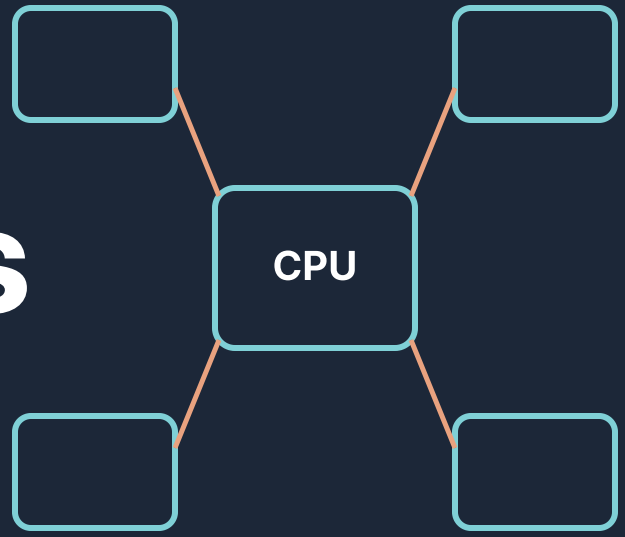
OPERATING SYSTEMS · LECTURE 12

Input / Output Systems

Drivers · buffering · sync & async I/O



Lecturer: Aidana Karibayeva

01 I/O devices

I/O devices exchange information between the computer and the outside world.

Human-readable

talk to the user

`monitor` `printer` `keyboard` `mouse`

Machine-readable

talk to electronic equipment

`disk` `DVD` `USB stick` `sensors`

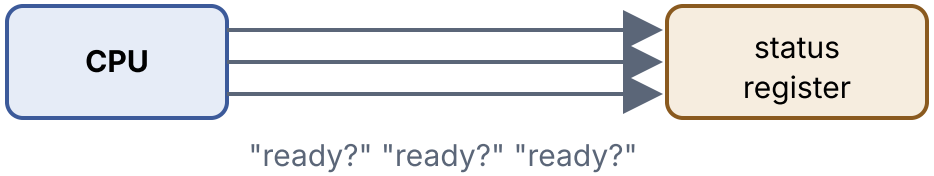
Communication

talk to other intelligent devices

`network card` `modem`

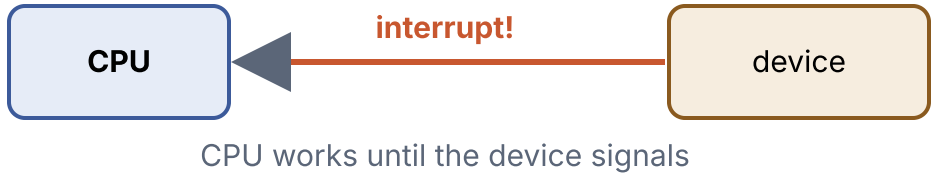
02 How the CPU learns that data has arrived

Polling



CPU checks the device status **periodically**. Example: PS/2 keyboard and mouse.

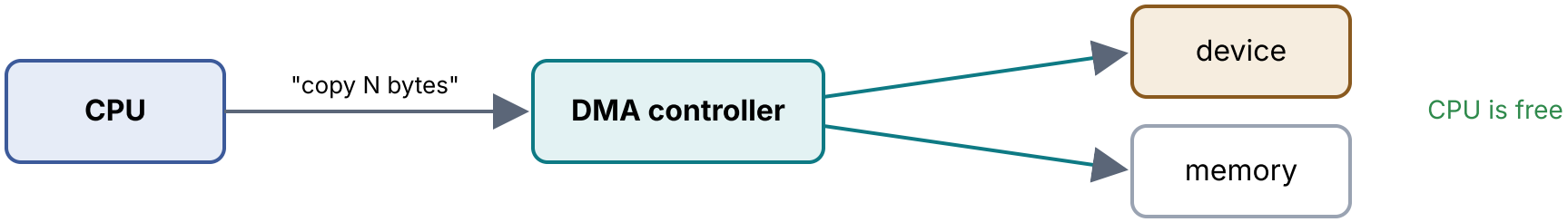
Interrupts



The device **signals** the CPU when it needs attention. Example: disk driver, network card.

