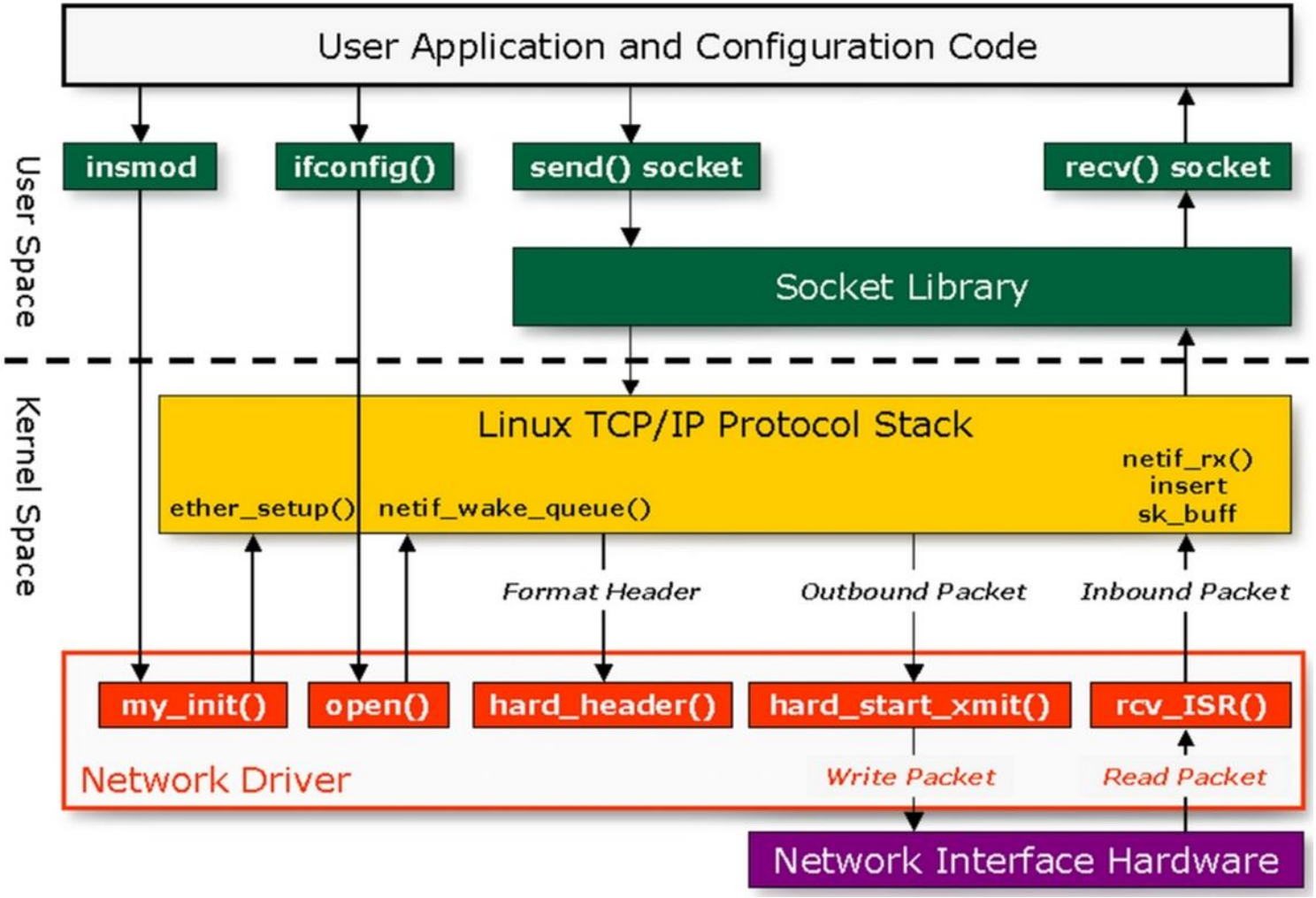
Лекция 3.2 Сетевой драйвер

Разработал: Максимов А.Н.

RISoft

Схема взаимодействия сетевого драйвера



На основе статьи Bill Weinberg Porting RTOS Device Drivers to Embedded Linu. http://www.linuxjournal.com/article/7355

RISoft

Сетевое устройство

сетевой устройство (сетевой интерфейс) - это точка соединения между пользовательским приложением и сетевой подсистемой

В современных системах Linux существует множество способов настройки сети. Наиболее популярными являются использование NetworkManager и Systemd. Однако иногда нам приходится делать это вручную, используя конфигурационный файл /etc/network/interfaces.

Сетевое устройство

Сетевой устройство (сетевой интерфейс) - это точка соединения между пользовательским приложением и сетевой подсистемой

$ ls /sys/class/net

eth0 lo wlan0

Сетевое устройство и утилита ip

```
$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen
    1000
   link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
   inet 127.0.0.1/8 scope host lo
      valid_lft forever preferred_lft forever
   inet6 ::1/128 scope host
      valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default
    qlen 1000
   link/ether 54:ee:74:c1:19:92 brd ff:ff:ff:ff:ff:ff
   inet 192.168.0.122/24 brd 192.168.0.255 scope global dynamic noprefixroute enp3s0
      valid_lft 80953sec preferred_lft 80953sec
   inet6 fe80::bb4c:8096:6:3695/64 scope link noprefixroute
```

```
# ifdown enp7s0
# ifup enp7s0
```

https://www.baeldung.com/linux/network-interface-configure

RISoft

Сетевое устройство и утилита ifconfig

```
ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0B:CD:1C:18:5A
          inet addr:172.16.25.126  Bcast:172.16.25.63  Mask:255.255.255.224
          inet6 addr: fe80::20b:cdff:fe1c:185a/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:2341604 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2217673 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:293460932 (279.8 MiB)  TX bytes:1042006549 (993.7 MiB)
          Interrupt:185 Memory:f7fe0000-f7ff0000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:5019066 errors:0 dropped:0 overruns:0 frame:0
          TX packets:5019066 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:2174522634 (2.0 GiB)  TX bytes:2174522634 (2.0 GiB)
```

Действия с утилитой ifconfig

Получение информации об интерфейсе

ifconfig eth0

Выключить интерфейс

ifconfig eth0 up

Включить интерфейс

ifconfig eth0 down

Задать IP адрес

ifconfig eth0 172.16.25.125

Изменить MAC адрес

ifconfig eth0 hw ether AA:BB:CC:DD:EE:FF

arp

```
root@kali:~# arp -a
_gateway (10.0.2.1) at 52:54:00:12:35:00 [ether] on eth0
root@kali:~# ping 10.0.2.4
PING 10.0.2.4 (10.0.2.4) 56(84) bytes of data.
64 bytes from 10.0.2.4: icmp_seq=1 ttl=64 time=0.541 ms
64 bytes from 10.0.2.4: icmp_seq=2 ttl=64 time=0.280 ms
^C
--- 10.0.2.4 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1031ms
rtt min/avg/max/mdev = 0.280/0.410/0.541/0.132 ms
root@kali:~# arp -a
? (10.0.2.4) at 08:00:27:ad:87:b3 [ether] on eth0
_gateway (10.0.2.1) at 52:54:00:12:35:00 [ether] on eth0
root@kali:~#
```

netcat

Netcat - это утилита, которая считывает и записывает данные по сетевым подключениям, используя протокол TCP или UDP.

