① 28.05

Лекция 28.05.

- NEXT_TXSOFTIRQ (transmit?) и NEXT_RX_SOFTIRQ (receive) предназначены для отложенной обработки входящих или отправляемых пакетов.
- Ранее описанные softirq составляют вектор (массив), кот в ядре Linux оформляется как стр-ра: (в коде ядра оформлен как стр-ра)

```
static struct softirq_action softirq_vec[NR_SOFTIRQS]
        __cacheline_aligned_in_smp;

char *softirq_to_name[NR_SOFTIRQS]={"HI", "TIMER",
(*)     "NEXT_TX", "NET_RX", "BLOCK", "BLOCKIOPOLL", "TASKLET",
        "SCHED", "HRTIMER" <> "RCU"};
```

- Меньшие номера имеют более высокий приоритет (по порядку перечисления идут соотв-ие номера)
- Писать свой softirq ниже по приоритету (уровню) чем tasklet нет смысла

Тасклет как отложенное действие видом softirq (это видно в (*)).

Для написания собств. softirq система предоставляет соотв. ф-ции ядра: →

28.05 ②

сюда передаем свой обработчик прерывания

```
void __init mysoftirq_init()
{
   ...
   request_irq (irq, XXX_interrupt, 0, "xxx", Null);
   open_softirq( XXX_SOFT_IRQ , XXX_handler, Null);
   ...
}
```

открываем softirq / создаем

```
void open_softirq (int nr, void (*action) struct softirq_action *))
{
   softirq - vec[nr] .action = action;
}
```

обработчик аппар. прерыв-я

```
/* interrupt handler */
static irqreturn_t XXX_interrupt (int irq, void *dev_id)
{
   {pending - добавление}
   ...
   /* mark softirq as pending */
   raise_softirq( XXX_SOFT_IRQ);
   ...
  return IRQ_HANDLER;
}
```

отмеченные softirq

В методичке есть картинка (входящие сетевые пакеты)

raise_softirq – в этой ф-ции вызывается raise_softirq_irqoff, которая выполняет основ. действие

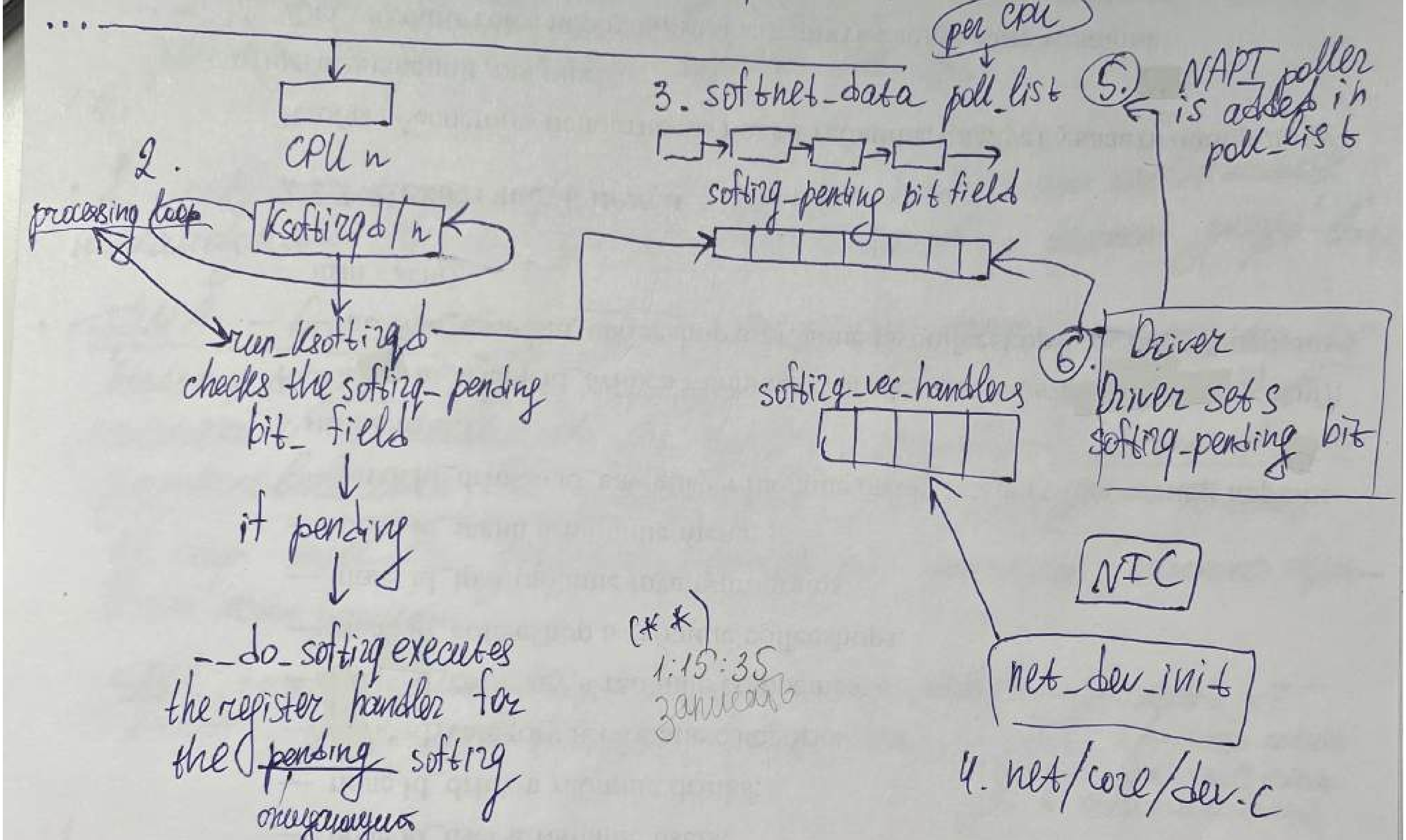
```
void raise_softirq(unsigned int nr)
{  unsigned long flags;
   local_irq_save(flags); // сохранение флагов предыдущего softirq
   raise_softirq_irqoff(nr);
   local_irq_restore(flags);
}
```

