Interruptions and communication

Gaël Thomas

CSC4508 – Operating Systems

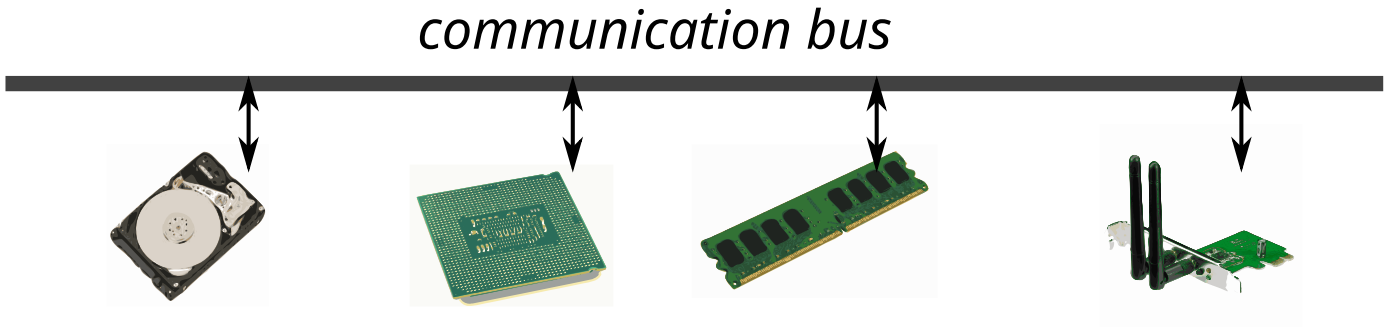
2022–2023

Outlines

1 Communication buses

1.1 Communication buses

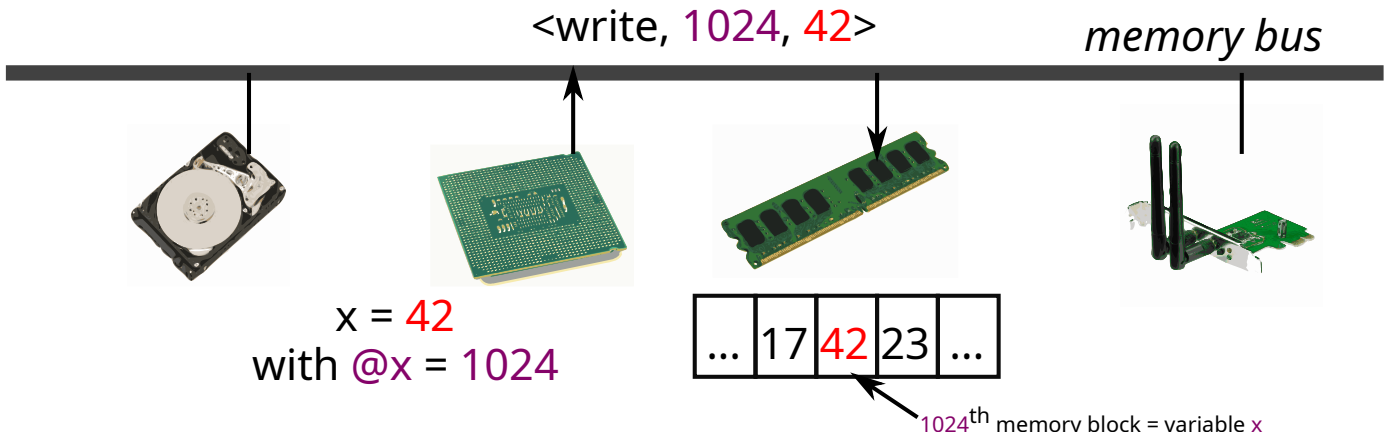
- Hardware components communicate via buses



- From a software point of view, 3 main buses
  - Memory bus: mainly to access memory
  - Input / output bus: messages from CPUs to devices
  - Interrupt bus: messages from peripherals to CPUs
- From the hardware point of view: a set of hardware buses with different protocols that can multiplex the software buses

