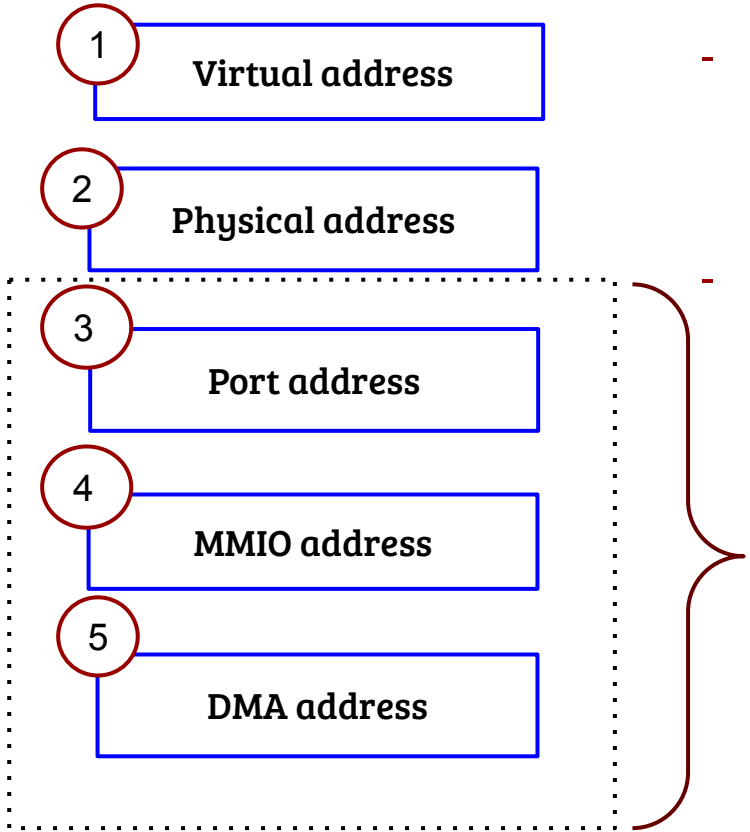
CS614: Linux Kernel Programming

PCI Devices and Drivers

Debadatta Mishra, CSE, IIT Kanpur

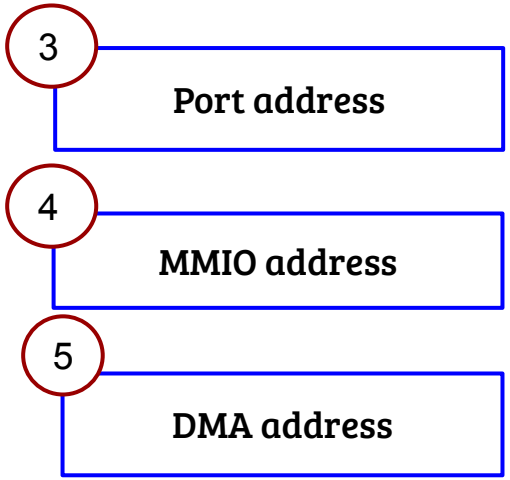
Recap: Address types in kernel



- What kind of addressing provides more flexibility to the OS, considering address as a resource?
- Can a device be operated only with DMA addressing?

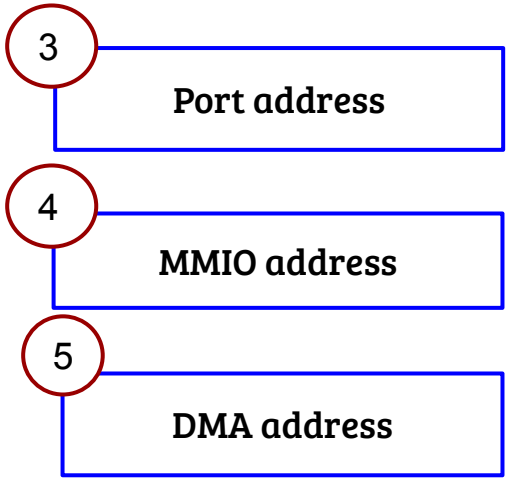
Used exclusively to operate and manage I/O devices

Flexibility in I/O Addressing



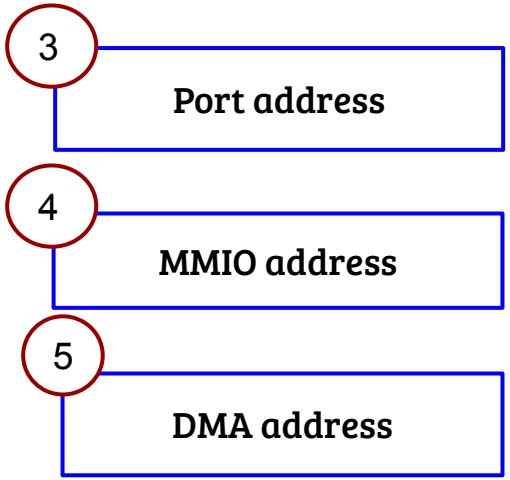
- What kind of addressing provides more flexibility to the OS, considering address as a resource?

