03_04_2024

- stvaranje CAN_RAW socketa

```
func NewReadWriteCloserForInterface(i *net.Interface) (ReadWriteCloser, error) {
        s, _ := syscall.Socket(syscall.AF_CAN, syscall.SOCK_RAW, unix.CAN_RAW)
        addr := &unix.SockaddrCAN{Ifindex: i.Index}
        if err := unix.Bind(s, addr); err != nil {
                return nil, err
        }

        f := os.NewFile(uintptr(s), fmt.Sprintf("fd %d", s))

        return &readWriteCloser{f}, nil
}
```

https://github.com/linux-can/can-utils/blob/master/include/linux/can.h
iz can.h

- postoji can-isotp tip socketa

```
#define CAN_RAW         1 /* RAW sockets */
#define CAN_BCM         2 /* Broadcast Manager */
#define CAN_TP16        3 /* VAG Transport Protocol v1.6 */
#define CAN_TP20        4 /* VAG Transport Protocol v2.0 */
#define CAN_MCNET       5 /* Bosch MCNet */
#define CAN_ISOTP       6 /* ISO 15765-2 Transport Protocol */
#define CAN_J1939       7 /* SAE J1939 */
#define CAN_NPROTO      8
```

ISO-TP

https://munich.dissec.to/kb/chapters/isotp/isotp-linux.html

iz -L zastavice isotpsend alata da se naslutiti da se link layer mora ispravno konfigurirati ovisno o tome koristi li se CAN 2.0 ili CAN FD:

```
❯ isotpsend

Usage: isotpsend [options] <CAN interface>
Options:
         -s <can_id>  (source can_id. Use 8 digits for extended IDs)
         -d <can_id>  (destination can_id. Use 8 digits for extended IDs)
         -x <addr>[:<rxaddr>]  (extended addressing / opt. separate rxaddr)
         -p [tx]:[rx]  (set and enable tx/rx padding bytes)
         -P <mode>     (check rx padding for (l)ength (c)ontent (a)ll)
         -t <time ns>  (frame transmit time (N_As) in nanosecs) (*)
         -f <time ns>  (ignore FC and force local tx stmin value in nanosecs)
         -D <len>      (send a fixed PDU with len bytes - no STDIN data)
         -l <num>      (send num PDUs - use 'i' for infinite loop)
         -g <usecs>    (wait given usecs before sending a PDU)
         -b            (block until the PDU transmission is completed)
         -S            (SF broadcast mode - for functional addressing)
         -C            (CF broadcast mode - no wait for flow controls)
         -L <mtu>:<tx_dl>:<tx_flags>  (link layer options for CAN FD)

CAN IDs and addresses are given and expected in hexadecimal values.
The pdu data is expected on STDIN in space separated ASCII hex values.
(*) = Use '-t ZERO' to set N_As to zero for Linux version 5.18+
```

https://github.com/hartkopp/can-isotp/blob/master/include/uapi/linux/can/isotp.h
https://github.com/linux-can/can-utils/blob/master/include/linux/can.h
Pretpostavljeno je da se koristi CAN 2.0

```
/* link layer default values => make use of Classical CAN frames */

#define CAN_ISOTP_DEFAULT_LL_MTU        CAN_MTU
#define CAN_ISOTP_DEFAULT_LL_TX_DL      CAN_MAX_DLEN
#define CAN_ISOTP_DEFAULT_LL_TX_FLAGS   0
```

```
#define CAN_MTU         (sizeof(struct can_frame))
#define CANFD_MTU       (sizeof(struct canfd_frame))
#define CANXL_MTU       (sizeof(struct canxl_frame))
#define CANXL_HDR_SIZE  (offsetof(struct canxl_frame, data))
#define CANXL_MIN_MTU   (CANXL_HDR_SIZE + 64)
#define CANXL_MAX_MTU   CANXL_MTU
```

```
/* CAN payload length and DLC definitions according to ISO 11898-1 */
#define CAN_MAX_DLC 8
#define CAN_MAX_RAW_DLC 15
#define CAN_MAX_DLEN 8

/* CAN FD payload length and DLC definitions according to ISO 11898-7 */
#define CANFD_MAX_DLC 15
#define CANFD_MAX_DLEN 64

/*
 * CAN XL payload length and DLC definitions according to ISO 11898-1
 * CAN XL DLC ranges from 0 .. 2047 => data length from 1 .. 2048 byte
 */
#define CANXL_MIN_DLC 0
#define CANXL_MAX_DLC 2047
#define CANXL_MAX_DLC_MASK 0x07FF
#define CANXL_MIN_DLEN 1
#define CANXL_MAX_DLEN 2048
```

