I/O Devices
You Need to Get Out More!

How does a computer connect with the outside world?
I/O Architecture
Interacting with a Device

Abstraction
(what the user sees)
Interacting with a Device

**Interface**
(what the OS sees)

**Internals**
(what is needed to implement the abstraction)
Interacting with a Device

<table>
<thead>
<tr>
<th>Registers</th>
<th>Status</th>
<th>Command</th>
<th>Data</th>
</tr>
</thead>
</table>

Internals
(what is needed to implement the abstraction)
Interacting with a Device

OS controls device by reading/writing registers

```
while (STATUS == BUSY)
    ; // wait until device is not busy
write data to DATA register
write command to COMMAND register
    // starts device and executes command
while (STATUS == BUSY)
    ; // wait until device is done with request
```
Tuning It Up

- CPU is polling
  - use interrupts
  - run another process while device is busy
  - what if device returns very quickly?

- CPU is copying all the data to and from DATA
  - use Direct Memory Access (DMA)

```c
while (STATUS == BUSY)
    ; // wait until device is not busy
write data to DATA register
write command to COMMAND register
    // starts device and executes command
while (STATUS == BUSY)
    ; // wait until device is done with request
```
From interrupt-driven I/O to DMA

Interrupt driven I/O

- Device ↔ CPU ↔ RAM

  for \((i = 1 \ldots n)\)
  - CPU issues read request
  - device interrupts CPU with data
  - CPU writes data to memory
From interrupt-driven I/O to DMA

**Interrupt driven I/O**
- Device ↔ CPU ↔ RAM
  - for $(i = 1 \ldots n)$
  - CPU issues read request
  - device interrupts CPU with data
  - CPU writes data to memory

**+ Direct Memory Access**
- Device ↔ RAM
  - CPU sets up DMA request
  - Device puts data on bus & RAM accepts it
  - Device interrupts CPU when done
Communicating with devices

- **Explicit I/O instructions** (privileged)
  - `in` and `out` instructions in x86

- **Memory-mapped I/O**
  - map device registers to memory location
  - use memory load and store instructions to read/write to registers
How can the OS handle a multitude of devices?

- **Abstraction!**
  - Encapsulate device specific interactions in a device driver
  - Implement device neutral interfaces above device drivers

- **Humans are about 70% water...**
  - ...OSs are about 70% device drivers!

---

**File System Stack (simplified)**

- **Application**
  - POSIX API [open, read, write, close, etc]
- **File System**
- **Generic Block Interface** [block read/write]
- **Block Cache**
- **Generic Block Layer**
- **Protocol-specific Block Interface**
- **Device Driver** [SCSI, ATA, etc]
- **Memory-mapped I/O, DMA, Interrupts**
- **Physical Device**