Лекция 16.05. Аппаратные прерывания 16.05 (1)

IBM 360 – 3е поколение машин

• Для освобождения процессора от ожидания завершения ввода запрошенных процессами данных, эту ф-цию управления внеш. устройствами возложили на каналы (появилась канальная архитектура). Каналы стали управлять внеш. устр-вами.

• Появилась задача информирования пр-сора о завершении операций вв/вывода, и эту задачу стали решать аппарат. прерывания – прерывания от устр-в вв/вывода.

(Прерыв-е от сис. таймера – это аппарат. прер-ие.)

PIC – программируемый контр-р прерываний

Мэйнфреймы, такие как IBM 360, имели канальную архит-ру, персональные компьютеры были реализованы с шинной арх-турой и управление внеш. устр-вами было передано спец. устр-вам – контроллерам.

• В современных системах применяется реализация прерываний от внеш. устр-в в виде сообщений.

• Управление внеш. устр-вами выполняют спец. программы-драйвера (для каждого типа устройства пишется спец. драйвер, в состав которого входит 1 обработчик прерываний для поддержания 1 ф-ции внеш. (устр-ва)

Этот обраб-к прер-я передаётся в качестве пар-ра

Ф-ция для установки (важное) собственного обработчика прерываний:

↓ это "обёртка"

```
static inline int __must_check(2) request_irq(unsigned int irq(1),
                                              irq_handler_t handler,
                                              unsigned long flags(3),
                                              (5)const char *name, void *dev(6))
{ return request_threaded_irq(irq, handler, NULL, flags, name, dev);
}
typedef irqreturn_t (*irq_handler_t)(int, void *);
```

(1) – номер линии прерывания

(2) – это макрос ядра Linux (он всегда предваряет важные ф-ции. Этот макрос срабатывает во время компиляции, чтобы система могла проверить допустимость выполнения-

(действие)

• Когда возникает прерывание по указанной линии прерывания, вызывается обработчик прерывания.

В 64-разрядных системах не используется таблица векторов прерываний первого Кбайта (начиная с 0-го адреса). Вместо этого в этих системах ∃ аппарат-ая поддержка: спец. таблица interrupt descriptor table; 16 специально выделенных прерываний, которые особым образом обрабатываются (для ускорения их обработки)

• вектор прерывания — это смещение к дескриптору прерывания в таблице векторов прерываний; а в таблице векторов прерываний уже находится смещение к обработчику прерыв-я; Обраб-к прерыв-я находится в опред. сегменте кода. В результате формирование абсолютного адреса обраб-ка прер-я, используя эти аппар. схемы, система переходит по этому адресу и начинает выполнение обработчика прерыв-я.

! Когда возникает прерывание, система через соотв-ющее аппарат. обеспечение переходит на выполнение обраб-ка прер-я

• Прерывание от клавиатуры возникает при отжатии клавиши, и будет вызван соотв. обраб-к прерыв-я

∴ Линия прерываний определяет номер прер-ия.

Номера прерыв-й от сис. таймера и клавиатуры — 0 и 1.
PS/2

(3) — флаги (регламентируют действия, кот. код может выполнять)

То, что мы подключаем к USB устр-ву, работает через эту шину.

Линия прерывания уже занята „родным" обработчиком → для этого ∃ спец. флаг (чтобы свой „повесить" на эту линию)

Флаги:

```
#define IRQF_SHARED        0x00000080  // говорит о возмож-ти разделения одной линии прерываний несколькими обработчиками прерываний
— " — IRQF_PROBE_SHARED    0x00000100
— " — __IRQF_TIMER         0x00000200  // в системе ∃ 1 быстрое прерывание, и это прерыв-е от сис. таймера
#define IRQF_TIMER(__IRQF_TIMER|IRQF_NO_SUSPEND|IRQF_NO_THREAD)(4)
```

(над 0x00000080: идентификатор)

IRQF_PROBE_SHARED — для доп. контроля, проверки прерыв-я

(4) — уточнение флага (говорит о том, что обраб-чик от сис. таймера не м.б. распараллелен (не может на разных ядрах выполняться) — условно выделяется 1 "главный" процессор, обрабатыв-й прер-е от сис. таймера)

· Базовая архитектура наших систем — SMP.

