- int rtdm_mutex_timedlock (rtdm_mutex_t *mutex, nanosecs_rel_t timeout, rtdm_toseq_t *timeout_seq)

  Request a mutex with timeout.

5.11.1 Define Documentation

5.11.1.1 #define RTDM_EXECUTE_ATOMICALLY(code_block)

Value:

```
{                                                                            \
    spl_t   s;                                                               \
                                                                             \
    xnlock_get_irqsave(&nklock, s);                                          \
    code_block;                                                              \
    xnlock_put_irqrestore(&nklock, s);                                       \
}
```

Execute code block atomically.

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

Parameters:

code_block Commands to be executed atomically

Note:

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

Environments:

This service can be called from:

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

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

5.11.1.2 #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.

5.11.1.3 #define rtdm_lock_get_irqsave(lock, context) rthal_spin_lock_irqsave(lock, context)

Acquire lock and disable preemption.

Parameters:

lock Address of lock variable

context name of local variable to store the context in

Environments:

This service can be called from:

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

Rescheduling: never.

5.11.1.4 #define rtdm_lock_init(lock) rthal_spin_lock_init(lock)

Dynamic lock initialisation.

Parameters:
lock Address of lock variable

Environments:

This service can be called from:

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

Rescheduling: never.

5.11.1.5 #define rtdm_lock_irqrestore(context) rthal_local_irq_restore(context)

Restore preemption state.

Parameters:
context name of local variable which stored the context

Environments:

This service can be called from:

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

Rescheduling: possible.

5.11.1.6 #define rtdm_lock_irqsave(context) rthal_local_irq_save(context)

Disable preemption locally.

Parameters:
context name of local variable to store the context in

Environments:

This service can be called from:

- Kernel module initialization/cleanup code

- Interrupt service routine
- Kernel-based task
- User-space task (RT, non-RT)

Rescheduling: never.

5.11.1.7 #define rtdm_lock_put(lock) rthal_spin_unlock(lock)

Release lock without preemption restoration.

Parameters:

lock Address of lock variable

Environments:

This service can be called from:

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

Rescheduling: never.

5.11.1.8 #define rtdm_lock_put_irqrestore(lock, context) rthal_spin_unlock_irqrestore(lock, context)

Release lock and restore preemption state.

Parameters:

lock Address of lock variable

context name of local variable which stored the context

Environments:

This service can be called from:

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

Rescheduling: possible.

5.11.2 Function Documentation

5.11.2.1 void rtdm_event_clear (rtdm_event_t * *event*)

Clear event state.

Parameters:

↔ *event* Event handle as returned by rtdm_event_init()

Environments:

This service can be called from:

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

Rescheduling: never.

5.11.2.2 void rtdm_event_destroy (rtdm_event_t * *event*)

Destroy an event.

Parameters:

↔ *event* Event handle as returned by rtdm_event_init()

Environments:

This service can be called from:

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

Rescheduling: possible.

5.11.2.3 void rtdm_event_init (rtdm_event_t * *event*, unsigned long *pending*)

Initialise an event.

Parameters:

↔ *event* Event handle

← *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.

5.11.2.4 void rtdm_event_pulse (rtdm_event_t * *event*)

Signal an event occurrence to currently listening waiters.

This function wakes up all current waiters of the given event, but it does not change the event state. Subsequently callers of rtdm_event_wait() or rtdm_event_timedwait() will therefore be blocked first.

Parameters:

↔ *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.

5.11.2.5 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:

↔ *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.

5.11.2.6 int rtdm_event_timedwait (rtdm_event_t * *event*, nanosecs_rel_t *timeout*, rtdm_toseq_t * *timeout_seq*)

Wait on event occurrence with timeout.

This function waits or tests for the occurence of the given event, taking the provided timeout into account. On successful return, the event is reset.

Parameters:

↔ ***event*** Event handle as returned by rtdm_event_init()

← ***timeout*** Relative timeout in nanoseconds, see RTDM_TIMEOUT_xxx for special values

↔ ***timeout_seq*** Handle of a timeout sequence as returned by rtdm_toseq_init() or rtdm_toseq_absinit(), or NULL

Returns:

0 on success, otherwise:

- -ETIMEDOUT is returned if the if the request has not been satisfied within the specified amount of time.
- -EINTR is returned if calling task has been unblock by a signal or explicitely via rtdm_task_unblock().
- -EIDRM is returned if *event* has been destroyed.
- -EPERM *may* be returned if an illegal invocation environment is detected.

Environments:

This service can be called from:

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

Rescheduling: possible.

5.11.2.7 int rtdm_event_wait (rtdm_event_t * *event*)

Wait on event occurrence.

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

Parameters:

↔ ***event*** Event handle as returned by rtdm_event_init()

Returns:

0 on success, otherwise:

- -EINTR is returned if calling task has been unblock by a signal or explicitely via rtdm_task_unblock().

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

Environments:

This service can be called from:

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

Rescheduling: possible.

5.11.2.8 void rtdm_mutex_destroy (rtdm_mutex_t * *mutex*)

Destroy a mutex.

Parameters:

↔ ***mutex*** Mutex handle as returned by rtdm_mutex_init()

Environments:

This service can be called from:

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

Rescheduling: possible.

5.11.2.9 void rtdm_mutex_init (rtdm_mutex_t * *mutex*)

Initialise a mutex.

This function initalises a basic mutex with priority inversion protection. "Basic", as it does not allow a mutex owner to recursively lock the same mutex again.

Parameters:

↔ ***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.

5.11.2.10 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:

↔ ***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.

5.11.2.11 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:

↔ ***mutex*** Mutex handle as returned by rtdm_mutex_init()

← ***timeout*** Relative timeout in nanoseconds, see RTDM_TIMEOUT_xxx for special values

↔ ***timeout_seq*** Handle of a timeout sequence as returned by rtdm_toseq_init() or rtdm_toseq_absinit(), or NULL

Returns:

0 on success, otherwise:

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

Environments:

This service can be called from:

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

Rescheduling: possible.

5.11.2.12 void rtdm_mutex_unlock (rtdm_mutex_t * *mutex*)

Release a mutex.

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

Parameters:

↔ ***mutex*** Mutex handle as returned by rtdm_mutex_init()

Environments:

This service can be called from:

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

Rescheduling: possible.

5.11.2.13 void rtdm_sem_destroy (rtdm_sem_t * *sem*)

Destroy a semaphore.

Parameters:

↔ ***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.

5.11.2.14 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:

↔ ***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 explicitely 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.

5.11.2.15 void rtdm_sem_init (rtdm_sem_t * *sem*, unsigned long *value*)

Initialise a semaphore.

Parameters:

↔ ***sem*** Semaphore handle

← ***value*** Initial value of the semaphore

Environments:

This service can be called from:

- Kernel module initialization/cleanup code

- Kernel-based task

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

Rescheduling: never.

5.11.2.16 int rtdm_sem_timeddown (rtdm_sem_t ∗ *sem*, nanosecs_rel_t *timeout*, rtdm_toseq_t ∗ *timeout_seq*)

Decrement a semaphore with timeout.

This function tries to decrement the given semphore's value if it is positive on entry. If not, the caller is blocked unless non-blocking operation was selected.

Parameters:

↔ ***sem*** Semaphore handle as returned by rtdm_sem_init()

← ***timeout*** Relative timeout in nanoseconds, see RTDM_TIMEOUT_xxx for special values

↔ ***timeout_seq*** Handle of a timeout sequence as returned by rtdm_toseq_init() or rtdm_toseq_absinit(), or NULL

Returns:

0 on success, otherwise:

- -ETIMEDOUT is returned if the if the request has not been satisfied within the specified amount of time.
- -EWOULDBLOCK is returned if *timeout* is negative and the semaphore value is currently not positive.
- -EINTR is returned if calling task has been unblock by a signal or explicitely 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.

5.11.2.17 void rtdm_sem_up (rtdm_sem_t ∗ *sem*)

Increment a semaphore.

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

Parameters:

↔ ***sem*** Semaphore handle as returned by rtdm_sem_init()

Environments:

This service can be called from:

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

Rescheduling: possible.

5.11.2.18 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:

↔ ***timeout_seq*** Timeout sequence handle

← ***timeout*** Relative timeout in nanoseconds, see RTDM_TIMEOUT_xxx for special values

Application Scenario:

```
int device_service_routine(...)
{
    rtdm_toseq_t timeout_seq;
    ...

    rtdm_toseq_init(&timeout_seq, timeout);
    ...
    while (received < requested) {
        ret = rtdm_event_timedwait(&data_available, timeout, &timeout_seq);
        if (ret < 0)    // including -ETIMEDOUT
            break;

        // receive some data
        ...
    }
    ...
}
```

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

Environments:

This service can be called from:

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

Rescheduling: never.

5.12 Interrupt Management Services

Collaboration diagram for Interrupt Management Services:

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.

Defines

- #define rtdm_irq_get_arg(irq_handle, type) ((type ∗)irq_handle → cookie)

  Retrieve IRQ handler argument.

Typedefs

- typedef int(∗ rtdm_irq_handler_t)(rtdm_irq_t ∗irq_handle)

  Interrupt handler.

Functions

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

  Register an interrupt handler.

- int rtdm_irq_free (rtdm_irq_t *irq_handle)

  Release an interrupt handler.

- int rtdm_irq_enable (rtdm_irq_t *irq_handle)

  Enable interrupt line.

- int rtdm_irq_disable (rtdm_irq_t *irq_handle)

  Disable interrupt line.

5.12.1 Define Documentation

5.12.1.1 #define rtdm_irq_get_arg(irq_handle, type) ((type *)irq_handle → cookie)

Retrieve IRQ handler argument.

Parameters:

irq_handle IRQ handle

type Type of the pointer to return

Returns:

The argument pointer registered on rtdm_irq_request() is returned, type-casted to the specified *type*.