Ожидание не заданном порту
nc -l 1234
nc 127.0.0.1 1234

Прием данных

nc -l 1234 > filename.out

Передача данных

printf "GET / HTTP/1.0\r\n\r\n" | nc host.example.com 80

Дополительно:
https://www.computerhope.com/unix/nc.htm

Перехват и анализ пакетов tcpdump

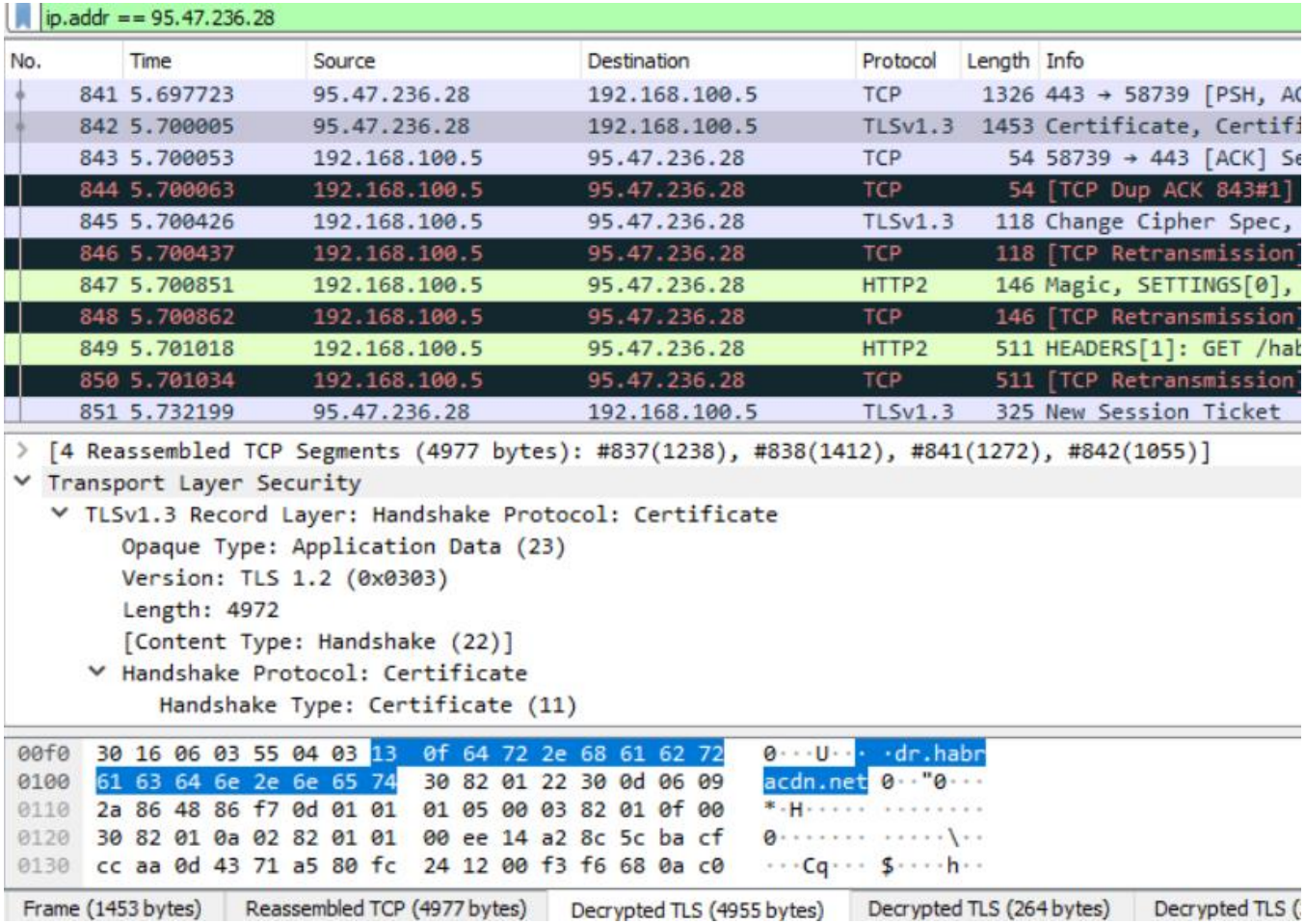
tcpdump

sudo tcpdump -i eth0 udp
sudo tcpdump -i eth0 host 10.10.1.1

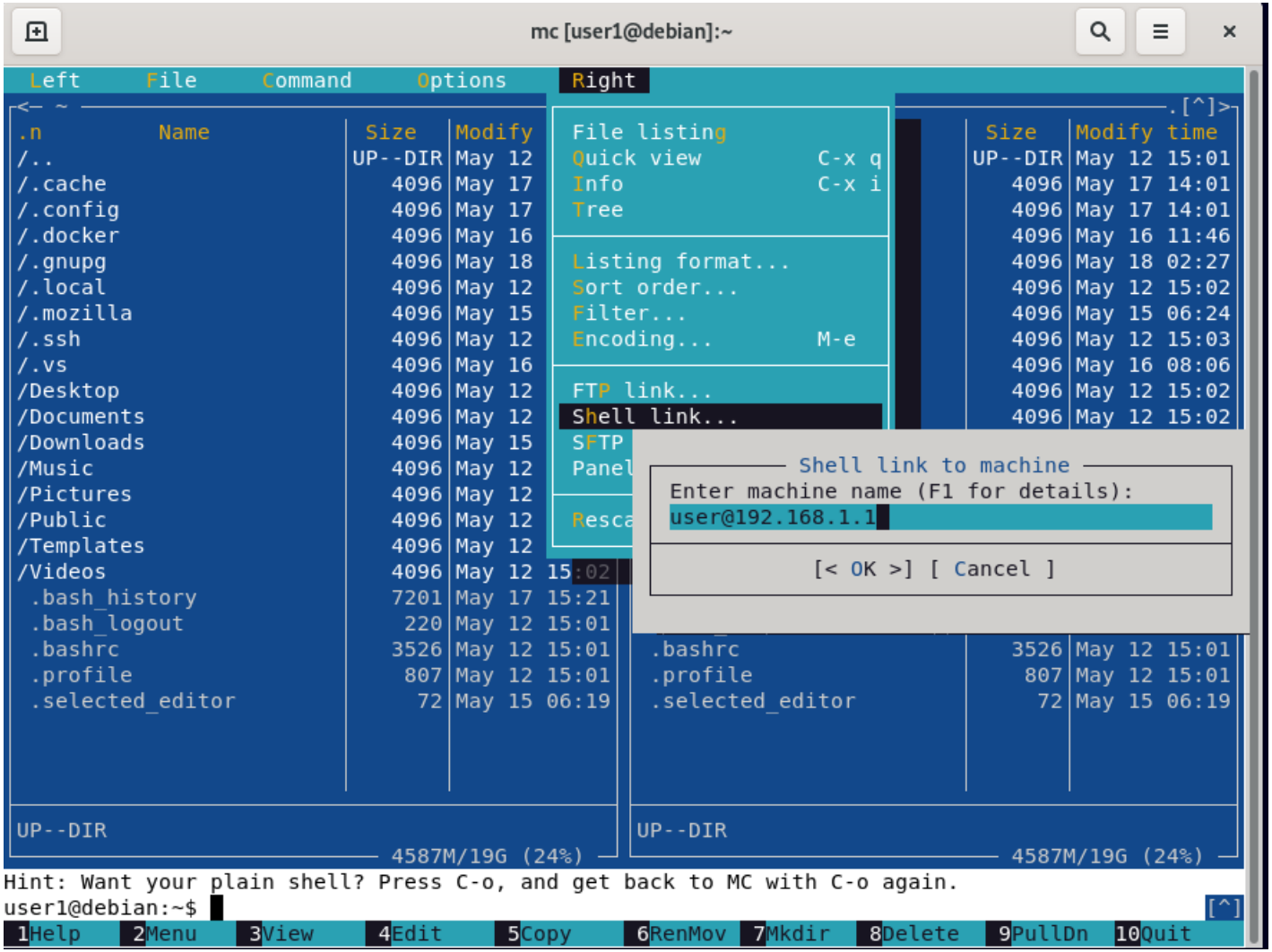
Дополнительно тут:
https://hackertarget.com/tcpdump-examples/