03 How data is transferred

METHOD	HOW IT WORKS	CPU INVOLVEMENT
Special-instruction I/O	special CPU instructions move data to/from the device	`high`
Memory-mapped I/O	a memory buffer is shared with the device; it interrupts the CPU when done	`medium`
DMA	a DMA controller reads/writes memory directly, without the CPU	`low`



04 Layers of I/O software

User-level I/O software
libraries such as stdio, spooling

Device-independent I/O software
common functions, uniform interface

Device drivers
device-specific code

Interrupt handlers
run when an interrupt arrives

Hardware
device + controller

LEVEL	EXAMPLE
User	print window in a text editor
Kernel	driver talks to the device controller
Hardware	the printer and its controller

> A request goes **down** the layers; the result comes back **up**.

05 Device drivers

Plug-in kernel modules that hide device-specific code behind a **standard interface**. Usually written by the device manufacturer.

What a driver does

1 · accept a request from the software above

2 · talk to the device controller, handle errors

3 · make sure the request completes

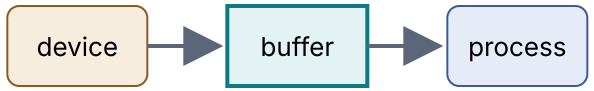
I/O services

Scheduling	order of device requests
Buffering	shared memory areas with devices
Caching	fast copies of device data
Spooling	output queue for a device
Error handling	handlers for device errors

06 I/O buffering

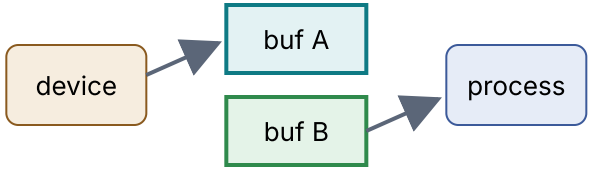
A buffer lets the device and memory exchange data **asynchronously**, so the process is not always suspended.

Single buffer



One block / line / byte. Next block is read while this one is processed; input may still block the process.

Double buffer



One buffer is filled while the other is emptied — no suspension unless both are full.

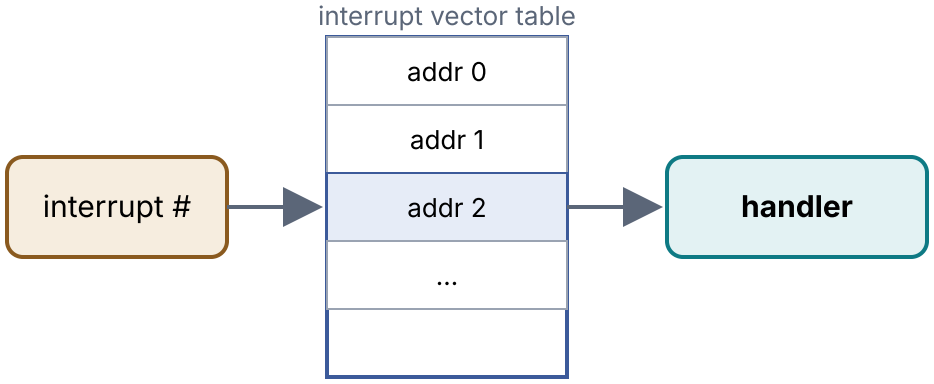
Circular buffer



A ring of buffers with **producer–consumer** access; suspension is very rare.

07 Interrupt handlers and higher layers

Interrupt handler (ISR)



The interrupt number selects a handler address in the **interrupt vector table**.

Device-independent software

`uniform driver interface`

`device naming (major / minor numbers)` `protection`

`common block size` `buffering` `storage allocation`

`dedicated devices` `error reporting`

User-space I/O software

Libraries (e.g. `stdio`) give a simpler interface to the kernel; spooling handles dedicated devices.

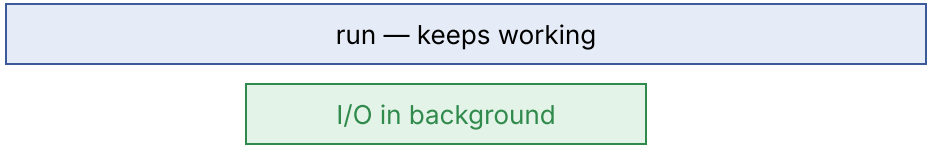
08 Synchronous vs asynchronous I/O

Synchronous



The application **waits** until I/O completes. Transfer is clocked at a fixed rate.

