ПО сетевых устройств

Трещановский Павел Александрович, к.т.н.

31.05.19

Ядро Linux

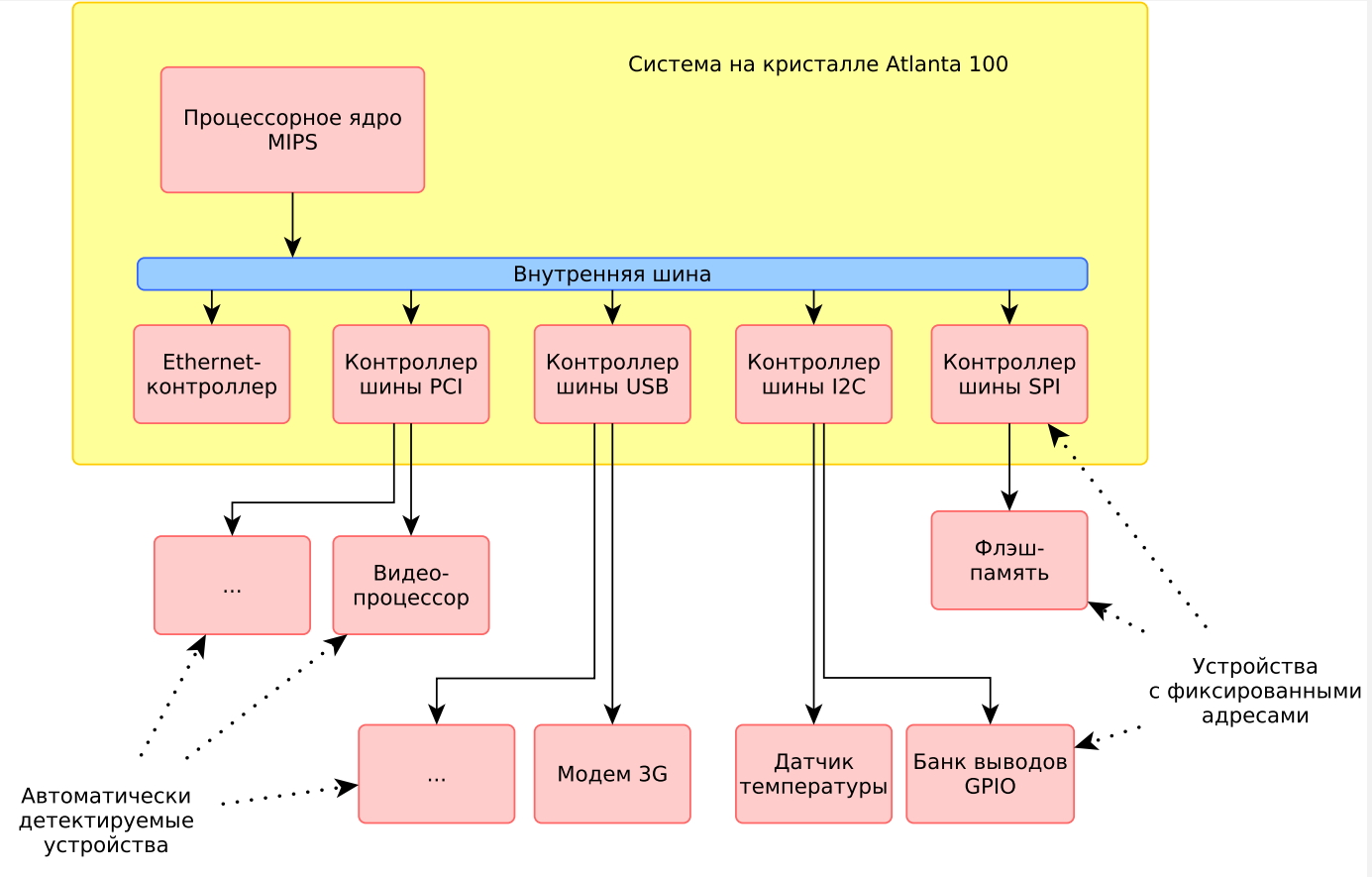
- Ядро имеет прямой доступ к аппаратной периферии.
- Ядро реагирует на внешние события (прерывания).
- Загрузчик записывает ядро в ОЗУ во время запуска системы. Далее ядро всегда целиком находится в ОЗУ.
- Основная работа ядра выполняется по запросам от приложений (системным вызовам). Похоже на библиотеку.
- Вспомогательная работа выполняется в процессах ядра (в выводе `ps` имена в квадратных скобках).
- Код ядра полностью автономен. Не используется даже стандартная библиотека C.
- После ошибки ядро не восстанавливается. Требуется перезагрузка.