Flexibility in I/O Addressing



- What kind of addressing provides more flexibility to the OS, considering address as a resource?
  DMA is allows maximum flexibility to the OS
- Can a device be initialized and operated only with DMA addressing?

Flexibility in I/O Addressing



- What kind of addressing provides more flexibility to the OS, considering address as a resource? DMA is allows maximum flexibility to the OS
- Can a device be initialized and operated only with DMA addressing? No, because the DMA setup requires MMIO/PIO access
- How can the OS manage PIO and MMIO addresses in a flexible manner?

PCI Subsystem

- PCI can be viewed as tree-like organization of I/O devices
- Each device mapped to PCI bus can be examined based on the IDs (device, vendor etc.)

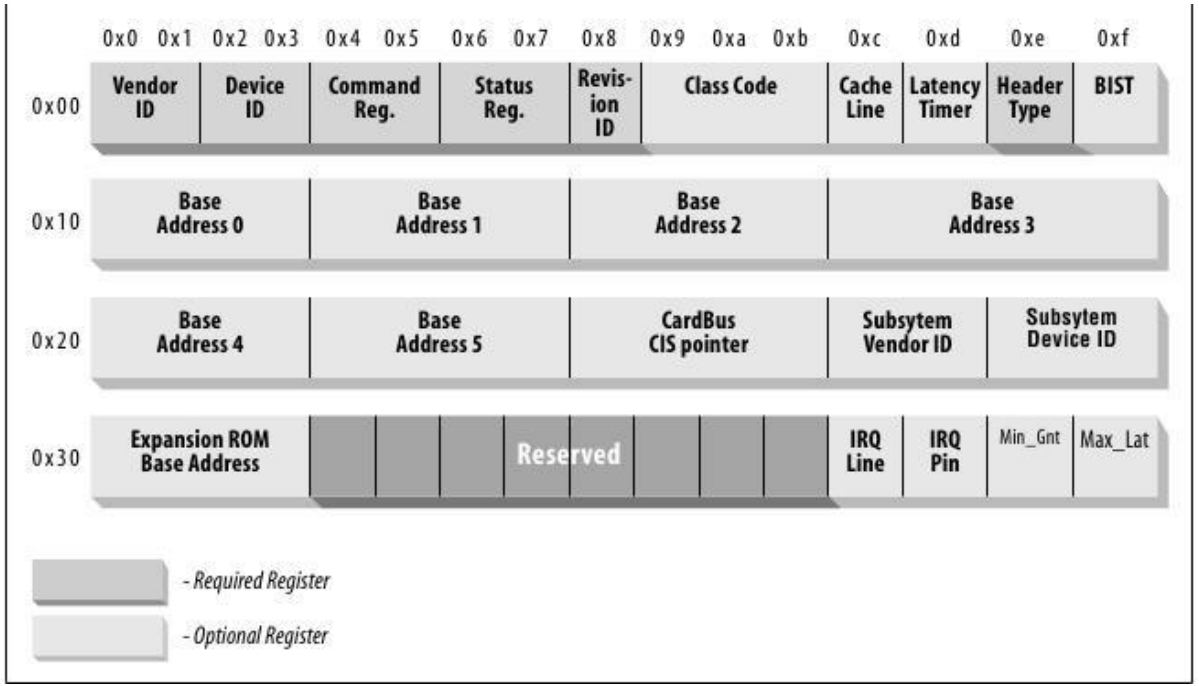


Image source: Linux Device Drivers, Ch12

PCI Layout

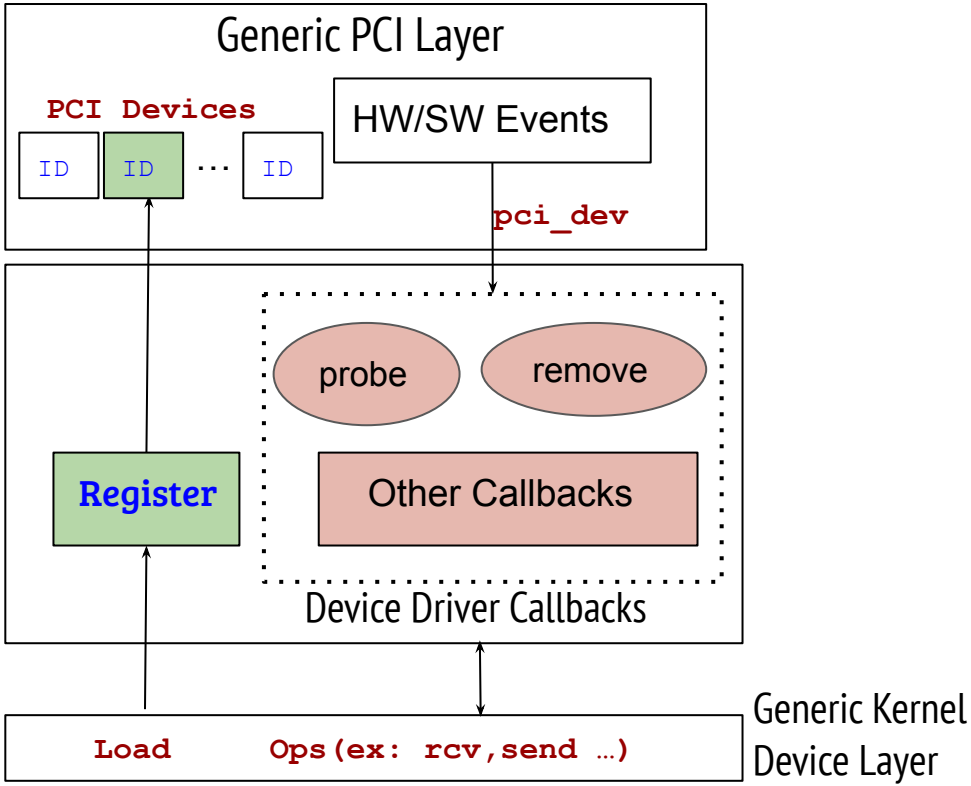
- PCI can be viewed as tree-like organization of I/O devices
