CAN Device Driver Internals

Pavel Pisa

pisa@cmp.felk.cvut.cz

CAN Device Driver Internals

by Pavel Pisa

The LinCAN is an implementation of the Linux device driver supporting more CAN controller chips and many CAN interface boards. Its implementation has long history already. The OCERA version of the driver adds new features, continuous enhancements and reimplementation of structure of the driver. The usage of the driver is tightly coupled to the virtual CAN API interface component which hides driver low level interface to the application programmers.

This book describes internals of the LinCAN driver.

Table of Contents

Chapter 1. Introduction

The LinCAN driver is the loadable module for the Linux kernel which implements CAN driver. The driver communicates and controls one or more CAN controllers chips. The each chip/CAN interface is represented to the applications as one or more CAN message objects through the character device interface. The application can open the character device and use `read/write` system calls for CAN messages transmission or reception through the connected message object. The parameters of the message object can be modified by the `IOCTL` system call. The closing of the character device releases resources allocated by the application.

The present version of the driver supports three most common CAN controllers:

- Intel i82527 chips
- Philips 82c200 chips
- Philips SJA1000 chips in standard and PeliCAN mode

The intelligent CAN/CANopen cards should be supported by future versions. One of such cards is P-CAN series of cards produced by Unicontrols. The driver contains support for more than ten CAN cards basic types with different combinations of the above mentioned chips. Not all card types are held by OCERA members, but CTU has and tested more SJA1000 type cards and will test some i82527 cards in near future.

Chapter 2. LinCAN Driver API

Driver API Overview

Each driver is a subsystem which has no direct application level API. The operating system is responsible for user space calls transformation into driver functions calls or dispatch routines invocations. The CAN driver is implemented as a character device with the standard device node names `/dev/can0`, `/dev/can1`, etc. The application program communicates with the driver through the standard system low level input/output primitives (`open`, `close`, `read`, `write`, `select` and `ioctl`). The CAN driver convention of usage of these functions is described in the next subsection.

The `read` and `write` functions need to transfer one or more CAN messages. The structure canmsg_t is defined for this purpose and is defined in include file `can/can.h`. The canmsg_t structure has next fields:

```
struct canmsg_t {
    short flags;
    int cob;
    unsigned long id;
    unsigned long timestamp;
    unsigned int length;
    unsigned char
    data[CAN_MSG_LENGTH];
} PACKED;
```

flags

> The flags field holds information about message type. The bit `MSG_RTR` marks remote transmission request messages. Writing of such message into the CAN message object handle results in transmission of the RTR message. The RTR message can be received by the read call if no buffer with corresponding ID is prefilled in the driver. The bit `MSG_EXT` indicates that the message with extended (bit 29 set) ID will be send or was received. The bit `MSG_OVR` is intended for fast indication of the reception message queue overfill.

cob

> The field reserved for a holding message communication object number. It could be used for serialization of received messages from more message object into one message queue in the future.

id

> CAN message ID.

timestamp

> The field intended for storing of the message reception time.

length

> The number of the data bytes send or received in the CAN message. The number of data load bytes is from 0 to 8.

data

> The byte array holding message data.

As was mentioned above, direct communication with the driver through system calls is not encouraged because this interface is partially system dependent and cannot be ported to all environments. The suggested alternative is to use OCERA provided VCA library which defines the portable and clean interface to the CAN driver implementation.

The other issue is addition of the support for new CAN interface boards and CAN controller chips. The subsection Board Support Functions describes template functions, which needs to be implemented for newly supported board. The template of board support can be found in the file `src/template.c`.

The other task for more brave souls is addition of the support for the unsupported chip type. The source supporting the SJA1000 chip in the PeliCAN mode can serve as an example. The full source of this chip support is stored in the file `src/sja1000p.c`. The subsection Chip Support Functions describes basic functions necessary for the new chip support.

CAN Driver File Operations

open

Name

`open` — message communication object open system call

Synopsis

```
int open (const char * pathname, int flags);
```

Arguments

`pathname`

The path to driver device node is specified there. The conventional device names for Linux CAN driver are `/dev/can0`, `/dev/can1`, etc.

`flags`

flags modifying style of open call. The standard `O_RDWR` mode should be used for CAN device. The mode `O_NOBLOCK` can be used with driver as well. This mode results in immediate return of `read` and `write`.

Description

Returns negative number in the case of error. Returns the file descriptor for named CAN message object in other cases.

close

Name

`close` — message communication object close system call

Synopsis

```
int close (int fd);
```

Arguments

`fd`

file descriptor to opened can message communication object

Description

Returns negative number in the case of error.

read

Name

`read` — reads received CAN messages from message object

Synopsis

```
ssize_t read(int fd, void * buf, size_t count);
```

Arguments

`fd`

file descriptor to opened can message communication object

`buf`

pointer to array of canmsg_t structures.

`count`

size of message array buffer in number of bytes

Description

Returns negative value in the case of error else returns number of read bytes which is multiple of canmsg_t structure size.

write

Name

`write` — writes CAN messages to message object for transmission

Synopsis

`ssize_t write(int fd, const void * buf, size_t count);`

Arguments

`fd`

file descriptor to opened can message communication object

`buf`

pointer to array of canmsg_t structures.

`count`

size of message array buffer in number of bytes. The parameter informs driver about number of messages prepared for transmission and should be multiple of canmsg_t structure size.

Description

Returns negative value in the case of error else returns number of bytes successfully stored into message object transmission queue. The positive returned number is multiple of canmsg_t structure size.

Chapter 3. Driver Internal Documentation

Basic Driver Data Structures

struct canhardware_t

Name

`struct canhardware_t` — structure representing pointers to all CAN boards

Synopsis

```
struct canhardware_t {
  int nr_boards;
  struct rtr_id * rtr_queue;
  spinlock_t rtr_lock;
  struct candevice_t * * candevice;
};
```

Members

nr_boards

number of present boards

rtr_queue

RTR - remote transmission request queue (expect some changes there)

rtr_lock

locking for RTR queue

candevice

array of pointers to CAN devices/boards

struct candevice_t

Name

`struct candevice_t` — CAN device/board structure

Synopsis

```
struct candevice_t {
  char * hwname;
  int candev_idx;
  unsigned long io_addr;
  unsigned long res_addr;
  unsigned long dev_base_addr;
  unsigned int flags;
  int nr_all_chips;
  int nr_82527_chips;
  int nr_sja1000_chips;
  struct chip_t * * chip;
  struct hwspecops_t * hwspecops;
  struct canhardware_t * hosthardware_p;
};
```