Environments:

This service can be called from:

- Interrupt service routine

Rescheduling: never.

5.12.2 Typedef Documentation

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

Interrupt handler.

Parameters:

← *irq_handle* IRQ handle as returned by rtdm_irq_request()

Returns:

0 or a combination of RTDM_IRQ_xxx flags

5.12.3 Function Documentation

5.12.3.1 int rtdm_irq_disable (rtdm_irq_t * *irq_handle*)

Disable interrupt line.

Parameters:

↔ *irq_handle* IRQ handle as returned by rtdm_irq_request()

Returns:

0 on success, otherwise negative error code

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.

5.12.3.2 int rtdm_irq_enable (rtdm_irq_t * *irq_handle*)

Enable interrupt line.

Parameters:

↔ *irq_handle* IRQ handle as returned by rtdm_irq_request()

Returns:

0 on success, otherwise negative error code

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.

5.12.3.3 int rtdm_irq_free (rtdm_irq_t * *irq_handle*)

Release an interrupt handler.

Parameters:

↔ *irq_handle* IRQ handle as returned by rtdm_irq_request()

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.

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

Register an interrupt handler.

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

Parameters:

↔ ***irq_handle*** IRQ handle

← ***irq_no*** Line number of the addressed IRQ

← ***handler*** Interrupt handler

← ***flags*** Registration flags, see RTDM_IRQTYPE_xxx for details

← ***device_name*** Device name to show up in real-time IRQ lists

← ***arg*** Pointer to be passed to the interrupt handler on invocation

Returns:

0 on success, otherwise:

- -EINVAL is returned if an invalid parameter was passed.
- -EBUSY is returned if the specified IRQ line is already in use.

Environments:

This service can be called from:

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

Rescheduling: never.

5.13 Non-Real-Time Signalling Services

Collaboration diagram for Non-Real-Time Signalling Services:

5.13.1 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.

Typedefs

- typedef void(∗ rtdm_nrtsig_handler_t)(rtdm_nrtsig_t nrt_sig)

  Non-real-time signal handler.

Functions

- int rtdm_nrtsig_init (rtdm_nrtsig_t ∗nrt_sig, rtdm_nrtsig_handler_t handler)

  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.

5.13.2 Typedef Documentation

5.13.2.1 typedef void(∗ rtdm_nrtsig_handler_t)(rtdm_nrtsig_t nrt_sig)

Non-real-time signal handler.

Parameters:

← ***nrt_sig*** signal handle as returned by rtdm_nrtsig_init()

Note:

The signal handler will run in soft-IRQ context of the non-real-time subsystem. Note the implications of this context, e.g. no invocation of blocking operations.

5.13.3 Function Documentation

5.13.3.1 void rtdm_nrtsig_destroy (rtdm_nrtsig_t * *nrt_sig*)

Release a non-realtime signal handler.

Parameters:
↔ ***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.

5.13.3.2 int rtdm_nrtsig_init (rtdm_nrtsig_t * *nrt_sig*, rtdm_nrtsig_handler_t *handler*)

Register a non-real-time signal handler.

Parameters:
↔ ***nrt_sig*** Signal handle
← ***handler*** Non-real-time signal handler

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.

5.13.3.3 void rtdm_nrtsig_pend (rtdm_nrtsig_t * *nrt_sig*)

Trigger non-real-time signal.

Parameters:
↔ ***nrt_sig*** Signal handle

Environments:

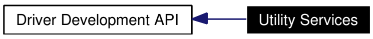
This service can be called from:

- 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.

5.14 Utility Services

Collaboration diagram for Utility Services:



Functions

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

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

- int rtdm_iomap_to_user (rtdm_user_info_t *user_info, unsigned long src_addr, size_t len, int prot, void **pptr, struct vm_operations_struct *vm_ops, void *vm_private_data)

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

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

  Unmap a user memory range.

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

  Real-time safe message printing on kernel console.

- void * rtdm_malloc (size_t size)

  Allocate memory block in real-time context.

- void rtdm_free (void *ptr)

  Release real-time memory block.

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

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

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

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

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

  Copy user-space memory block to specified buffer.

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

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

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

  Copy specified buffer to user-space memory block.

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

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

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

  Copy user-space string to specified buffer.

- int rtdm_in_rt_context (void)

  Test if running in a real-time task.

5.14.1 Function Documentation

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

Copy user-space memory block to specified buffer.

Parameters:

← ***user_info*** User information pointer as passed to the invoked device operation handler

← ***dst*** Destination buffer address

← ***src*** Address of the user-space memory block

← ***size*** Size of the memory block

Returns:

0 on success, otherwise:

- -EFAULT is returned if an invalid memory area was accessed.

Note:

Before invoking this service, verify via rtdm_read_user_ok() that the provided user-space address can securely be accessed.

Environments:

This service can be called from:

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

Rescheduling: never.

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

Copy specified buffer to user-space memory block.

Parameters:

← ***user_info*** User information pointer as passed to the invoked device operation handler

← *dst* Address of the user-space memory block

← *src* Source buffer address

← *size* Size of the memory block

Returns:

0 on success, otherwise:

- -EFAULT is returned if an invalid memory area was accessed.

Note:

Before invoking this service, verify via rtdm_rw_user_ok() that the provided user-space address can securely be accessed.

Environments:

This service can be called from:

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

Rescheduling: never.

5.14.1.3 void rtdm_free (void * *ptr*)

Release real-time memory block.

Parameters:

← *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.

5.14.1.4 int rtdm_in_rt_context (void)

Test if running in a real-time task.

Returns:

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

Environments:

This service can be called from:

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

Rescheduling: never.

5.14.1.5 int rtdm_iomap_to_user (rtdm_user_info_t * *user_info*, unsigned long *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:

← ***user_info*** User information pointer as passed to the invoked device operation handler

← ***src_addr*** physical I/O address to be mapped

← ***len*** Length of the memory range

← ***prot*** Protection flags for the user's memory range, typically either PROT_READ or PROT_READ|PROT_WRITE

↔ ***pptr*** Address of a pointer containing the desired user address or NULL on entry and the finally assigned address on return

← ***vm_ops*** vm_operations to be executed on the vma_area of the user memory range or NULL

← ***vm_private_data*** Private data to be stored in the vma_area, primarily useful for vm_operation handlers

Returns:

0 on success, otherwise (most common values):

- -EINVAL is returned if an invalid start address, size, or destination address was passed.
- -ENOMEM is returned if there is insufficient free memory or the limit of memory mapping for the user process was reached.
- -EAGAIN is returned if too much memory has been already locked by the user process.
- -EPERM *may* be returned if an illegal invocation environment is detected.

Note:

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

Environments:

This service can be called from:

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

Rescheduling: possible.

5.14.1.6 void* rtdm_malloc (size_t *size*)

Allocate memory block in real-time context.

Parameters:

← ***size*** Requested size of the memory block

Returns:

The pointer to the allocated block is returned on success, NULL otherwise.

Environments:

This service can be called from:

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

Rescheduling: never.

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

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

Parameters:

← ***user_info*** User information pointer as passed to the invoked device operation handler

← ***src_addr*** Kernel virtual address to be mapped

← ***len*** Length of the memory range

← ***prot*** Protection flags for the user's memory range, typically either PROT_READ or PROT_READ|PROT_WRITE

↔ ***pptr*** Address of a pointer containing the desired user address or NULL on entry and the finally assigned address on return

← ***vm_ops*** vm_operations to be executed on the vma_area of the user memory range or NULL

← ***vm_private_data*** Private data to be stored in the vma_area, primarily useful for vm_-operation handlers

Returns:

0 on success, otherwise (most common values):

- -EINVAL is returned if an invalid start address, size, or destination address was passed.
- -ENOMEM is returned if there is insufficient free memory or the limit of memory mapping for the user process was reached.
- -EAGAIN is returned if too much memory has been already locked by the user process.
- -EPERM *may* be returned if an illegal invocation environment is detected.

Note:

This service only works on memory regions allocated via kmalloc() or vmalloc(). To map physical I/O memory to user-space use rtdm_iomap_to_user() instead.

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

Environments:

This service can be called from:

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

Rescheduling: possible.

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

Unmap a user memory range.

Parameters:

← ***user_info*** User information pointer as passed to rtdm_mmap_to_user() when requesting to map the memory range

← ***ptr*** User address or the memory range

← ***len*** Length of the memory range

Returns:

0 on success, otherwise:

- -EINVAL is returned if an invalid address or size was passed.
- -EPERM *may* be returned if an illegal invocation environment is detected.

Environments:

This service can be called from:

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

Rescheduling: possible.

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

Real-time safe message printing on kernel console.

Parameters:

← ***format*** Format string (conforming standard `printf()`)

... Arguments referred by *format*

Returns:

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

Environments:

This service can be called from:

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

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

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

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

Parameters:

← ***user_info*** User information pointer as passed to the invoked device operation handler

← ***ptr*** Address of the user-provided memory block

← ***size*** Size of the memory block

Returns:

Non-zero is return when it is safe to read from the specified memory block, 0 otherwise.

Environments:

This service can be called from:

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

Rescheduling: never.

5.14.1.11 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:

← ***user_info*** User information pointer as passed to the invoked device operation handler

← ***ptr*** Address of the user-provided memory block

← ***size*** Size of the memory block

Returns:

Non-zero is return when it is safe to read from or write to the specified memory block, 0 otherwise.

Environments:

This service can be called from:

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

Rescheduling: never.

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

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

Parameters:

← ***user_info*** User information pointer as passed to the invoked device operation handler

← ***dst*** Destination buffer address

← ***src*** Address of the user-space memory block

← ***size*** Size of the memory block

Returns:

0 on success, otherwise:

- -EFAULT is returned if an invalid memory area was accessed.