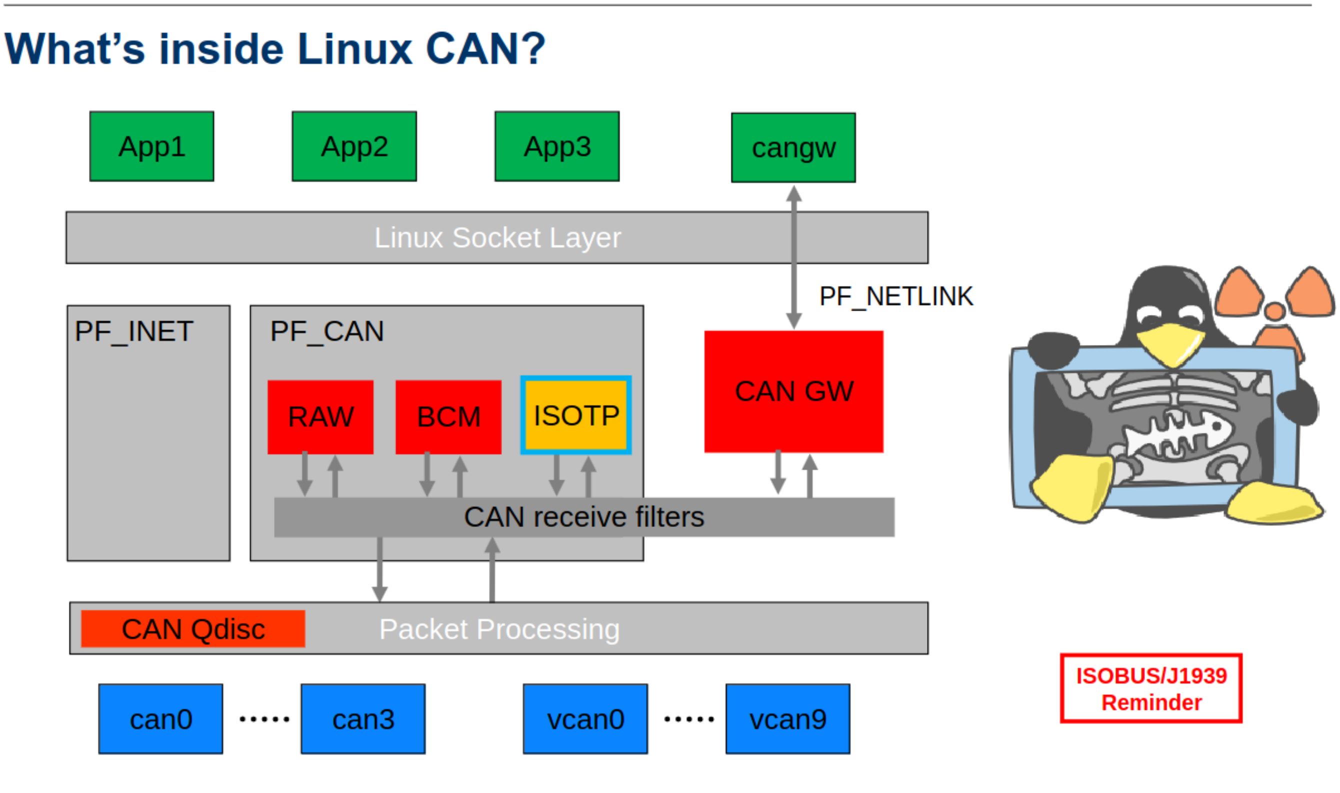
CAN FD kao LL se moze konfigurirati setsockopt pozivom (iz isotpsend.c)

```
    if (llopts.tx_dl) {
            if (setsockopt(s, SOL_CAN_ISOTP, CAN_ISOTP_LL_OPTS, &llopts, sizeof(llopts)) < 0) {
                perror("link layer sockopt");
                exit(1);
            }
    }
```

04_04_2024

ISOTP CANFD linux prezentacija

https://s3.eu-central-1.amazonaws.com/cancia-de/documents/proceedings/slides/hartkopp_slides_15icc.pdf



What's inside Linux CAN?