Members

hwname

text string with board type

candev_idx

board index in canhardware_t.candevice[]

io_addr

IO/physical MEM address

res_addr

optional reset register port

dev_base_addr

CPU translated IO/virtual MEM address

flags

board flags: `PROGRAMMABLE_IRQ` .. interrupt number can be programmed into board

nr_all_chips

number of chips present on the board

nr_82527_chips

number of Intel 8257 chips

nr_sja1000_chips

number of Philips SJA100 chips

chip

array of pointers to the chip structures

hwspecops

pointer to board specific operations

hosthardware_p

pointer to the root hardware structure

Description

The structure represent configuration and state of associated board. The driver infrastructure prepares this structure and calls board type specific `board_register` function. The board support provided register function fills right function pointers in *`hwspecops`* structure. Then driver setup calls functions `init_hw_data`, `init_chip_data`, `init_chip_data`, `init_obj_data` and `program_irq`. Function `init_hw_data` and `init_chip_data` have to specify number and types of connected chips or objects respectively. The use of *`nr_all_chips`* is preferred over use of fields *`nr_82527_chips`* and *`nr_sja1000_chips`* in the board non-specific functions. The *`io_addr`* and *`dev_base_addr`* is filled from module parameters to the same value. The request_io function can fix-up *`dev_base_addr`* field if virtual address is different than bus address.

struct chip_t

Name

`struct chip_t` — CAN chip state and type information

Synopsis

```
struct chip_t {
  char * chip_type;
  int chip_idx;
  int chip_irq;
  unsigned long chip_base_addr;
  unsigned int flags;
  int clock;
  void (* write_register (unsigned char data,unsigned long address);
  unsigned (* read_register (unsigned long address);
  unsigned short sja_cdr_reg;
  unsigned short sja_ocr_reg;
  unsigned short int_cpu_reg;
  unsigned short int_clk_reg;
  unsigned short int_bus_reg;
  struct msgobj_t * * msgobj;
  struct chipspecops_t * chipspecops;
  struct candevice_t * hostdevice;
  int max_objects;
};
```

Members

chip_type

text string describing chip type

chip_idx

index of the chip in candevice_t.chip[] array

chip_irq

chip interrupt number if any

chip_base_addr

chip base address in the CPU IO or virtual memory space

flags

chip flags: `CHIP_CONFIGURED` .. chip is configured, `CHIP_SEGMENTED` .. access to the chip is segmented (mainly for i82527 chips)

clock

chip base clock frequency in Hz

write_register

write chip register function copy -

read_register

read chip register function copy

sja_cdr_reg

SJA specific register - holds hardware specific options for the Clock Divider register. Options defined in the sja1000.h file: `CDR_CLKOUT_MASK`, `CDR_CLK_OFF`, `CDR_RXINPEN`, `CDR_CBP`, `CDR_PELICAN` sja_ocr_reg

SJA specific register - hold hardware specific options for the Output Control register. Options defined in the sja1000.h file: `OCR_MODE_BIPHASE`, `OCR_MODE_TEST`, `OCR_MODE_NORMAL`, `OCR_MODE_CLOCK`, `OCR_TX0_LH`, `OCR_TX1_ZZ`.

int_cpu_reg

Intel specific register - holds hardware specific options for the CPU Interface register. Options defined in the i82527.h file: `iCPU_CEN`, `iCPU_MUX`, `iCPU_SLP`, `iCPU_PWD`, `iCPU_DMC`, `iCPU_DSC`, `iCPU_RST`.

int_clk_reg

Intel specific register - holds hardware specific options for the Clock Out register. Options defined in the i82527.h file: `iCLK_CD0`, `iCLK_CD1`, `iCLK_CD2`, `iCLK_CD3`, `iCLK_SL0`, `iCLK_SL1`.

int_bus_reg

Intel specific register - holds hardware specific options for the Bus Configuration register. Options defined in the i82527.h file: `iBUS_DR0`, `iBUS_DR1`, `iBUS_DT1`, `iBUS_POL`, `iBUS_CBY`.

msgobj

array of pointers to individual communication objects

chipspecops

pointer to the set of chip specific object filled by `init_chip_data` function

hostdevice

pointer to chip hosting board

max_objects

maximal number of communication objects connected to this chip

Description

The fields *`write_register`* and *`read_register`* are copied from corresponding fields from ***`hwspecops`*** structure (chip->hostdevice->hwspecops->write_register and chip->hostdevice->hwspecops->read_register) to speedup `can_write_reg` and `can_read_reg` functions.

struct msgobj_t

Name

`struct msgobj_t` — structure holding communication object state

Synopsis

```
struct msgobj_t {
  unsigned long obj_base_addr;
  unsigned int minor;
  unsigned int object;
  unsigned long flags;
  int ret;
  struct canque_ends_t * qends;
  struct canque_edge_t * tx_qedge;
  struct canque_slot_t * tx_slot;
  int tx_retry_cnt;
  struct canmsg_t rx_msg;
  struct chip_t * hostchip;
  atomic_t obj_used;
  struct list_head obj_users;
};
```

Members

obj_base_addr

minor

associated device minor number

object

object number in chip_t structure +1

flags

message object flags

ret

field holding status of the last Tx operation

qends

pointer to message object corresponding ends structure

tx_qedge

edge corresponding to transmitted message

tx_slot

slot holding transmitted message, slot is taken from `canque_test_outslot` call and is freed by `canque_free_outslot` or rescheduled `canque_again_outslot`

tx_retry_cnt

transmission attempt counter

rx_msg

temporary storage to hold received messages before calling to `canque_filter_msg2edges`

hostchip

pointer to the &chip_t structure this object belongs to

obj_used

counter of users (associated file structures for Linux userspace clients) of this object

obj_users

list of user structures of type &canuser_t.

struct canuser_t

Name

`struct canuser_t` — structure holding CAN user/client state

Synopsis

```
struct canuser_t {
  struct list_head peers;
  struct canque_ends_t * qends;
  struct file * file;
  struct msgobj_t * msgobj;
  struct canque_edge_t * rx_edge0;
  int magic;
};
```

Members

peers

for connection into list of object users

qends

pointer to the ends structure corresponding for this user

file

pointer to open device file state structure

msgobj

communication object the user is connected to

rx_edge0

default receive queue for filter IOCTL

magic

magic number to check consistency when pointer is retrieved from file private field

struct hwspecops_t

Name

`struct hwspecops_t` — hardware/board specific operations

Synopsis

```
struct hwspecops_t {
  int (* request_io (struct candevice_t *candev);
  int (* release_io (struct candevice_t *candev);
  int (* reset (struct candevice_t *candev);
  int (* init_hw_data (struct candevice_t *candev);
  int (* init_chip_data (struct candevice_t *candev, int chipnr);
  int (* init_obj_data (struct chip_t *chip, int objnr);
  int (* program_irq (struct candevice_t *candev);
  void (* write_register (unsigned char data,unsigned long address);
  unsigned (* read_register (unsigned long address);
};
```