1.2 The memory bus

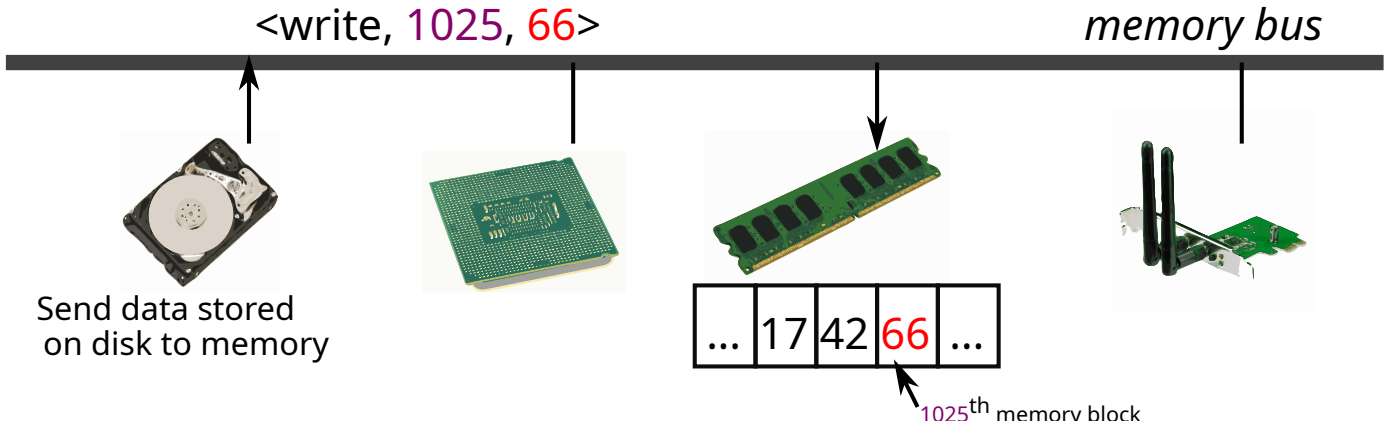
- Processors use the memory bus for reads / writes
  - Sender: the processor or a peripheral
  - Receiver: most often memory, but can also be a device (*memory-mapped IO*)



1.2.1 DMA: Direct Memory Access

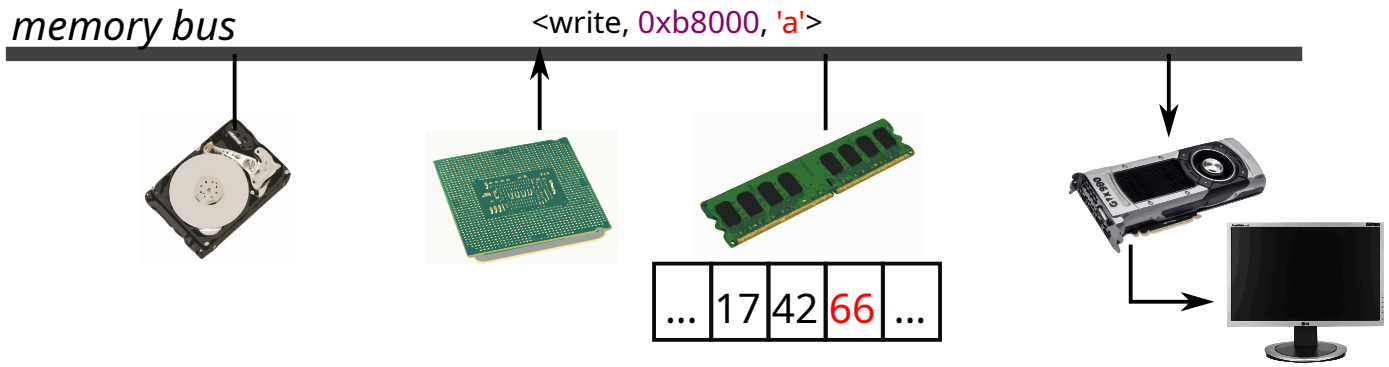
- Devices use the memory bus for reads/writes
  - Sender: a processor or a peripheral
  - Receiver: most often memory, but can also be a device (*memory-mapped IO*)
- The DMA controller manages the transfer between peripherals or memory
  - The processor configures the DMA controller
  - The DMA controller performs the transfer
  - When finished, the DMA controller generates an interrupt