App1

App2

App3

cangw

Linux Socket Layer

PF_NETLINK

PF_INET

PF_CAN

RAW

BCM

ISOTP

CAN GW

CAN receive filters

CAN Qdisc

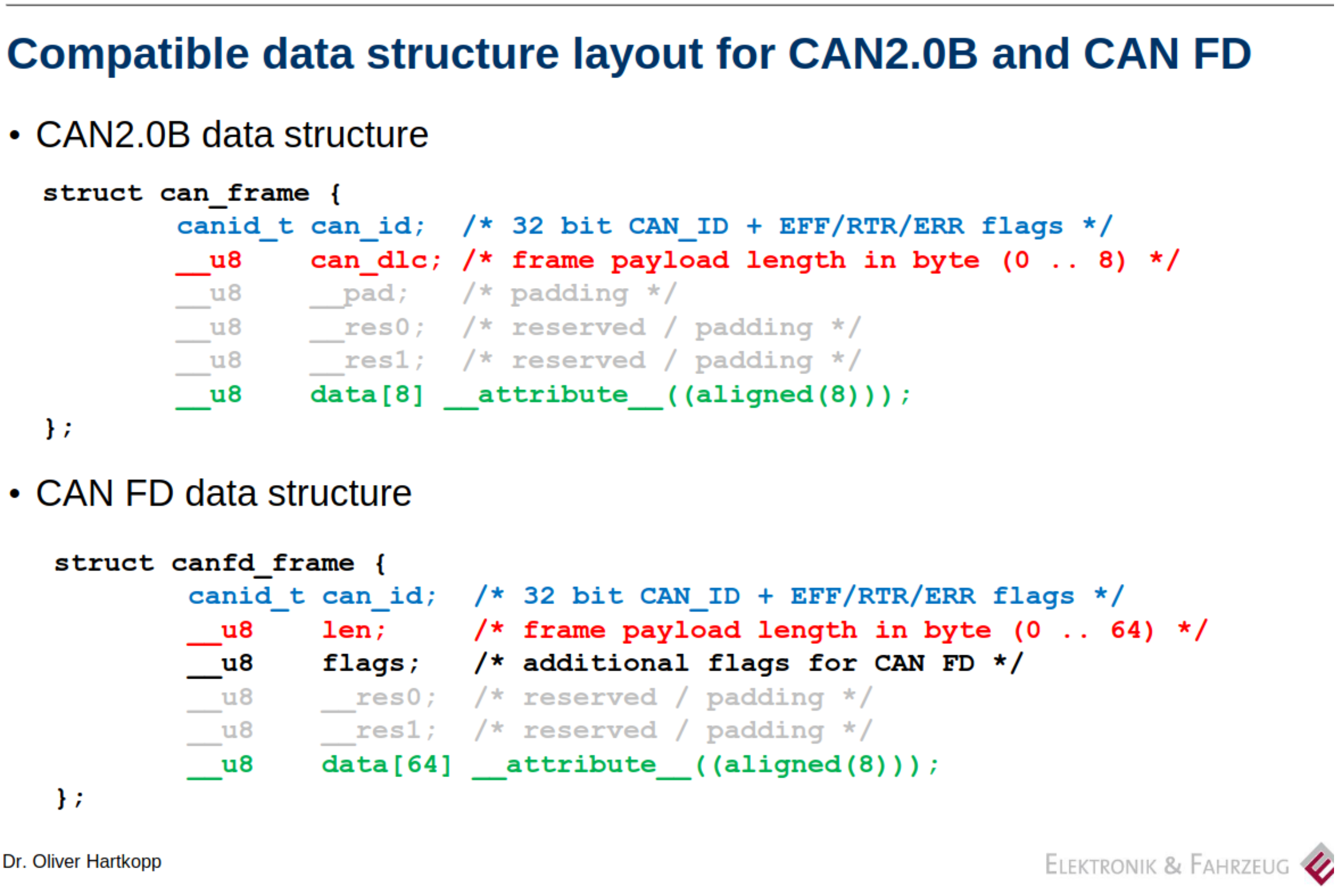
Packet Processing

can0 ····· can3

vcan0 ····· vcan9

ISOBUS/J1939 Reminder

Inace CAN FD i CAN imaju razlicito mapiranje DLC (Data length code) na duljinu podataka, u socketcanu je to rijeseno:



Compatible data structure layout for CAN2.0B and CAN FD

- CAN2.0B data structure

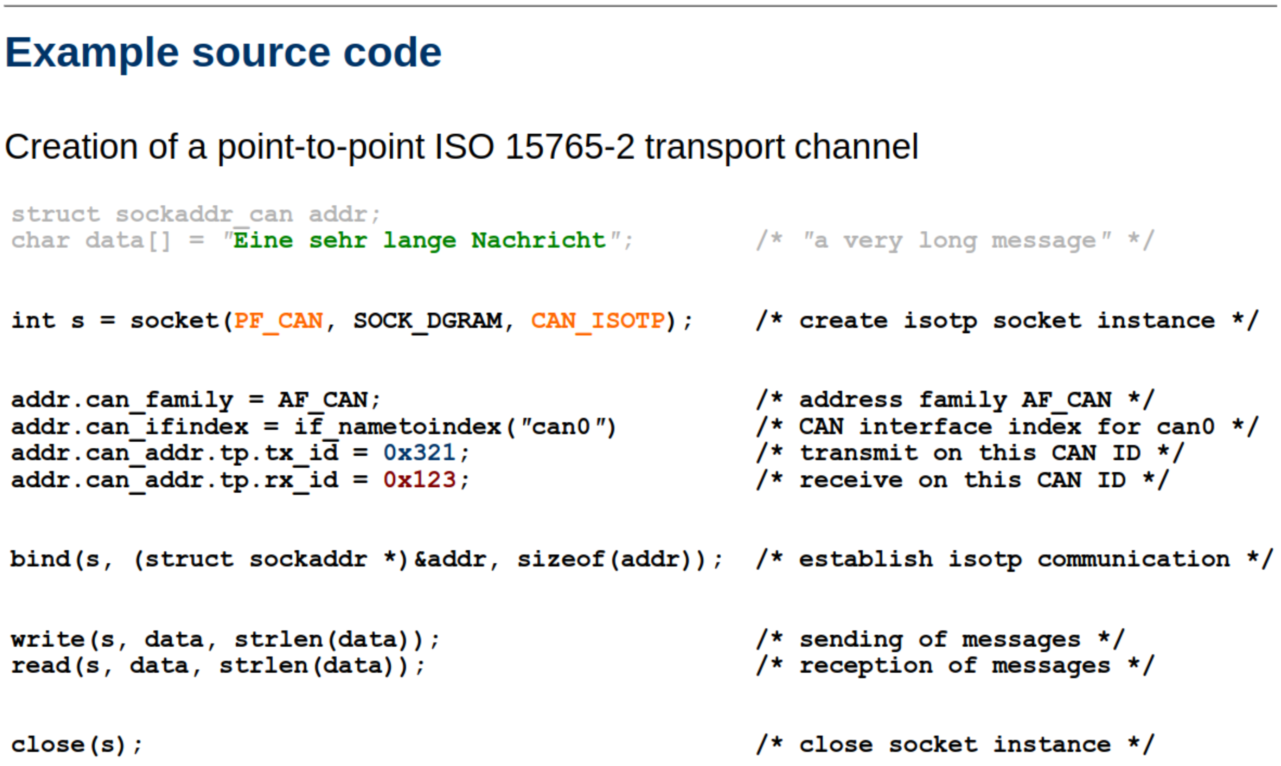
```
struct can_frame {
        canid_t can_id;  /* 32 bit CAN_ID + EFF/RTR/ERR flags */
        __u8    can_dlc; /* frame payload length in byte (0 .. 8) */
        __u8    __pad;   /* padding */
        __u8    __res0;  /* reserved / padding */
        __u8    __res1;  /* reserved / padding */
        __u8    data[8] __attribute__((aligned(8)));
};
```