- Each device mapped to PCI bus can be examined based on the IDs (device, vendor etc.)
- Devices can be found by querying the PCI controller and scanning the mapped devices though the nested laying of
  - Domain
  - Bus
  - Device
  - Function

PCI Layout

- PCI can be viewed as tree-like organization of I/O devices
- Each device mapped to PCI bus can be examined based on the IDs (device, vendor etc.)
- Devices can be found by querying the PCI controller and scanning the mapped devices though the nested laying of
  - Domain
  - Bus
  - Device
  - Function
- Linux kernel pre-creates this list and invokes the probe method of the matching driver when a driver is registered
- The “lspci” user space utility (and the /sys/bus/... interface) can be used to examine

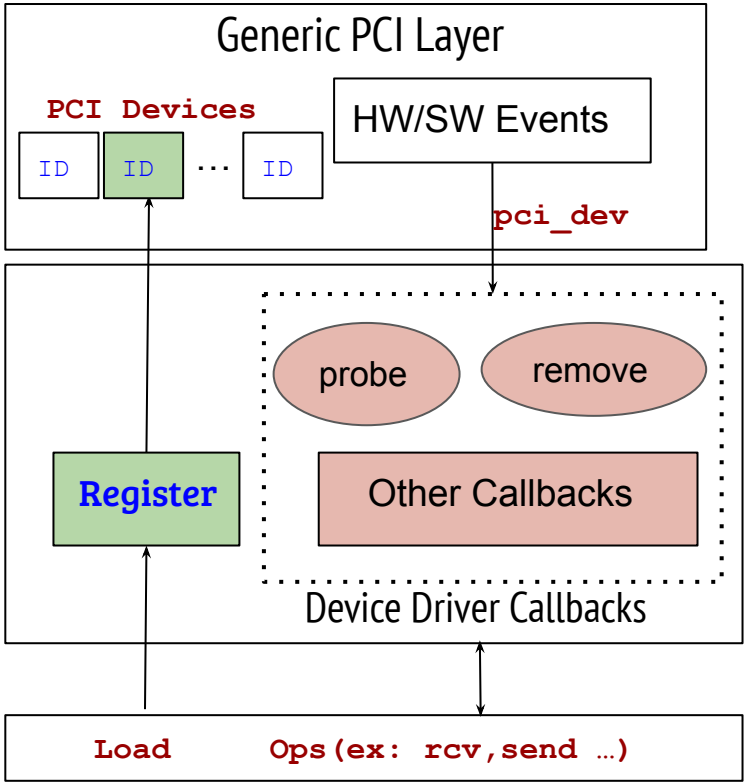
Linux PCI device driver

- A PCI device driver must register itself using an object of type “struct pci_driver”