Members

request_io

reserve io or memory range for can board

release_io

free reserved io memory range

reset

hardware reset routine

init_hw_data

called to initialize &candevice_t structure, mainly *`res_add`*, *`nr_all_chips`*, *`nr_82527_chips`*, *`nr_sja1000_chips`* and *`flags`* fields

init_chip_data

called initialize each &chip_t structure, mainly *`chip_type`*, *`chip_base_addr`*, *`clock`* and chip specific registers. It is responsible to setup &chip_t->*`chipspecops`* functions for non-standard chip types (type other than “i82527”, “sja1000” or “sja1000p”)

init_obj_data

called initialize each &msgobj_t structure, mainly *`obj_base_addr`* field.

program_irq

program interrupt generation hardware of the board if flag `PROGRAMMABLE_IRQ` is present for specified device/board

write_register

low level write register routine

read_register

low level read register routine

struct chipspecops_t

Name

`struct chipspecops_t` — can controller chip specific operations

Synopsis

```
struct chipspecops_t {
  int (* chip_config (struct chip_t *chip);
  int (* baud_rate (struct chip_t *chip, int rate, int clock, int sjw,int sampl_pt, int
  int (* standard_mask (struct chip_t *chip, unsigned short code,unsigned short mask);
  int (* extended_mask (struct chip_t *chip, unsigned long code,unsigned long mask);
  int (* message15_mask (struct chip_t *chip, unsigned long code,unsigned long mask);
  int (* clear_objects (struct chip_t *chip);
  int (* config_irqs (struct chip_t *chip, short irqs);
  int (* pre_read_config (struct chip_t *chip, struct msgobj_t *obj);
  int (* pre_write_config (struct chip_t *chip, struct msgobj_t *obj,struct canmsg_t *m
  int (* send_msg (struct chip_t *chip, struct msgobj_t *obj,struct canmsg_t *msg);
  int (* remote_request (struct chip_t *chip, struct msgobj_t *obj);
  int (* check_tx_stat (struct chip_t *chip);
  int (* wakeup_tx (struct chip_t *chip, struct msgobj_t *obj);
  int (* enable_configuration (struct chip_t *chip);
  int (* disable_configuration (struct chip_t *chip);
  int (* set_btregs (struct chip_t *chip, unsigned short btr0,unsigned short btr1);
  int (* start_chip (struct chip_t *chip);
  int (* stop_chip (struct chip_t *chip);
  irqreturn_t (* irq_handler (int irq, void *dev_id, struct pt_regs *regs);
};
```

Members

chip_config

CAN chip configuration

baud_rate

set communication parameters

standard_mask

setup of mask for message filtering

extended_mask

setup of extended mask for message filtering

message15_mask

set mask of i82527 message object 15

clear_objects

clears state of all message object residing in chip

config_irqs

tunes chip hardware interrupt delivery

pre_read_config

prepares message object for message reception

pre_write_config

prepares message object for message transmission

send_msg

initiate message transmission

remote_request

configures message object and asks for RTR message

check_tx_stat

checks state of transmission engine

wakeup_tx

wakeup TX processing

enable_configuration

enable chip configuration mode

disable_configuration

disable chip configuration mode

set_btregs

configures bitrate registers

start_chip

starts chip message processing

stop_chip

stops chip message processing

irq_handler

interrupt service routine

Board Support Functions

The functions, which should be implemented for each supported board, are described in the next section. The functions are prefixed by boardname. The prefix *`template`* has been selected for next description.

template_request_io

Name

`template_request_io` — reserve io or memory range for can board

Synopsis

```
int template_request_io (struct candevice_t * candev);
```

Arguments

`candev`

pointer to candevice/board which asks for io. Field *`io_addr`* of *`candev`* is used in most cases to define start of the range

Description

The function `template_request_io` is used to reserve the io-memory. If your hardware uses a dedicated memory range as hardware control registers you will have to add the code to reserve this memory as well. `IO_RANGE` is the io-memory range that gets reserved, please adjust according your hardware. Example: #define IO_RANGE 0x100 for i82527 chips or #define IO_RANGE 0x20 for sja1000 chips in basic CAN mode.

Return Value

The function returns zero on success or `-ENODEV` on failure

File

src/template.c

template_release_io

Name

`template_release_io` — free reserved io memory range

Synopsis

```
int template_release_io (struct candevice_t * candev);
```

Arguments

`candev`

pointer to candevice/board which releases io

Description

The function `template_release_io` is used to free reserved io-memory. In case you have reserved more io memory, don't forget to free it here. IO_RANGE is the io-memory range that gets released, please adjust according your hardware. Example: #define IO_RANGE 0x100 for i82527 chips or #define IO_RANGE 0x20 for sja1000 chips in basic CAN mode.

Return Value

The function always returns zero

File

src/template.c

template_reset

Name

`template_reset` — hardware reset routine

Synopsis

```
int template_reset (struct candevice_t * candev);
```

Arguments

candev

Pointer to candevice/board structure

Description

The function `template_reset` is used to give a hardware reset. This is rather hardware specific so I haven't included example code. Don't forget to check the reset status of the chip before returning.

Return Value

The function returns zero on success or `-ENODEV` on failure

File

src/template.c

template_init_hw_data

Name

`template_init_hw_data` — Initialize hardware cards

Synopsis

```
int template_init_hw_data (struct candevice_t * candev);
```

Arguments

candev

Pointer to candevice/board structure

Description

The function `template_init_hw_data` is used to initialize the hardware structure containing information about the installed CAN-board. `RESET_ADDR` represents the io-address of the hardware reset register. `NR_82527` represents the number of Intel 82527 chips on the board. `NR_SJA1000` represents the number of Philips sja1000 chips on the board. The flags entry can currently only be `PROGRAMMABLE_IRQ` to indicate that the hardware uses programmable interrupts.