- CAN FD data structure

```
struct canfd_frame {
        canid_t can_id;  /* 32 bit CAN_ID + EFF/RTR/ERR flags */
        __u8    len;     /* frame payload length in byte (0 .. 64) */
        __u8    flags;   /* additional flags for CAN FD */
        __u8    __res0;  /* reserved / padding */
        __u8    __res1;  /* reserved / padding */
        __u8    data[64] __attribute__((aligned(8)));
};
```

Dr. Oliver Hartkopp

Elektronik & Fahrzeug



Example source code

Creation of a point-to-point ISO 15765-2 transport channel

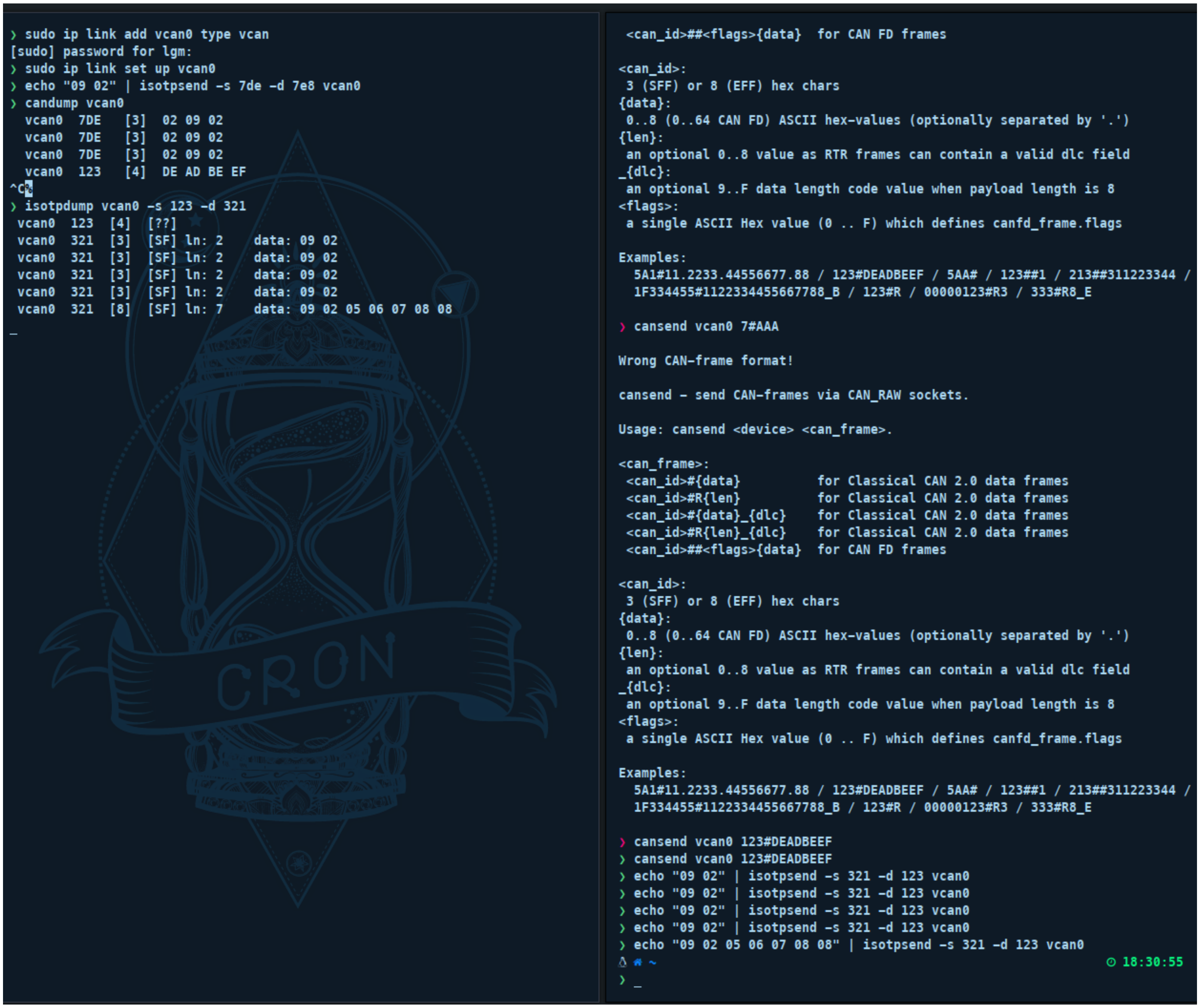
```
struct sockaddr_can addr;
char data[] = "Eine sehr lange Nachricht";          /* "a very long message" */

int s = socket(PF_CAN, SOCK_DGRAM, CAN_ISOTP);    /* create isotp socket instance */

addr.can_family = AF_CAN;                         /* address family AF_CAN */
addr.can_ifindex = if_nametoindex("can0")         /* CAN interface index for can0 */
addr.can_addr.tp.tx_id = 0x321;                   /* transmit on this CAN ID */
addr.can_addr.tp.rx_id = 0x123;                   /* receive on this CAN ID */

bind(s, (struct sockaddr *)&addr, sizeof(addr));  /* establish isotp communication */

write(s, data, strlen(data));                     /* sending of messages */
read(s, data, strlen(data));                      /* reception of messages */

close(s);                                         /* close socket instance */
```

