Рязанова, лекция #6 Аппаратные прерывания.

Понятно, что для разработчика имеют значение функции, которые вызываются, но также важно понимать особенности этого процесса, процесса завершения получения данных от устройства, аппаратные прерывания от устройств приходят на контроллер прерываний в итоге контроллер формирует сигнал прерывания, поступающий на процессор и процессор начинает обрабатывать это прерывание.

Механизм аппаратных прерываний - это естественный асинхронный параллелизм.

Общая модель обработки аппаратных прерывания является принципиально-архитектурно-зависимой. Например, в MS-DOS использовался контроллер прерываний peek, из которого формировался вектор прерывания для адресации смещения по схеме 4 байта. Номер аппаратного прерывания использовался в качество смещения в таблице векторов, которая начиналась с нуля. В 32-разрядных системах использовался apeek, который формировал вектор прерывания, который является смещением к 8 байтовому дескриптору прерывания в таблице дескрипторов пребываний (IDT).

Основная функция аппаратных прерываний заключается в том, чтобы позволить периферийным устройствам взаимодействовать с процессором, чтобы информировать его о завершении некоторых задач, или о возникновении ошибочных ситуаций или каких-то других событий, которые требуют внимания со стороны системы. В результате возникновения прерываний, используя соответствующую схему адресации обработчиков прерываний, такой обработчик от конкретного устройства начинает выполняться. Очень часто такой обработчик называют (ISR) Interrupt Service R???. Обработчик прерывания выполняется в режиме ядра в системном контексте, так как прерванный процесс, обычно, не имеет никакого отношения к возникшему в системе пребыванию, его обработчик не должен обращаться к контексту процесса. По этой причине он не обладает правом блокировки. Однако, прерывание оказывает некоторое влияние на выполнение текущего процесса. Время, потраченное на обработку прерывания, является частью кванта, выделенного процессу. Например, обработчик прерывания от системного таймера использует

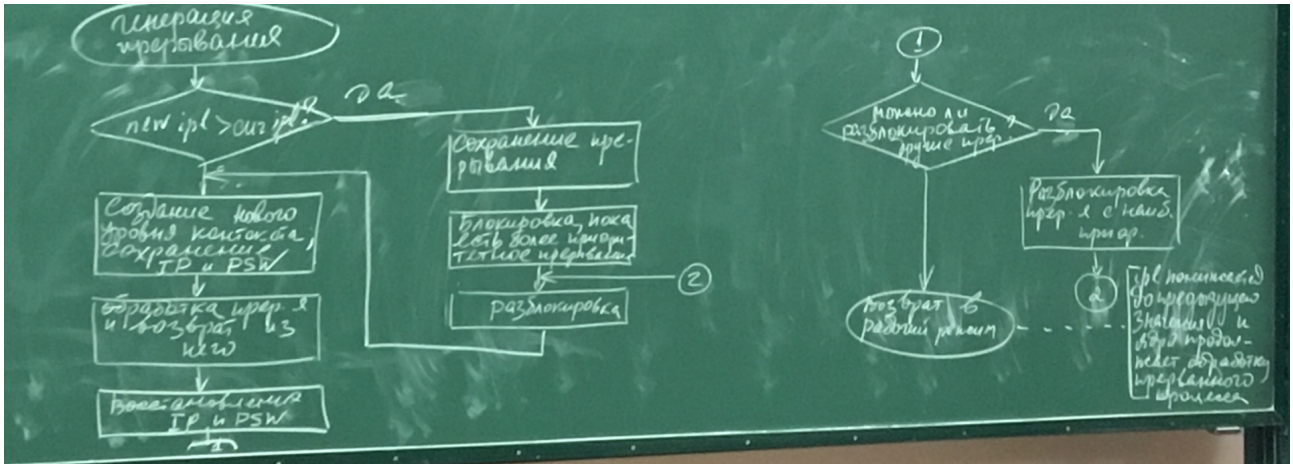
тики текущего процесса, и поэтому нуждается в доступе к его структуре. В UNIX это struct proc.