Linux PCI device driver

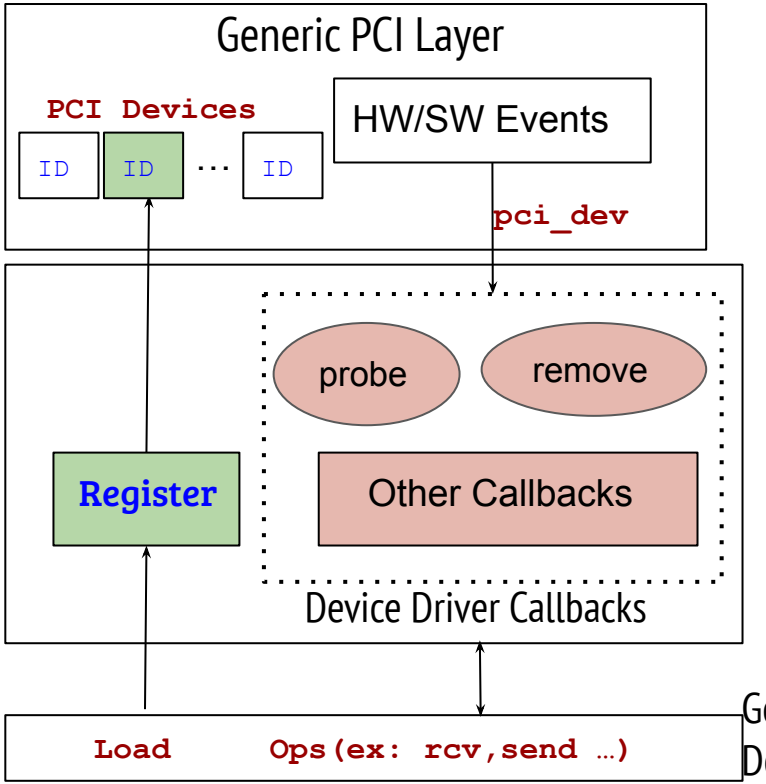
- A PCI device driver must register itself using an object of type “struct pci_driver”



- While registering a driver for a PCI device, an ID table containing a list of ID entries (*vendor, device, subvendor, subdevice*) are passed to the PCI layer to match a device for this driver
- A probe method (part of pci_driver structure) is registered as a call back

Linux PCI device driver

- A PCI device driver must register itself using an object of type “struct pci_driver”



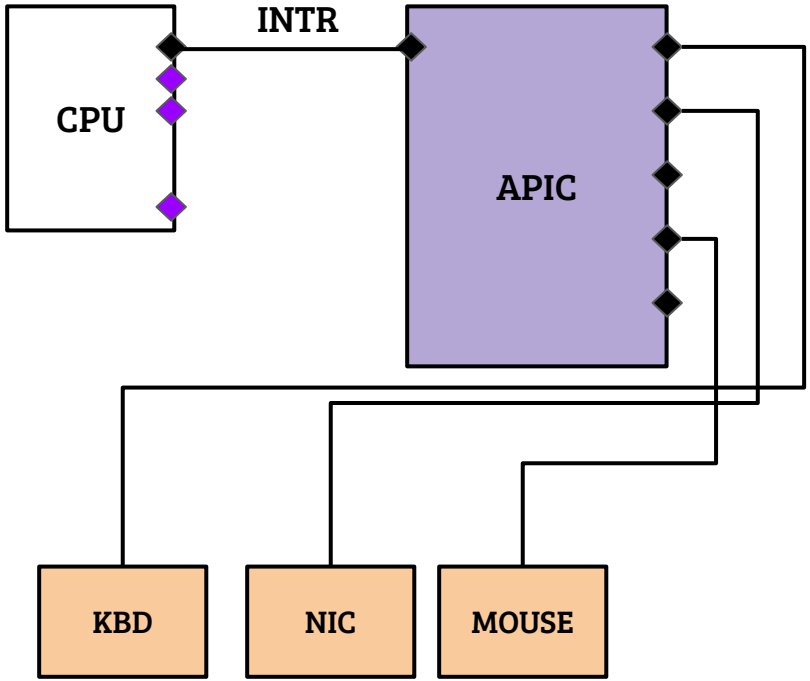
- While registering a driver for a PCI device, an ID table containing a list of ID entries (*vendor, device, subvendor, subdevice*) are passed to the PCI layer to match a device for this driver
- A probe method (part of pci_driver structure) is registered as a call back
- The generic PCI layer invokes the probe method to allow the device driver to perform device and software initializations (device API for the generic device layer)