Note:

This service is a combination of rtdm_read_user_ok and rtdm_copy_from_user.

Environments:

This service can be called from:

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

Rescheduling: never.

5.14.1.13 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:

← ***user_info*** User information pointer as passed to the invoked device operation handler

← ***dst*** Address of the user-space memory block

← ***src*** Source buffer address

← ***size*** Size of the memory block

Returns:

0 on success, otherwise:

- -EFAULT is returned if an invalid memory area was accessed.

Note:

This service is a combination of rtdm_rw_user_ok and rtdm_copy_to_user.

Environments:

This service can be called from:

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

Rescheduling: never.

5.14.1.14 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:

← ***user_info*** User information pointer as passed to the invoked device operation handler

← ***dst*** Destination buffer address

← ***src*** Address of the user-space string

← ***count*** Maximum number of bytes to copy, including the trailing ’0’

Returns:

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

- -EFAULT is returned if an invalid memory area was accessed.

Note:

This services already includes a check of the source address, calling rtdm_read_user_ok() for *src* explicitly is not required.

Environments:

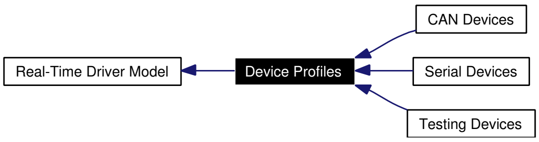
This service can be called from:

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

Rescheduling: never.

5.15 Device Profiles

Collaboration diagram for Device Profiles:



5.15.1 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.

Modules

- CAN Devices
- Serial Devices
- Testing Devices

Data Structures

- struct rtdm_device_info

  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_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 rtdm_device_info rtdm_device_info_t

  Device information.

5.15.2 Define Documentation

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

Retrieve information about a device or socket.

Parameters:

→ *arg* Pointer to information buffer (struct rtdm_device_info)

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

Purge internal device or socket buffers.

Parameters:

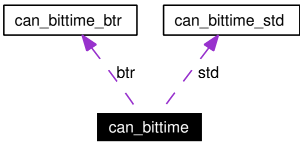
← *arg* Purge mask, see RTDM_PURGE_xxx_BUFFER

Chapter 6

Xenomai RTDM skin API Data Structure Documentation

6.1 can_bittime Struct Reference

Collaboration diagram for can_bittime:



6.1.1 Detailed Description

Custom CAN bit-time definition.

Examples:
rtcanconfig.c.

Data Fields

- can_bittime_type_t type

  Type of bit-time definition.

- can_bittime_std std

  Standard bit-time.

- can_bittime_btr btr

  Hardware-spcific BTR bit-time.

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

- include/rtdm/rtcan.h

6.2 can_bittime_btr Struct Reference

6.2.1 Detailed Description

Hardware-specific BTR bit-times.

Data Fields

- uint8_t btr0

  Bus timing register 0.

- uint8_t btr1

  Bus timing register 1.

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

- include/rtdm/rtcan.h

6.3 can_bittime_std Struct Reference

6.3.1 Detailed Description

Standard bit-time parameters according to Bosch.

Data Fields

- uint32_t brp

  Baud rate prescaler.

- uint8_t prop_seg

  from 1 to 8

- uint8_t phase_seg1

  from 1 to 8

- uint8_t phase_seg2

  from 1 to 8

- uint8_t sjw:7

  from 1 to 4

- uint8_t sam:1

  1 - enable triple sampling

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

- include/rtdm/rtcan.h

6.4 can_filter Struct Reference

6.4.1 Detailed Description

Filter for reception of CAN messages.

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

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

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

Examples:
rtcan_rtt.c.

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.

6.4.2 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:
rtcan_rtt.c, and 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.

Examples:
rtcan_rtt.c, and rtcanrecv.c.

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

- include/rtdm/rtcan.h

6.5 can_frame Struct Reference

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.

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.

6.5.2 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.

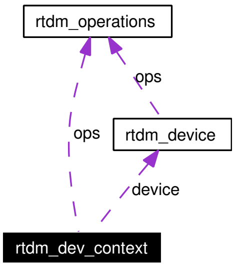
Examples:
rtcan_rtt.c.

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

- include/rtdm/rtcan.h

6.6 rtdm_dev_context Struct Reference

Collaboration diagram for rtdm_dev_context:



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.

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.

- rtdm_operations * ops

  Set of active device operation handlers.

- rtdm_device * device

  Reference to owning device.

- rtdm_devctx_reserved reserved

  Data stored by RTDM inside a device context (internal use only).

- char dev_private [0]

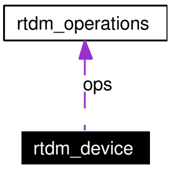
  Begin of driver defined context data structure.

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

- include/rtdm/rtdm_driver.h

6.7 rtdm_device Struct Reference

Collaboration diagram for rtdm_device:



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.

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 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 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.

- 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.

- proc_dir_entry * proc_entry

  Set to device's /proc root entry after registration, do not modify.

- int device_id

  Driver definable device ID.

- rtdm_dev_reserved reserved

  Data stored by RTDM inside a registered device (internal use only).

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

- include/rtdm/rtdm_driver.h

6.8 rtdm_device_info Struct Reference

6.8.1 Detailed Description

Device information.

Data Fields

- int device_flags

  Device flags, see Device Flags for details.

- int device_class

  Device class ID, see RTDM_CLASS_xxx.

- int device_sub_class

  Device sub-class, either RTDM_SUBCLASS_GENERIC or a RTDM_SUBCLASS_xxx definition of the related Device Profile.

- int profile_version

  Supported device profile version.

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

- include/rtdm/rtdm.h

6.9 rtdm_operations Struct Reference

6.9.1 Detailed Description

Device operations.

Data Fields

Common Operations

- rtdm_close_handler_t close_rt
  Close handler for real-time contexts (optional).
- rtdm_close_handler_t close_nrt
  Close handler for non-real-time contexts (required).
- rtdm_ioctl_handler_t ioctl_rt
  IOCTL from real-time context (optional).
- rtdm_ioctl_handler_t ioctl_nrt
  IOCTL from non-real-time context (optional).

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).

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

- include/rtdm/rtdm_driver.h

6.10 rtser_config Struct Reference

6.10.1 Detailed Description

Serial device configuration.

Examples:

cross-link.c.

Data Fields

- int config_mask

  mask specifying valid fields, see RTSER_SET_xxx

- int baud_rate

  baud rate, default RTSER_DEF_BAUD

- int parity

  number of parity bits, see RTSER_xxx_PARITY

- int data_bits

  number of data bits, see RTSER_xxx_BITS

- int stop_bits

  number of stop bits, see RTSER_xxx_STOPB

- int handshake

  handshake mechanisms, see RTSER_xxx_HAND

- int fifo_depth

  reception FIFO interrupt threshold, see RTSER_FIFO_xxx

- nanosecs_rel_t rx_timeout

  reception timeout, see RTSER_TIMEOUT_xxx for special values

- nanosecs_rel_t tx_timeout

  transmission timeout, see RTSER_TIMEOUT_xxx for special values

- nanosecs_rel_t event_timeout

  event timeout, see RTSER_TIMEOUT_xxx for special values

- int timestamp_history

  enable timestamp history, see RTSER_xxx_TIMESTAMP_HISTORY

- int event_mask

  event mask to be used with RTSER_RTIOC_WAIT_EVENT, see RTSER_EVENT_xxx

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

- include/rtdm/rtserial.h

6.11 rtser_event Struct Reference

6.11.1 Detailed Description

Additional information about serial device events.

Examples:

cross-link.c.

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

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

- include/rtdm/rtserial.h

6.12 rtser_status Struct Reference

6.12.1 Detailed Description

Serial device status.

Data Fields

- int line_status

  line status register, see RTSER_LSR_xxx

- int modem_status

  modem status register, see RTSER_MSR_xxx

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

- include/rtdm/rtserial.h

6.13 sockaddr_can Struct Reference

6.13.1 Detailed Description

Socket address structure for the CAN address family.

Examples:

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

Data Fields

- sa_family_t can_family

  CAN address family, must be `AF_CAN`*.*

- int can_ifindex

  Interface index of CAN controller.

6.13.2 Field Documentation

6.13.2.1 int sockaddr_can::can_ifindex

Interface index of CAN controller.

See SIOCGIFINDEX.

Examples:

rtcan_rtt.c, and rtcanrecv.c.

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

- include/rtdm/rtcan.h

Chapter 7

Xenomai RTDM skin API File Documentation

7.1 include/rtdm/rtcan.h File Reference

7.1.1 Detailed Description

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

Note:

Copyright (C) 2006 Wolfgang Grandegger <wg@grandegger.com>
Copyright (C) 2005, 2006 Sebastian Smolorz <Sebastian.Smolorz@stud.uni-hannover.de>

This RTDM CAN device profile header is based on:

include/linux/can.h, include/linux/socket.h, net/can/pf_can.h in linux-can.patch, a CAN socket framework for Linux

Copyright (C) 2004, 2005, Robert Schwebel, Benedikt Spranger, Marc Kleine-Budde, Pengutronix

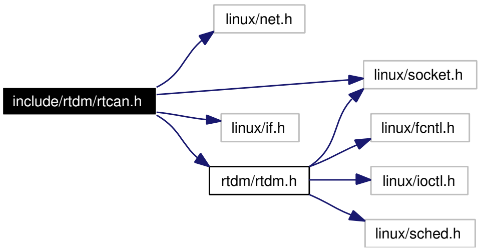
This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU

General Public License for more details.

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

Include dependency graph for rtcan.h:



Data Structures

- struct can_bittime_std

    Standard bit-time parameters according to Bosch.

- struct can_bittime_btr

    Hardware-specific BTR bit-times.

- struct can_bittime

    Custom CAN bit-time definition.

- struct can_filter

    Filter for reception of CAN messages.