Перехват и анализ пакетов wireshark

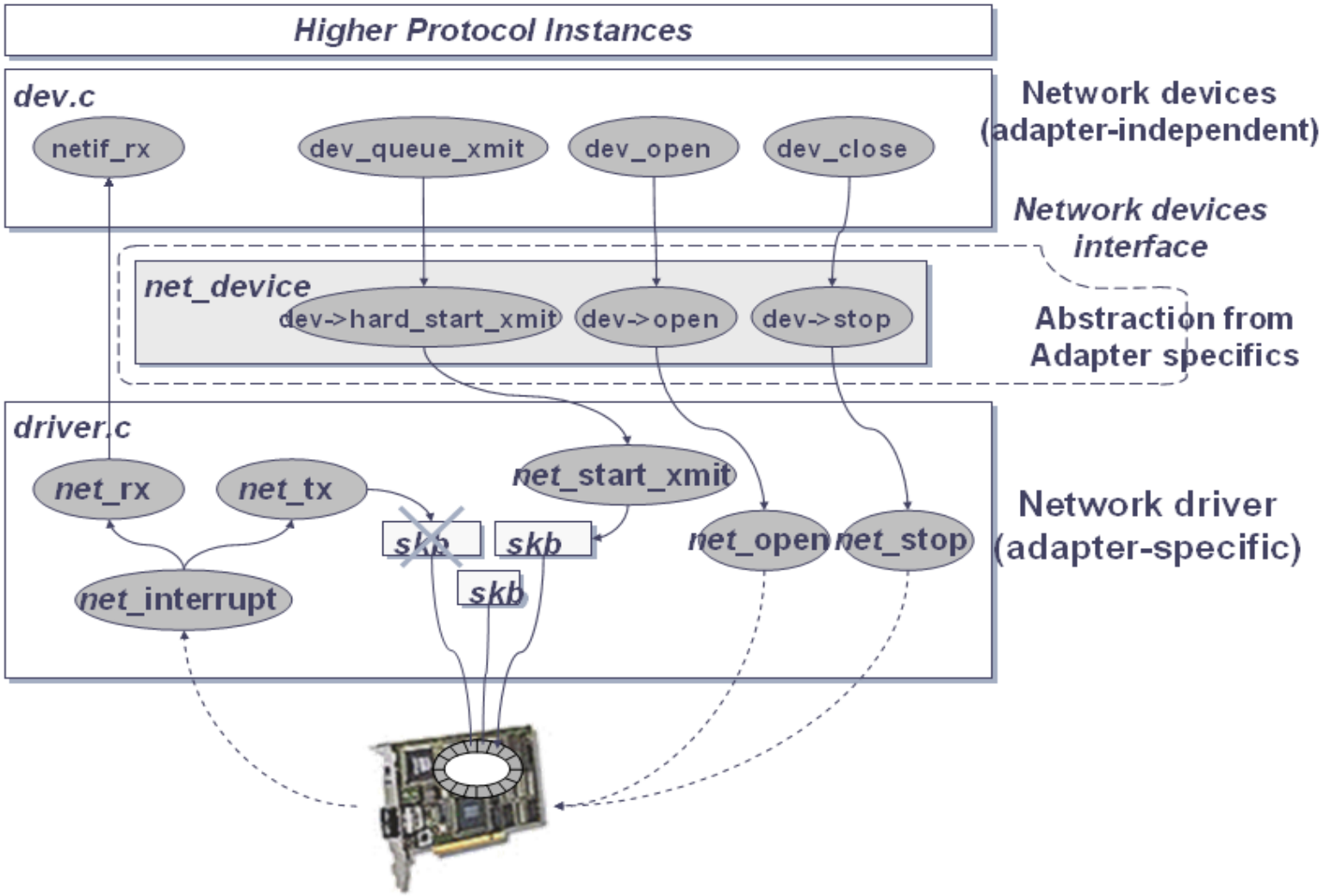
#apt-get update
#apt-get install wireshark tshark



Копирование файлов scp



Структура сетевого драйвера



На основе лекций Jenifier How.http://lion.cs.uiuc.edu/courses/cs498hou_fall05/lectures

Softirq

Есть несколько видов:

HI_SOFTIRQ, TIMER_SOFTIRQ, NET_TX_SOFTIRQ,NET_RX__SOFTIRQ, SCSI_SOFTIRQ, TASKLET_SOFTIRQ

Структура данных softirq_action

action, data

Возможны следующие операции:

Инициализация open_softirq

Активация rise_softirq

Выполнение do_softirq

Если irq_exit, ksoftirqd

Свойства Softirq

- Число softirq фиксированное и определяется при компиляции (см.Linux/include/linux/interrupt.h)
- Softirqs запускаются в порядке приоритета.
- Softirqs привязаны к ядрам CPU.
- Гарантируется, что softirq будет запущено на том ядре на котором оно запланировано к выполнению.
- Выполняются в контексте прерывния и имеют все его ограничения
- Не могут быть выполнены
- Выполняются атомарно
- На разных ядрах могут быть запушены несколько softirq даже одного тип

