Тема №7. Драйвер простого устройства

Обработка прерываний

Существует два типа взаимодействий между CPU и остальной аппаратной частью компьютера. Первый -- передача команд аппаратным средствам, второй -- прием ответов от аппаратуры. Второй тип взаимодействия -- прерывания, является наиболее тяжелым в обработке, потому что прерывания возникают тогда, когда это удобно устройству, а не CPU. Аппаратные устройства обычно имеют весьма ограниченный объем ОЗУ, и если не считать поставляемую ими информацию немедленно, то она может потеряться.

В Linux аппаратные прерывания называются **IRQ** (сокращенно от Interrupt ReQuests -- Запросы на Прерывание). [14] Имеется два типа IRQ: "короткие" и "длинные". "Короткие" IRQ занимают очень короткий период времени, в течение которого работа операционной системы будет заблокирована, а так же будет невозможна обработка других прерываний. "Длинные" IRQ могут занять довольно продолжительное время, в течение которого могут обрабатываться и другие прерывания (но не прерывания из того же самого устройства). Поэтому, иногда бывает благоразумным разбить выполнение работы на исполняемую внутри обработчика прерываний (т.е. подтверждение прерывания, изменение состояния и пр.) и работу, которая может быть отложена на некоторое время (например постобработка данных, активизация процессов, ожидающих эти данные и т.п.). Если это возможно, лучше объявлять обработчики прерывания "длинными".

Когда CPU получает прерывание, он останавливает любые процессы (если это не более приоритетное прерывание, тогда обработка пришедшего прерывания произойдет только тогда, когда более приоритетное будет завершено), сохраняет некоторые параметры в стеке и вызывает обработчик прерывания. Это означает, что не все действия допустимы внутри обработчика прерывания, потому что система находится в неизвестном состоянии. Решение проблемы: обработчик прерывания определяет -- что должно быть сделано немедленно (обычно что-то прочитать из устройства или что-то послать ему), а затем запланировать обработку поступившей информации на более позднее время (это называется "bottom halves" -- "нижние половины") и вернуть управление. Ядро гарантирует вызов "нижней половины" так быстро, насколько это возможно. Когда это произойдет, то наш обработчик -- "нижняя половина", уже не будет стеснен какими-то рамками и ему будет доступно все то, что доступно обычным модулям ядра.

Устанавливается обработчик прерывания вызовом `request_irq`. Ей передаются номер IRQ, имя функции-обработчика, флаги, имя для `/proc/interrupts` и дополнительный параметр для обработчика прерываний. Флаги могут включать `SA_SHIRQ`, чтобы указать, что прерывание может обслуживаться несколькими обработчиками (обычно, по той простой причине, что на одном IRQ может "сидеть" несколько устройств) и `SA_INTERRUPT`, чтобы указать, что это "короткое" прерывание. Эта функция установит обработчик только в том случае, если на заданном IRQ еще нет обработчика прерывания, или если существующий обработчик зарегистрировал совместную обработку прерывания флагом `SA_SHIRQ`.

Во время обработки прерывания, из функции-обработчика прерывания, мы можем получить данные от устройства и затем, с помощью `queue_task_irq`, `tq_immediate` и `mark_bh(BH_IMMEDIATE)`, запланировать "нижнюю половину". В

ранних версиях Linux имелся массив только из 32 "нижних половин", теперь же, одна из них (а именно BH_IMMEDIATE) используется для обслуживания целого списка "нижних половин" драйверов. Вызов `mark_bh(BH_IMMEDIATE)` как раз и вставляет "нижнюю половину" драйвера в этот список, планируя таким образом ее исполнение.

Клавиатура на архитектуре Intel

Для демонстрации работоспособности кода сначала придется отключить стандартный обработчик прерываний от клавиатуры, а так как этот символ объявлен как `static` (в файле `drivers/char/keyboard.c`), то нет никакого способа восстановить обработчик. Поэтому, прежде чем вы дадите команду **insmod**, перейдите в другую консоль и дайте команду **sleep 120; reboot**, если ваша файловая система представляет для вас хоть какую-нибудь ценность.