Модули ядра

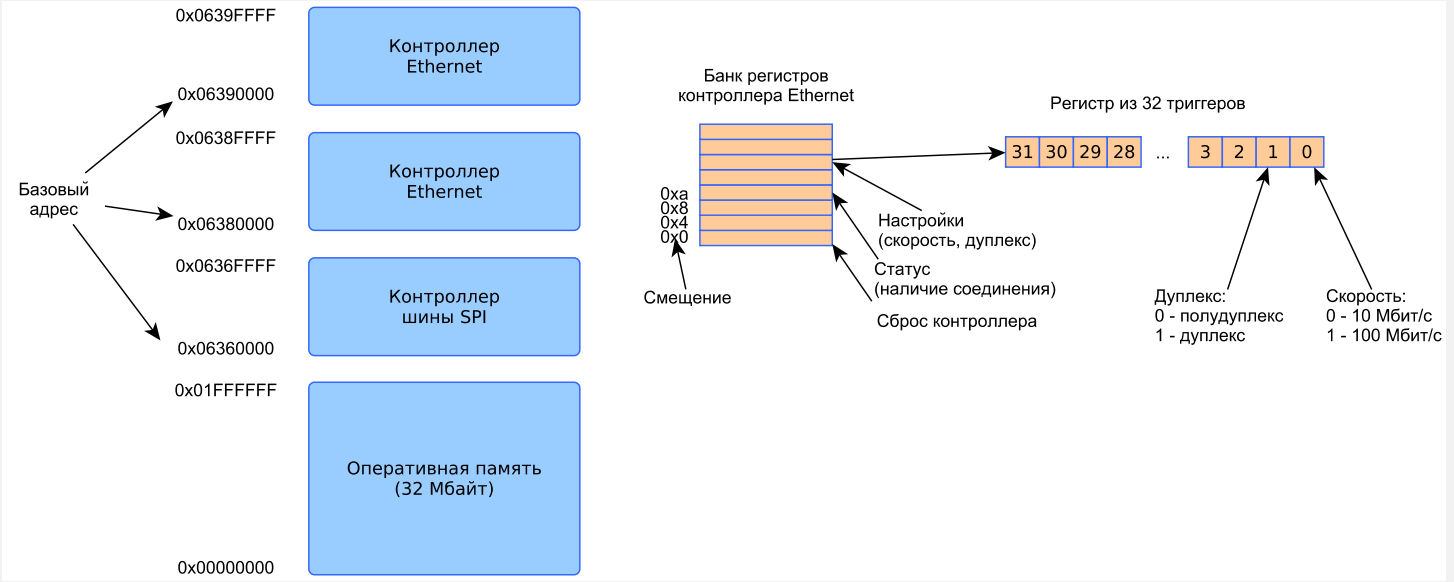
Загрузка и выгрузка из ядра - командами `insmod` и `rmmod`. Просмотр - `lsmod`.