- struct sockaddr_can

    Socket address structure for the CAN address family.

- struct can_frame

    Raw CAN frame.

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.

CAN controller modes

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

- #define CAN_CTRLMODE_LISTENONLY 0x1

  Listen-Only mode.

- #define CAN_CTRLMODE_LOOPBACK 0x2

  Loopback mode.

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.

IOCTLs

CAN device IOCTLs

- #define SIOCGIFINDEX _IOWR(RTIOC_TYPE_CAN, 0x00, struct ifreq)

  Get CAN interface index by name.

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

  Set baud rate.

- #define SIOCGCANBAUDRATE _IOWR(RTIOC_TYPE_CAN, 0x02, struct ifreq)

  Get baud rate.

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

  Set custom bit time parameter.

- #define SIOCGCANCUSTOMBITTIME _IOWR(RTIOC_TYPE_CAN, 0x04, struct ifreq)

  Get custum bit-time parameters.

- #define SIOCSCANMODE _IOW(RTIOC_TYPE_CAN, 0x05, struct ifreq)

  Set operation mode of CAN controller.

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

  Get current state of CAN controller.

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

  Set special controller modes.

- #define SIOCGCANCTRLMODE _IOWR(RTIOC_TYPE_CAN, 0x08, struct ifreq)

  Get special controller modes.

- #define RTCAN_RTIOC_TAKE_TIMESTAMP _IOW(RTIOC_TYPE_CAN, 0x09, int)

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

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

  Specify a reception timeout for a socket.

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

  Specify a transmission timeout for a socket.

Error mask

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

- #define CAN_ERR_TX_TIMEOUT 0x00000001U

  TX timeout (netdevice driver).

- #define CAN_ERR_LOSTARB 0x00000002U

  Lost arbitration (see data[0]).

- #define CAN_ERR_CRTL 0x00000004U

  Controller problems (see data[1]).

- #define CAN_ERR_PROT 0x00000008U

  Protocol violations (see data[2], data[3]).

- #define CAN_ERR_TRX 0x00000010U

  Transceiver status (see data[4]).

- #define CAN_ERR_ACK 0x00000020U

  Received no ACK on transmission.

- #define CAN_ERR_BUSOFF 0x00000040U

  Bus off.

- #define CAN_ERR_BUSERROR 0x00000080U

  Bus error (may flood!).

- #define CAN_ERR_RESTARTED 0x00000100U

  Controller restarted.

- #define CAN_ERR_MASK 0x1FFFFFFFU

  Omit EFF, RTR, ERR flags.

Arbitration lost error

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

- #define CAN_ERR_LOSTARB_UNSPEC 0x00

  unspecified else bit number in bitstream

Controller problems

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

- #define CAN_ERR_CRTL_UNSPEC 0x00

  unspecified

- #define CAN_ERR_CRTL_RX_OVERFLOW 0x01

  RX buffer overflow.

- #define CAN_ERR_CRTL_TX_OVERFLOW 0x02

  TX buffer overflow.

- #define CAN_ERR_CRTL_RX_WARNING 0x04

  reached warning level for RX errors

- #define CAN_ERR_CRTL_TX_WARNING 0x08

  reached warning level for TX errors

- #define CAN_ERR_CRTL_RX_PASSIVE 0x10

  reached passive level for RX errors

- #define CAN_ERR_CRTL_TX_PASSIVE 0x20

  reached passive level for TX errors

Protocol error type

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

- #define CAN_ERR_PROT_UNSPEC 0x00

  unspecified

- #define CAN_ERR_PROT_BIT 0x01

  single bit error

- #define CAN_ERR_PROT_FORM 0x02

  frame format error

- #define CAN_ERR_PROT_STUFF 0x04

  bit stuffing error

- #define CAN_ERR_PROT_BIT0 0x08

  unable to send dominant bit

- #define CAN_ERR_PROT_BIT1 0x10

  unable to send recessive bit

- #define CAN_ERR_PROT_OVERLOAD 0x20

  bus overload

- #define CAN_ERR_PROT_ACTIVE 0x40

  active error announcement

- #define CAN_ERR_PROT_TX 0x80

  error occured on transmission

Protocol error location

Error in the data[3] field of struct can_frame.

- #define CAN_ERR_PROT_LOC_UNSPEC 0x00

  unspecified

- #define CAN_ERR_PROT_LOC_SOF 0x03

  start of frame

- #define CAN_ERR_PROT_LOC_ID28_21 0x02

  ID bits 28 - 21 (SFF: 10 - 3).

- #define CAN_ERR_PROT_LOC_ID20_18 0x06

  ID bits 20 - 18 (SFF: 2 - 0).

- #define CAN_ERR_PROT_LOC_SRTR 0x04

  substitute RTR (SFF: RTR)

- #define CAN_ERR_PROT_LOC_IDE 0x05

  identifier extension

- #define CAN_ERR_PROT_LOC_ID17_13 0x07

  ID bits 17-13.

- #define CAN_ERR_PROT_LOC_ID12_05 0x0F

  ID bits 12-5.

- #define CAN_ERR_PROT_LOC_ID04_00 0x0E

  ID bits 4-0.

- #define CAN_ERR_PROT_LOC_RTR 0x0C

  RTR.

- #define CAN_ERR_PROT_LOC_RES1 0x0D

  reserved bit 1

- #define CAN_ERR_PROT_LOC_RES0 0x09

  reserved bit 0

- #define CAN_ERR_PROT_LOC_DLC 0x0B

  data length code

- #define CAN_ERR_PROT_LOC_DATA 0x0A

  data section

- #define CAN_ERR_PROT_LOC_CRC_SEQ 0x08

  CRC sequence.

- #define CAN_ERR_PROT_LOC_CRC_DEL 0x18

CRC delimiter.

- #define CAN_ERR_PROT_LOC_ACK 0x19

  ACK slot.

- #define CAN_ERR_PROT_LOC_ACK_DEL 0x1B

  ACK delimiter.

- #define CAN_ERR_PROT_LOC_EOF 0x1A

  end of frame

- #define CAN_ERR_PROT_LOC_INTERM 0x12

  intermission

Protocol error location

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

- #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

CAN protocols

Possible protocols for PF_CAN protocol family

- enum CAN_PROTO { CAN_PROTO_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 states

States a CAN controller can be in.

- enum CAN_STATE {
  CAN_STATE_ACTIVE = 0, CAN_STATE_BUS_WARNING, CAN_STATE_BUS_PASSIVE, CAN_STATE_BUS_OFF,
  CAN_STATE_SCANNING_BAUDRATE, CAN_STATE_STOPPED, CAN_STATE_SLEEPING }

Defines

- #define AF_CAN 29

  CAN address family.

- #define PF_CAN AF_CAN

  CAN protocol family.

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 can_filter can_filter_t

  Filter for reception of CAN messages.

- typedef can_frame can_frame_t

  Raw CAN frame.

Enumerations

- enum CAN_BITTIME_TYPE { CAN_BITTIME_STD, CAN_BITTIME_BTR }

  Supported CAN bit-time types.

7.2 include/rtdm/rtdm.h File Reference

7.2.1 Detailed Description

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

Note:

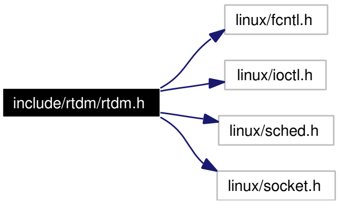
Copyright (C) 2005, 2006 Jan Kiszka <jan.kiszka@web.de>
Copyright (C) 2005 Joerg Langenberg <joerg.langenberg@gmx.net>

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.

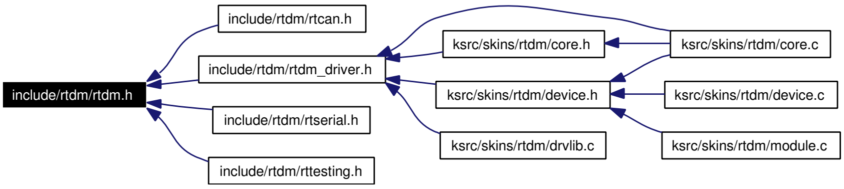
Xenomai is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

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.

Include dependency graph for rtdm.h:



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



Data Structures

- struct rtdm_device_info

  Device information.

API Versioning

- #define RTDM_API_VER 6

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_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 rtdm_device_info rtdm_device_info_t

  Device information.

7.3 include/rtdm/rtdm_driver.h File Reference

7.3.1 Detailed Description

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

Note:

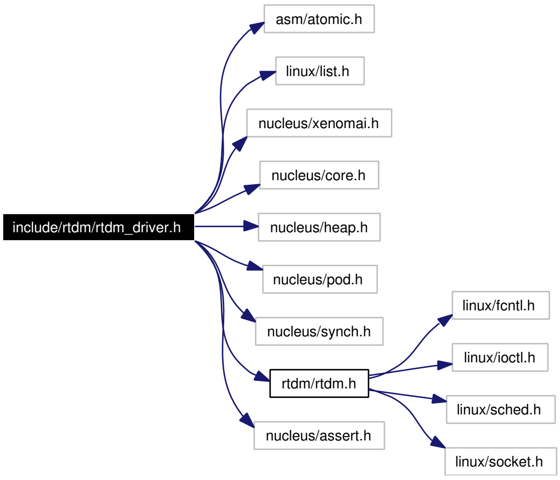
Copyright (C) 2005, 2006 Jan Kiszka <jan.kiszka@web.de>
Copyright (C) 2005 Joerg Langenberg <joerg.langenberg@gmx.net>

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.

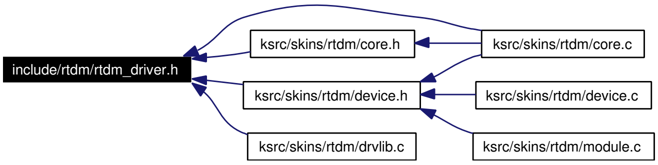
Xenomai is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

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.