Return Value

The function always returns zero

File

src/template.c

template_init_chip_data

Name

`template_init_chip_data` — Initialize chips

Synopsis

```
int template_init_chip_data (struct candevice_t * candev, int chipnr);
```

Arguments

`candev`

Pointer to candevice/board structure

`chipnr`

Number of the CAN chip on the hardware card

Description

The function `template_init_chip_data` is used to initialize the hardware structure containing information about the CAN chips. `CHIP_TYPE` represents the type of CAN chip. `CHIP_TYPE` can be “i82527” or “sja1000”. The *`chip_base_addr`* entry represents the start of the 'official' memory map of the installed chip. It's likely that this is the same as the *`io_addr`* argument supplied at module loading time. The *`clock`* entry holds the chip clock value in Hz. The entry *`sja_cdr_reg`* holds hardware specific options for the Clock Divider register. Options defined in the `sja1000`.h file: `CDR_CLKOUT_MASK`, `CDR_CLK_OFF`, `CDR_RXINPEN`, `CDR_CBP`, `CDR_PELICAN` The entry *`sja_ocr_reg`* holds hardware specific options for the Output Control register. Options defined in the `sja1000`.h file: `OCR_MODE_BIPHASE`, `OCR_MODE_TEST`, `OCR_MODE_NORMAL`, `OCR_MODE_CLOCK`, `OCR_TX0_LH`, `OCR_TX1_ZZ`. The entry *`int_clk_reg`* holds hardware specific options for the Clock Out register. Options defined in the `i82527`.h file: `iCLK_CD0`, `iCLK_CD1`, `iCLK_CD2`, `iCLK_CD3`, `iCLK_SL0`, `iCLK_SL1`. The entry *`int_bus_reg`* holds hardware specific options for the Bus Configuration register. Options defined in the `i82527`.h file: `iBUS_DR0`, `iBUS_DR1`, `iBUS_DT1`, `iBUS_POL`, `iBUS_CBY`. The entry *`int_cpu_reg`* holds hardware specific options for the cpu interface register. Options defined in the `i82527`.h file: `iCPU_CEN`, `iCPU_MUX`, `iCPU_SLP`, `iCPU_PWD`, `iCPU_DMC`, `iCPU_DSC`, `iCPU_RST`.

Return Value

The function always returns zero

File

src/template.c

template_init_obj_data

Name

`template_init_obj_data` — Initialize message buffers

Synopsis

```
int template_init_obj_data (struct chip_t * chip, int objnr);
```

Arguments

`chip`

Pointer to chip specific structure

`objnr`

Number of the message buffer

Description

The function `template_init_obj_data` is used to initialize the hardware structure containing information about the different message objects on the CAN chip. In case of the sja1000 there's only one message object but on the i82527 chip there are 15. The code below is for a i82527 chip and initializes the object base addresses The entry *`obj_base_addr`* represents the first memory address of the message object. In case of the sja1000 *`obj_base_addr`* is taken the same as the chips base address. Unless the hardware uses a segmented memory map, flags can be set zero.

Return Value

The function always returns zero

File

src/template.c

template_program_irq

Name

`template_program_irq` — program interrupts

Synopsis

```
int template_program_irq (struct candevice_t * candev);
```

Arguments

`candev`

Pointer to candevice/board structure

Description

The function `template_program_irq` is used for hardware that uses programmable interrupts. If your hardware doesn't use programmable interrupts you should not set the *`candevices_t`*->flags entry to `PROGRAMMABLE_IRQ` and leave this function unedited. Again this function is hardware specific so there's no example code.

Return value

The function returns zero on success or `-ENODEV` on failure

File

src/template.c

template_write_register

Name

`template_write_register` — Low level write register routine

Synopsis

```
void template_write_register (unsigned char data, unsigned long
address);
```

Arguments

`data`

data to be written

`address`

memory address to write to

Description

The function `template_write_register` is used to write to hardware registers on the CAN chip. You should only have to edit this function if your hardware uses some specific write process.

Return Value

The function does not return a value

File

src/template.c

template_read_register

Name

`template_read_register` — Low level read register routine

Synopsis

```
unsigned template_read_register (unsigned long address);
```

Arguments

`address`

memory address to read from

Description

The function `template_read_register` is used to read from hardware registers on the CAN chip. You should only have to edit this function if your hardware uses some specific read process.

Return Value

The function returns the value stored in *`address`*

File

src/template.c

Chip Support Functions

The controller chip specific functions are described in the next section. The functions should be prefixed by chip type. Because documentation of chip functions has been retrieved from the actual SJA1000 PeliCAN support, the function prefix is *`sja1000p`*.

sja1000p_enable_configuration

Name

`sja1000p_enable_configuration` — enable chip configuration mode

Synopsis

```
int sja1000p_enable_configuration (struct chip_t * chip);
```

Arguments

`chip`

pointer to chip state structure

sja1000p_disable_configuration

Name

`sja1000p_disable_configuration` — disable chip configuration mode

Synopsis

```
int sja1000p_disable_configuration (struct chip_t * chip);
```

Arguments

`chip`

pointer to chip state structure

sja1000p_chip_config

Name

`sja1000p_chip_config` — can chip configuration

Synopsis

```
int sja1000p_chip_config (struct chip_t * chip);
```

Arguments

`chip`

pointer to chip state structure

Description

This function configures chip and prepares it for message transmission and reception. The function resets chip, resets mask for acceptance of all messages by call to `sja1000p_extended_mask` function and then computes and sets baudrate with use of function `sja1000p_baud_rate`.

Return Value

negative value reports error.

File

src/sja1000p.c

sja1000p_extended_mask

Name

`sja1000p_extended_mask` — setup of extended mask for message filtering

Synopsis

```
int sja1000p_extended_mask (struct chip_t * chip, unsigned long code,
unsigned long mask);
```

Arguments

`chip`

pointer to chip state structure

`code`

can message acceptance code

`mask`

can message acceptance mask

Return Value

negative value reports error.

File

src/sja1000p.c

sja1000p_baud_rate

Name

`sja1000p_baud_rate` — set communication parameters.

Synopsis

```
int sja1000p_baud_rate (struct chip_t * chip, int rate, int clock, int
sjw, int sampl_pt, int flags);
```

Arguments

`chip`

pointer to chip state structure

`rate`

baud rate in Hz

`clock`

frequency of sja1000 clock in Hz (ISA osc is 14318000)

`sjw`

synchronization jump width (0-3) prescaled clock cycles

`sampl_pt`

sample point in % (0-100) sets (TSEG1+1)/(TSEG1+TSEG2+2) ratio

`flags`

fields `BTR1_SAM`, `OCMODE`, `OCPOL`, `OCTP`, `OCTN`, `CLK_OFF`, `CBP`

Return Value

negative value reports error.

File

src/sja1000p.c

sja1000p_read

Name

`sja1000p_read` — reads and distributes one or more received messages

Synopsis

```
void sja1000p_read (struct chip_t * chip, struct msgobj_t * obj);
```

Arguments

`chip`

pointer to chip state structure

`obj`

pinter to CAN message queue information

File

src/sja1000p.c

sja1000p_pre_read_config

Name

`sja1000p_pre_read_config` — prepares message object for message reception

Synopsis

```
int sja1000p_pre_read_config (struct chip_t * chip, struct msgobj_t *
obj);
```

Arguments

`chip`

pointer to chip state structure

`obj`

pointer to message object state structure

Return Value

negative value reports error. Positive value indicates immediate reception of message.

File

src/sja1000p.c

sja1000p_pre_write_config

Name

`sja1000p_pre_write_config` — prepares message object for message transmission

Synopsis

```
int sja1000p_pre_write_config (struct chip_t * chip, struct msgobj_t *
obj, struct canmsg_t * msg);
```

Arguments

`chip`

pointer to chip state structure

`obj`

pointer to message object state structure

`msg`

pointer to CAN message

Description

This function prepares selected message object for future initiation of message transmission by `sja1000p_send_msg` function. The CAN message data and message ID are transfered from *`msg`* slot into chip buffer in this function.

