

Рязанова, лекция

...///
/////

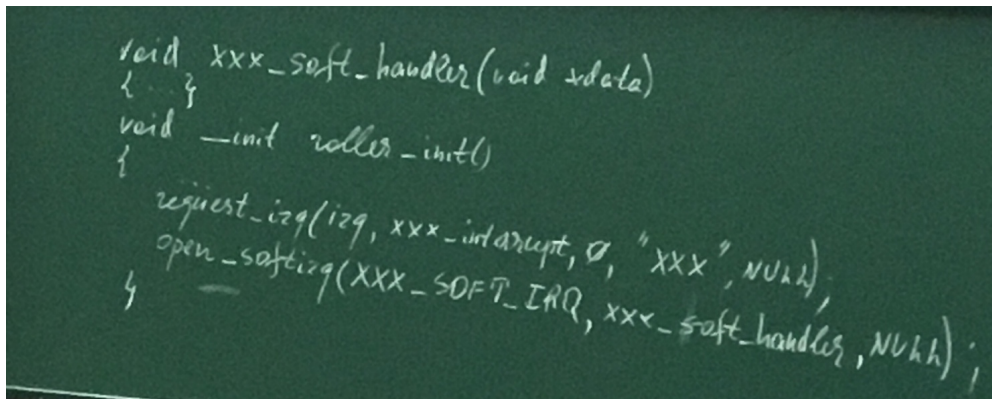
Добавить новый уровень разработчика - XXX_SOFT_IRQ, можно только перекомпилировав ядро.

Отложенное прерывание с меньшим номером выполняется дальше, то есть его приоритет выше.

Для создания нового уровня soft_IRQ нужно

1. Определить новый тип прерывания, вписав его в константу XXX_SOFT_IRQ в перечислении
2. Вовремя инициализации модуля должен быть зарегистрирован обработчик отложенного прерывания с помощью вызова open_softirq

soft_irq - действия, которые завершают действие прерывания.

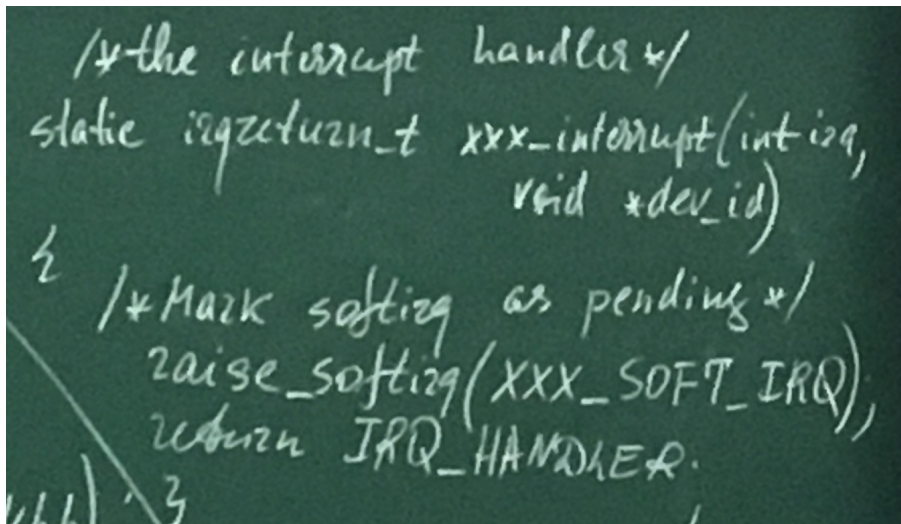


```
void xxx_soft_handler(void *data)
{
}
void __init roller_init()
{
    request_irq(irq, xxx_interrupt, 0, "xxx", NULL);
    open_softirq(XXX_SOFT_IRQ, xxx_soft_handler, NULL);
}
```

Функция open_softirq имеет три параметра: XXX_SOFT_IRQ - индекс, второй параметр обработчик, третий параметр значение поля data.