Include dependency graph for rtdm_driver.h:



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



Data Structures

- struct rtdm_operations

Device operations.

- struct rtdm_dev_context

  Device context.

- struct rtdm_device

  RTDM device.

Device Flags

Static flags describing a RTDM device

- #define RTDM_EXCLUSIVE 0x0001

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

- #define RTDM_NAMED_DEVICE 0x0010

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

- #define RTDM_PROTOCOL_DEVICE 0x0020

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

- #define RTDM_DEVICE_TYPE_MASK 0x00F0

  Mask selecting the device type.

Context Flags

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

- #define RTDM_CREATED_IN_NRT 0

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

- #define RTDM_CLOSING 1

  Set by RTDM when the device is being closed.

- #define RTDM_USER_CONTEXT_FLAG 8

  Lowest bit number the driver developer can use freely.

Driver Versioning

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

- #define RTDM_DEVICE_STRUCT_VER 4

  Version of struct rtdm_device.

- #define RTDM_CONTEXT_STRUCT_VER 3

  Version of struct rtdm_dev_context.

- #define RTDM_SECURE_DEVICE 0x80000000

  Flag indicating a secure variant of RTDM (not supported here).

- #define RTDM_DRIVER_VER(major, minor, patch) (((major & 0xFF) << 16) | ((minor & 0xFF) << 8) | (patch & 0xFF))

  Version code constructor for driver revisions.

- #define RTDM_DRIVER_MAJOR_VER(ver) (((ver) >> 16) & 0xFF)

  Get major version number from driver revision code.

- #define RTDM_DRIVER_MINOR_VER(ver) (((ver) >> 8) & 0xFF)

  Get minor version number from driver revision code.

- #define RTDM_DRIVER_PATCH_VER(ver) ((ver) & 0xFF)

  Get patch version number from driver revision code.

Global Lock across Scheduler Invocation

- #define RTDM_EXECUTE_ATOMICALLY(code_block)

  Execute code block atomically.

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) rthal_spin_unlock(lock)

  Release lock without preemption restoration.

- #define rtdm_lock_get_irqsave(lock, context) rthal_spin_lock_irqsave(lock, context)

  Acquire lock and disable preemption.

- #define rtdm_lock_put_irqrestore(lock, context) rthal_spin_unlock_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.

RTDM_IRQTYPE_xxx

Interrupt registrations flags

- #define RTDM_IRQTYPE_SHARED XN_ISR_SHARED

  Enable IRQ-sharing with other real-time drivers.

- #define RTDM_IRQTYPE_EDGE XN_ISR_EDGE

  Mark IRQ as edge-triggered, relevant for correct handling of shared edge-triggered IRQs.

RTDM_IRQ_xxx

Return flags of interrupt handlers

- #define RTDM_IRQ_NONE XN_ISR_NONE

  Unhandled interrupt.

- #define RTDM_IRQ_HANDLED XN_ISR_HANDLED

  Denote handled interrupt.

Task Priority Range

Maximum and minimum task priorities

- #define **RTDM_TASK_LOWEST_PRIORITY** XNCORE_LOW_PRIO
- #define **RTDM_TASK_HIGHEST_PRIORITY** XNCORE_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)

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 ∗arg)

  IOCTL handler.

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

  Read handler.

- typedef ssize_t(∗ rtdm_write_handler_t)(struct rtdm_dev_context ∗context, rtdm_user_info_t ∗user_info, const void ∗buf, size_t nbyte)

  Write handler.

- typedef ssize_t(∗ rtdm_recvmsg_handler_t)(struct rtdm_dev_context ∗context, rtdm_user_info_t ∗user_info, struct msghdr ∗msg, int flags)

  Receive message handler.

- typedef ssize_t(∗ rtdm_sendmsg_handler_t)(struct rtdm_dev_context ∗context, rtdm_user_info_t ∗user_info, const struct msghdr ∗msg, int flags)

  Transmit message handler.

Defines

- #define rtdm_irq_get_arg(irq_handle, type) ((type ∗)irq_handle → cookie)

  Retrieve IRQ handler argument.

Typedefs

- typedef int(∗ rtdm_irq_handler_t)(rtdm_irq_t ∗irq_handle)

  Interrupt handler.

- typedef void(∗ rtdm_nrtsig_handler_t)(rtdm_nrtsig_t nrt_sig)

  Non-real-time signal handler.

- typedef void(∗ rtdm_task_proc_t)(void ∗arg)

Real-time task procedure.

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.

- rtdm_dev_context * rtdm_context_get (int fd)

  Resolve file descriptor to device context.

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

  Intialise and start a real-time task.

- 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.

- 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_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_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.

7.4 include/rtdm/rtserial.h File Reference

7.4.1 Detailed Description

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

Note:

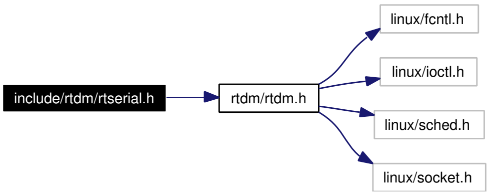
Copyright (C) 2005, 2006 Jan Kiszka <jan.kiszka@web.de>

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.

Include dependency graph for rtserial.h:



Data Structures

- struct rtser_config

  Serial device configuration.

- struct rtser_status

  Serial device status.

- struct rtser_event

  Additional information about serial device events.

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
- #define RTSER_1_5_STOPB 0x01

  valid only in combination with 5 data bits

- #define **RTSER_2_STOPB** 0x01
- #define **RTSER_DEF_STOPB** RTSER_1_STOPB

RTSER_xxx_HAND

Handshake mechanisms

- #define **RTSER_NO_HAND** 0x00
- #define **RTSER_RTSCTS_HAND** 0x01
- #define **RTSER_DEF_HAND** RTSER_NO_HAND

RTSER_FIFO_xxx

Reception FIFO interrupt threshold

- #define **RTSER_FIFO_DEPTH_1** 0x00
- #define **RTSER_FIFO_DEPTH_4** 0x40
- #define **RTSER_FIFO_DEPTH_8** 0x80
- #define **RTSER_FIFO_DEPTH_14** 0xC0
- #define **RTSER_DEF_FIFO_DEPTH** RTSER_FIFO_DEPTH_1

RTSER_TIMEOUT_xxx

Special timeout values, see also RTDM_TIMEOUT_xxx

- #define **RTSER_TIMEOUT_INFINITE** RTDM_TIMEOUT_INFINITE
- #define **RTSER_TIMEOUT_NONE** RTDM_TIMEOUT_NONE
- #define **RTSER_DEF_TIMEOUT** RTDM_TIMEOUT_INFINITE

RTSER_xxx_TIMESTAMP_HISTORY

Timestamp history control

- #define **RTSER_RX_TIMESTAMP_HISTORY** 0x01
- #define **RTSER_DEF_TIMESTAMP_HISTORY** 0x00

RTSER_EVENT_xxx

Events bits

- #define **RTSER_EVENT_RXPEND** 0x01
- #define **RTSER_EVENT_ERRPEND** 0x02
- #define **RTSER_EVENT_MODEMHI** 0x04
- #define **RTSER_EVENT_MODEMLO** 0x08
- #define **RTSER_DEF_EVENT_MASK** 0x00

RTSER_SET_xxx

Configuration mask bits

- #define **RTSER_SET_BAUD** 0x0001
- #define **RTSER_SET_PARITY** 0x0002
- #define **RTSER_SET_DATA_BITS** 0x0004
- #define **RTSER_SET_STOP_BITS** 0x0008
- #define **RTSER_SET_HANDSHAKE** 0x0010
- #define **RTSER_SET_FIFO_DEPTH** 0x0020
- #define **RTSER_SET_TIMEOUT_RX** 0x0100
- #define **RTSER_SET_TIMEOUT_TX** 0x0200
- #define **RTSER_SET_TIMEOUT_EVENT** 0x0400
- #define **RTSER_SET_TIMESTAMP_HISTORY** 0x0800
- #define **RTSER_SET_EVENT_MASK** 0x1000

RTSER_LSR_xxx

Line status bits

- #define **RTSER_LSR_DATA** 0x01
- #define **RTSER_LSR_OVERRUN_ERR** 0x02
- #define **RTSER_LSR_PARITY_ERR** 0x04

- #define **RTSER_LSR_FRAMING_ERR** 0x08
- #define **RTSER_LSR_BREAK_IND** 0x10
- #define **RTSER_LSR_THR_EMTPY** 0x20
- #define **RTSER_LSR_TRANSM_EMPTY** 0x40
- #define **RTSER_LSR_FIFO_ERR** 0x80
- #define **RTSER_SOFT_OVERRUN_ERR** 0x0100

RTSER_MSR_xxx

Modem status bits

- #define **RTSER_MSR_DCTS** 0x01
- #define **RTSER_MSR_DDSR** 0x02
- #define **RTSER_MSR_TERI** 0x04
- #define **RTSER_MSR_DDCD** 0x08
- #define **RTSER_MSR_CTS** 0x10
- #define **RTSER_MSR_DSR** 0x20
- #define **RTSER_MSR_RI** 0x40
- #define **RTSER_MSR_DCD** 0x80

RTSER_MCR_xxx

Modem control bits

- #define **RTSER_MCR_DTR** 0x01
- #define **RTSER_MCR_RTS** 0x02
- #define **RTSER_MCR_OUT1** 0x04
- #define **RTSER_MCR_OUT2** 0x08
- #define **RTSER_MCR_LOOP** 0x10

IOCTLs

Serial device IOCTLs

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

  Get serial device configuration.

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

  Set serial device configuration.

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

  Get serial device status.

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

  Get serial device's modem contol register.