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/slab.h>

static char *buf;
static int example_init(void)
{
       printk("Hello world\n");
       buf = kzalloc(100, GFP_KERNEL);
       if (!buf)
              return -ENOMEM;
       return 0;
}
static void example_exit(void)
{
       free(buf);
}

module_init(example_init);
module_exit(example_exit);
```

Аппаратные шины



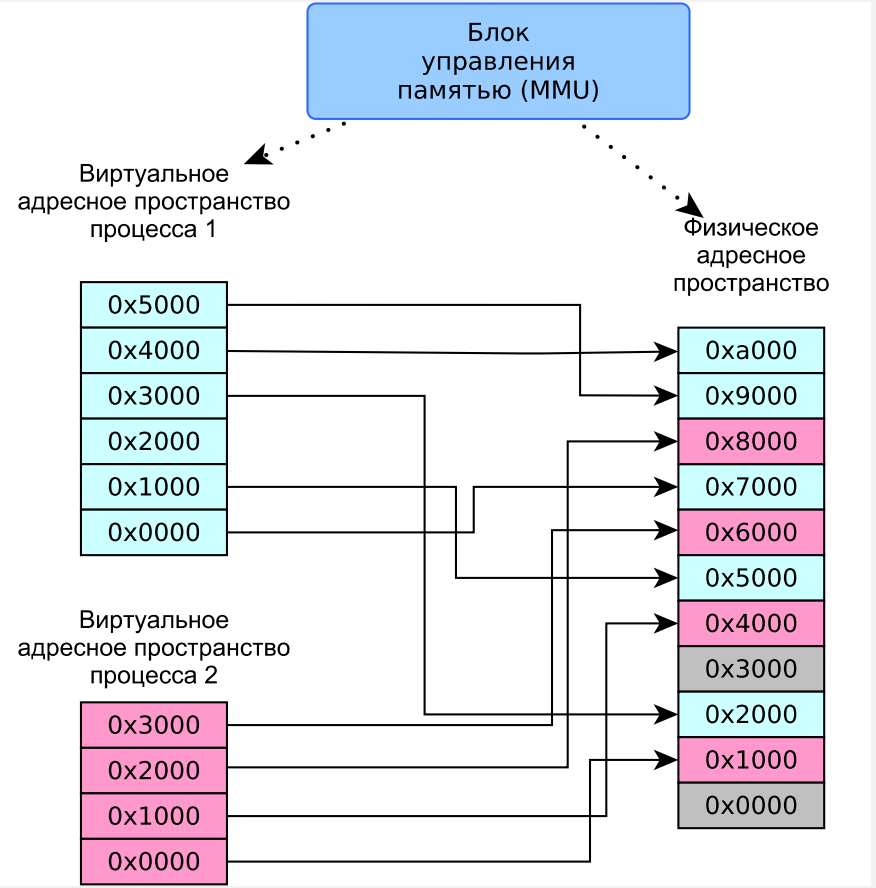
Структура физического адресного пространства



Обращение к регистрам в C

```
#define DEVICE_BASE_ADDR    0x06380000
#define DEVICE_CONFIG_REG   0x0
#define DEVICE_STATUS_REG   0x1
#define DEVICE_INTR_REG     0x2

void read_register(void)
{
       void *base_addr = (void *)DEVICE_BASE_ADDR;
       unsigned int config, status;

       config = *(unsigned int *)(base_addr + DEVICE_CONFIG_REG);
       status = *(unsigned int *)(base_addr + DEVICE_STATUS_REG);
}
```

Структура физического адресного пространства



Регистрация и связывание драйвера

```
#include <linux/platform_device.h>
#include <linux/of_device.h>

static const struct of_device_id a100_example_of_match[] = {
	{ .compatible = "ctlm,a100-example", }, {}
};
/* platform - название внутренней процессорной шины. */
static struct platform_driver a100_example_drv = {
	.driver = {
		.owner = THIS_MODULE,
		.name = "a100_example",
		.of_match_table = a100_example_of_match, /* Признак для связывания */
	},
};
static int a100_example_init(void)
{
	return platform_driver_register(&a100_example_drv);
}
static void a100_example_exit(void)
{
	platform_driver_unregister(&a100_example_drv);
}
```

Дерево устройств (Device Tree)

- Структура данных, описывающая недетектируемые устройства (SPI, I2C, устройства на внутренней шине) и отношения между ними.
- Хранится в ПЗУ устройства. Во время запуска системы загрузчик передает дерево устройств ядру.
- of_ - OpenFirmware (стандарт, в котором первоначальном был описан формат Device Tree).
- Пример:

```
soc {
        /* ... */
        a100-example {
                compatible = "ctlm,a100-example";
                reg = <0x063C0000 0x010000>; /* Базовый адрес и размер
                                                банка регистров. */
                interrupts = <4>, <5>;
        };
};
```

Инициализация устройства

```
#include <linux/io.h>

static int a100_example_probe(struct platform_device *pdev)
{
	struct resource *res;
	void __iomem *base;

	/* Получение базового адреса банка регистров. */
	res = platform_get_resource(pdev, IORESOURCE_MEM, 0);

	/* Отображение в виртуальное адресное пространство. */
	base = ioremap(res->start, resource_size(res));
	return 0;
}
static int a100_example_remove(struct platform_device *pdev) {/* ... */}

static struct platform_driver a100_example_drv = {
	.probe = a100_example_probe,
	.remove = a100_example_remove,
};
```

Привязка объектов к `struct platform_device`

```
static int a100_example_probe(struct platform_device *pdev)
{
	struct a100_example_data *data;
	struct resource *res;
	void __iomem *base;

	data = kzalloc(sizeof(struct a100_example_data), GFP_KERNEL);
	/* ... */
	data->base = ioremap(res->start, resource_size(res));

	platform_set_drvdata(pdev, data);
	return 0;
}
static int a100_example_remove(struct platform_device *pdev)
{
	struct a100_example_data *data = platform_get_drvdata(pdev);

	iounmap(data->base);
	free(data);
	return 1;
}
```

Чтение и запись регистров

- Чтение регистра с базовым адресом `base` и смещением 5:

```
#include <linux/io.h>

void __iomem *base;
u32 reg;
reg = readl(base + 5);
```

- Запись значения `val` в регистр с базовым адресом `base` и смещением 5:

```
u32 val;
writel(val, base + 5);
```

- Установка бита 6 и снятие бита 7:

```
u32 reg;
reg = readl(base + 5);
reg |= (1 << 6);
reg &= ~(1 << 7);
writel(reg, base + 5);
```

Информация об устройстве в sysfs

- Список драйверов для шины platform:

```
# ls /sys/bus/platform/drivers
```

- Список устройств на шины platform:

```
# ls /sys/bus/platform/devices
```

- Каждому устройству соответствует каталог в sysfs.
- Файлы в каталоге - атрибуты.
- Все файлы виртуальные. Данные не хранятся на жестком диске.

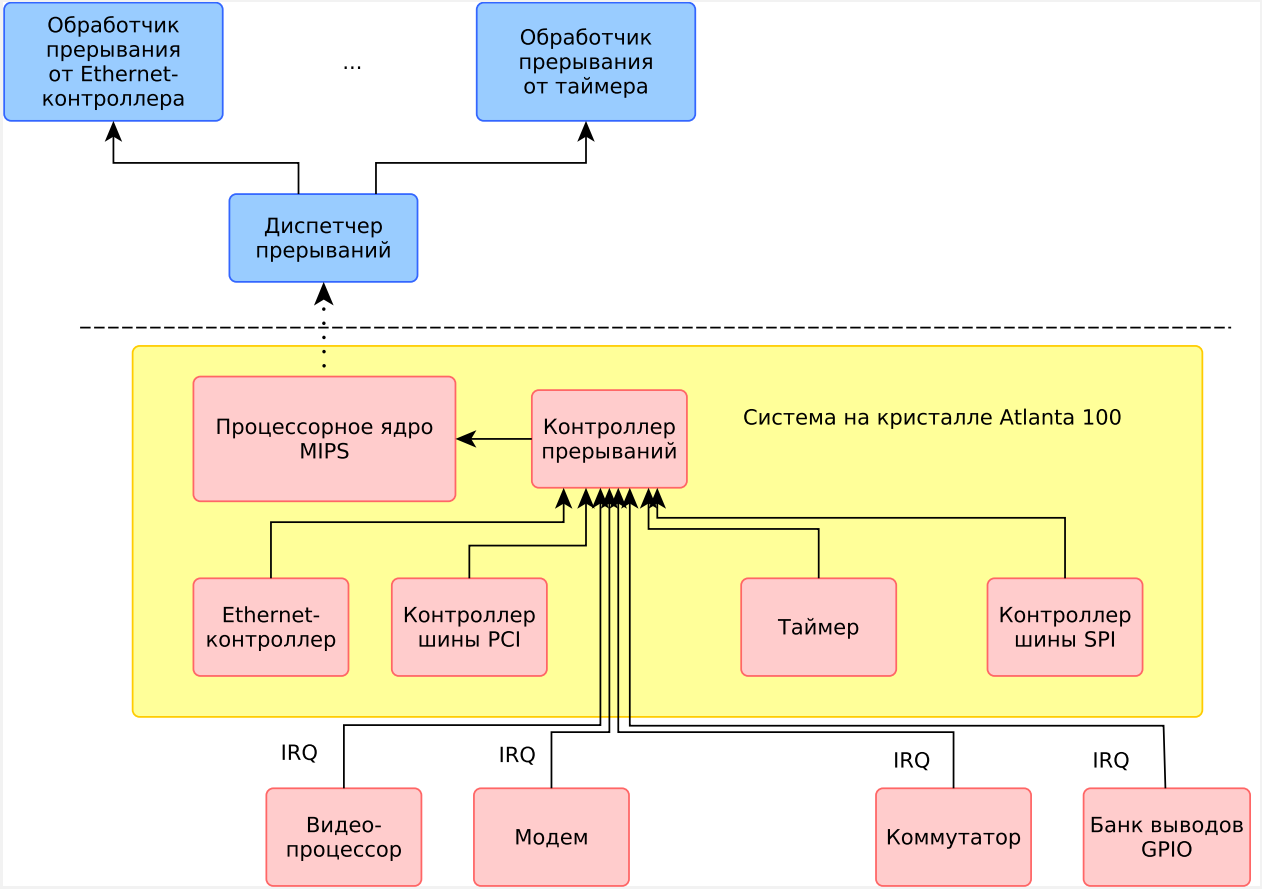
Определение новых атрибутов

```
ssize_t example_show(struct device *dev, struct device_attribute *attr, char *buf)
{
	struct a100_example_data *data = platform_get_drvdata(to_platform_device(dev));

	/* То, что записывается в buf, передается приложению через системный
	вызов read. */
	return sprintf(buf, "%s", "Hello world");
}

ssize_t example_store(struct device *dev, struct device_attribute *attr,
		      const char *buf, size_t len)
{
	struct a100_example_data *data = platform_get_drvdata(to_platform_device(dev));
	char *endp;

	/* То, что приложение передает в системный вызов write, оказывается в buf. */
	data->value = kstrtoul(buf, &endp, 0);

	return len;
}

static DEVICE_ATTR_RW(example); /* Имя атрибута совпадает с префиксом имен функций. */
```

Регистрация атрибутов

```
static DEVICE_ATTR_RW(example0);
static DEVICE_ATTR_RO(example1);

static struct device_attribute *example_attrs[] = {
       &dev_attr_example0,
       &dev_attr_example1,
       NULL,
};
static struct attribute_group example_group = {
       .name = "example group",
       .attrs = example_attrs,
}

