Linux Device Drivers

IO Ports

- IO registers can be memory mapped or IO mapped based on arch.
    - x86 -- IO mapped -- separate address space for IO registers.
        - Serial port 1: address = 0x3F8 and irq number = 4
        - Serial port 2: address = 0x2F8 and irq number = 3
        - Parallel port: address = 0x378
        - Keyboard controller: address = 0x60
            - http://embeddedguruji.blogspot.com/2019/01/linux-device-driver-to-disableenable.html
    - ARM -- Memory mapped -- combined address space for IO registers and memory.
        - AM335x: For each GPIO there are 4KB address ranges (memory mapped). This 4KB have sevaral addresses at definite offset for controlling GPIO operations (for particular bits).
- HAL macros to write/read from IO port registers --> inb(), outb(), inl(), outl(), ...
- To access IO ports from the user-space ioperm() syscall can be used.
- HAL functions to write/read from IO memory registers --> ioread8(), iowrite8(), ioread32(), iowrite32(), ...
- Notes:
    - Every module e.g. GPIO module has its own memory map i.e. physical address specified in the processor's technical reference manual.
    - First you need to check if the memory region is being used or not using check_mem_region(). This step is deprecated in newer kernel.
    - If it is free, request access to this memory region using request_mem_region(), then map the GPIO module using ioremap() or ioremap_nocache() (map bus memory into CPU space), which returns a void*.
    - The returned address is not guaranteed to be usable directly as a virtual address; it is only usable by ioread*|iowrite*|read*|write*, etc. functions.
    - Use ioread8|16|32/iowrite8|16|32 functions to read or write from/to i/o ports.
    - Finally you need to iounmap() to unmap the memory and then you need to release memory region using release_mem_region().
- Reference: LDD -- Communicating with Hardware

Writing Hardware Device Drivers

- Method 1:

  - Acquire IO port addresses using request_region() -- in module initialization
    - terminal> sudo cat /proc/ioports
  - Initialize the device using inb()/outb() -- in open()
  - Read/write data on IO device using inb()/outb() -- in read()/write()
  - De-initialize the device using inb()/outb() -- in release()
  - Release IO port addresses using release_region() -- in module exit
  - Note: If existing device driver is already occupying the port, then your driver request_region() will fail. In this case, existing driver should be blacklisted (/etc/modprobe.d/blacklist.conf) and then your driver can control the device.
- Method 2:
  - Implement your device driver depending on existing device driver/kernel sub-system i.e. your device operations invokes the functions exported by existing device driver/kernel sub-system.
  - Note: Here you don't have direct access to IO ports (no request_region()) and you should not blacklist existing device driver.

Linux GPIO sub-system

- Linux GPIO sub-system internally access hardware ports/memory for the given board and controls GPIO ports (input/output).
- The same GPIO APIs can be used for any board or any arch e.g. BBB, RPi, etc.
- Refer slides.

Interrupt handling

- Device interrupts are handled by device driver.
  - step 1: Implement ISR.
  - step 2: Register ISR -- request_irq().
  - step 3: Unregister ISR -- free_irq().
- ISR is executed in Interrupt context -- must not sleep.
- Heavy processing and blocking task should be done in Bottom halfs.
  - Soft IRQ
  - Tasklets
  - Work queue