Этот пример захватывает обработку IRQ 1 -- прерывание от клавиатуры на архитектуре Intel. При получении прерывания обработчик читает состояние клавиатуры (inb(0x64)) и скан-код нажатой клавиши. Затем, как только ядро сочтет возможным, оно вызывает `got_char` (она играет роль "нижней половины"), которая выводит, через `printk`, код клавиши (младшие семь бит скан-кода) и признак "нажата/отпущена" (8-й бит скан-кода -- 0 или 1 соответственно).

```
/*
 *  intrpt.c - Обработчик прерываний.
 *
 *  Copyright (C) 2001 by Peter Jay Salzman
 */


/*
 * Standard in kernel modules
 */
#include <linux/kernel.h>    /* Все-таки мы работаем с ядром! */
#include <linux/module.h>    /* Необходимо для любого модуля */
#include <linux/workqueue.h> /* очереди задач */
#include <linux/sched.h>     /* Взаимодействие с планировщиком
*/
#include <linux/interrupt.h> /* определение irqreturn_t */
#include <asm/io.h>

#define MY_WORK_QUEUE_NAME "WQsched.c"

static struct workqueue_struct *my_workqueue;

/*
 * Эта функция вызывается ядром, поэтому в ней будут безопасны
все действия
 * которые допустимы в модулях ядра.
 */
static void got_char(void *scancode)
{
  printk("Scan Code %x %s.\n",
```

```
        (int)*((char *)scancode) & 0x7F,
        *((char *)scancode) & 0x80 ? "Released" : "Pressed");
}

/*
 * Обработчик прерываний от клавиатуры. Он считывает информацию
с клавиатуры
 * и передает ее менее критичной по времени исполнения части,
 * которая будет запущена сразу же, как только ядро сочтет это
возможным.
 */
irqreturn_t irq_handler(int irq, void *dev_id, struct pt_regs
*regs)
{
  /*
   * Эти переменные объявлены статическими, чтобы имелась
возможность
   * доступа к ним (посредством указателей) из "нижней
половины".
   */
  static int initialised = 0;
  static unsigned char scancode;
  static struct work_struct task;
  unsigned char status;

  /*
   * Прочитать состояние клавиатуры
   */
  status = inb(0x64);
  scancode = inb(0x60);

  if (initialised == 0) {
    INIT_WORK(&task, got_char, &scancode);
    initialised = 1;
  } else {
    PREPARE_WORK(&task, got_char, &scancode);
  }

  queue_work(my_workqueue, &task);

  return IRQ_HANDLED;
}

/*
 * Инициализация модуля - регистрация обработчика прерывания
 */
int init_module()
{
  my_workqueue = create_workqueue(MY_WORK_QUEUE_NAME);

  /*
   * Поскольку стандартный обработчик прерываний от клавиатуры
```

```
не может
   * сосуществовать с таким как наш, то придется запретить его
   * (освободить IRQ) прежде, чем что либо сделать.
   * Но поскольку мы не знаем где он находится в ядре, то мы
лишены
   * возможности переустановить его - поэтому компьютер придется
перезагрузить
   * после опробования этого примера.
   */
        free_irq(1, NULL);

  /*
   * Подставить свой обработчик (irq_handler) на IRQ 1.
   * SA_SHIRQ означает, что мы допускаем возможность совместного
   * обслуживания этого IRQ другими обработчиками.
   */
  return request_irq(1,     /* Номер IRQ */
         irq_handler,       /* наш обработчик */
         SA_SHIRQ,
         "test_keyboard_irq_handler",
         (void *)(irq_handler));
}

/*
 * Завершение работы
 */
void cleanup_module()
{
  /*
   * Эта функция добавлена лишь для полноты изложения.
   * Она вообще бессмысленна, поскольку я не вижу способа
   * восстановить стандартный обработчик прерываний от
клавиатуры
   * поэтому необходимо выполнить перезагрузку системы.
   */
  free_irq(1, NULL);
}

/*
 * некоторые функции, относящиеся к work_queue
 * доступны только если модуль лицензирован под GPL
 */
MODULE_LICENSE("GPL");
```

Задание:

1. Ознакомиться с представленным материалом.
2. Реализовать драйвер в виде модуля ядра.
3. Подключить драйвер, как модуль на уровне ядра.
4. Продемонстрировать корректность работы драйвера.