Важно отметить, что контекст процесса не полностью защищен от обработчика прерывания. Поэтому неверно написанный обработчик может нанести вред любой части адресного пространства процесса.

Кроме аппаратных прерываний в системе также существуют программные прерывания (системные вызовы) и исключения (юниксоиды говорят программные и системные прерывания).

Очевидно, что разные типы прерываний и имеют разную важность для работы системы. Кроме того, они могут иметь разный объем кода, часто довольно большой. Под большим объемом понимается объем, который требуется для вычислений в течение нескольких тиков системного таймера. В силу этого в UNIX/LINUX вводятся уровни приоритетов прерываний (ipl) interrupt priority level.

Рассмотрим общий *алгоритм обработки прерываний*, предлагаемый юниксоидами: (в нашем ГОСТе нет у стрелок писать не положено)



Аппаратные прерывания имеют очень высокий уровень приоритетов, из которых прерывание от системного таймера имеет наивысший приоритет.

Очевидно, что прерывания от внешних устройств являются высокоприоритетными, так как должны быть обработаны быстрее всех, клавиатура и мышь имеют наиболее высокий уровень привелегий для того, чтобы их обрабатывали сразу, потому что это интеркативные устройства.

Говоря о прерываниях, речь идет о драйверах. Каждое устройство имеет один драйвер и если устройство использует прерывание, то этот драйвер регистрирует один обработчик прерывания. Драйверы могут

регистрировать обработчик прерывания и разрешить определенную линию обработки прерывания, посредством функции request_irq

int request_irq(unsigned int irq, irqreturn_t (*handler)(int, void*, struct pt_regs*), unsigned long irqflags, const char *devname, void *dev_id) *- регистрирует обработчик прерывания*

Можно обратиться к книжке Alessandro Rubni, Yonathan Corlet Linux Device Drivers

extern void free_irq(unsigned int irq, void *dev) - освобождает обработчик прерывания, чтобы он больше не выполнялся.

Первый параметр irq определяет номер прерывания. Для некоторых устройств, например, для legacy device, таких как системный таймер и клавиатура, эта величина устанавливается аппаратно (hard coded). Для большинства других устройств она устанавливается динамически.

Второй параметр handler - указатель на обработчик прерывания, который обслуживает данное прерывание. Именно эта функция будет вызвана, когда возникнет соответствующее прерывание.

```
enum irqreturn {
	IRQ_NONE = (0<<0), // прерывание было не от соответствующего девайса или прерывание не было обработано
	IRQ_HANDLED = (1<<0), // прерывание было обработано соответствующим устройством
	IRQ_WAKE_THREAD = (1<<1) // обработчик запрашивает пробуждения нити обработчика
};

typedef enum irqreturn irqreturn_t;
#define IRQ_RETVAL(x) ((x) ? IRQ_HANDLER: IRQ_NONE)
```

Важно, чтобы функция free_irq, освобождала именно то устройство, драйвер которого установил этот обработчик. void *dev - указатель на девайс для этого, это поле добавили в более поздних версиях LINUX.

Возвращаясь к requst_irq: третий параметр irqflags.
Замечание: **Начиная с 2.6.19 версий ядра, все флаги были заменены, старое название было SA_* -> IRQF_*, это сигнализирует о актуальности литературы.**

Важнейшим флагом, который мы будем использовать в лабах - IRQF_SHARED.

Флаги:

- IRQF_SHARED - разрешает разделение линии прерывания, то есть совместное использование линии IRQ разными устройствами
- IRQF_PROBE_SHARED - устанавливается абонентами, вызывающими данные действия, если они предполагают возможность нестыковок при совместном использовании использовании линии IRQ
- IRQF_TIMER - флаг, маскирующий прерывание как прерывание от таймера
- IRQF_PERCPU - прерывание, закрепленное монопольно за конкретным процессором

Четвертый параметр dev_name, ASCII текст, представляющие связь устройства с прерыванием, например может иметь значение keyboard, эта строка использует название /proc/irq, а также /proc/interrupts.