Очевидно, что функция обработчик soft_irq должна соответствовать правильному прототипу

3. Зарегистрированная soft_irq должна быть поставлена в очередь на выполнение. Для этого оно должно быть возбуждено с помощью функции raise_softirq - имерация отложенного прерывания. Обычно обработчик аппаратного прерывания, то есть верхняя половина перед возвратом управления возбуждает свой обработчик отложенного прерывания



```
/* the interrupt handler */
static irqreturn_t xxx_interrupt(int irq,
                                void *dev_id)
{
    /* Mark softirq as pending */
    raise_softirq(XXX_SOFT_IRQ),
    return IRQ_HANDLER.
}
```

Проверка ожидающих выполнения отложенных прерываний и их запуск осуществляется в следующих

случаях:

1. При возврате прерывания
2. В контексте ksoftirqd
3. В любом коде ядра, в котором явно проверяются и запускаются ожидающие обработчики отложенных прерываний, как это делается например в сетевой подсистеме

Независимо от метода вызова `soft_irq` его выполнение осуществляется функцией `do_soft_irq`, которая в цикле проверяет наличие отложенных прерываний. Несмотря на то, что у `softirq` есть приоритеты, `softirq` никогда не вытесняет другой `softirq`. Единственное событие, которое может вытеснить `softirq`, это аппаратное прерывание. При этом на другом процессоре может выполняться другой обработчик этого же `softirq`. В результате для `softirq` остро стоит проблема взаимного исключения. Это означает, что функция должна быть реинтерабельной, а если есть критический участок, то он должен быть защищен. Демон `ksoftirqd` это поток ядра каждого процессора, то есть `per cpu`.

Когда машина нагружена мягкими прерываниями, которые, как правило, обслуживаются по возвращении из аппаратного прерывания, то возможна ситуация когда такое `softirq` переключается значительно быстрее чем может быть обслужено. Это связано с тем, что возможны ситуации, в которых `IRQ` приходят очень быстро, одно за другим, и в результате `os` не может закончить обслуживание одного до прихода другого. Например, это может произойти когда сетевая карта с высокой скоростью получает пакеты в течение короткого промежутка времени. В результате операционная система не может с этим справиться и создает очередь для последовательной обработки `softirq` с помощью специального процесса, который называется `ksoftirqd`.

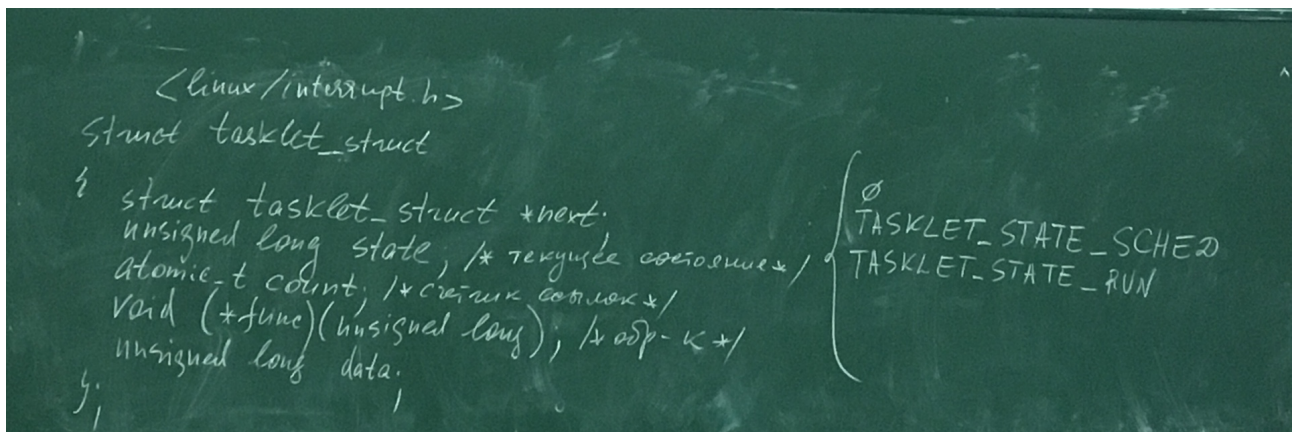
Сб, 27 апреля

Если ksoftirqd занимает больше чем какой-то достаточно небольшой процент процессорного времени, это говорит о том, что машина находится под большой нагрузкой прерываний.