05_04_2024

https://github.com/aakash-s45/ic/tree/master

- python kuksa sdk
  - https://github.com/eclipse-kuksa/kuksa-python-sdk/blob/main/docs/cli.md
- val server
  - https://github.com/eclipse/kuksa.val/tree/master/kuksa-val-server

ISO TP isprobavanje



```
❯ sudo ip link add vcan0 type vcan
[sudo] password for lgm:
❯ sudo ip link set up vcan0
❯ echo "09 02" | isotpsend -s 7de -d 7e8 vcan0
❯ candump vcan0
  vcan0  7DE   [3]  02 09 02
  vcan0  7DE   [3]  02 09 02
  vcan0  7DE   [3]  02 09 02
  vcan0  123   [4]  DE AD BE EF
^C%
❯ isotpdump vcan0 -s 123 -d 321
 vcan0  123  [4]  [??]
 vcan0  321  [3]  [SF] ln: 2    data: 09 02
 vcan0  321  [3]  [SF] ln: 2    data: 09 02
 vcan0  321  [3]  [SF] ln: 2    data: 09 02
 vcan0  321  [3]  [SF] ln: 2    data: 09 02
 vcan0  321  [8]  [SF] ln: 7    data: 09 02 05 06 07 08 08

_
```

```
 <can_id>##<flags>{data}  for CAN FD frames

<can_id>:
 3 (SFF) or 8 (EFF) hex chars
{data}:
 0..8 (0..64 CAN FD) ASCII hex-values (optionally separated by '.')
{len}:
 an optional 0..8 value as RTR frames can contain a valid dlc field
_{dlc}:
 an optional 9..F data length code value when payload length is 8
<flags>:
 a single ASCII Hex value (0 .. F) which defines canfd_frame.flags

Examples:
  5A1#11.2233.44556677.88 / 123#DEADBEEF / 5AA# / 123##1 / 213##311223344 /
  1F334455#1122334455667788_B / 123#R / 00000123#R3 / 333#R8_E

❯ cansend vcan0 7#AAA

Wrong CAN-frame format!

cansend - send CAN-frames via CAN_RAW sockets.

Usage: cansend <device> <can_frame>.

<can_frame>:
 <can_id>#{data}          for Classical CAN 2.0 data frames
 <can_id>#R{len}          for Classical CAN 2.0 data frames
 <can_id>#{data}_{dlc}    for Classical CAN 2.0 data frames
 <can_id>#R{len}_{dlc}    for Classical CAN 2.0 data frames
 <can_id>##<flags>{data}  for CAN FD frames

<can_id>:
 3 (SFF) or 8 (EFF) hex chars
{data}:
 0..8 (0..64 CAN FD) ASCII hex-values (optionally separated by '.')
{len}:
 an optional 0..8 value as RTR frames can contain a valid dlc field
_{dlc}:
 an optional 9..F data length code value when payload length is 8
<flags>:
 a single ASCII Hex value (0 .. F) which defines canfd_frame.flags

Examples:
  5A1#11.2233.44556677.88 / 123#DEADBEEF / 5AA# / 123##1 / 213##311223344 /
  1F334455#1122334455667788_B / 123#R / 00000123#R3 / 333#R8_E

❯ cansend vcan0 123#DEADBEEF
❯ cansend vcan0 123#DEADBEEF
❯ echo "09 02" | isotpsend -s 321 -d 123 vcan0
❯ echo "09 02" | isotpsend -s 321 -d 123 vcan0
❯ echo "09 02" | isotpsend -s 321 -d 123 vcan0
❯ echo "09 02" | isotpsend -s 321 -d 123 vcan0
❯ echo "09 02 05 06 07 08 08" | isotpsend -s 321 -d 123 vcan0
  ~                                                          18:30:55
❯ _
```

candump koji cita iz CAN_RAW socketa moze procitati ISO_TP frameove, ali i isotprecv koji cita iz CAN_ISOTP socketa moze procitati CAN poruke neovisno jesu li formirane u skladu s isotp standardom.

06_04_2024

https://github.com/CaringCaribou/caringcaribou/blob/master/documentation/uds.md
UDS moze biti na bilo kojem arbitration ID-u te bi simulirani ECU-ovi trebali raditi s postojecim alatima primjerice caring caribou

socketcan go

https://gist.github.com/FabianInostroza/b64ba3e2c85de136552a03d6b03b90d1

implementacija u pythonu

obzirom da je python puno popularniji i rasireniji nego Go, koristit cu ga za stvaranje konfigurabilnih predlozaka za ECU-ove

https://docs.python.org/3/library/socket.html#socket-families
socket families

- A tuple `(interface, )` is used for the `AF_CAN` address family, where *interface* is a string representing a network interface name like `'can0'`. The network interface name `''` can be used to receive packets from all network interfaces of this family.
  - `CAN_ISOTP` protocol require a tuple `(interface, rx_addr, tx_addr)` where both additional parameters are unsigned

long integer that represent a CAN identifier (standard or extended).
  - `CAN_J1939` protocol require a tuple `(interface, name, pgn, addr)` where additional parameters are 64-bit unsigned integer representing the ECU name, a 32-bit unsigned integer representing the Parameter Group Number (PGN), and an 8-bit integer representing the address.