$\Longrightarrow$ The processor can execute instructions during an I/O



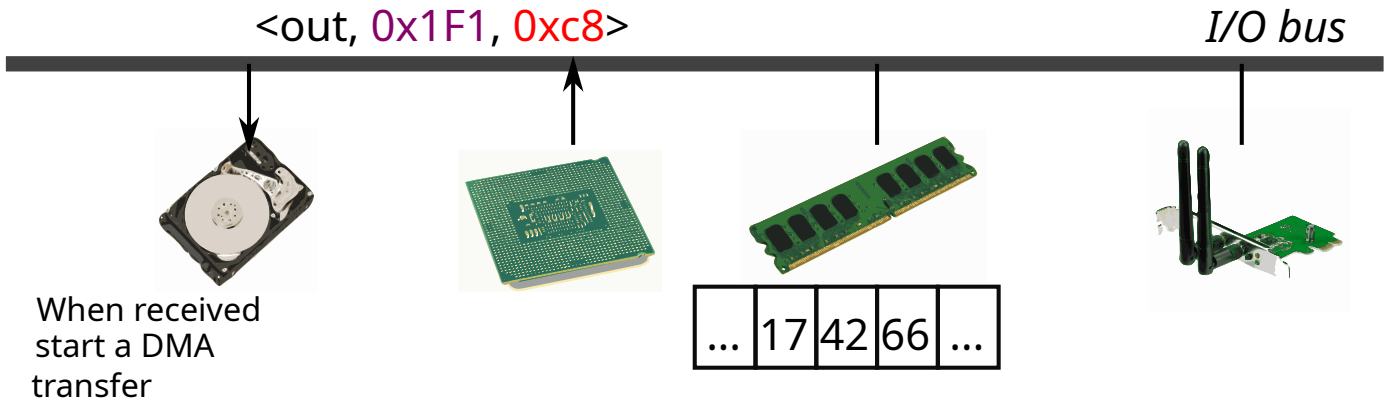
1.2.2 MMIO: Memory-Mapped IO

- Processors use memory bus to access devices
  - Sender: a processor or a peripheral
  - Receiver: most often memory, but can also be a device (*memory-mapped IO*)
- Device memory is *mapped* in memory
  - When the processor accesses this memory area, the data is transferred from / to the device



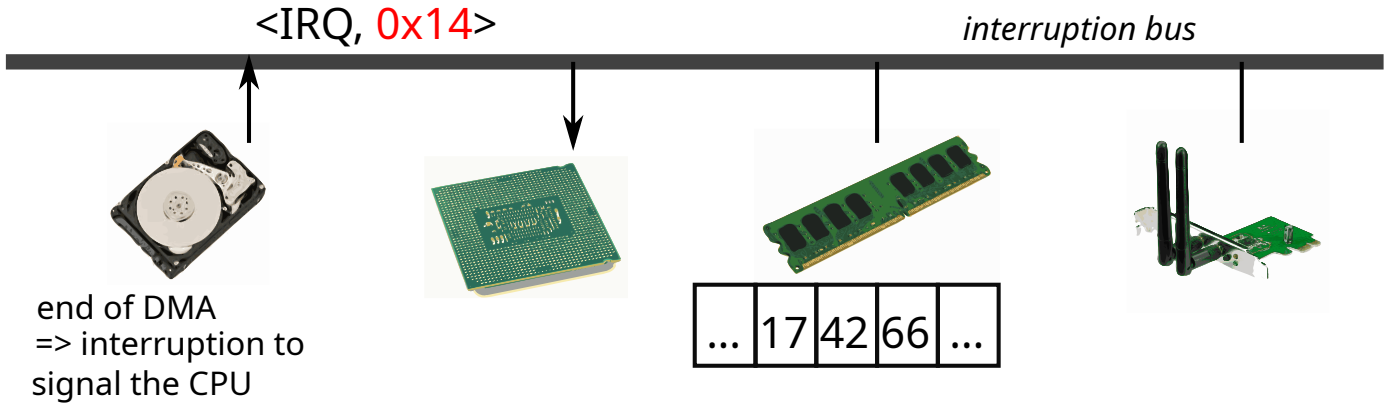
1.3 The input / output bus

- Request / response protocol, special instructions `in`/`out`
  - Sender: aprocessor
  - Receiver: a peripheral
  - Examples: activate the caps-lock LED, start a DMA transfer, read the key pressed on a keyboard ...