- #define RTSER_RTIOC_SET_CONTROL _IOW(RTIOC_TYPE_SERIAL, 0x04, int)

  Set serial device's modem contol register.

- #define RTSER_RTIOC_WAIT_EVENT _IOR(RTIOC_TYPE_SERIAL, 0x05, struct rtser_-event)

  Wait on serial device events according to previously set mask.

Typedefs

- typedef rtser_config rtser_config_t

  Serial device configuration.

- typedef rtser_status rtser_status_t

  Serial device status.

- typedef rtser_event rtser_event_t

  Additional information about serial device events.

7.5 include/rtdm/rttesting.h File Reference

7.5.1 Detailed Description

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

Note:

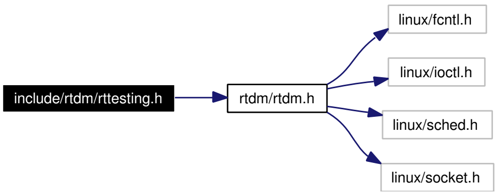
Copyright (C) 2005 Jan Kiszka <jan.kiszka@web.de>

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.

Include dependency graph for rttesting.h:



IOCTLs

Testing device IOCTLs

- #define **RTTST_RTIOC_INTERM_BENCH_RES** _IOWR(RTIOC_TYPE_TESTING, 0x00, struct rttst_interm_bench_res)
- #define **RTTST_RTIOC_TMBENCH_START** _IOW(RTIOC_TYPE_TESTING, 0x10, struct rttst_tmbench_config)
- #define **RTTST_RTIOC_TMBENCH_STOP** _IOWR(RTIOC_TYPE_TESTING, 0x11, struct rttst_overall_bench_res)
- #define **RTTST_RTIOC_IRQBENCH_START** _IOW(RTIOC_TYPE_TESTING, 0x20, struct rttst_irqbench_config)
- #define **RTTST_RTIOC_IRQBENCH_STOP** _IO(RTIOC_TYPE_TESTING, 0x21)
- #define **RTTST_RTIOC_IRQBENCH_GET_STATS** _IOR(RTIOC_TYPE_TESTING, 0x22, struct rttst_irqbench_stats)
- #define **RTTST_RTIOC_IRQBENCH_WAIT_IRQ** _IO(RTIOC_TYPE_TESTING, 0x23)
- #define **RTTST_RTIOC_IRQBENCH_REPLY_IRQ** _IO(RTIOC_TYPE_TESTING, 0x24)
- #define **RTTST_RTIOC_SWTEST_SET_TASKS_COUNT** _IOW(RTIOC_TYPE_TESTING, 0x30, unsigned long)
- #define **RTTST_RTIOC_SWTEST_SET_CPU** _IOW(RTIOC_TYPE_TESTING, 0x31, unsigned long)

- #define **RTTST_RTIOC_SWTEST_REGISTER_UTASK** _IOW(RTIOC_TYPE_TESTING, 0x32, struct rttst_swtest_task)
- #define **RTTST_RTIOC_SWTEST_CREATE_KTASK** _IOWR(RTIOC_TYPE_TESTING, 0x33, struct rttst_swtest_task)
- #define **RTTST_RTIOC_SWTEST_PEND** _IOR(RTIOC_TYPE_TESTING, 0x34, struct rttst_swtest_task)
- #define **RTTST_RTIOC_SWTEST_SWITCH_TO** _IOR(RTIOC_TYPE_TESTING, 0x35, struct rttst_swtest_dir)
- #define **RTTST_RTIOC_SWTEST_GET_SWITCHES_COUNT** _IOR(RTIOC_TYPE_TESTING, 0x36, unsigned long)
- #define **RTTST_RTIOC_SWTEST_GET_LAST_ERROR** _IOR(RTIOC_TYPE_TESTING, 0x37, struct rttst_swtest_error)

7.6 ksrc/skins/rtdm/device.c File Reference

7.6.1 Detailed Description

Real-Time Driver Model for Xenomai, device management.

Note:

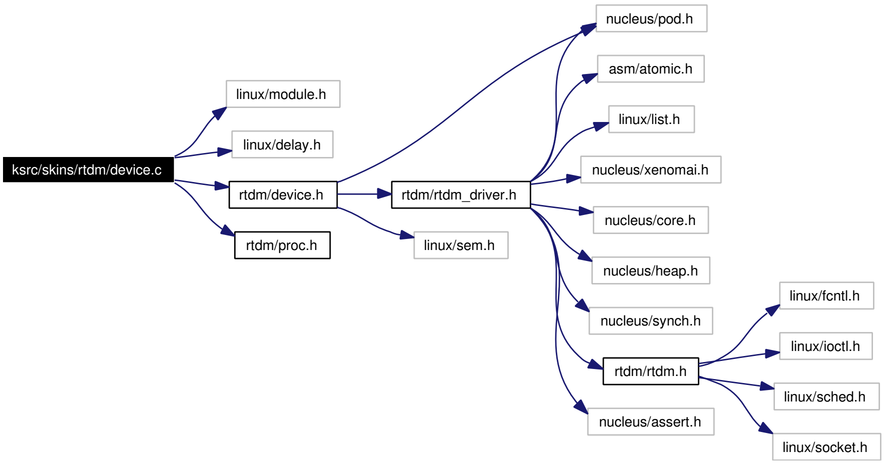
Copyright (C) 2005 Jan Kiszka <jan.kiszka@web.de>
Copyright (C) 2005 Joerg Langenberg <joerg.langenberg@gmx.net>

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.

Include dependency graph for device.c:



Functions

- int rtdm_dev_register (struct rtdm_device *device)

  Register a RTDM device.

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

  Unregisters a RTDM device.

7.7 ksrc/skins/rtdm/drvlib.c File Reference

7.7.1 Detailed Description

Real-Time Driver Model for Xenomai, driver library.

Note:

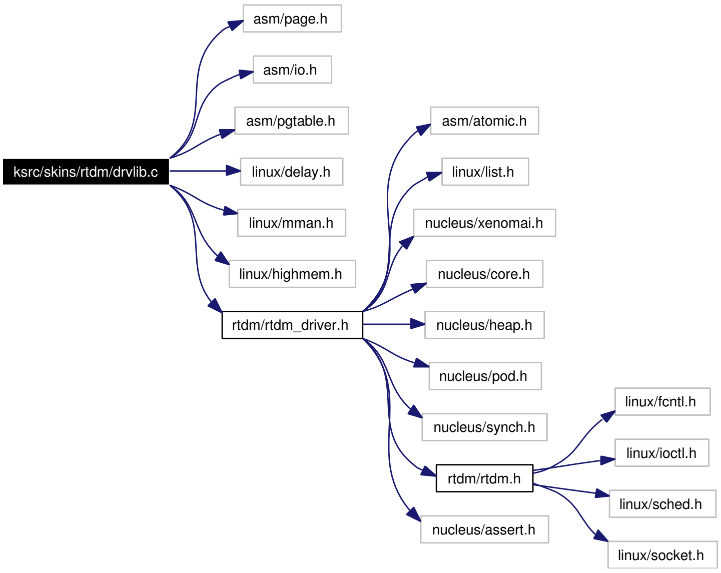
Copyright (C) 2005 Jan Kiszka <jan.kiszka@web.de>
Copyright (C) 2005 Joerg Langenberg <joerg.langenberg@gmx.net>

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.

Include dependency graph for drvlib.c:



Timeout Sequence Management

- void rtdm_toseq_init (rtdm_toseq_t *timeout_seq, nanosecs_rel_t timeout)

    Initialise a timeout sequence.

Event Services

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

Initialise an event.

- void rtdm_event_destroy (rtdm_event_t *event)

  Destroy an event.

- void rtdm_event_pulse (rtdm_event_t *event)

  Signal an event occurrence to currently listening waiters.

- void rtdm_event_signal (rtdm_event_t *event)

  Signal an event occurrence.

- int rtdm_event_wait (rtdm_event_t *event)

  Wait on event occurrence.

- int rtdm_event_timedwait (rtdm_event_t *event, nanosecs_rel_t timeout, rtdm_toseq_t *timeout_seq)

  Wait on event occurrence with timeout.

- void rtdm_event_clear (rtdm_event_t *event)

  Clear event state.

Semaphore Services

- void rtdm_sem_init (rtdm_sem_t *sem, unsigned long value)

  Initialise a semaphore.

- void rtdm_sem_destroy (rtdm_sem_t *sem)

  Destroy a semaphore.

- int rtdm_sem_down (rtdm_sem_t *sem)

  Decrement a semaphore.

- int rtdm_sem_timeddown (rtdm_sem_t *sem, nanosecs_rel_t timeout, rtdm_toseq_t *timeout_seq)

  Decrement a semaphore with timeout.

- void rtdm_sem_up (rtdm_sem_t *sem)

  Increment a semaphore.

Mutex Services

- void rtdm_mutex_init (rtdm_mutex_t *mutex)

  Initialise a mutex.

- void rtdm_mutex_destroy (rtdm_mutex_t *mutex)

  Destroy a mutex.

- void rtdm_mutex_unlock (rtdm_mutex_t *mutex)

  Release a mutex.

- int rtdm_mutex_lock (rtdm_mutex_t *mutex)

  Request a mutex.

- int rtdm_mutex_timedlock (rtdm_mutex_t *mutex, nanosecs_rel_t timeout, rtdm_toseq_t *timeout_seq)

  Request a mutex with timeout.

Functions

- nanosecs_abs_t rtdm_clock_read (void)

  Get system time.

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

  Intialise and start a real-time task.

- void rtdm_task_destroy (rtdm_task_t *task)

  Destroy a real-time task.

- void rtdm_task_set_priority (rtdm_task_t *task, int priority)

  Adjust real-time task priority.

- int rtdm_task_set_period (rtdm_task_t *task, nanosecs_rel_t period)

  Adjust real-time task period.