Схемы (условные):

28.05 ③

- В smp архитек-ре равноправные процессоры. Они все могут параллельно обрабатывать входящие пакеты.
- Есть ksoftirq daemon per cpu — на cpu

(создание ksoftirqd потока) один на cpu
1. create ksoftirqd kernel thread (per CPU)



(**) 1:15:35 записать

Пояснение: ksoftirqd отслеживает сеть ядра, выполняя петлю цикла

1. создание потока ksoftirqd
2. этот поток крутится в петле цикла; этот цикл в функц run_ksoftirqd. Функц run_ksoftirqd проверяет добавленные в очередь события (softirq pending bit field), чтобы их обработать

4. –установка соответствующего девайса. У девайса д.б. драйвер.

6. Драйвер устанавливает этот список, т.е. pending bit.

5. Napi – new api. New-api появилось вследствие тем, что интенсивность

28.05

работы с сетью в совр. системах резко возросла. Каждый входящий пакет приходит на сетевую карточку. Сет. карта — это девайс, а с девайсами система штатно работает через прерыв-я, но прерыв-е это очень затрат. дей-е в системе.

(3-) Системы в этом смысле поменялись (меняется начинка, его быстродействие, ёмкость памяти и т.д.) → → и на это отреагировали введением NAPI: замена прерываний опросом (polling)

poll list per cpu — лист опроса на cpu

*

- В совр. системах прямой доступ к памяти один из основ. механизмов взаимод-я процессоров с внеш. устр-вами

- DMA — Direct memory access: если бы этого механизма не было, то для передачи данных от внеш. устр-ств в память и из памяти на внеш. устр-ва постоянно задействовались бы регистры процессора. То есть пр-сор был бы загружен передачей данных, вместо обработки этих данных.

- DMA стало основным средством передачи данных от внеш. устройств в операт. память. (чтобы пр-сор мог обраб-ть данные, они д.б. в опер памяти)

!! → Пакет помещён в кольцевой буфер; кольц. буфер находится в операт. памяти — помещается он туда через DMA. Как раз обозначается добавление бита в эту очередь битов (что..?). Пакет, кот. требует обработки. Обрабатывать его будет отложенный обработчик. (под softirq будет выполняться)
NET RX TX softirq

- Управляет выполнением этого softirq RX — ksoftirqdaemon, он управляет. Он работает с этой битовой очередью, которая обозначает, что ∃ событие, которое требует обработки

28.05

Сетевая подсистема Linux построена по примеру стека BSD. В ней приём и передача данных на транспортном и сетевом уровне происходит с помощью интерфейса сокетов.

- В совр. системах ∃ napi (new api); основ. цель napi — сократить кол-во прерываний, генерируемых при получении пакетов. В napi механизм прерываний сочетается с механизмом опроса. В „napi совместимых драйверах" прерывания отключаются, когда на интерфейс приходит пакет. Обработчик в этом случае только вызывает RX_SCHEDULE, что гарантирует обработку пакета в дальнейшем.

и

Тасклеты.

