... airplane, fuel injection system of a car, …

Non-traditional devices: joystick, robot actuators, flying surfaces of an airplane.

Traditional devices: disk drive, printer, keyboard, mouse, display

- The device itself
  - The device itself
    - Controller: receives commands from the system bus, translates them into device
    - Data-out: data being sent from the CPU to the device
    - Data-in: data being sent from the device to the CPU
    - Control: command to perform
    - Status indicators: device busy, data ready, or error condition
- A device port typically consisting of 4 registers
- Multiple devices
- System bus: allows the device to communicate with the CPU, typically shared by

**Key components**

**Architecture of I/O Systems**

- How can the OS improve the performance of I/O?
- How does the OS implement these services?
- What I/O services does the OS provide?
- How does I/O hardware influence the OS?

**Today: I/O Systems**
What happens if the device is slow compared to the CPU?

- Good choice if data must be handled promptly, like for a modern or keyboard.
- CPU does not observe the change to idle and reads the data if it was an input operation.
- If the operation succeeded, the controller changes the status to idle.
- Controller reads the command register and performs the command, placing a value in
- CPU sets status to command-ready = controller sets status to busy
- CPU sets the command register and displays if it is an output operation.
- CPU busy-waits until the status is idle.

**Communication using Polling**

---

**I/O Services Provided by OS**

- Device drivers to implement device-specific behaviors.
  - Example: error handling and failure recovery associated with devices (command resets, reads
  - I/O scheduling
  - Buffering, caching, and spooling to allow efficient communication with devices.
  - Device allocation.
  - Operations appropriate to the files and devices.
  - Access control.
  - Naming of files and devices. (On Unix, devices appear as files in the /dev directory)
Transfer I/O:
CPU somewhat, but still providing better performance than if the CPU had to do the
- DMA controller and the CPU compete for the memory bus, slowing down the
  transfer.
- DMA controller is complete. Instead of when each byte is ready
  the DMA controller operates the bus and interrupts the CPU when the entire
- DMA uses the DMA the locations of the source and destination of the transfer.
- DMA controller that can write directly to memory. Instead of

\[ \text{Solution: Direct memory access (DMA)} \]

For devices that transfer large volumes of data at a time (like a disk

Direct Memory Access

Communication using Interrupts

On an I/O interrupt:

- Start the next operation for that device.
- If the last command was an input operation, retrieve the data from the device register.
- Determine which device caused the interrupt.

\[ \text{On an I/O interrupt:} \]

Rather than using busy waiting, the device can interrupt the CPU when it

\[ \text{Communications using Interrupts} \]
• The DMA controller interrupts the CPU when the transfer is done.

physical memory

• It is transferred over the bus by the DMA controller into a buffer in

• A disk buffer stores a block when it is read from the disk.

• Data temporarily before transferring to/from the CPU.

I/O devices typically contain a small on-board memory where they can store

1/0 Buffering

1/0 Buffering

Device characteristics:

- Examples: keyboard (sequential, character), disk (block, random or sequential)
- Operations: input, output, or both
- Synchronized
- Shareable or dedicated
- The OS implements blocking
- Most devices are synchronous, while I/O system calls are synchronous.

Timing: synchronous or asynchronous

Access method: sequential or random access

- Transfer unit: character or block

The OS provides a high-level interface to devices, greatly simplifying the

Application Programmer's View of I/O Devices

Application Programmer's View of I/O Devices
What should happen when we write to a cache?

- If (block in memory) return value from memory
  - Example: Read (disk address)
- Else read from disk (if needed)
  - Example: Write (disk address)
- Else allocate space in memory, read block from disk, and update value in memory
  - Write-back policy (write to the fastest memory containing the block, write to disk).
- Write-through policy (write to all levels of memory containing the block, including the disk).

**Caching**

```

Why buffer on the OS side?

- Example: Compute the contents of a display in a buffer (slow) and then zapping the buffer to the screen (fast)
- To cope with devices that have different data transfer sizes.
- To cope with speed mismatches between device and CPU.
```

- Improve disk performance by reducing the number of disk accesses.

**Cache**
Utilization:
- Increase physical memory to reduce amount of time paging and thereby improve CPU utilization.
- Increase the number of devices to reduce contention for a single device and thereby decrease computation from the main CPU by using DMA controllers.
- Reduce interrupt frequency by using large data transfers.
- Reduce data copying by caching in memory.

Approaches to Improving Performance:
- I/O is瓶颈 supported via system calls and interrupt handling. Which are slow.
- Conversion from multiple processes.
- Slow devices and slow communication links.

Summary

Putting the Pieces Together - A Typical Read Call

4. When the process gets the CPU, it begins execution following the system call.

3. OS transfers the data to the user process and places the process in the ready queue.

2. OS checks if data is in a buffer, if not,

1. User process requests a read from a device.

(d) DMA controller transfers the data from the device.
(c) DMA controller transfers the data to the kernel buffer when it has all been retrieved.
(b) Device driver tells the DMA controller what to do and blocks itself.
(a) OS tells the device driver to perform input.