- int rtdm_task_wait_period (void)

  Wait on next real-time task period.

- int rtdm_task_unblock (rtdm_task_t *task)

  Activate a blocked real-time task.

- rtdm_task_t * rtdm_task_current (void)

  Get current real-time task.

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

  Wait on a real-time task to terminate.

- 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.

- void rtdm_task_busy_sleep (nanosecs_rel_t delay)

  Busy-wait a specified amount of time.

- 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)

  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, unsigned long src_addr, size_t len, int prot, void **pptr, struct vm_operations_struct *vm_ops, void *vm_private_data)

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

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

  Unmap a user memory range.

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

  Real-time safe message printing on kernel console.

- void * rtdm_malloc (size_t size)

  Allocate memory block in real-time context.

- void rtdm_free (void *ptr)

  Release real-time memory block.

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

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

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

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

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

Copy user-space memory block to specified buffer.

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

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

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

  Copy specified buffer to user-space memory block.

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

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

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

  Copy user-space string to specified buffer.

- int rtdm_in_rt_context (void)

  Test if running in a real-time task.

7.8 ksrc/skins/rtdm/module.c File Reference

7.8.1 Detailed Description

Real-Time Driver Model for Xenomai.

Note:

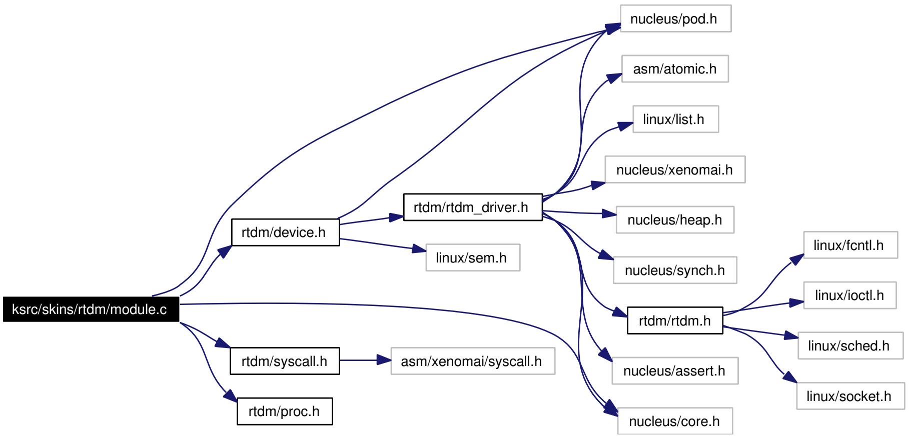
Copyright (C) 2005, 2006 Jan Kiszka <jan.kiszka@web.de>
Copyright (C) 2005 Joerg Langenberg <joerg.langenberg@gmx.net>

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.

Include dependency graph for module.c:



7.9 ksrc/skins/rtdm/core.c File Reference

7.9.1 Detailed Description

Real-Time Driver Model for Xenomai, device operation multiplexing.

Note:

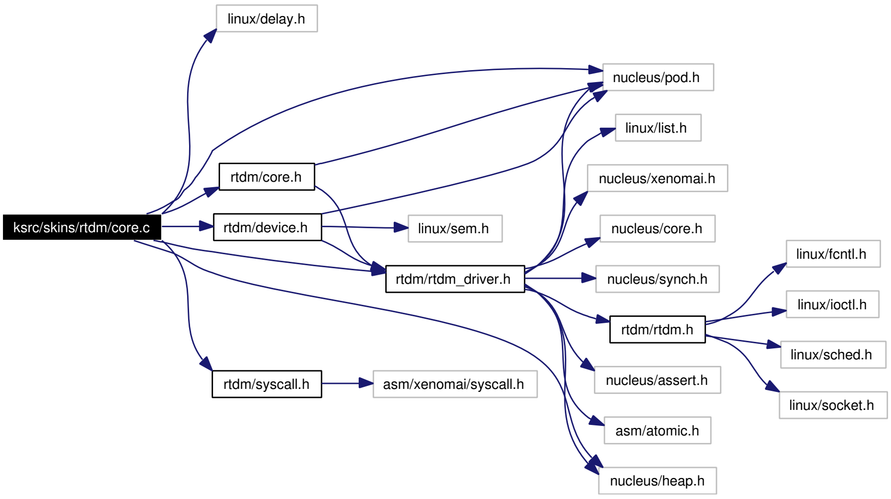
Copyright (C) 2005 Jan Kiszka <jan.kiszka@web.de>
Copyright (C) 2005 Joerg Langenberg <joerg.langenberg@gmx.net>

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.

Include dependency graph for core.c:



Functions

- rtdm_dev_context * rtdm_context_get (int fd)

  Resolve file descriptor to device context.

- 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.

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

  Open a device.

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

  Create a socket.

- int rtdm_close (int fd)

  Close a device or socket.

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

  Issue an IOCTL.

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

  Read from device.

- ssize_t rtdm_write (int fd, const void *buf, size_t nbyte)

  Write to device.

- ssize_t rtdm_recvmsg (int fd, struct msghdr *msg, int flags)

  Receive message from socket.

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

  Receive message from socket.

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

  Receive message from socket.

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

  Transmit message to socket.

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

  Transmit message to socket.

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

  Transmit message to socket.

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

  Bind to local address.

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

  Connect to remote address.

- int rtdm_listen (int fd, int backlog)

  Listen for incomming connection requests.

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

  Accept a connection requests.

- int rtdm_shutdown (int fd, int how)

  Shut down parts of a connection.

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

  Get socket option.

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

  Set socket option.

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

  Get local socket address.

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

  Get socket destination address.

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

  Open a device.

- int rt_dev_socket (int protocol_family, int socket_type, int protocol)

  Create a socket.

- int rt_dev_close (int fd)

  Close a device or socket.

- int rt_dev_ioctl (int fd, int request,...)

  Issue an IOCTL.

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

  Read from device.

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

  Write to device.

- ssize_t rt_dev_recvmsg (int fd, struct msghdr *msg, int flags)

  Receive message from socket.

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

  Receive message from socket.

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

  Receive message from socket.

- ssize_t rt_dev_sendmsg (int fd, const struct msghdr *msg, int flags)

  Transmit message to socket.

- ssize_t rt_dev_sendto (int fd, const void *buf, size_t len, int flags, const struct sockaddr *to, socklen_t tolen)

  Transmit message to socket.

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

    Transmit message to socket.

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

    Bind to local address.

- int rt_dev_connect (int fd, const struct sockaddr *serv_addr, socklen_t addrlen)

    Connect to remote address.

- int rt_dev_listen (int fd, int backlog)

    Listen for incomming connection requests.

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

    Accept a connection requests.

- int rt_dev_shutdown (int fd, int how)

    Shut down parts of a connection.

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

    Get socket option.

- int rt_dev_setsockopt (int fd, int level, int optname, const void *optval, socklen_t optlen)

    Set socket option.

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

    Get local socket address.

- int rt_dev_getpeername (int fd, struct sockaddr *name, socklen_t *namelen)

    Get socket destination address.

Chapter 8

Xenomai RTDM skin API Example Documentation

8.1 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,
};

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 implicitely 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));
                        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;

        printf(WTASK_PREFIX "exit\n");
}

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(RTASK_PREFIX "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;

        printf(RTASK_PREFIX "exit\n");
}

int main(int argc, char* argv[])
{
        int err = 0;

        signal(SIGTERM, catch_signal);
        signal(SIGINT, catch_signal);

        /* no memory-swapping for this programm */
        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(-err));
                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(-err));
                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:
        cleanup_all();
        return err;
}
```

8.2 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 <sys/mman.h>

#include <rtdm/rtcan.h>

static unsigned int cycle = 10000; /* 10 ms */
static can_id_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>\n"
            "Options:\n"
            " -h, --help     This help\n"
            " -r, --repeater Repeater, send back received messages\n"
            " -i, --id=ID    CAN Identifier (default = 0x1)\n"
            " -c, --cycle    Cycle time in us (default = 10000us)\n",
            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);

    pthread_setschedparam(pthread_self(), SCHED_FIFO, &param);

    clock_gettime(CLOCK_MONOTONIC, &next_period);

    while(1) {
        next_period.tv_nsec += cycle * 1000;
        if (next_period.tv_nsec >= 1000000000) {
            next_period.tv_nsec = 0;
            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 = time.tv_sec * 1000000000LL + time.tv_nsec;

        /* Transmit the message containing the local time */
        if (send(txsock, (void *)&frame, sizeof(can_frame_t), 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,
                                0, 0, 0};
    pthread_setschedparam(pthread_self(), SCHED_FIFO, &param);

    rtt_stat.counts_per_sec = 1000000 / cycle;

    while (1) {
        if (recv(rxsock, (void *)&frame, sizeof(can_frame_t), 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(can_frame_t), 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 = (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", required_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’:
            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, 0)) < 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 definiton 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);
    printf("Cycle time: %d us\n", cycle);
    printf("All RTT timing figures are in us.\n");

    /* Create statistics message queue */
    snprintf(mqname, sizeof(mqname), "/rtcan_rtt-%d", getpid());
    mqattr.mq_flags   = 0;
    mqattr.mq_maxmsg  = 100;
    mqattr.mq_msgsize = sizeof(struct rtt_stat);
    mq = mq_open(mqname, O_RDWR | O_CREAT | O_EXCL, 0600, &mqattr);
    if (mq == (mqd_t)-1) {
        perror("opening mqueue failed");
        goto 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));
        goto failure3;
    }

    if (!repeater) {
        /* Create transitter RT-thread */
        ret = pthread_create(&txthread, &thattr, &transmitter, NULL);
        if (ret) {
            fprintf(stderr, "%s: pthread_create(transmitter) failed\n",
                    strerror(-ret));
            goto failure4;
        }
    }

    pthread_setschedparam(pthread_self(), SCHED_FIFO, &param);

    if (repeater)
        printf("Messages\n");
    else
        printf("Messages RTTlast 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("%8d\n", rxcount);
        } else {
            rtt_avg = ((rtt_stat.rtt_sum - rtt_stat.rtt_sum_last) /
                       rtt_stat.counts_per_sec);
            printf("%8d %7ld %7ld %7ld %7ld %8d\n", rxcount,
                   (long)(rtt_stat.rtt / 1000), (long)(rtt_avg / 1000),
                   (long)(rtt_stat.rtt_min / 1000),
                   (long)(rtt_stat.rtt_max / 1000),
                   overruns);
        }
    }

    /* This call also leaves primary mode, required for socket cleanup. */
    printf("shutting down\n");

    /* Important: First close the sockets! */
    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;
}
```