Пятый параметр dev_id, используется прежде всего для разделения (share) линий прерываний. Когда обработчик выполняется, dev_id обеспечивает уникальные cookie-файлы (смысл в том), чтобы выполнить удаление только нужного обработчика прерывания соответствующей линии прерываний. Указатель dev_id имеет тип void, то есть он может указывать на что угодно, но обычно используется указатель на структуру, специфичную для устройства.

В случае успеха request_irq возвращает 0, ненулевая величина означает ошибку, и в этом случае обработчик прерывания не регистрируется. Обычная ошибка EBUSY значит, что данная линия пребывания уже используется или не был указан флаг SHARED.

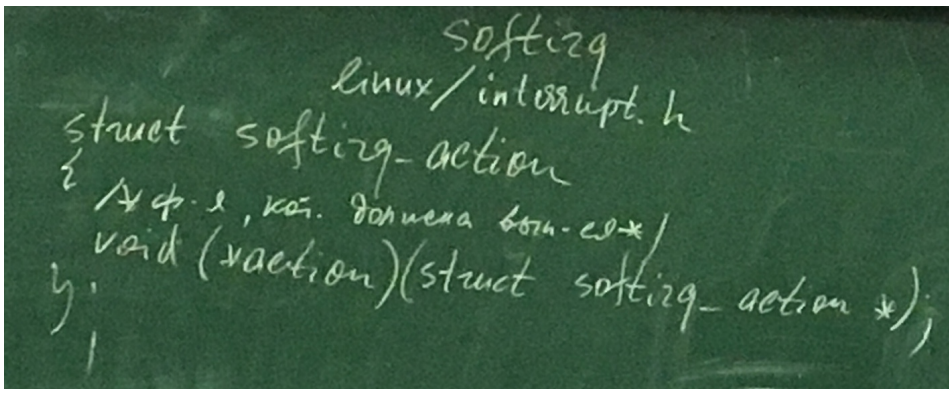
Процедура обработки быстрых прерываний очень короткая, поэтому прерывает текущую активность ненадолго. При выполнении таких прерываний, запрещены все прерывания на локальном процессоре, а также запрещены прерывания по данной линии прерываний. В старых версиях ядра такие обработчики прерываний регистрировались с флагом IRQF_INTERRUPT. В новых версиях ядра быстрым прерыванием является единственное - IRQF_TIMER.

Медленные прерывания делятся на две части: верхнюю и нижнюю половины. Это историческое название - top & bottom half. Фактически, top half это быстрое прерывание со всеми вытекающими, то есть верхняя половина должна заканчиваться как можно быстрее, так как она блокирует всю активность на локальном процессоре, то есть выполняется при защищенных прерываниях.

Нижняя половина это отложенное действие, которое как правило идет сразу после верхней, но имеет другой уровень приоритета и может быть прервана кем угодно, если называть вещи своими именами.

Однако, в отличие от реального обработчика быстрого прерывания, top half делит свои завершения, должна инициализировать последующее выполнение botton half, а именно должна поставить соответствующий обработчик в очередь на выполнение, причем в соответствующую очередь, в соответствии с типом отложенного действия. Заканчивается классическим IR_RET'ом.

В настоящее время обработчики нижних половин бывают трех видов: softirq, tasklet, workqueue. Мягкие или гибкие пребывания - tasklet, workqueue'ы. Softirq прерывания определяются статически во время компиляции ядра. В файле linux/interrupt.h определена структура struct softirq_action. В версии ядра 4.9 в этой структуре определена только функция которая должна выполняться.



В версии ядра 2.?? существовало поле данных

В файле kernel/softirq.c определен массив из 32 экземпляров этой структуры.

NR_SOFTIRQS - число задействованных номером - 32. Очевидно, что имеется возможность создать 32 softirq обработчика, в настоящее время определено десять.