Return Value

negative value reports error.

File

src/sja1000p.c

sja1000p_send_msg

Name

`sja1000p_send_msg` — initiate message transmission

Synopsis

```
int sja1000p_send_msg (struct chip_t * chip, struct msgobj_t * obj,
struct canmsg_t * msg);
```

Arguments

`chip`

pointer to chip state structure

`obj`

pointer to message object state structure

`msg`

pointer to CAN message

Description

This function is called after `sja1000p_pre_write_config` function, which prepares data in chip buffer.

Return Value

negative value reports error.

File

src/sja1000p.c

sja1000p_check_tx_stat

Name

`sja1000p_check_tx_stat` — checks state of transmission engine

Synopsis

```
int sja1000p_check_tx_stat (struct chip_t * chip);
```

Arguments

`chip`

pointer to chip state structure

Return Value

negative value reports error. Positive return value indicates transmission under way status. Zero value indicates finishing of all issued transmission requests.

File

src/sja1000p.c

sja1000p_set_btregs

Name

`sja1000p_set_btregs` — configures bitrate registers

Synopsis

```
int sja1000p_set_btregs (struct chip_t * chip, unsigned short btr0,
unsigned short btr1);
```

Arguments

`chip`

pointer to chip state structure

`btr0`

bitrate register 0

`btr1`

bitrate register 1

Return Value

negative value reports error.

File

src/sja1000p.c

sja1000p_start_chip

Name

`sja1000p_start_chip` — starts chip message processing

Synopsis

```
int sja1000p_start_chip (struct chip_t * chip);
```

Arguments

`chip`
: pointer to chip state structure

Return Value

negative value reports error.

File

src/sja1000p.c

sja1000p_stop_chip

Name

`sja1000p_stop_chip` — stops chip message processing

Synopsis

```
int sja1000p_stop_chip (struct chip_t * chip);
```

Arguments

`chip`
: pointer to chip state structure

Return Value

negative value reports error.

File

src/sja1000p.c

sja1000p_remote_request

Name

`sja1000p_remote_request` — configures message object and asks for RTR message

Synopsis

```
int sja1000p_remote_request (struct chip_t * chip, struct msgobj_t *
obj);
```

Arguments

`chip`

pointer to chip state structure

`obj`

pointer to message object structure

Return Value

negative value reports error.

File

src/sja1000p.c

sja1000p_standard_mask

Name

`sja1000p_standard_mask` — setup of mask for message filtering

Synopsis

```
int sja1000p_standard_mask (struct chip_t * chip, unsigned short code,
unsigned short mask);
```

Arguments

`chip`

pointer to chip state structure

`code`

can message acceptance code

`mask`

can message acceptance mask

Return Value

negative value reports error.

File

src/sja1000p.c

sja1000p_clear_objects

Name

`sja1000p_clear_objects` — clears state of all message object residing in chip

Synopsis

```
int sja1000p_clear_objects (struct chip_t * chip);
```

Arguments

`chip`

pointer to chip state structure

Return Value

negative value reports error.

File

src/sja1000p.c

sja1000p_config_irqs

Name

`sja1000p_config_irqs` — tunes chip hardware interrupt delivery

Synopsis

```
int sja1000p_config_irqs (struct chip_t * chip, short irqs);
```

Arguments

`chip`

pointer to chip state structure

`irqs`

requested chip IRQ configuration

Return Value

negative value reports error.

File

src/sja1000p.c

sja1000p_irq_write_handler

Name

`sja1000p_irq_write_handler` — part of ISR code responsible for transmit events

Synopsis

```
void sja1000p_irq_write_handler (struct chip_t * chip, struct msgobj_t
* obj);
```

Arguments

`chip`

pointer to chip state structure

`obj`

pointer to attached queue description

Description

The main purpose of this function is to read message from attached queues and transfer message contents into CAN controller chip. This subroutine is called by `sja1000p_irq_write_handler` for transmit events.

File

src/sja1000p.c

sja1000p_irq_handler

Name

`sja1000p_irq_handler` — interrupt service routine

Synopsis

```
irqreturn_t sja1000p_irq_handler (int irq, void * dev_id, struct
pt_regs * regs);
```

Arguments

`irq`

interrupt vector number, this value is system specific

`dev_id`

driver private pointer registered at time of `request_irq` call. The CAN driver uses this pointer to store relationship of interrupt to chip state structure - *`struct`* chip_t

`regs`

system dependent value pointing to registers stored in exception frame

Description

Interrupt handler is activated when state of CAN controller chip changes, there is message to be read or there is more space for new messages or error occurs. The receive events results in reading of the message from CAN controller chip and distribution of message through attached message queues.

File

src/sja1000p.c

sja1000p_wakeup_tx

Name

`sja1000p_wakeup_tx` — wakeups TX processing

Synopsis

```
int sja1000p_wakeup_tx (struct chip_t * chip, struct msgobj_t * obj);
```

Arguments

chip

pointer to chip state structure

obj

pointer to message object structure

Return Value

negative value reports error.

File

src/sja1000p.c

CAN Queues Structures and Functions

struct canque_slot_t

Name

`struct canque_slot_t` — one CAN message slot in the CAN FIFO queue

Synopsis

```
struct canque_slot_t {
  struct canque_slot_t * next;
  unsigned long slot_flags;
  struct canmsg_t msg;
};
```

Members

next

pointer to the next/younger slot

slot_flags

space for flags and optional command describing action associated with slot data

msg

space for one CAN message

Description

This structure is used to store CAN messages in the CAN FIFO queue.

struct canque_fifo_t

Name

`struct canque_fifo_t` — CAN FIFO queue representation

Synopsis

```
struct canque_fifo_t {
  unsigned long fifo_flags;
  unsigned long error_code;
  struct canque_slot_t * head;
  struct canque_slot_t ** tail;
  struct canque_slot_t * flist;
  struct canque_slot_t * entry;
  spinlock_t fifo_lock;
};
```

Members

fifo_flags

this field holds global flags describing state of the FIFO. `CAN_FIFOF_ERROR` is set when some error condition occurs. `CAN_FIFOF_ERR2BLOCK` defines, that error should lead to the FIFO block state. `CAN_FIFOF_BLOCK` state blocks insertion of the next messages. `CAN_FIFOF_OVERRUN` attempt to acquire new slot, when FIFO is full. `CAN_FIFOF_FULL` indicates FIFO full state. `CAN_FIFOF_EMPTY` indicates no allocated slot in the FIFO. `CAN_FIFOF_DEAD` condition indication. Used when FIFO is beeing destroyed.

error_code

futher description of error condition

head

pointer to the FIFO head, oldest slot

tail

pointer to the location, where pointer to newly inserted slot should be added

flist

pointer to list of the free slots associated with queue

entry

pointer to the memory allocated for the list slots.

fifo_lock

the lock to ensure atomicity of slot manipulation operations.