8.3 rtcanconfig.c

```
/*
 * Program to configuring the CAN controller
 *
 * Copyright (C) 2006 Wolfgang Grandegger <wg@grandegger.com>
 *
 * Copyright (C) 2005, 2006 Sebastian Smolorz
 *                          <Sebastian.Smolorz@stud.uni-hannover.de>
 *
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>
#include <string.h>
#include <time.h>
#include <errno.h>
#include <getopt.h>
#include <sys/mman.h>

#include <rtdm/rtcan.h>

static void print_usage(char *prg)
{
    fprintf(stderr,
            "Usage: %s <can-interface> [Options] [up|down|start|stop|sleep]\n"
            "Options:\n"
            " -v, --verbose            be verbose\n"
            " -h, --help               this help\n"
            " -c, --ctrlmode=M1:M2:... listenonly or loopback mode\n"
            " -b, --baudrate=BPS       baudrate in bits/sec\n"
            " -B, --bittime=BTR0:BTR1  BTR or standard bit-time\n"
            " -B, --bittime=BRP:PROP_SEG:PHASE_SEG1:PHASE_SEG2:SJW:SAM\n",
            prg);
}

can_baudrate_t string_to_baudrate(char *str)
{
    can_baudrate_t baudrate;
    if (sscanf(str, "%i", &baudrate) != 1)
        return -1;
    return baudrate;
}

int string_to_mode(char *str)
{
    if ( !strcmp(str, "up") || !strcmp(str, "start") )
        return CAN_MODE_START;
    else if ( !strcmp(str, "down") || !strcmp(str, "stop") )
        return CAN_MODE_STOP;
    else if ( !strcmp(str, "sleep") )
```

```
        return CAN_MODE_SLEEP;
    return -EINVAL;
}

int string_to_ctrlmode(char *str)
{
    if ( !strcmp(str, "listenonly") )
        return CAN_CTRLMODE_LISTENONLY;
    else if ( !strcmp(str, "loopback") )
        return CAN_CTRLMODE_LOOPBACK;

    return 0;
}

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);

        case 'v':
            verbose = 1;
            break;

        case 'b':
            new_baudrate = string_to_baudrate(optarg);
            if (new_baudrate == -1) {
                print_usage(argv[0]);
                exit(0);
            }
            break;

        case 'B':
            ptr = optarg;
            while (1) {
                bittime_data[bittime_count++] = strtoul(ptr, NULL, 0);
                if (!(ptr = strchr(ptr, ':')))
                    break;
                ptr++;
            }
            if (bittime_count != 2 && bittime_count != 6) {
                print_usage(argv[0]);
                exit(0);
            }
            break;

        case ’c’:
            new_ctrlmode |= string_to_ctrlmode(optarg);
            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;
        }
    }

    can_fd = rt_dev_socket(PF_CAN, SOCK_RAW, 0);
    if (can_fd < 0) {
        fprintf(stderr, "Cannot open RTDM CAN socket. Maybe driver not loaded? \n");
        return can_fd;
    }

    ret = rt_dev_ioctl(can_fd, SIOCGIFINDEX, &ifr);
    if (ret) {
        fprintf(stderr,"Can’t get interface index for %s, code = %d\n", ifname, ret);
        return ret;
    }


    if (new_baudrate != -1) {
        if (verbose)
            printf("baudrate: %d\n", new_baudrate);
        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;
        }

    }

    if (set_ctrlmode != 0) {
        ctrlmode = (can_ctrlmode_t *)&ifr.ifr_ifru;
        *ctrlmode = new_ctrlmode;
        if (verbose)
            printf("ctrlmode: %#x\n", new_ctrlmode);
        ret = rt_dev_ioctl(can_fd, SIOCSCANCTRLMODE, &ifr);
        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;
}
```

8.4 rtcanrecv.c

```
#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/rtcan.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;

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) {
            fprintf(stderr, "rt_dev_close: %s\n", strerror(-ret));
        }
        rt_task_delete(&rt_task_desc);
    }
    }

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");
                continue;
            case -EBADF:
                if (verbose)
                    printf("rt_dev_recv: aborted because socket was closed");
                break;
            default:
                fprintf(stderr, "rt_dev_recv: %s\n", strerror(-ret));
            }
            break;
        }

        if (print && (count % print) == 0) {
            printf("#%d: (%d) ", count, addr.can_ifindex);
            if (with_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);

        case 'p':
            print = strtoul(optarg, NULL, 0);
            break;

        case 'v':
            verbose = 1;
            break;

        case ’e’:
            err_mask = strtoul(optarg, NULL, 0);
            break;

        case ’f’:
            ptr = optarg;
            while (1) {
                id = strtoul(ptr, NULL, 0);
                ptr = strchr(ptr, ’:’);
                if (!ptr) {
                    fprintf(stderr, "filter must be applied in the form id:mask[:id:mask]...\n");
                    exit(1);
                }
                ptr++;
                mask = strtoul(ptr, NULL, 0);
                ptr = strchr(ptr, ’:’);
                add_filter(id, mask);
                if (!ptr)
                    break;
                ptr++;
            }
            break;

        case ’t’:
            timeout = (nanosecs_rel_t)strtoul(optarg, NULL, 0) * 1000000;
            break;

        case ’R’:
            timestamp_rel = 1;
        case ’T’:
            with_timestamp = 1;
            break;

        default:
            fprintf(stderr, "Unknown option %c\n", opt);
            break;
        }
    }

    ret = rt_dev_socket(PF_CAN, SOCK_RAW, 0);
    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;
}
```

8.5 rtcansend.c

```
#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/rtcan.h>

extern int optind, opterr, optopt;

static void print_usage(char *prg)
{
    fprintf(stderr,
            "Usage: %s <can-interface> [Options] <can-msg>\n"
            "<can-msg> can consist of up to 8 bytes given as a space separated list\n"
            "Options:\n"
            " -i, --identifier=ID   CAN Identifier (default = 1)\n"
            " -r  --rtr             send remote request\n"
            " -e  --extended        send extended frame\n"
            " -l  --loop=COUNT      send message COUNT times\n"
            " -c, --count           message count in data[0-3]\n"
            " -d, --delay=MS        delay in ms (default = 1ms)\n"
            " -s, --send            use send instead of sendto\n"
            " -t, --timeout=MS      timeout in ms\n"
            " -L, --loopback=0|1    switch local loopback off or on\n"
            " -v, --verbose         be verbose\n"
            " -p, --print=MODULO    print every MODULO message\n"
            " -h, --help            this help\n",
            prg);
}


RT_TASK rt_task_desc;

static int s=-1, dlc=0, rtr=0, extended=0, verbose=0, loops=1;
static SRTIME delay=1000000;
static int count=0, print=1, use_send=0, loopback=-1;
static nanosecs_rel_t timeout = 0;
static struct can_frame frame;
static struct sockaddr_can to_addr;


void cleanup(void)
{
    int ret;

    if (verbose)
        printf("Cleaning up...\n");

    usleep(100000);

    if (s >= 0) {
        ret = rt_dev_close(s);
        s = -1;
        if (ret) {
            fprintf(stderr, "rt_dev_close: %s\n", strerror(-ret));
        }
        rt_task_delete(&rt_task_desc);
    }
```

```
}

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 definiton 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");
                break;
            case -EBADF:
                if (verbose)
                    printf("rt_dev_send(to): aborted because socket was closed");
                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>", frame.can_id & CAN_EFF_MASK);
            else
                printf("<0x%03x>", frame.can_id & CAN_SFF_MASK);
            printf(" [%d]", frame.can_dlc);
            for (j = 0; j < frame.can_dlc; j++) {
                printf(" %02x", frame.data[j]);
            }
            printf("\n");
        }
    }
}

int main(int argc, char **argv)
{
    int i, opt, ret;
    struct 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);

        case ’p’:
            print = strtoul(optarg, NULL, 0);

        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;
        }
    }

    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, 0);
    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 definiton 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] = strtoul(argv[i], NULL, 0);
            dlc++;
            if( dlc == 8 )
                break;
        }
        frame.can_dlc = dlc;
    }

    if (rtr)
        frame.can_id |= CAN_RTR_FLAG;

    if (extended)
        frame.can_id |= CAN_EFF_FLAG;

    if (timeout) {
        if (verbose)
            printf("Timeout: %lld ns\n", (long long)timeout);
        ret = rt_dev_ioctl(s, RTCAN_RTIOC_SND_TIMEOUT, &timeout);
        if (ret) {
            fprintf(stderr, "rt_dev_ioctl SND_TIMEOUT: %s\n", strerror(-ret));
            goto failure;
        }
    }

    snprintf(name, sizeof(name), "rtcansend-%d", getpid());
    ret = rt_task_shadow(&rt_task_desc, name, 1, 0);
    if (ret) {
        fprintf(stderr, "rt_task_shadow: %s\n", strerror(-ret));
        goto failure;
    }

    rt_task();

    cleanup();
    return 0;

 failure:
    cleanup();
    return -1;
}
```

Index