Laurentiu Mihalcea

Iuliana Prodan

Daniel Baluta

| Public | NXP, and the NXP logo are trademarks of NXP B.V. All other product or service names are the property of their respective owners. © 2024 NXP B.V.

00

Day 3: Interacting with hardware: buttons, gpio and leds

Secondary header

Kernel executions contexts

- Process context
  - Originates from a system call or a kernel thread
  - Can sleep
  - Has full access to user memory space
- Interrupt context
  - Triggered by hardware
  - Cannot sleep or block
  - Must be fast and minimal

Preemptive kernel vs userspace

- User space preemption
- Kernel space preemption

Interrupts handling

- Why do we need interrupts?
- How it works?
  - Use request_irq to associate an interrupt with a handler
  - Watch for restriction on implementing the handler
  - free_irq() when done

Synchronization between contexts

- Interrupt context can preempt process context!
- Challenges?
  - we cannot sleep in interrupt contexts!
  - spinlock
  - mutex

GPIO and interrupts

```
button_gpio = devm_gpiod_get(&pdev->dev, NULL, GPIOD_IN);
if (IS_ERR(button_gpio)) {
        dev_err(&pdev->dev, "Failed to get GPIO\n");
        return PTR_ERR(button_gpio);
}

irq = gpiod_to_irq(button_gpio);

ret = devm_request_irq(&pdev->dev, irq, button_irq_handler, IRQF_TRIGGER_FALLING,
                       "button_irq", NULL);
```

```
button {
        compatible = "lkss,gpio-button";
        gpios = <&gpio2 3 GPIO_ACTIVE_HIGH>; // Example: GPIO2.IO[3]
        status = "disabled";
};
```

Communication with hardware

- Hardware registers
  - Control
  - Status
  - Data
- Devices usually connected to a bus
  - SPI, I2C, UART, CAN
- Registers can be accessed via
  - Ports
  - Memory mapped IO