https://setuptools.pypa.io/en/latest/userguide/quickstart.html#setup-py
https://carpentries-incubator.github.io/python_packaging/instructor/03-building-and-installing.html

07_04_2024

Kako socketcan koriste postojeci alati?
Python-can biblioteka:
https://github.com/hardbyte/python-can/blob/main/can/interfaces/socketcan/socketcan.py#L84

- nema opciju za iso-tp sockete
- koristi python structove i packing za slanje can_frameova

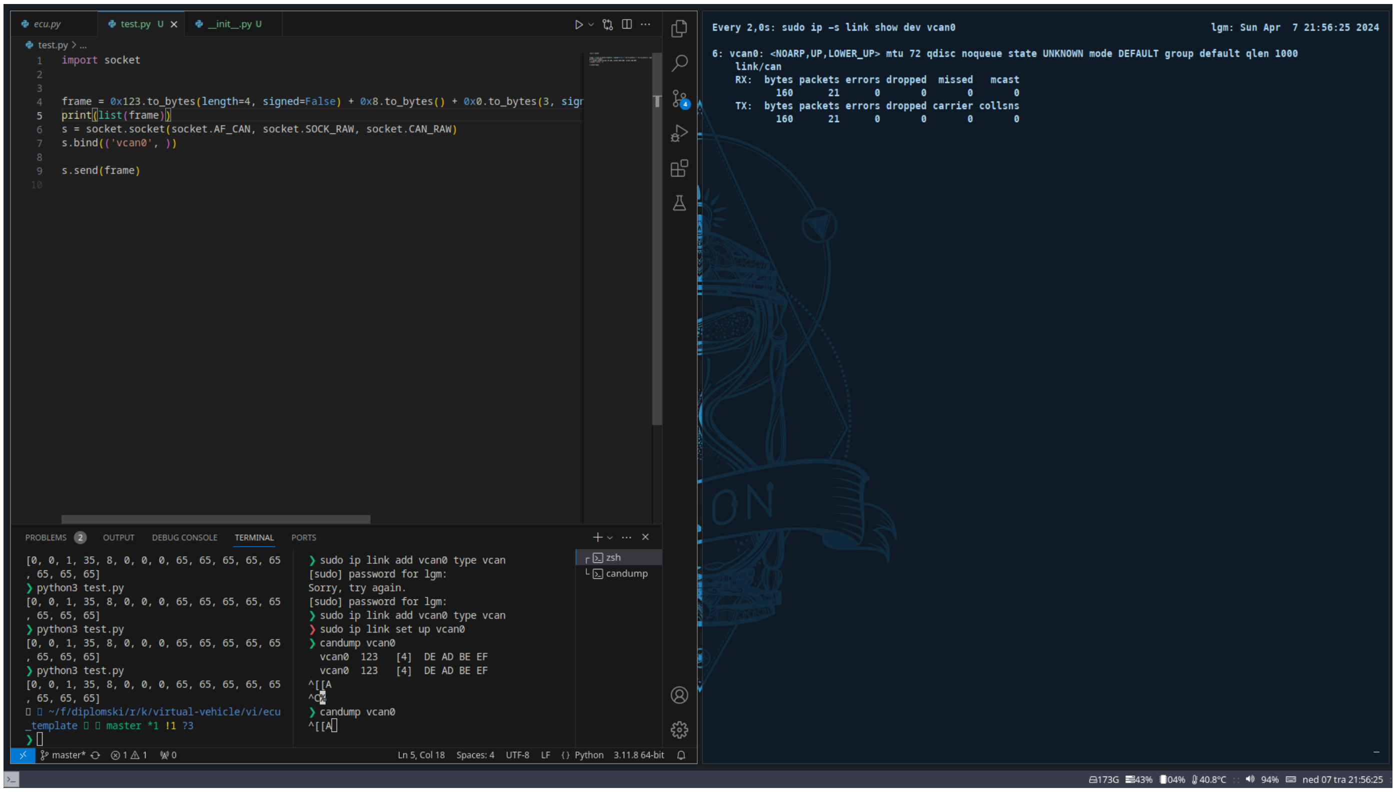
Caringcaribou:

- ne koristi socketcan iso-tp nego vlastorucnu implementaciju
- python-can

IsoTP paket:
https://can-isotp.readthedocs.io/en/latest/isotp/socket.html#examples