(5) — параметр name — это строка символов, где можно „мое любимое словечко". (раньше было device-name)

(6) — этот параметр, позволяет указать имя нашего обраб-ка прерыв-я, чтобы в последствии иметь возможность освободить линию irq от нашего обработчика.

```
extern const void *free_irq(unsigned int, void *);
```

<u>дескриптор действия прерывания</u>

```
struct irqaction                    #include <asm/irq.h>
{
  irq_handler_t handler; //обработчик прер-ия
  void *dev_id; //идентификатор устройства
  void __percpu *percpu_dev_id; //идентификатор, привязанный к cpu
  struct irqaction *next; //объекты irq_action связаны в односвязный список
  irq_handler_t thread_fn;
  struct task_struct *thread; //система находит драйвер выполняет как поток
  ...
  unsigned int irq; //номер линии прерывания
  unsigned int flags;
  unsigned long thread_flags;
  ...
  const char *name;
  const proc_dir_entry *dir; //pointer to the proc/irq/NN/name entry
};
```

В соврем. системах аппарат. прерывания сигнализируются сообщением и это наз-ся <u>MSI</u> — message signal interrupt

В соврем. версиях ядра Linux ∃ 1 <u>быстрое прерывание</u> — от систем. таймера => все остальные прерывания <u>медленные</u>

Быстрые и медленные прерывания

Проблема: ∀ обработчик аппар. прер-е в системе выполняется на высочайшем уровне приоритета. В результате никакая другая работа в системе выполняться не может, пока не завершится выполнение обраб-ка прерыв-ие → это может влиять на отзывчивость системы.

Все медленные прерыв-е (все прерывые кроме сис. таймера) делятся на 2 части: · top half (верхняя половина) — это аппар. прер-ие
отложенные действие — · bottom half (нижняя половина)

· Чтобы оно завершалось как можно быстрее, в обраб-ке аппар. пр-ие оставляет ся минимально необходимый объём действий.

Минимал. объём действий обраб-ка прер-ие : сохранение полученных данных в буфере ядра. А уже остальные действие по завершению обработки данного прерыв-ие выполняет нижняя половина.

В совр системах ∃ 3 типа нижних половин: отложенные действие

1) softirqs (гибкие прерывания)
2) tasklets
3) work queue (works)

Посмотреть установленные в системе обраб-ки прер-й:

cat /proc/interrupts

1й столбец это № IRQ к-во прерываний cpu контроллер

0: IO_APIC_edge timer

1: IO_APIC_edge i8042 ← тут наш будет

идентифи-катор контр-ра

выполнять мин. объём действий по обслуживанию внеш. устр-ва

В Лр: наш обраб-к прерывания будет инициализ-ть отложенное действие и установленный в системе обраб-к прер-е от клавы

Синхронизация выполнение тасклета (работ из очереди работ)

16.05 (5)

Softirq <linux/interrupt.h>

(в переводе „гибкие прер-ия")

```
struct softirq_action
{ /* ф-ция, кот. должна выполняться */
  void (*action)(struct softirq_action *);
};
```

(softirq — отложенное дей-ие, которые определяются в системе статические, т.е. во время компиляции ядра)

• Кроме этой стр-ры в ядре также определён массив из 32 экземпляров данной стр-ры:

static struct softirq_action softirq_vec [NR_SOFTIRQS];
32

↓

Это значит, что в ядре имеется возможность создания 32 обработчиков softirq. (В настоящее время в ядре определено 10 обраб-ков)

• Именно обработчики softirq работают с сетевой подсистемой

Виды softirq:

Индекс	Приоритет	Значение
HI_SOFTIRQ	0	Высокоприоритетн. softirq
TIMER_SOFTIRQ	1	таймеры (программируемые таймеры, для контроля по времени выполнения действий в системе)
NET_TX_SOFTIRQ	2	отправка сетевых пакетов
NET_RX_SOFTIRQ	3	приём сетевых пакетов
BLOCK_SOFTIRQ	4	блочные устройства
BLOCK_IOPOLL_SOFTIRQ	5	
TASKLET_SOFTIRQ	6	
SHED_SOFTIRQ	7	
HR TIMER_SOFTIRQ	8	не использ.
RCU_SOFTIRQ	9	

• Часть пакетов, поступающих на компьютер, — транзитные

• По сети ∀ информ-ция передаётся в виде пакетов