Useful Kernel PCI helpers

- Most PCI device drivers read and examine the BAR registers
- Reading the PCI configuration for any device (@PCI controller)
  - pci_read_config_byte/word/dword(pci_dev, offset, into)
  - pci_write_config_byte/word/dword(pci_dev, offset, from)
- Most PCI device drivers read and examine the BAR registers (BAR0, BAR1… Bar5)
  - pci_resource_flags(pci_dev, bar)
  - Type of resource (IO or MEM) can be examined, accordingly used for PIO or MMIO
  - pci_request_regions to check the I/O "address" resource availability
  - pci_resource_start(pci_dev, bar) returns handle to start of an I/O resource
- For MMIO resources
  - pci_ioremap_bar(pci_dev, barnum)
  - Returns a VA handle to operate on the device

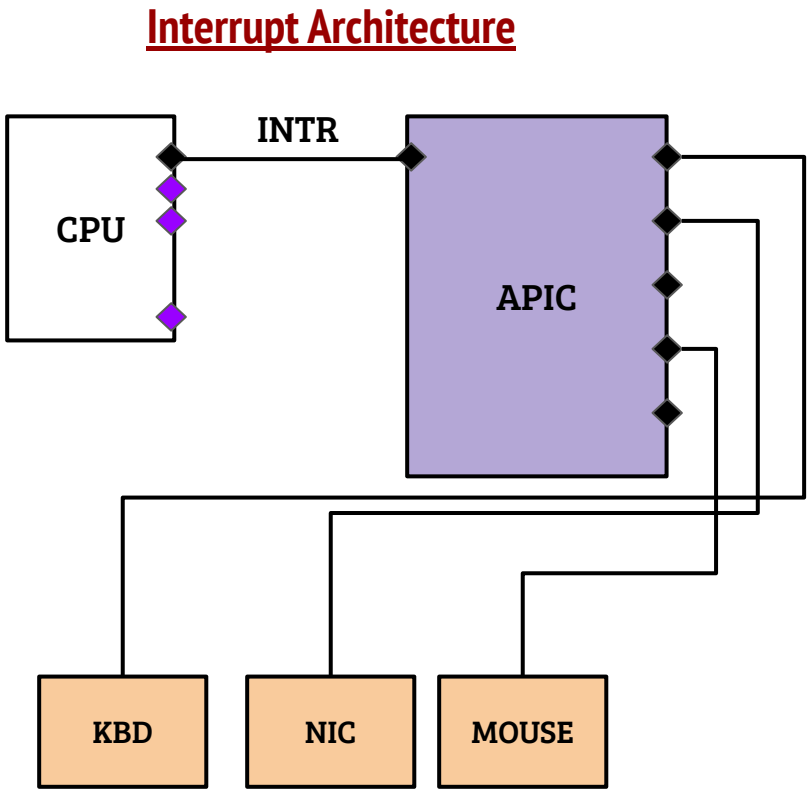
Recap: Simplified Interrupt handling

Interrupt Architecture



- How does PCI fit into this?