static int a100_example_probe(struct platform_device *pdev)
{
       /* ... */

       sysfs_create_group(&pdev->dev.kobj, example_group);
}
```

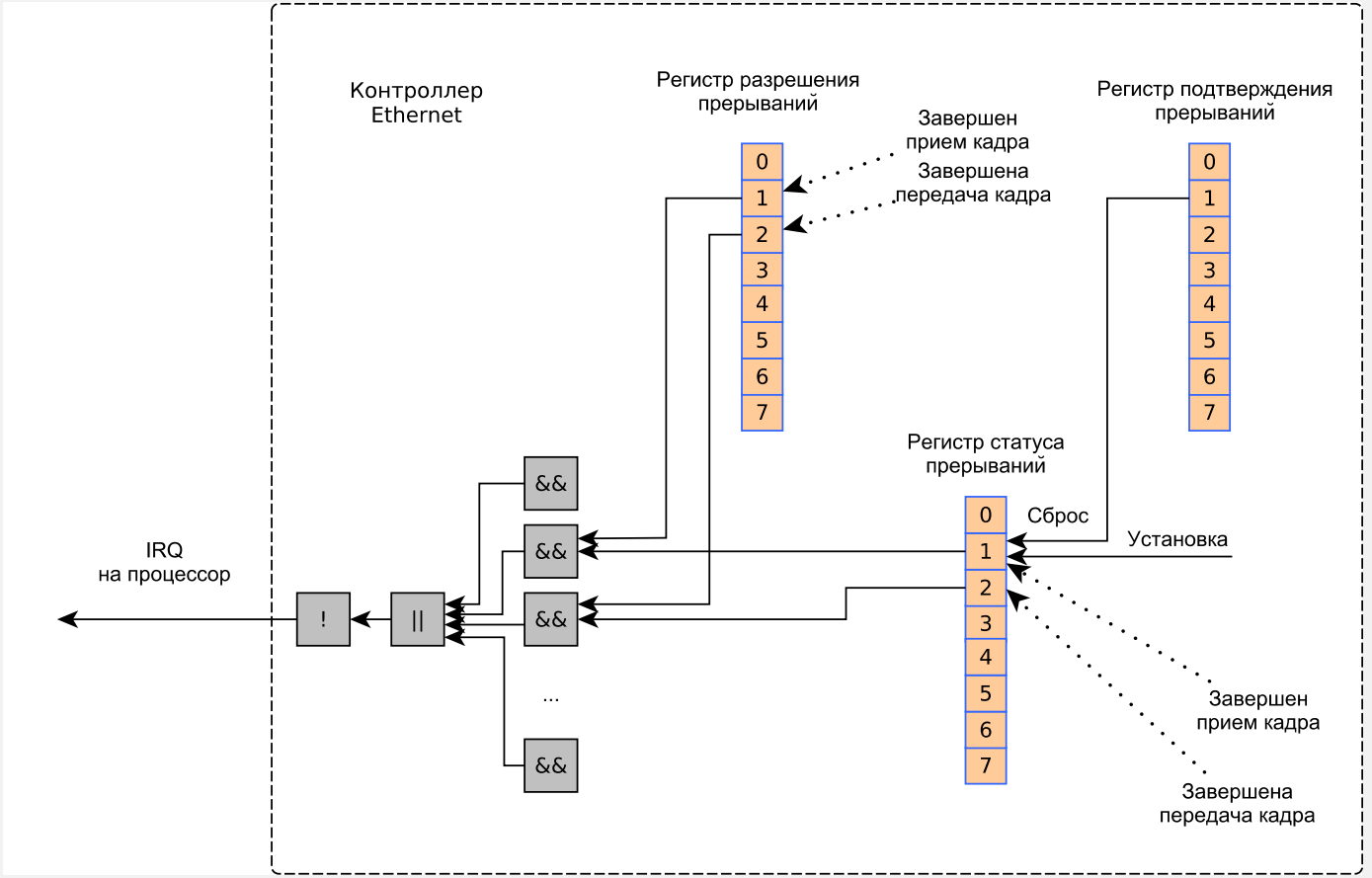
Обработка событий



Регистрация обработчика прерывания

```
#include <linux/interrupt.h>

static irqreturn_t a100_example_isr(int irq, void *id)
{
       struct a100_example_data *data = id;
       return IRQ_HANDLED;
}
static int a100_example_probe(struct platform_device *pdev)
{
       struct a100_example_data *data;
       int irq;

       /* Получение внутреннего номера для 0-го прерывания. */
       irq = platform_get_irq(pdev, 0);

       /* Регистрация обработчика для прерывания irq. Обработчику при вызове
          будет передан указатель data. */
       request_irq(irq, a100_example_isr, 0, "a100-example-isr", data);
       return 0;
}
```

Мультиплексирование прерываний



Аппаратный таймер