Iz nekog razloga slanje rucno sastavljenih can frameova koristenjem socket paketa nije prikazano na candump ispisu, ali se mijenja statistika interfacea:

```
frame = 0x123.to_bytes(length=4, signed=False) + 0x8.to_bytes() + 0x0.to_bytes(3, signed=False) +
bytes("AAAAAAAA", "ascii").ljust(8, b"\x00")
print(frame.hex())
s = socket.socket(socket.AF_CAN, socket.SOCK_RAW, socket.CAN_RAW)
s.bind(('vcan0', ))

print(s.send(frame))
```



```python
import socket


frame = 0x123.to_bytes(length=4, signed=False) + 0x8.to_bytes() + 0x0.to_bytes(3, sigr
print(list(frame))
s = socket.socket(socket.AF_CAN, socket.SOCK_RAW, socket.CAN_RAW)
s.bind(('vcan0', ))

s.send(frame)
```

```
[0, 0, 1, 35, 8, 0, 0, 0, 65, 65, 65, 65, 65
, 65, 65, 65]
❯ python3 test.py
[0, 0, 1, 35, 8, 0, 0, 0, 65, 65, 65, 65, 65
, 65, 65, 65]
❯ python3 test.py
[0, 0, 1, 35, 8, 0, 0, 0, 65, 65, 65, 65, 65
, 65, 65, 65]
❯ python3 test.py
[0, 0, 1, 35, 8, 0, 0, 0, 65, 65, 65, 65, 65
, 65, 65, 65]
  ~/f/diplomski/r/k/virtual-vehicle/vi/ecu
_template    master *1 !1 ?3
❯
```

```
❯ sudo ip link add vcan0 type vcan
[sudo] password for lgm:
Sorry, try again.
[sudo] password for lgm:
❯ sudo ip link add vcan0 type vcan
❯ sudo ip link set up vcan0
❯ candump vcan0
  vcan0  123   [4]  DE AD BE EF
  vcan0  123   [4]  DE AD BE EF
^[[A
^C%
❯ candump vcan0
^[[A
```

```
Every 2,0s: sudo ip -s link show dev vcan0                    lgm: Sun Apr  7 21:56:25 2024

6: vcan0: <NOARP,UP,LOWER_UP> mtu 72 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000
    link/can
    RX:  bytes packets errors dropped  missed   mcast
           160      21      0       0       0       0
    TX:  bytes packets errors dropped carrier collsns
           160      21      0       0       0       0
```

Usporedba s kodom iz python-can-a:

```python
CAN_FRAME_HEADER_STRUCT = struct.Struct("=IBB2x")

can_id = 0x123
flags = 0
max_len = 8
data = bytes("AAAAAAAA", "ascii").ljust(max_len, b"\x00")
result = CAN_FRAME_HEADER_STRUCT.pack(can_id, 8, flags) + data

print(result.hex())

frame = 0x123.to_bytes(length=4, signed=False, byteorder="little") + 0x8.to_bytes() + 0x0.to_bytes(3,
signed=False) + bytes("AAAAAAAA", "ascii").ljust(8, b"\x00")
print(frame.hex())
s = socket.socket(socket.AF_CAN, socket.SOCK_RAW, socket.CAN_RAW)
s.bind(('vcan0', ))

print(s.send(frame))
```

```
❯ python3 test.py
b'#\x01\x00\x00\x08\x00\x00\x00AAAAAAAA'
b'\x00\x00\x01#\x08\x00\x00\x00AAAAAAAA'
16
❯ python3 test.py
23010000080000004141414141414141
00000123080000004141414141414141
```

razlog je u razlici u endianessu, popravljen kod:

```python
frame = 0x123.to_bytes(length=4, signed=False, byteorder="little") + 0x8.to_bytes() + 0x0.to_bytes(3,
signed=False) + bytes("AAAAAAAA", "ascii").ljust(8, b"\x00")
```

ili jos bolje:

```
frame = 0x123.to_bytes(length=4, signed=False, byteorder=sys.byteorder) + 0x8.to_bytes() +
0x0.to_bytes(3, signed=False) + bytes("AAAAAAAA", "ascii").ljust(8, b"\x00")
```

iso-tp python lib

https://github.com/hardbyte/python-can/issues/45#issuecomment-451158673
slucajno sam naisao na ovaj github issue i primjetio da se zasebni iso-tp library (za koji sam mislio da je samo wrapper oko socket API-ja) moze direktno koristiti s python-canom, a ostvaren je u aplikacijskom sloju (odnosno bez koristenja podrske kernela)

- koristit cu ovo za pocetak, a ako mi ostane vremena cu napraviti fork python-cana i dodati direktno podrsku za iso-tp sockete
  - takodjer, iso-tp kernel modul nije automatski ucitan u velikom broju linux distribucija kao can_raw

podrska za can fd

can*fd* (bool)_□

default: False

When set to `True`, transmitted messages will be CAN FD. CAN 2.0 when `False`.

Setting this parameter to `True` does not change the behavior of the `TransportLayer` except that outputted message will have their `is_fd` property set to `True`. This parameter is just a convenience to integrate more easily with python-can

dodatno o UDS-u

Addressing mode: For communicating with the ECU, the diagnostic tool uses either Physical addressing or Functional addressing method.
– *Physical addressing* is the kind of addressing where the Diagnostics tool communicates with a single ECU.
– *Functional addressing* is where the Diagnostics tool communicates with multiple ECUs.

neki uds libraryiji koji nisu prikladni za koristenje s python-can-om

- https://uds.readthedocs.io/en/latest/pages/knowledge_base.html
  - UDS knowledge base
- https://python-uds.readthedocs.io/en/latest/index.html]