1.4 The interrupt bus - principle

- Used to signal an event to a processor
  - Sender: a peripheral or a processor
  - Receiver: a processor
  - Examples: keyboard key pressed, end of a DMA transfer, millisecond elapsed ...
  - **IRQ** (*Interrupt ReQuest*): interruption number. Identifies the sending device



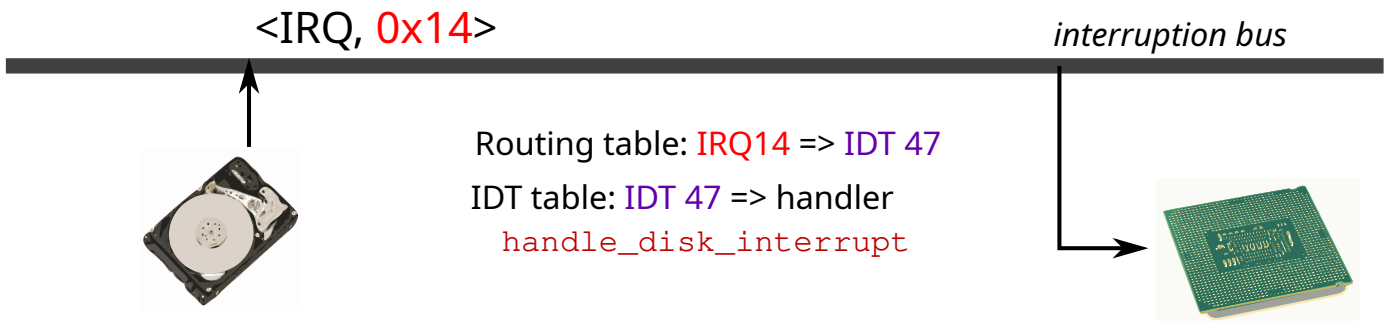
2 Interruptions

2.1 Receiving an interrupt

- Two tables configured by the kernel to handle reception
  - **Routing table**: associate an **IRQ** with an **IDT** number
  - **IDT table** (*interrupt descriptor table*): associate an **IDT** number to a function called **interrupt handler**
- Two tables allow more flexibility than a single table which associates an IRQ number directly with a manager
- Useful in particular with multicore (see the rest of the lecture)

2.2 Receiving an interrupt: example

- A device sends an **IRQ** (for example `0x14`)
- The **routing table** associates `IRQ14` with `IDT47`
- The **IDT table** indicates that `IDT47` is managed by the function `handle_disk_interrupt`

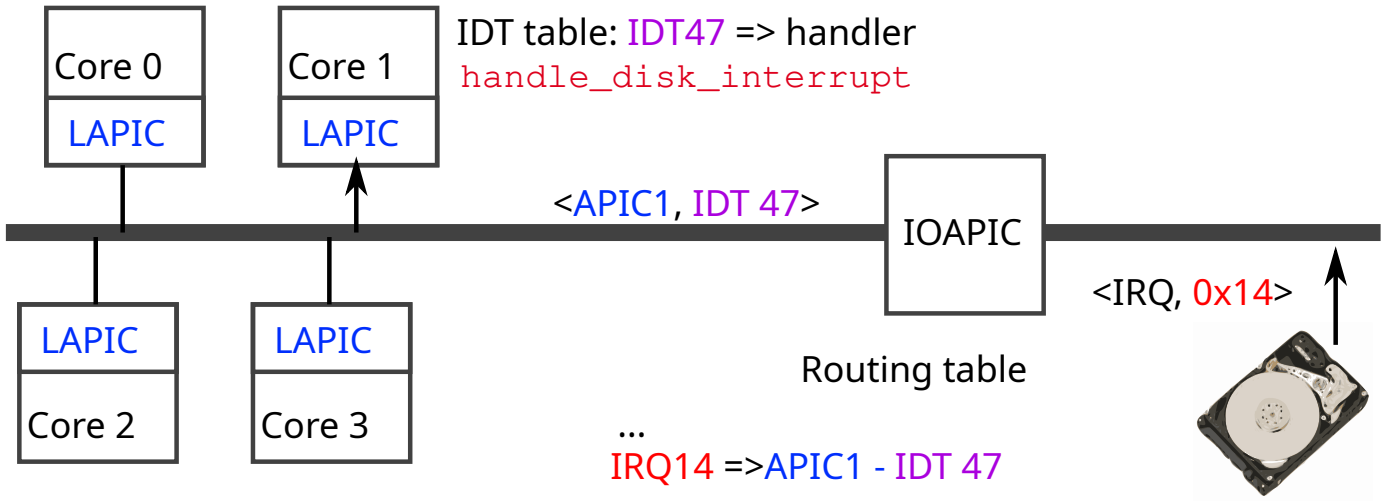


2.3 Receiving an interrupt (continued)

- In the processor, after executing **each instruction**
  - Check if an interrupt has been received
  - If so, find the address of the associated handler
  - Switch to kernel mode and run the interrupt handler
  - Then swicth back to the previous mode and continue the execution
- Note: a handler can be run **anytime**
  - Problem of concurrent access between handlers and the rest of the kernel code
  - Solution: masking interruptions (`cli` / `sti`)