- Тасклет — частный случай реализации softirq. Тасклеты базируются на softirq.
- Но softirq можно сериализировать (т.е. один и тот же тип softirq может выполняться параллельно на разных пр-ссорах), поэтому в softirq д.б. реализовано взаимоисключение.
- Тасклеты сериализоваться не могут, т.е. один тип тасклета может выполняться только на одном процессоре (параллельно выполняться не может) => тасклеты не требуют реализации взаимоисключения.
- Но разные тасклеты могут выполняться параллельно.
- Один и тот же тасклет может выполняться в системе в единств. экземпляре. Разные тасклеты могут выполняться ||.

Т.о. тасклеты не могут использ-ся для обработки таких напряженных действий, подобных сетевыми пакетами. Т.е. тасклеты явл. компромиссом м/у производительностью и простотой использования.

- Softirq регистрируются статически в ядре при инициализации системы. Тасклеты же могут быть зарегистрированы и статически, и динамически.

28.05

В ядре есть стр-ра :

```
< linux/interrupt.h>
struct tasklet_struct
{
  struct tasklet_struct *next; // связ. список
  unsigned long state; //текущ. состояние
  atomic_t count; //счётчик ссылок на тасклет
  void(*func)(unsigned long); //обработчик
  unsigned long data;
};
```

Значение поле state

```
enum {
  TASKLET_STATE_SCHED,
  TASKLET_STATE_RUN
};
```

имеет смысл только для SMP

- Если count == 0, то тасклет можно выполнять, иначе нельзя выполнять
- Для <u>статического</u> создание тасклета определены 2 макроса:

(1) DECLARE_TASKLET(name, func, data); ← данные для ф-ции func

и

(2) DECLARE_TASKLET_DISABLED(name, func, data);

← Оба макроса статически создают экземпляр стр-ры struct tasklet с именем name

(1) - создаёт тасклет, у кот. count = 0 и этот тасклет разрешён

(2) - создаёт тасклет, у кот. count = 1 и этот тасклет не разрешен

- Динамически тасклеты создаются с помощью ф-ции:

вызов ф-ции — tasklet_init(pt, tasklet_func, data);

pt — указ-ль на стр-ру tasklet_struct

tasklet_func — обработчик

data — передаваемые данные

объявление — extern void tasklet_init(struct tasklet_struct *pt, void (*func)(unsigned long), unsigned long data);

Планирование тасклетов

28.05.

⑦

· В init мы можем динамически зарегистрировать свой тасклет, а в обработчике прерывания запланировать выполнение ф-ции тасклета.

ей передается единств. аргумент – указ-ль на tasklet_struct → tasklet_schedule(); // очередь tasklet_vec
tasklet_hi_schedule(); // очередь tasklet_hi_vec

· Запланированные тасклеты будут храниться в 2х разных связ. списках:

1) тасклеты с обычным приоритетом в очередь tasklet_vec
2) тасклеты высокоприоритетные в очереди tasklet_hi_vec

· Одна из важнейших особенностей тасклетов: тасклеты не могут блокироваться.

Поэтому даже если в тасклете необходимо написать взаимоисключение, то семафоры не м.б. использованы в тасклетах. Использ. только spin lock.

· В smp архитек-рах могут использ-ся специально определённые ф-ции:
CONFIG_SMP
· tasklet_trylock();
· tasklet_unlock();
← в теле этих ф-ций вызывается команда test_and_set

этот api устарел

· В elixir написано: · Тасклеты – это многопоточ. аналог BHs (bottom half soft irq).
Отличие tasklet от softirq: тасклет выполн. только на 1 пр-ссоре одновременно
Отличие tasklet от BHs: разные тасклеты могут запускаться одновременно на разн. про-ссорах

· Ф-ция tasklet_schedule() гарантирует, что тасклет будет когда-нибудь выполнен на одном из пр-ссоров. Если тасклет уже запланирован, но его выполнение ещё не произошло, он будет выполнен только 1 раз. (т.е. если повторно заплани р. тасклет, он будет выполнен 1 раз)

Тасклет строго организован по отношению к самому себе, но не по отношению к другим тасклетам.

Дополнение к (**): (объяснение про нумерацию 1.2.,..)

Если рассматривать во времени, то сначала мы запускаем систему. Когда запускается система, начинает выполняться ksoftirq daemon (поэтому 1.). Он (daemon) начинает крутиться в очереди, инициализируя список битов (событий). Несмотря на то, что эта картинка демонстрирует работу с сетевой карточкой (драйвером сетевой карточки), ~~[illegible]~~ эта битовая очередь должна содержать биты всех softirq, потому что ksoftirq daemon не обрабатывает только события, связ. с сетевой подсистемой (3.)

Потом подключается сетевая подсистема, device, драйвер и уже он начинает (драйвер) в очередь битов заносить информацию о событии (поступление пакета — это событие).