Tasklet.

Отмечается, что tasklet'ы это частный случай softirq, но это отложенные прерывания, для которых обработчик, то есть tasklet не может выполняться одновременно на нескольких процессорах, в отличие от softirq. Для них по этой причине не нужно реализовывать средства взаимного исключения.

Tasklet'ы проще всего понимать как простые softirq. Разные Tasklet'ы могут выполняться параллельно на нескольких процессорах, но одного типа не могут. Поэтому tasklet'ы являются компромиссами по производительности и простотой. Tasklet'ы могут быть зарегистрированы в системе как статически, так и динамически.



TASKLET_STATE_SCHED - запланирован, __RUN - выполняется.

Count счетчик ссылок на tasklet, если 0, то tasklet, разрешен и может выполняться если он запланирован, иначе он запрещен и выполняться не может.

Для того, чтобы запланировать tasklet на выполнение должна быть вызвана функция tasklet_schedule(). Аналогично softirq планирование tasklet'ов на выполнение выполняет interrupt_handler. Для оптимизации tasklet обрабатывается на том процессоре, на котором его запланировал обработчик прерывания.

Сб, 27 апреля

Статические tasklet'ы создаются с помощью двух макросов, которые определены в linux/interrupts.h - DECLARE_TASKLET(name, func, data); DECLARE_TASKLET_DISABLED(name, func, data). Оба макроса статически создают экземпляр структуры struct tasklet. Соответственно с именем name, вызываемой функцией func и какими-то данными data. Первый макрос создает tasklet, у которого поле count равно нулю и соответственно этот tasklet разрешен, второй создает экземпляр с count равным 1, и, соответственно такой tasklet будет запрещен.

При динамическом создании tasklet'а объявляется указатель на структуру.

```
struct tasklet_struct *mytasklet;
```

Для инициализации tasklet'а вызывается функция:
tasklet_init(mytasklet, tasklet_handler, data);

//tasklet_handler prototype:

```
Void tasklet_handler(unsigned long data);
```

В обработчиках tasklet'ов нельзя использовать семафоры, так как они не могут блокироваться. Если используются одинаковые данные с обработчиком прерывания или с другим tasklet'ом, то они должны быть защищены spin блокировкой.

Планирование tasklet'ов.

Tasklet'ы могут быть запланированы с помощью следующих функций:

```
tasklet_schedule(); tasklet_hi_schedule();
```

Этим функциям передается единственный аргумент - указатель на структуру tasklet_struct.

Например tasklet_schedule(&irq_tasklet);

Запланированные на выполнение tasklet'ы хранятся в связанных списках. Обычные tasklet'ы будут находиться в связанном списке - tasklet_vec, а высокоприоритетные в списке tasklet_hi_vec, оба списка состоят из экземпляров структуры tasklet_struct.

После того как tasklet запланирован он будет запущен только один раз даже если он был запланирован на выполнение несколько раз. Для

Сб, 27 апреля

отключения заданного tasklet'а используется функция `tasklet_disable()` или функция `tasklet_disable_nosync()`. Первая функция не сможет отменить tasklet, который уже выполняется, вторая может прервать выполнение tasklet'а. Для активации tasklet'а используется функция `tasklet_enable()`.

На tasklet'ах определены специальные функции блокировки - `tasklet_trylock()`, `tasklet_unlock()`. Внутри этой функции вызывается команда

```
static inline int tasklet_trylock(struct
                                tasklet_struct *t)
{
    return !test_and_set_bit(TASKLET_STATE_
                             RUN, &(t)->state);
}
```

```
static inline void tasklet_unlock(struct tasklet_struct *t)
{
    smp_mb_before_atomic();
    clear_bit(TASKLET_STATE_RUN, &(t)->state);
}
```