Asynchronous



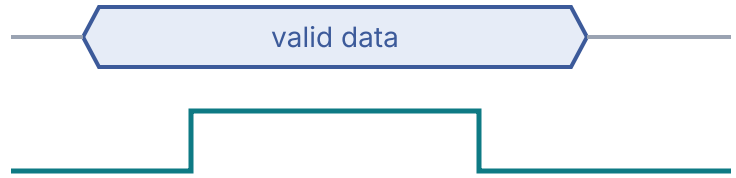
I/O runs **in the background** at an unpredictable rate; good for slow or serial devices.

Problem of asynchronous transfer: no time slot, so it is unclear if the data on the bus is fresh. Solutions: **strobe** and **handshaking**.

09

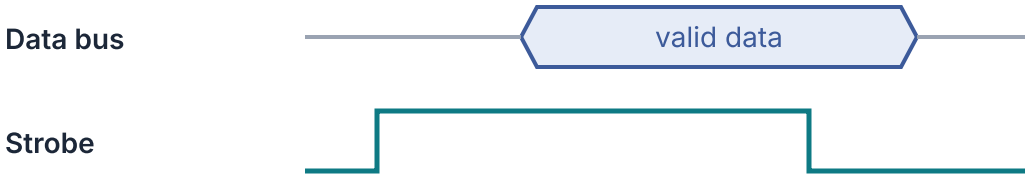
Strobe mechanism

Source-initiated



Source puts data on the bus → strobe ON → destination reads → strobe OFF.

Destination-initiated

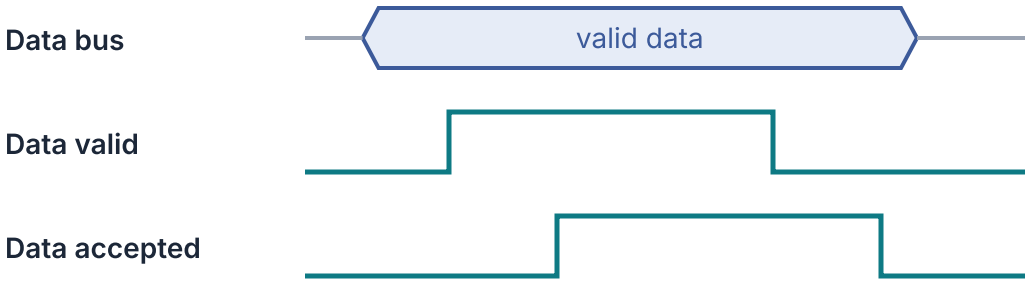


Strobe ON first (request) → source puts data → destination reads → strobe OFF.

Weakness: no confirmation. The source **assumes** the data was read; the destination **assumes** the data was placed.

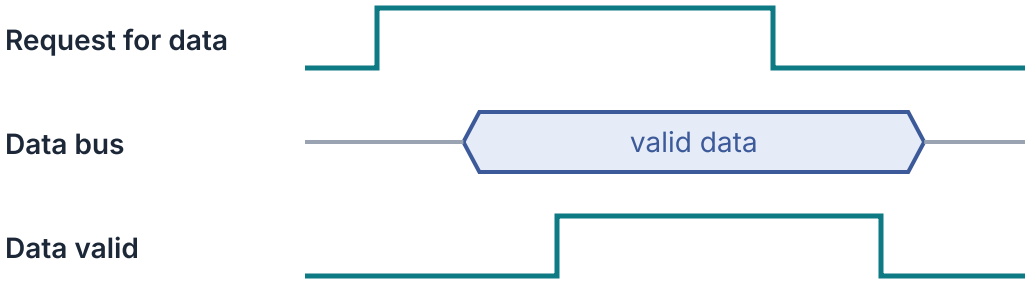
10 Handshaking

Source-initiated



Data → **data valid** ON → **data accepted** ON → data valid OFF → data accepted OFF.

Destination-initiated



Request for data ON → data on bus → **data valid** ON → request OFF → data valid OFF.

> Each side **confirms** the other's action — this fixes the weakness of the strobe.

11

Asynchronous I/O: pros and cons

Advantages

Flexibility	adapts to any transfer speed
Efficiency	no clock synchronisation overhead
Lower latency	more responsive system
Error handling	errors detected during transfer
Compatibility	works with many devices

Disadvantages

Complexity	needs interrupt-driven I/O
Overhead	CPU monitors interrupt signals
Latency	waiting for interrupts
Synchronisation	order with many devices
Compatibility	not for fixed-rate protocols