Функции сетевого драйвера

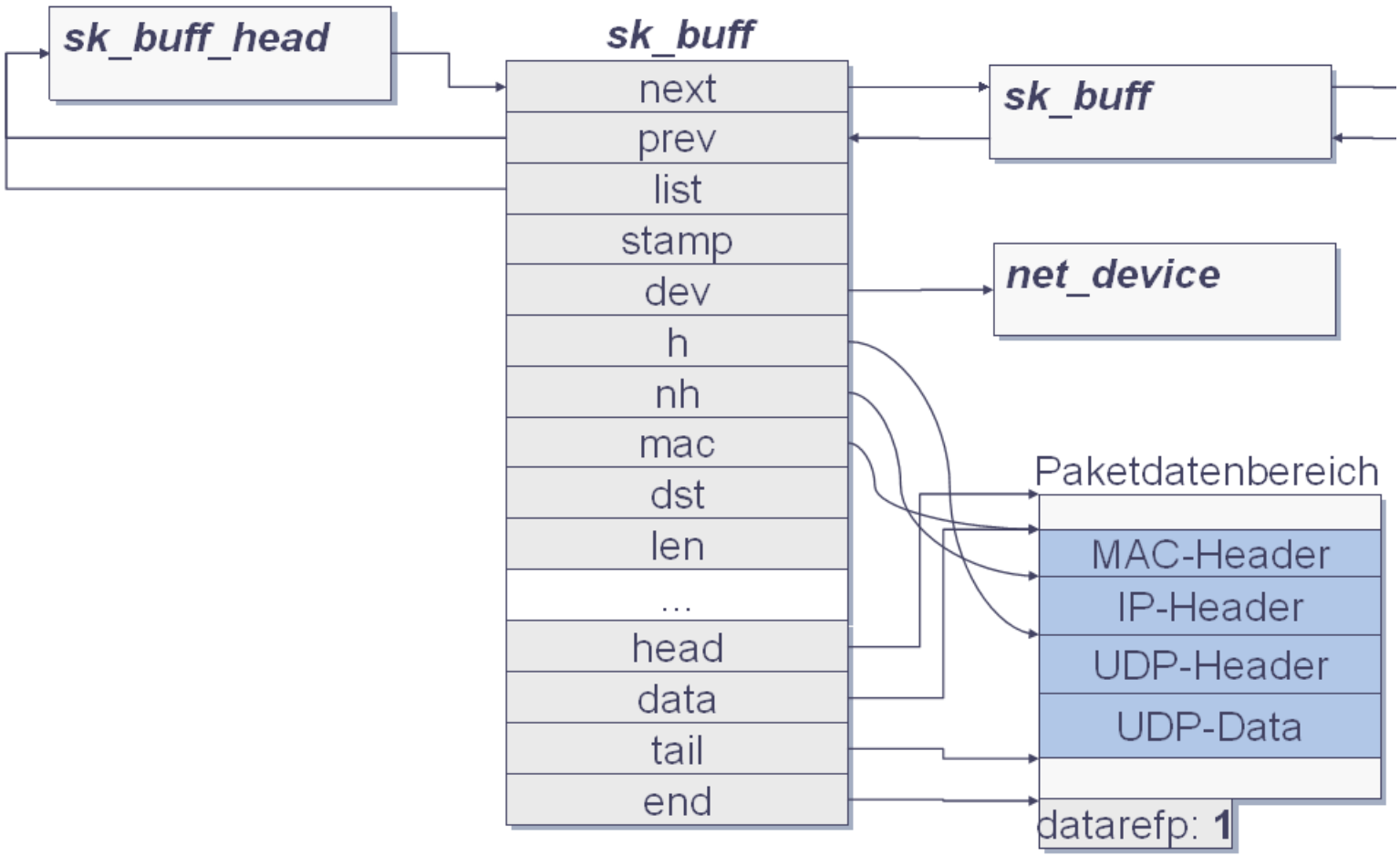
Сверху: Интерфейс со стеком сетевых протоколов

Снизу: Интерфейс с оборудованием (сетевой адаптер)

Необходимо обеспечить унифицированный доступ к оборудованию (которое может быть различным)

Предоставить сервисы для стеков протоколов (которые могут быть различными)

Представление пакета



Буфер сокета skb_buff

- Структура используется для работы с пакетом в сетевой подсистеме;
- Создаётся когда приложение передаёт данные в сокет или когда данные принимаются от сетевого адаптера (dev_alloc_skb() is invoked);
- Каждый сетевой уровень добавляет свой заголовок;
- Пакет копируется только два раза в момент передачи с уровня пользователя и при передаче в сетевой адаптер.

Функции для работы с sk_buff

Создание и освобождение:

struct sk_buff ***netdev_alloc_skb**(struct net_device *dev, unsigned int length) - Allocate a new sk_buff. The buffer has reserved headroom built in. The built in space is used for optimisations (reserved space size = NET_SKB_PAD).

void **dev_kfree_skb**(struct sk_buff *skb) — Free skb_buff.

Манипулирование указателями:

void **skb_reserve**(struct sk_buff *skb, int len) Increase the headroom of an empty &sk_buff by reducing the tail room. This is only allowed for an empty buffer.

unsigned char ***skb_put**(struct sk_buff *skb, unsigned int len) - Add data to an sk_buff

*char ****skb_pull**(struct sk_buff *skb, unsigned int len)*

*void ****skb_trim**(struct sk_buff *skb, unsigned int len)*

Получение информации:

unsigned int **skb_headroom**(const struct sk_buff *skb) - Return the number of bytes of free space at the head of an sk_buff

int **skb_tailroom**(const struct sk_buff *skb) - Return the number of bytes of free space at the tail of an sk_buff

Получение доступа к данным:

void **skb_copy_to_linear_data**(struct sk_buff *skb,const void *from,const unsigned int len)

void **skb_copy_to_linear_data_offset**(struct sk_buff *skb,const int offset,const void *from,const unsigned int len);

и другие

RISoft

Пример использвания

8390too

```
skb = netdev_alloc_skb(dev, pkt_size +
      NET_IP_ALIGN);
if (likely(skb)) {
  skb_reserve (skb, NET_IP_ALIGN);    /* 16 byte align
      the IP fields. */
```