2.4 Interruptions and multicore processors

- XAPIC protocol on pentium (x2APIC since Intel Core processors)
  - Each core has a number called APIC number (*Advanced Programmable Interrupt Controller*)
  - Each core handles interrupts via its LAPIC (*local APIC*)
  - An IOAPIC routes an interrupt to a given LAPIC
    - Routing table configured by the system kernel

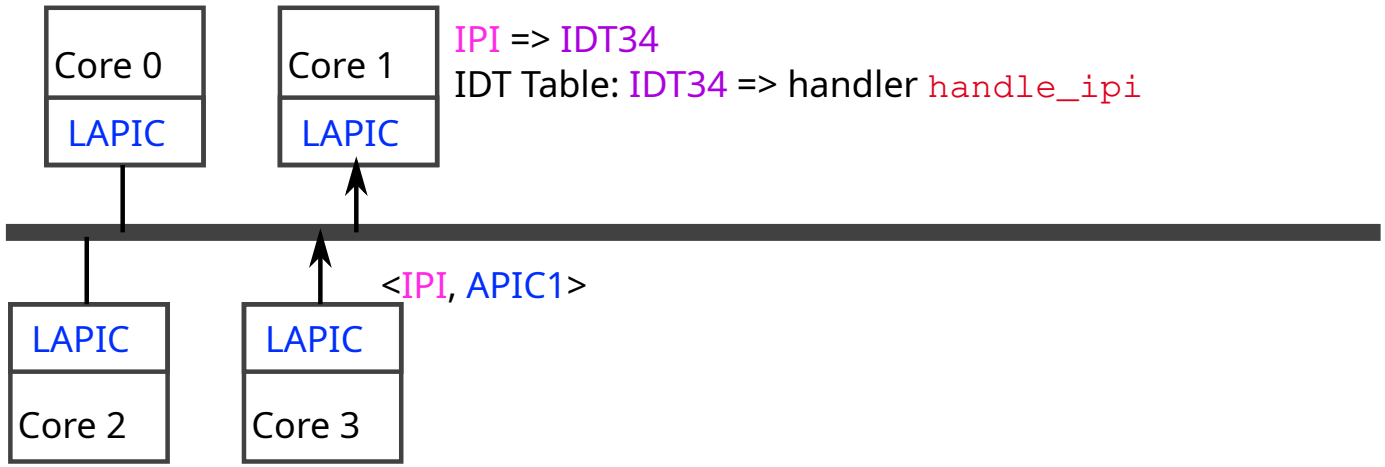


2.5 MSI: Message Signaling Interrupt

- MSI: direct interrupt from a device to a LAPIC without passing through the IOAPIC
  - The kernel must configure the device so that it knows which LAPIC / IDT pair should be generated
  - Used when the need for performance is important

2.6 Inter-core communication

- One core can send an interrupt to another core
  - Called Inter-Processor Interrupt (IPI)
  - LAPIC x sends an IPI to LAPIC y
  - In LAPIC y, receiving an IPI is associated with an IDT number



2.7 IDT table

- Table that associates a handler with each IDT number
  - Used by **interrupts** as seen previously
  - But also for a **system call**: `int 0x64` simply generates the interrupt `IDT 0x64`
  - But also to catch *faults* when executing instructions
    - a division by zero generates the interrupt `IDT 0x00`, an access illicit memory (`SIGSEGV`) the interrupt `IDT 0x0e` etc.
- The IDT table is therefore the table that contains all of the entry points to the kernel
  - From the software via the system call
  - From material for other IDTs

2.8 Time management: two sources

- **Jiffies**: global time source to update the date
  - A device (e.g. HPET) regularly sends IRQ
  - Received by a single core which updates the date
- **Tick**: core-local time source used for scheduling
  - LAPIC regularly generates an interrupt to its core
  - The system associates an IDT number and a handler with this interruption
  - Less precise than the **jiffies**