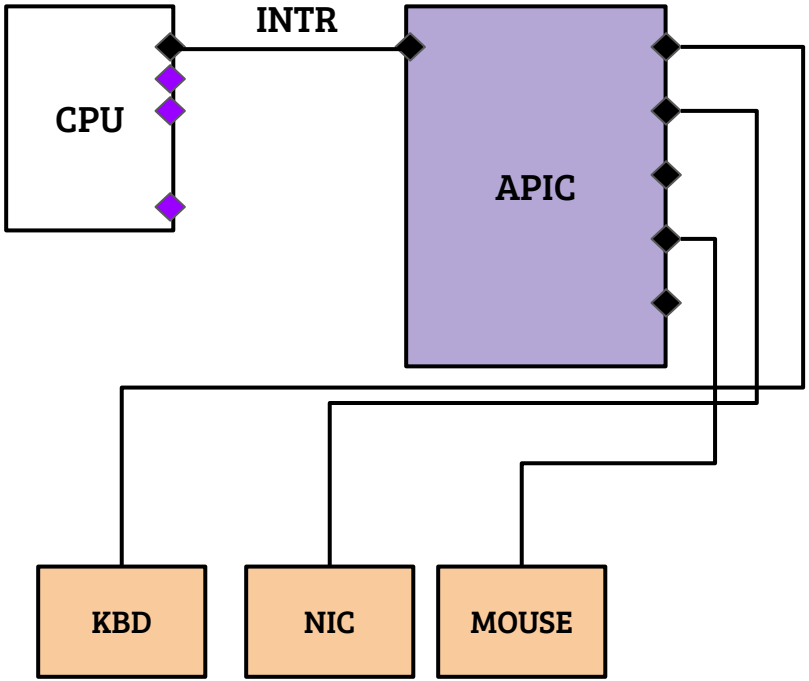
Recap: Simplified Interrupt handling



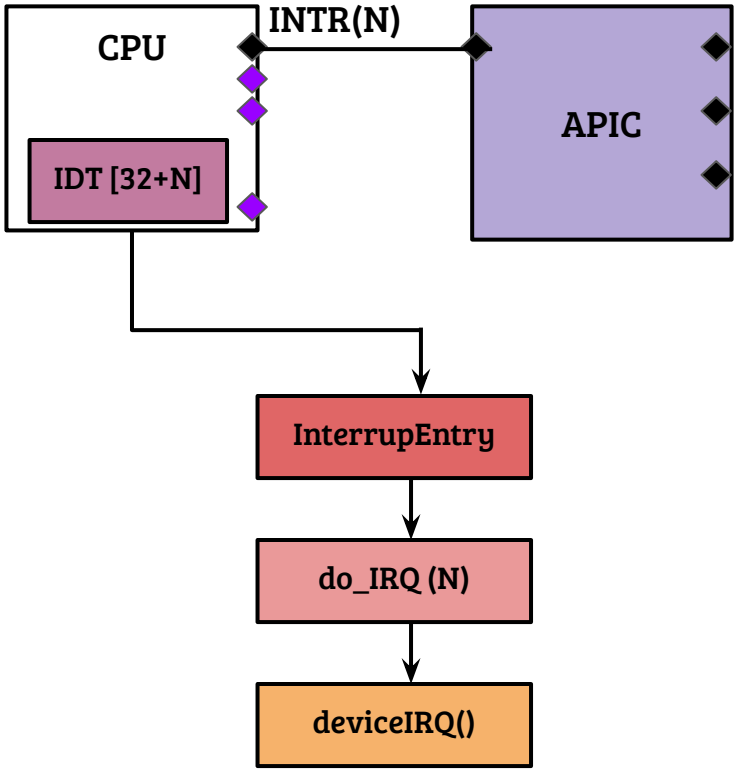
- How does PCI fit into this?
- A device connected through PCI can use upto four interrupt PINs
- Each PIN can be independently forwarded to the core interrupt controllers (e.g, APIC or IOAPIC)
- Typically, IRQs are shared in PCI devices

Recap: Simplified Interrupt handling

Interrupt Architecture



Software Interfacing



Interrupts in PCI devices

- Examining interrupt capability
  - Reading the PCI config using pci_read_config_byte (IRQ pin and IRQ line) directly
  - Using the PCI helper APIs such as
    - pci_alloc_irq_vectors
    - pci_irq_vector
    - request_irq

Interrupts in PCI devices

- Examining interrupt capability
  - Reading the PCI config using pci_read_config_byte (IRQ pin and IRQ line) directly
  - Using the PCI helper APIs such as
    - pci_alloc_irq_vectors
    - pci_irq_vector
    - request_irq
- Interrupt handler
  - The device level callback for interrupt handling is registered during request_irq
  - The handler code must determine if the IRQ belongs to the device, why? And How?

Interrupts in PCI devices

- Examining interrupt capability
  - Reading the PCI config using pci_read_config_byte (IRQ pin and IRQ line) directly
  - Using the PCI helper APIs such as
    - pci_alloc_irq_vectors
    - pci_irq_vector
    - request_irq
- Interrupt handler
  - The device level callback for interrupt handling is registered during request_irq
  - The handler code must determine if the IRQ belongs to the device, why? And How?
    - IRQ may be shared across many devices
    - By checking the interrupt status register of the device