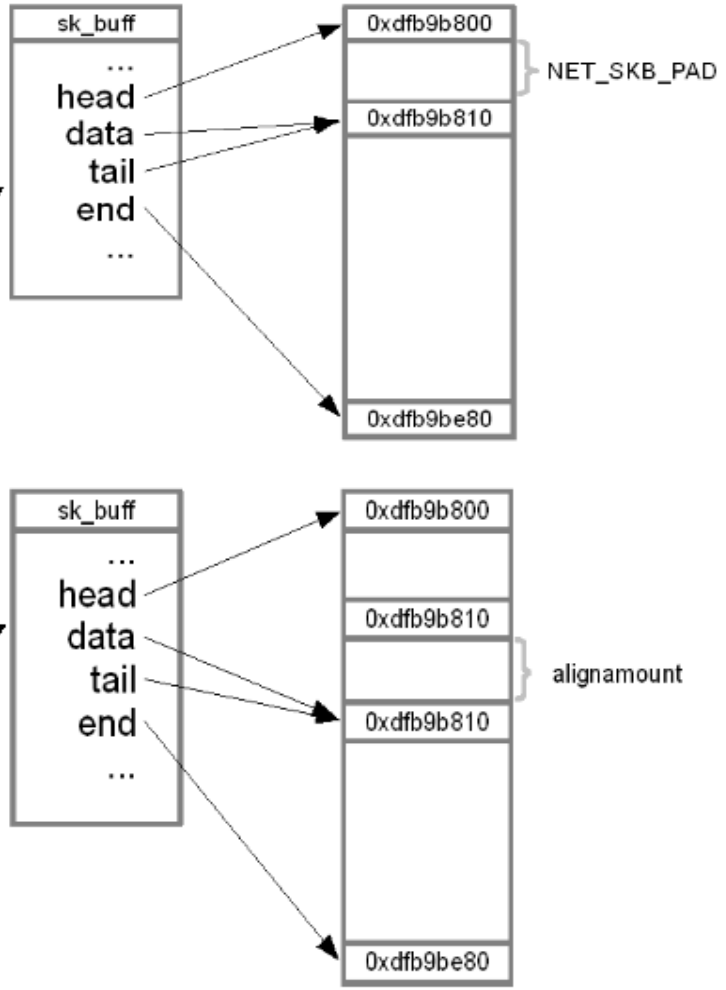
gianfar

gianfar.h

```
 #define RXBUF_ALIGNMENT 64
 #define DEFAULT_RX_BUFFER_SIZE  1536
```

gianfar.c

```
/* We have to allocate the skb, so keep trying till we
      succeed */
skb = netdev_alloc_skb(dev, priv->rx_buffer_size +
      RXBUF_ALIGNMENT);
if (!skb)
     return NULL;
alignamount = RXBUF_ALIGNMENT -
     (((unsigned long) skb->data) &
      (RXBUF_ALIGNMENT - 1));
/* We need the data buffer to be aligned properly.  We
      will reserve as many bytes as needed to align the
      data properly */
skb_reserve(skb, alignamount);
```



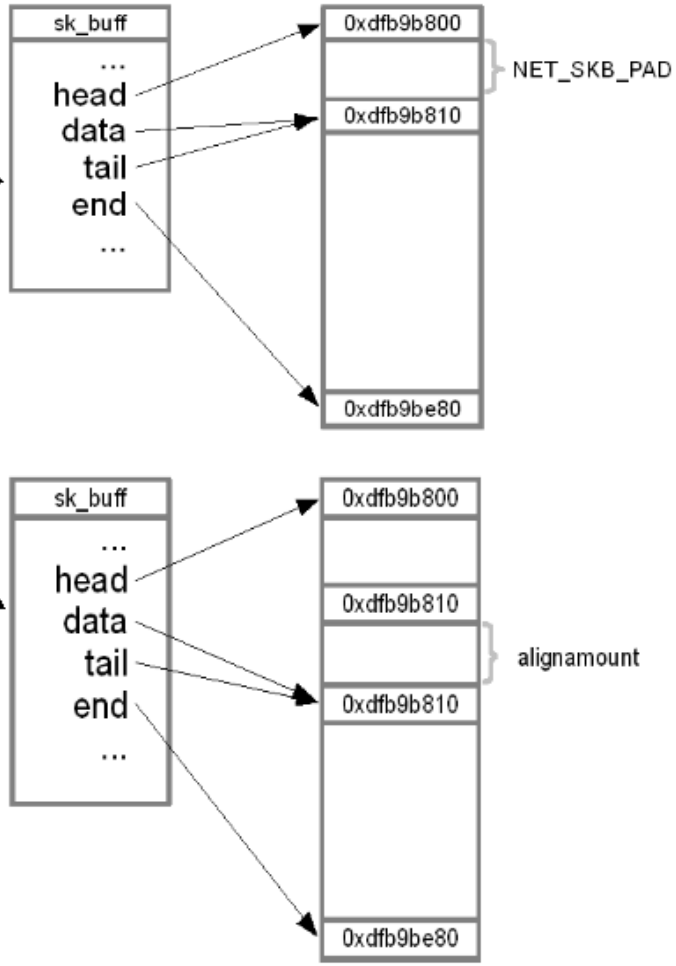
Пример использвания 2

8139too

skb_copy_to_linear_data (skb, &rx_ring[ring_offset + 4], pkt_size);
skb_put (skb, pkt_size);

gianfar

gianfar.c
skb_put(skb, length);



Заготовка сетевого драйвера

netframe.c

```
#include <linux/module.h>
#include <linux/etherdevice.h>
…
typedef  struct {
    /* driver's private data */
    } MY_DRIVERDATA;

char modname[ ] = "netframe";
struct net_device *netdev;
```

my_open()

my_stop()

my_hard_start_xmit()

my_isr()

my_get_info()

my_init my_exit

Функции специфичные для сетевого дрейвера

Функции инициализации и деинициализации модуля

На основе статьи www.cs.usfca.edu/~cruse/cs686s08/Lecture3 Programming Gigabit Ethernet

RISoft

Изменения структуры net_device

В «новых» ядрах структура net_device несколько изменена:

```
struct net_device {
  char              name[IFNAMSIZ];
// I/O specific fields
  unsigned long     mem_end;      /* shared mem end      */
  unsigned long     mem_start;    /* shared mem start    */
  unsigned long     base_addr;    /* device I/O address  */
  unsigned int      irq;                                  /* device IRQ number   */
...
  const struct net_device_ops *netdev_ops; // Management operations
...
};
struct net_device_ops {
  int       (*ndo_init)(struct net_device *dev);
  void      (*ndo_uninit)(struct net_device *dev);
  int       (*ndo_open)(struct net_device *dev);
  int       (*ndo_stop)(struct net_device *dev);
  netdev_tx_t           (*ndo_start_xmit) (struct sk_buff *skb,   struct net_device *dev);
  u16           (*ndo_select_queue)(struct net_device *dev,    struct sk_buff *skb);
....
};
```

Определено в include\linux\netdevice.h (linux 2.6.33.4)

RISoft

Назначение функций

open — Функция открытия устройства вызывается в момент инициализации сетевого устройства командой ifconfig (например, "ifconfig eth0 up"). В этом методе обычно выделяются системные ресурсы (I/O ports, IRQ, DMA, etc.), инициализируется аппаратная часть и увеличивается счётчик использования.

stop — функция остановки интерфейса. Вызывается при деактивации интерфейса при помощи ifconfig (for example, "ifconfig eth0 down"). Метод должен деактивировать аппаратное устройство и освободить выделенные ресурсы (I/O ports, IRQ, DMA, etc.).

hard_start_xmit — функция передачи пакетов в сеть. Первый аргумент — указатель на передаваемый пакет sk_buff.

get_stats — предоставляет доступ к статистике об интерфейсе Функция вызывается при вызове команды ifconfig ("ifconfig eth0").