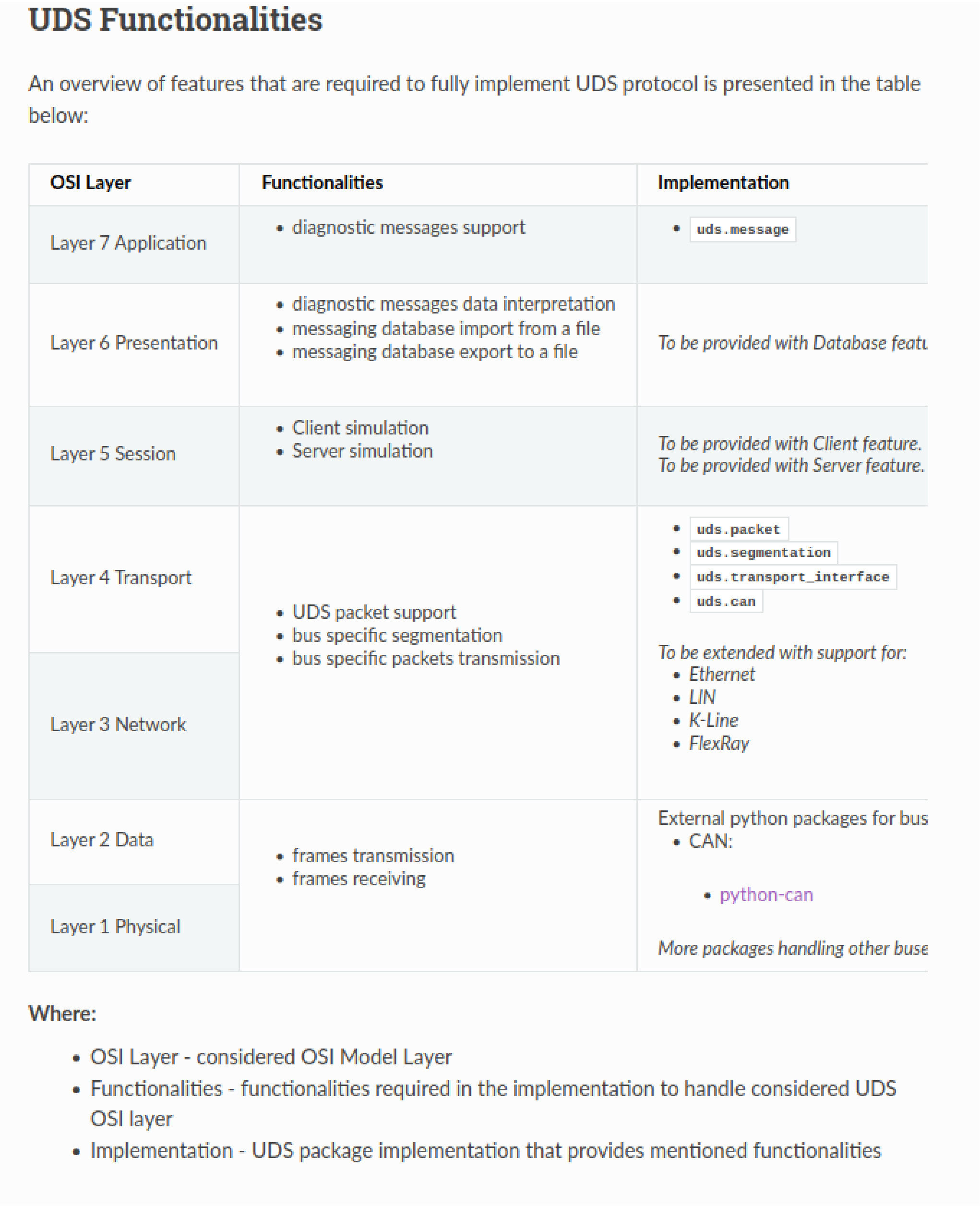
UDS Standards

UDS is defined by multiple standards which are the main source of information and requirements about this protocol. Full list of standards is included in the table below:

OSI Layer	Common	CAN	FlexRay	Ethernet	K-Line
Layer 7 Application	ISO 14229-1 ISO 27145-3	ISO 14229-3	ISO 14229-4	ISO 14229-5	ISO 142
Layer 6 Presentation	ISO 27145-2				
Layer 5 Session	ISO 14229-2				
Layer 4 Transport	ISO 27145-4	ISO 15765-2	ISO 10681-2	ISO 13400-2	Not appl
Layer 3 Network					
Layer 2 Data		ISO 11898-1	ISO 17458-2	ISO 13400-3	ISO 142
Layer 1 Physical		ISO 11898-2 ISO 11898-3	ISO 17458-4		ISO 142

Where:

- OSI Layer - OSI Model Layer for which standards are relevant
- Common - standards mentioned in this column are always relevant for UDS communication regardless of bus used
- CAN - standards which are specific for UDS on CAN implementation
- FlexRay - standards which are specific for UDS on FlexRay implementation
- Ethernet - standards which are specific for UDS on IP implementation
- K-Line - standards which are specific for UDS on K-Line implementation
- LIN - standards which are specific for UDS on LIN implementation



UDS Functionalities

An overview of features that are required to fully implement UDS protocol is presented in the table below:

OSI Layer	Functionalities	Implementation
Layer 7 Application	• diagnostic messages support	• `uds.message`
Layer 6 Presentation	• diagnostic messages data interpretation • messaging database import from a file • messaging database export to a file	*To be provided with Database featu*
Layer 5 Session	• Client simulation • Server simulation	*To be provided with Client feature.* *To be provided with Server feature.*
Layer 4 Transport	• UDS packet support • bus specific segmentation • bus specific packets transmission	• `uds.packet` • `uds.segmentation` • `uds.transport_interface` • `uds.can` *To be extended with support for:* • *Ethernet* • *LIN* • *K-Line* • *FlexRay*
Layer 3 Network		
Layer 2 Data	• frames transmission • frames receiving	External python packages for bus • CAN: • python-can *More packages handling other buse*
Layer 1 Physical		

Where:

- OSI Layer - considered OSI Model Layer
- Functionalities - functionalities required in the implementation to handle considered UDS OSI layer
- Implementation - UDS package implementation that provides mentioned functionalities