Description

This structure represents CAN FIFO queue. It is implemented as a single linked list of slots prepared for processing. The empty slots are stored in single linked list (*`flist`*).

canque_fifo_get_inslot

Name

`canque_fifo_get_inslot` — allocate slot for the input of one CAN message

Synopsis

```
int canque_fifo_get_inslot (struct canque_fifo_t * fifo, struct
canque_slot_t ** slotp, int cmd);
```

Arguments

`fifo`

pointer to the FIFO structure

`slotp`

pointer to location to store pointer to the allocated slot.

`cmd`

optional command associated with allocated slot.

Return Value

The function returns negative value if there is no free slot in the FIFO queue.

canque_fifo_put_inslot

Name

`canque_fifo_put_inslot` — releases slot to further processing

Synopsis

```
int canque_fifo_put_inslot (struct canque_fifo_t * fifo, struct
canque_slot_t * slot);
```

Arguments

`fifo`

pointer to the FIFO structure

`slot`

pointer to the slot previously acquired by `canque_fifo_get_inslot`.

Return Value

The nonzero return value indicates, that the queue was empty before call to the function. The caller should wake-up output side of the queue.

canque_fifo_abort_inslot

Name

`canque_fifo_abort_inslot` — release and abort slot

Synopsis

```
int canque_fifo_abort_inslot (struct canque_fifo_t * fifo, struct
canque_slot_t * slot);
```

Arguments

`fifo`

pointer to the FIFO structure

`slot`

pointer to the slot previously acquired by `canque_fifo_get_inslot`.

Return Value

The nonzero value indicates, that fifo was full

canque_fifo_test_outslot

Name

`canque_fifo_test_outslot` — test and get ready slot from the FIFO

Synopsis

```
int canque_fifo_test_outslot (struct canque_fifo_t * fifo, struct
canque_slot_t ** slotp);
```

Arguments

`fifo`

pointer to the FIFO structure

`slotp`

pointer to location to store pointer to the oldest slot from the FIFO.

Return Value

The negative value indicates, that queue is empty. The positive or zero value represents command stored into slot by the call to the function `canque_fifo_get_inslot`. The successfully acquired FIFO output slot has to be released by the call `canque_fifo_free_outslot` or `canque_fifo_again_outslot`.

canque_fifo_free_outslot

Name

`canque_fifo_free_outslot` — free processed FIFO slot

Synopsis

```
int canque_fifo_free_outslot (struct canque_fifo_t * fifo, struct
canque_slot_t * slot);
```

Arguments

`fifo`

pointer to the FIFO structure

`slot`

pointer to the slot previously acquired by `canque_fifo_test_outslot`.

Return Value

The returned value informs about FIFO state change. The mask `CAN_FIFOF_FULL` indicates, that the FIFO was full before the function call. The mask `CAN_FIFOF_EMPTY` informs, that last ready slot has been processed.

canque_fifo_again_outslot

Name

`canque_fifo_again_outslot` — interrupt and postpone processing of the slot

Synopsis

```
int canque_fifo_again_outslot (struct canque_fifo_t * fifo, struct
canque_slot_t * slot);
```

Arguments

`fifo`

pointer to the FIFO structure

`slot`

pointer to the slot previously acquired by `canque_fifo_test_outslot`.

Return Value

The function cannot fail..

struct canque_edge_t

Name

`struct canque_edge_t` — CAN message delivery subsystem graph edge

Synopsis

```
struct canque_edge_t {
  struct canque_fifo_t fifo;
  unsigned long filtid;
  unsigned long filtmask;
  struct list_head inpeers;
  struct list_head outpeers;
  struct canque_ends_t * inends;
  struct canque_ends_t * outends;
  atomic_t edge_used;
  int edge_prio;
  int edge_num;
};
```

Members

fifo

place where primitive *`struct`* canque_fifo_t FIFO is located.

filtid

the possible CAN message identifiers filter.

filtmask

the filter mask, the comparison considers only *`filtid`* bits corresponding to set bits in the *`filtmask`* field.

inpeers

the lists of all peer FIFOs connected by their input side (*`inends`*) to the same terminal (*`struct`* canque_ends_t).

outpeers

the lists of all peer FIFOs connected by their output side (*`outends`*) to the same terminal (*`struct`* canque_ends_t).

inends

the pointer to the FIFO input side terminal (*`struct`* canque_ends_t).

outends

the pointer to the FIFO output side terminal (*`struct`* canque_ends_t).

edge_used

the atomic usage counter, mainly used for safe destruction of the edge.

edge_prio

the assigned queue priority from the range 0 to `CANQUEUE_PRIO_NR-1`

edge_num

edge sequential number intended for debugging purposes only

Description

This structure represents one direction connection from messages source (*`inends`*) to message consumer (*`outends`*) fifo ends hub. The edge contains &struct canque_fifo_t for message fifo implementation.

struct canque_ends_t

Name

`struct canque_ends_t` — CAN message delivery subsystem graph vertex (FIFO ends)

Synopsis

```
struct canque_ends_t {
  struct list_head * active;
  struct list_head idle;
  struct list_head inlist;
  spinlock_t ends_lock;
  void (* notify (struct canque_ends_t *qends, struct canque_edge_t *qedge, int what);
  void * context;
  union endinfo;
};
```

Members

active

the array of the lists of active edges directed to the ends structure with ready messages. The array is indexed by the edges priorities.

idle

the list of the edges directed to the ends structure with empty FIFOs.

inlist

the list of outgoing edges input sides.

ends_lock

the lock synchronizing operations between threads accessing same ends structure.

notify

pointer to notify procedure. The next state changes are notified. `CANQUEUE_NOTIFY_EMPTY` (out->in call) - all slots are processed by FIFO out side. `CANQUEUE_NOTIFY_SPACE` (out->in call) - full state negated => there is space for new message. `CANQUEUE_NOTIFY_PROC` (in->out call) - empty state negated => out side is requested to process slots. `CANQUEUE_NOTIFY_NOUSR` (both) - notify, that the last user has released the edge usage called with some lock to prevent edge disappear. `CANQUEUE_NOTIFY_DEAD` (both) - edge is in progress of deletion. `CANQUEUE_NOTIFY_ATACH` (both) - new edge has been attached to end. `CANQUEUE_NOTIFY_FILTCH` (out->in call) - edge filter rules changed `CANQUEUE_NOTIFY_ERROR` (out->in call) - error in messages processing.

context

space to store ends user specific information

endinfo

space to store some other ends usage specific informations mainly for waking-up by the notify calls.