Приём пакетов осуществляется в обработчике прерываний — нет метода в net_device

Пример module_init

```
#include <linux/netdevice.h>
typedef    struct { /* внутренние данные драйвера */ } MY_DRIVERDATA;
struct net_device      *netdev;
static const struct net_device_ops my_netdev_ops = {
        .ndo_open               = my_open,
        .ndo_stop               = my_close,
        .ndo_set_mac_address    = my_set_mac_address,
        .ndo_start_xmit                 = my_start_xmit,
} ;
static  int  init_module( void )  {
    netdev = alloc_etherdev( sizeof( MY_DRIVERDATA ) );
    if ( !netdev ) return -ENOMEM;

    netdev->netdev_ops = &my_netdev_ops;
    return  register_netdev( netdev );
}
```

Пример module_exit()

```
struct net_device    *netdev;

static  void  __exit  my_exit( void ) {
   unregister_netdev( netdev );
   free_netdev( netdev );
}
```

Открытие и закрытие устройства

- Сетевое устройство открывается администратором при помощи команд:

ifconfig eth0 up

ifconfig eth0 down

- Для передаче устройству параметров используется ifconfig, который в свою очередь использует ioctl

open()

- Метод вызывается, когда администратор при помощи ifconfig инициализирует адаптер (назначает IP адрес и включает (UP) интерфейс)
- В методе выполняется:

Сброс аппаратуры сетевого адаптера в известное состояние

Инициализация очередей пакетов ядра

Пример open

```
int  my_open( struct net_device *dev )
{
   /* Инициализировать  'private' данные */
   /* подготовить «железо» к работе */
   /* установить обработчик прерываний */
   /* разрешить NIC генерировать прерывания */

   netif_start_queue( netdev );
   return 0;  //SUCCESS
}
```

stop()

- Ядро linux вызывает эту функцию, когда сетевой интерфейс отключается (down)
- Отключение может произойти, если: вызван ifconfig eth0
- модуль драйвера выгружен rmmod

Пример stop()

```
int  my_stop( struct net_device *dev )
 {
   netif_stop_queue( netdev );

   /* kill any previously scheduled 'tasklets' (or other deferred work) */
   /* turn off the NIC's transmit and receive engines */
   /* disable the NIC's ability to generate interrupts */
   /* delete the NIC's Interrupt Service Routine */

   return 0;  //SUCCESS
 }
```

Посылка пакетов hard_start_xmit()

Функция вызывается, когда есть данные для передаче NIC

Ядро передаёт драйверу указатель на буфер сокета struct sk_buff, содержащий пакет для передачи

Open выполняет:

- инициировать передачу данных;
- обновить статистику;
- освободить структуру sk_buff.

Пример hard_start_xmit()

```
int  my_hard_start_xmit( struct sk_buff *skb,  struct net_device *dev )
 {
    /* Код для инициализации передачи апаратурой */

   dev->trans_start       = jiffies;
   dev->stats.tx_packets += 1;
   dev->stats.tx_bytes += skb->len;

    dev_kfree_skb( skb );
    return 0;  //SUCCESS
 }
```

Приём пакетов

- Аппаратура сетевого адаптера принимает пакеты асинхронно
- Для определения момента прихода данных используются прерывания

Что происходит при приёме пакета

Сетевой адаптер генерирует прерывание

Выделяется память для буфера сокета dev_alloc_skb ()

Определяется тип протокола

skb->protocol = eth_type_trans(skb, dev)

Буфер сокета помещается во входную очередь

Обновляется время приема последнего пакета

netif_rx(skb);

dev->last_rx = jiffies;

Пример обработки принятых данных

```
void  my_rxhandler( unsigned long data )  {
   struct net_device  *dev = (struct net_device *)data;
   struct sk_buff        *skb;
   int          rxbytes = 60;    // just an artificial value here
   skb = dev_alloc_skb( rxbytes + 2 );

   skb->dev = dev;
   skb->protocol = eth_type_trans( skb, dev );
   skb->ip_summed = CHECKSUM_NONE;
   dev->stats.rx_packets += 1;
   dev->stats.rx_bytes += rxbytes;

   netif_rx( skb );
 }
```

Задание

- Разработать сетевой драйвер, который должен
  - Открываться
  - иметь MAC адрес
  - иметь возможность имитировать отправку пакетов. Т.е. пакет должен получаться в hard_start_xmit,
  - Для полученного пакета должно при помощи pr_info печататься сообщение

Литература

1. An In-Depth Guide to iptables, the Linux Firewall https://www.booleanworld.com/depth-guide-iptables-linux-firewall/
2. Packet filtering in Linux - iptables, nftables and firewalld https://wyssmann.com/blog/2021/07/packet-filtering-in-linux-iptables-nftables-and-firewalld/
3. Nftables - Packet flow and Netfilter hooks in detail https://thermalcircle.de/doku.php?id=blog:linux:nftables_packet_flow_netfilter_hooks_detail
4. Почему сообщество разработчиков ядра заменяет iptables на BPF? https://habr.com/ru/company/otus/blog/646335/
5. Extending the Kernel with eBPF https://source.android.com/docs/core/architecture/kernel/bpf
6. eBPF Documentation https://ebpf.io/what-is-ebpf/
7. https://evilshit.wordpress.com/2013/12/17/how-to-set-up-a-stateful-firewall-with-iptables/
8. Использование связки iptables – geoip https://vds-admin.ru/security/iptables-geoip
9. xtables-addons: фильтруем пакеты по странам https://habr.com/ru/company/selectel/blog/511392/
10. How to Log IPTables - Send messages to rsyslog or journalctl https://www.putorius.net/how-to-log-iptables-send-messages-to.html
11. psad - Intrusion Detection with iptables Logs https://github.com/mrash/psad
12. https://cipherdyne.org/psad/
13. https://libellux.com/psad/#install-psad-from-source
14. Connman plugin that provides D-Bus API for controlling iptables rules. https://github.com/sailfishos/sailfish-connman-plugin-iptables
15. https://xakep.ru/2017/02/15/firewalld/
16. **https://opensource.com/article/18/10/iptables-tips-and-tricks**
17. **https://www.cyberciti.biz/tips/linux-iptables-examples.html**
18. Переход с iptables на nftables. Краткий справочник https://habr.com/ru/company/ruvds/blog/580648/
19. Get Started with nftables https://www.linode.com/docs/guides/how-to-use-nftables/
20. ZERO TRUST ARCHITECTURE WITH WIREGUARD https://www.procustodibus.com/blog/2022/08/zero-trust-architecture-with-wireguard/
21. https://home.regit.org/netfilter-en/nftables-quick-howto/
22. https://www.youtube.com/watch?v=EGKhIljDPCw
23. https://thermalcircle.de/doku.php?id=blog:linux:connection_tracking_1_modules_and_hooks
24. https://linux-audit.com/nftables-beginners-guide-to-traffic-filtering/
25. Гайд по межсетевому экранированию (nftables) https://habr.com/ru/post/684524/ (тут и про модели угроз)