Description

Structure represents place to connect edges to for CAN communication entity. The zero, one or more incoming and outgoing edges can be connected to this structure.

canque_notify_inends

Name

`canque_notify_inends` — request to send notification to the input ends

Synopsis

```
void canque_notify_inends (struct canque_edge_t * qedge, int what);
```

Arguments

`qedge`

pointer to the edge structure

`what`

notification type

canque_notify_outends

Name

`canque_notify_outends` — request to send notification to the output ends

Synopsis

```
void canque_notify_outends (struct canque_edge_t * qedge, int what);
```

Arguments

`qedge`

pointer to the edge structure

`what`

notification type

canque_notify_bothends

Name

`canque_notify_bothends` — request to send notification to the both ends

Synopsis

```
void canque_notify_bothends (struct canque_edge_t * qedge, int what);
```

Arguments

`qedge`

pointer to the edge structure

`what`

notification type

canque_activate_edge

Name

`canque_activate_edge` — mark output end of the edge as active

Synopsis

```
void canque_activate_edge (struct canque_ends_t * inends, struct
canque_edge_t * qedge);
```

Arguments

`inends`

input side of the edge

`qedge`

pointer to the edge structure

Description

Function call moves output side of the edge from idle onto active edges list.

canque_filtid2internal

Name

`canque_filtid2internal` — converts message ID and filter flags into internal format

Synopsis

```
unsigned int canque_filtid2internal (unsigned long id, int filtflags);
```

Arguments

`id`

CAN message 11 or 29 bit identifier

`filtflags`

CAN message flags

Description

This function maps message ID and `MSG_RTR`, `MSG_EXT` and `MSG_LOCAL` into one 32 bit number

canque_fifo_flush_slots

Name

`canque_fifo_flush_slots` — free all ready slots from the FIFO

Synopsis

```
int canque_fifo_flush_slots (struct canque_fifo_t * fifo);
```

Arguments

`fifo`

pointer to the FIFO structure

Description

The caller should be prepared to handle situations, when some slots are held by input or output side slots processing. These slots cannot be flushed or their processing interrupted.

Return Value

The nonzero value indicates, that queue has not been empty before the function call.

canque_fifo_init_slots

Name

`canque_fifo_init_slots` — initialize one CAN FIFO

Synopsis

```
int canque_fifo_init_slots (struct canque_fifo_t * fifo, int slotsnr);
```

Arguments

`fifo`

pointer to the FIFO structure

`slotsnr`

number of requested slots

Return Value

The negative value indicates, that there is no memory to allocate space for the requested number of the slots.

canque_fifo_done

Name

`canque_fifo_done` — frees slots allocated for CAN FIFO

Synopsis

```
int canque_fifo_done (struct canque_fifo_t * fifo);
```

Arguments

`fifo`

pointer to the FIFO structure

canque_get_inslot

Name

`canque_get_inslot` — finds one outgoing edge and allocates slot from it

Synopsis

```
int canque_get_inslot (struct canque_ends_t * qends, struct
canque_edge_t ** qedgep, struct canque_slot_t ** slotp, int cmd);
```

Arguments

`qends`

ends structure belonging to calling communication object

`qedgep`

place to store pointer to found edge

`slotp`

place to store pointer to allocated slot

`cmd`

command type for slot

Description

Function looks for the first non-blocked outgoing edge in *`qends`* structure and tries to allocate slot from it.

Return Value

If there is no usable edge or there is no free slot in edge negative value is returned.

canque_get_inslot4id

Name

`canque_get_inslot4id` — finds best outgoing edge and slot for given ID

Synopsis

```
int canque_get_inslot4id (struct canque_ends_t * qends, struct
canque_edge_t ** qedgep, struct canque_slot_t ** slotp, int cmd,
unsigned long id, int prio);
```

Arguments

`qends`

ends structure belonging to calling communication object

`qedgep`

place to store pointer to found edge

`slotp`

place to store pointer to allocated slot

`cmd`

command type for slot

`id`

communication ID of message to send into edge

`prio`

optional priority of message

Description

Function looks for the non-blocked outgoing edge accepting messages with given ID. If edge is found, slot is allocated from that edge. The edges with non-zero mask are preferred over edges open to all messages. If more edges with mask accepts given message ID, the edge with highest priority below or equal to required priority is selected.

Return Value

If there is no usable edge or there is no free slot in edge negative value is returned.

canque_put_inslot

Name

`canque_put_inslot` — schedules filled slot for processing

Synopsis

```
int canque_put_inslot (struct canque_ends_t * qends, struct
canque_edge_t * qedge, struct canque_slot_t * slot);
```

Arguments

`qends`

ends structure belonging to calling communication object

`qedge`

edge slot belong to

`slot`

pointer to the prepared slot

Description

Puts slot previously acquired by `canque_get_inslot` or `canque_get_inslot4id` function call into FIFO queue and activates edge processing if needed.

Return Value

Positive value informs, that activation of output end has been necessary

canque_abort_inslot

Name

`canque_abort_inslot` — aborts preparation of the message in the slot

Synopsis

```
int canque_abort_inslot (struct canque_ends_t * qends, struct
canque_edge_t * qedge, struct canque_slot_t * slot);
```

Arguments

`qends`

ends structure belonging to calling communication object

`qedge`

edge slot belong to

`slot`

pointer to the previously allocated slot

Description

Frees slot previously acquired by `canque_get_inslot` or `canque_get_inslot4id` function call. Used when message copying into slot fails.

Return Value

Positive value informs, that queue full state has been negated.

canque_filter_msg2edges

Name

`canque_filter_msg2edges` — sends message into all edges which accept its ID

Synopsis

```
int canque_filter_msg2edges (struct canque_ends_t * qends, struct
canmsg_t * msg);
```

Arguments

`qends`

ends structure belonging to calling communication object

`msg`

pointer to CAN message

Description

Sends message to all outgoing edges connected to the given ends, which accepts message communication ID.

Return Value

Returns number of edges message has been send to

canque_test_outslot

Name

`canque_test_outslot` — test and retrieve ready slot for given ends

Synopsis

```
int canque_test_outslot (struct canque_ends_t * qends, struct
canque_edge_t ** qedgep, struct canque_slot_t ** slotp);
```

Arguments

`qends`

ends structure belonging to calling communication object

`qedgep`

place to store pointer to found edge

`slotp`

place to store pointer to received slot

Description

Function takes highest priority active incoming edge and retrieves oldest ready slot from it.