Назначение NAPI

- Уменьшить загрузку процессора
- Не генерировать прерывание для каждого пакета
- Обработчик прерывания только фиксирует факт наличия пакета и планирует выполнения обработки асинхронным потоком (poll)
- Если стек переполнен, то лишние пакеты могут быть отброшены до помещения в стек
- NAPI можно не использовать

Функции обработчика прерываний

- Прерывание возникает по приходу пакета
- Дальнейшие прерываний запрещаются
- Запланировать выполнение функции опроса пакетов при помощи вызова:
- void netif_rx_schedule(struct net_device* dev)

Функция poll

int (*poll)(struct net_device* dev, int *budget)

Выполняемые функции (аналогично обработчику IRQ):

Выделяется память для буфера сокета dev_alloc_skb ()

Определяется тип протокола

skb->protocol = eth_type_trans(skb, dev)

Буфер сокета помещается во входную очередь

netif_receive_skb(struct skb_buf* skb) (Вместо netif_rx)

Обновляется время приема последнего пакета

dev->last_rx = jiffies;

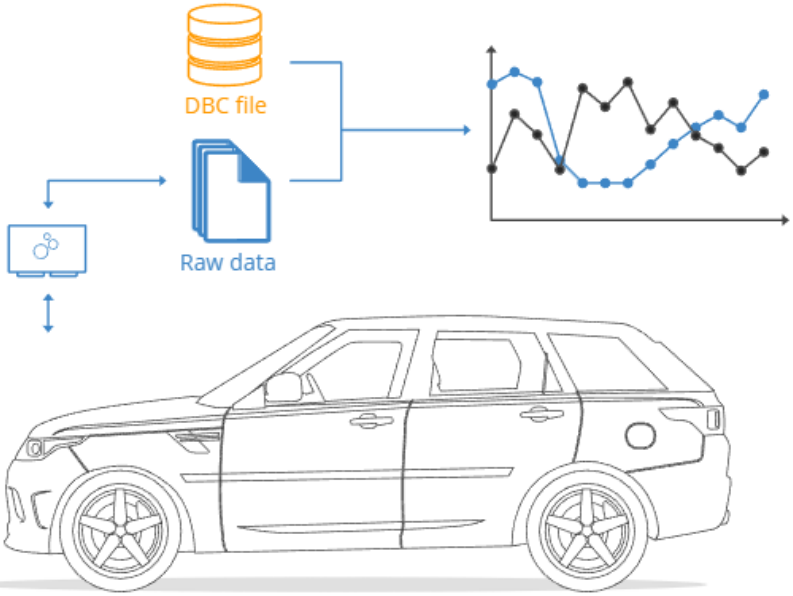
А еще дополнительно.....

Функция poll (2)

- В структуру net_device добавлено поле quota. Максимальное число пакетов, которое стек протоколов готов принять от драйвера. Если квота исчерпана, то poll не передает пакеты в ядро;
- В параметр budget — количество пакетов, которые драйвер может обработать;
- Уменьшить dev->quota и budget на число обработанных пакетов
- Если остались пакеты для обработки, то poll возвращает 1
- Если обработаны все пакеты, то:
- - разрешить прерывания;
- - выключить опрос (void netif_rx_copmplete (struct net_device* dev))
- - вернуть 0

CAN

CAN (англ. Controller Area Network — сеть контроллеров) — стандарт промышленной сети, ориентированный, прежде всего, на объединение в единую сеть различных исполнительных устройств и датчиков.

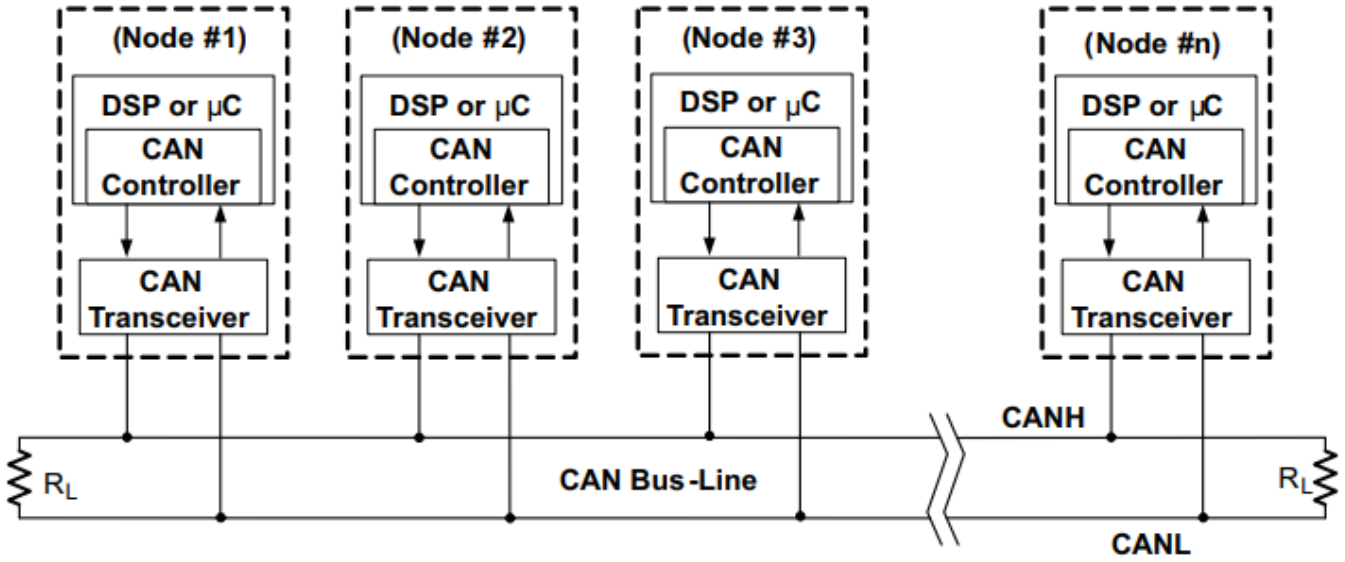


The CAN bus history in short

1986: Bosch developed the CAN protocol as a solution
1991: Bosch published CAN 2.0 (CAN 2.0A: 11 bit, 2.0B: 29 bit)
1993: CAN is adopted as international standard (ISO 11898)
2003: ISO 11898 becomes a standard series
2012: Bosch released the CAN FD 1.0 (flexible data rate)
2015: The CAN FD protocol is standardized (ISO 11898-1)
2016: The physical CAN layer for data rates up to 5 Mbit/s standardized in ISO 11898-2

hhttps://habr.com/ru/company/unet/blog/371293/
https://habr.com/ru/post/442184/
https://www.csselectronics.com/pages/python-can-bus-api

Официальная документация про socker can



Документация

- https://www.kernel.org/doc/Documentation/networking/can.txt
- Примеры
- https://github.com/craigpeacock/CAN-Examples

CAN frame

Standard CAN frame with 11 bits identifier (CAN 2.0A), which is the type used in most cars

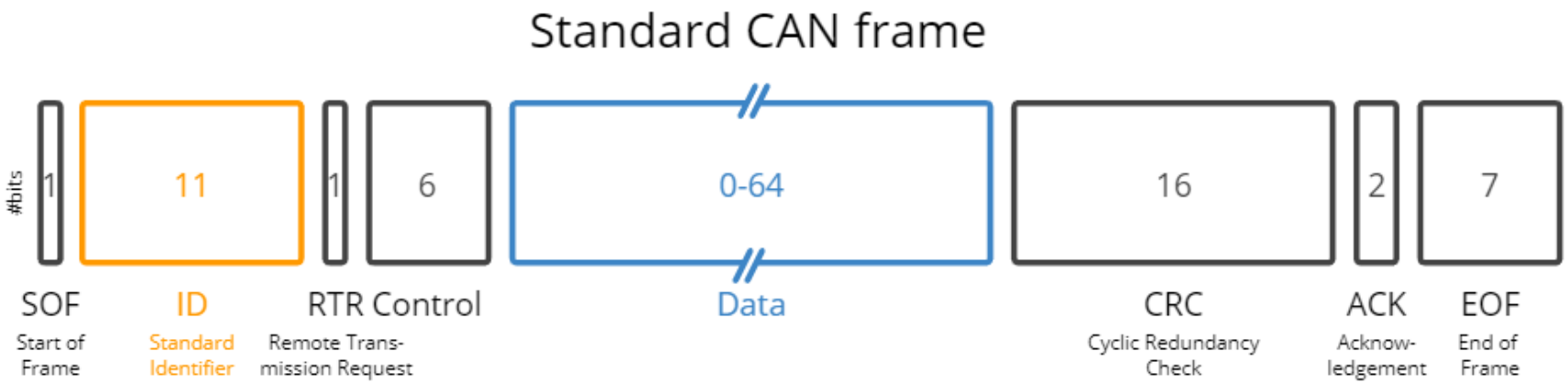
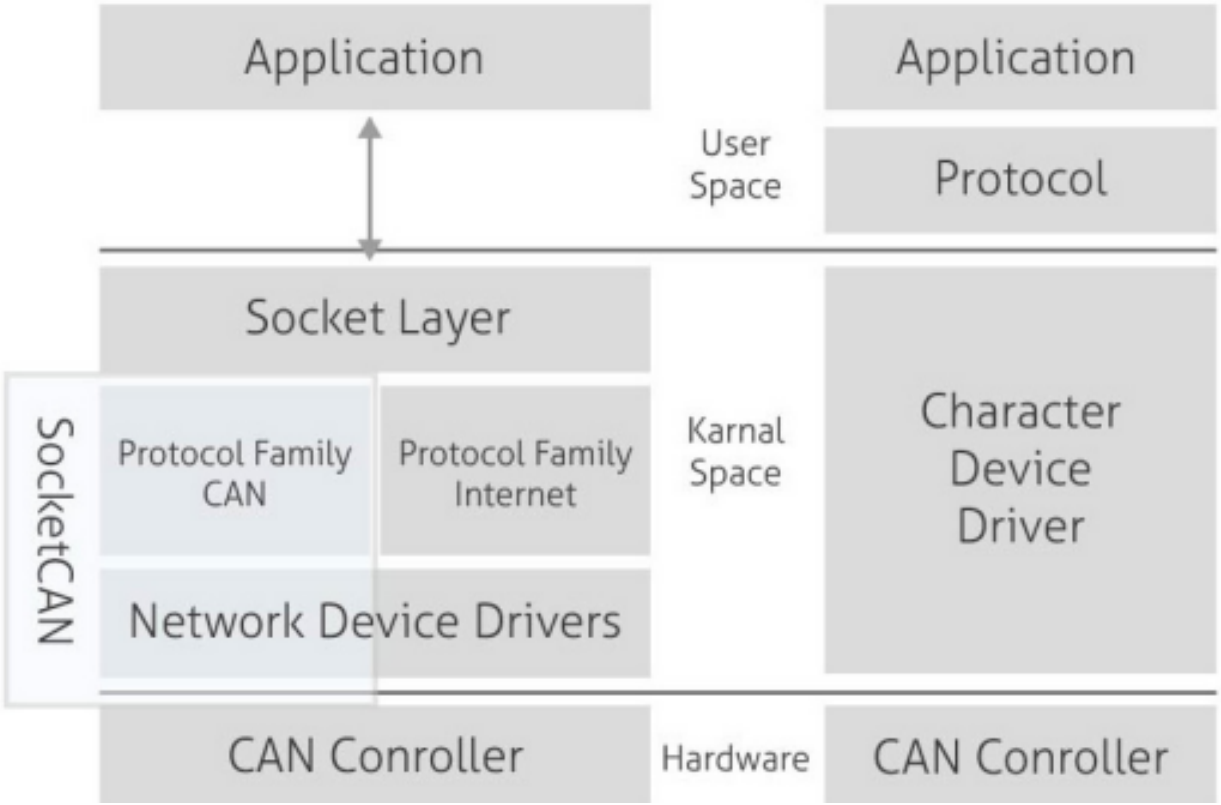


Схема взаимодействия сетевого драйвера



На основе статьи Bill Weinberg Porting RTOS Device Drivers to Embedded Linu. http://www.linuxjournal.com/article/7355

Примеры кода

Фрейм

```
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)));
};
```

Отправка:

```
struct can_frame frame;
frame.can_id = 0x555;
frame.can_dlc = 5;
sprintf(frame.data, "Hello");
if (write(s, &frame, sizeof(struct can_frame)) != sizeof(struct can_frame)) {
   perror("Write");
  return 1;
}
```

Примеры кода

Прием:

```
int nbytes;
struct can_frame frame;
nbytes = read(s, &frame, sizeof(struct can_frame));
if (nbytes < 0) {
   perror("Read");
   return 1;
}
printf("0x%03X [%d] ",frame.can_id, frame.can_dlc);
for (i = 0; i < frame.can_dlc; i++)
   printf("%02X ",frame.data[i]);
printf("\r\n");
```

Примеры кода – фильтрация пакетов

```
struct can_filter rfilter[1];
rfilter[0].can_id   = 0x550;
rfilter[0].can_mask = 0xFF0;
//rfilter[1].can_id   = 0x200;
//rfilter[1].can_mask = 0x700;
setsockopt(s, SOL_CAN_RAW, CAN_RAW_FILTER, &rfilter, sizeof(rfilter));
```

Интерфейс vcan

```
sudo ip link add dev vcan0 type vcan
sudo ifconfig vcan0 up
```

Утилиты canutils

Как поставить:
sudo apt-get install can-utils

Что входит в состав:

candump – Dump can packets – display, filter and log to disk.
canplayer – Replay CAN log files.
cansend – Send a single frame.
cangen – Generate random traffic.
canbusload – Display the current CAN bus utilisation.

Примеры использования изоляции ядер

Intel DPDK (http://dpdk.org/)