Return Value

Negative value informs, that there is no ready output slot for given ends. Positive value is equal to the command slot has been allocated by the input side.

canque_free_outslot

Name

`canque_free_outslot` — frees processed output slot

Synopsis

```
int canque_free_outslot (struct canque_ends_t * qends, struct
canque_edge_t * qedge, struct canque_slot_t * slot);
```

Arguments

`qends`

ends structure belonging to calling communication object

`qedge`

edge slot belong to

`slot`

pointer to the processed slot

Description

Function releases processed slot previously acquired by `canque_test_outslot` function call.

Return Value

Return value informs if input side has been notified to know about change of edge state

canque_again_outslot

Name

`canque_again_outslot` — reschedule output slot to process it again later

Synopsis

```
int canque_again_outslot (struct canque_ends_t * qends, struct
canque_edge_t * qedge, struct canque_slot_t * slot);
```

Arguments

`qends`

ends structure belonging to calling communication object

`qedge`

edge slot belong to

`slot`

pointer to the slot for re-processing

Description

Function reschedules slot previously acquired by `canque_test_outslot` function call for second time processing.

Return Value

Function cannot fail.

canque_set_filt

Name

`canque_set_filt` — sets filter for specified edge

Synopsis

```
int canque_set_filt (struct canque_edge_t * qedge, unsigned long
filtid, unsigned long filtmask, int filtflags);
```

Arguments

`qedge`

pointer to the edge

`filtid`

ID to set for the edge

`filtmask`

mask used for ID match check

`filtflags`

required filer flags

Return Value

Negative value is returned if edge is in the process of delete.

canque_flush

Name

`canque_flush` — fluesh all ready slots in the edge

Synopsis

```
int canque_flush (struct canque_edge_t * qedge);
```

Arguments

`qedge`

pointer to the edge

Description

Tries to flush all allocated slots from the edge, but there could exist some slots associated to edge which are processed by input or output side and cannot be flushed at this moment.

Return Value

The nonzero value indicates, that queue has not been empty before the function call.

canqueue_ends_init_gen

Name

`canqueue_ends_init_gen` — subsystem independent routine to initialize ends state

Synopsis

```
int canqueue_ends_init_gen (struct canque_ends_t * qends);
```

Arguments

qends

pointer to the ends structure

Return Value

Cannot fail.

canqueue_notify_kern

Name

`canqueue_notify_kern` — notification callback handler for Linux userspace clients

Synopsis

```
void canqueue_notify_kern (struct canque_ends_t * qends, struct
canque_edge_t * qedge, int what);
```

Arguments

qends

pointer to the callback side ends structure

qedge

edge which invoked notification

what

notification type

canqueue_ends_init_kern

Name

`canqueue_ends_init_kern` — Linux userspace clients specific ends initialization

Synopsis

```
int canqueue_ends_init_kern (struct canque_ends_t * qends);
```

Arguments

`qends`

pointer to the callback side ends structure

canque_get_inslot4id_wait_kern

Name

`canque_get_inslot4id_wait_kern` — find or wait for best outgoing edge and slot for given ID

Synopsis

```
int canque_get_inslot4id_wait_kern (struct canque_ends_t * qends,
struct canque_edge_t ** qedgep, struct canque_slot_t ** slotp, int
cmd, unsigned long id, int prio);
```

Arguments

`qends`

ends structure belonging to calling communication object

`qedgep`

place to store pointer to found edge

`slotp`

place to store pointer to allocated slot

`cmd`

command type for slot

`id`

communication ID of message to send into edge

`prio`

optional priority of message

Description

Same as `canque_get_inslot4id`, except, that it waits for free slot in case, that queue is full. Function is specific for Linux userspace clients.

Return Value

If there is no usable edge negative value is returned.

canque_get_outslot_wait_kern

Name

`canque_get_outslot_wait_kern` — receive or wait for ready slot for given ends

Synopsis

```
int canque_get_outslot_wait_kern (struct canque_ends_t * qends, struct
canque_edge_t ** qedgep, struct canque_slot_t ** slotp);
```

Arguments

`qends`

ends structure belonging to calling communication object

`qedgep`

place to store pointer to found edge

`slotp`

place to store pointer to received slot

Description

The same as `canque_test_outslot`, except it waits in the case, that there is no ready slot for given ends. Function is specific for Linux userspace clients.

Return Value

Negative value informs, that there is no ready output slot for given ends. Positive value is equal to the command slot has been allocated by the input side.

canque_sync_wait_kern

Name

`canque_sync_wait_kern` — wait for all slots processing

Synopsis

```
int canque_sync_wait_kern (struct canque_ends_t * qends, struct
canque_edge_t * qedge);
```

Arguments

`qends`

ends structure belonging to calling communication object

`qedge`

pointer to edge

Description

Functions waits for ends transition into empty state.

Return Value

Positive value indicates, that edge empty state has been reached. Negative or zero value informs about interrupted wait or other problem.

canque_new_edge_kern

Name

`canque_new_edge_kern` — allocate new edge structure in the Linux kernel context

Synopsis

```
struct canque_edge_t * canque_new_edge_kern (int slotsnr);
```

Arguments

`slotsnr`

required number of slots in the newly allocated edge structure

Return Value

Returns pointer to allocated slot structure or `NULL` if there is not enough memory to process operation.

canqueue_connect_edge

Name

`canqueue_connect_edge` — connect edge between two communication entities

Synopsis

```
int canqueue_connect_edge (struct canque_edge_t * qedge, struct
canque_ends_t * inends, struct canque_ends_t * outends);
```

Arguments

`qedge`

pointer to edge

`inends`

pointer to ends the input of the edge should be connected to

`outends`

pointer to ends the output of the edge should be connected to

Return Value

Negative value informs about failed operation.

canqueue_disconnect_edge

Name

`canqueue_disconnect_edge` — disconnect edge from communicating entities

Synopsis

```
int canqueue_disconnect_edge (struct canque_edge_t * qedge);
```

Arguments

`qedge`

pointer to edge

Return Value

Negative value means, that edge is used and cannot be disconnected. Operation has to be delayed.

canqueue_disconnect_edge_kern

Name

`canqueue_disconnect_edge_kern` — disconnect edge from communicating entities with wait

Synopsis

```
int canqueue_disconnect_edge_kern (struct canque_ends_t * qends, struct
canque_edge_t * qedge);
```

Arguments

`qends`

ends structure belonging to calling communication object

`qedge`

pointer to edge

Description

Same as `canqueue_disconnect_edge`, but tries to wait for state with zero use counter.

Return Value

Negative value means, that edge is used and cannot be disconnected yet. Operation has to be delayed.

canqueue_ends_done_kern

Name

`canqueue_ends_done_kern` — finalizing of the ends structure for Linux kernel clients

Synopsis

```
int canqueue_ends_done_kern (struct canque_ends_t * qends, int sync);
```

Arguments

`qends`

pointer to ends structure

`sync`

flag indicating, that user wants to wait for processing of all remaining messages

Return Value

Function should be designed such way to not fail.