I/O Devices - General (ch. 36+37)

Operating Systems

Based on: Three Easy Pieces by Arpaci-Dusseaux

Moshe Sulamy

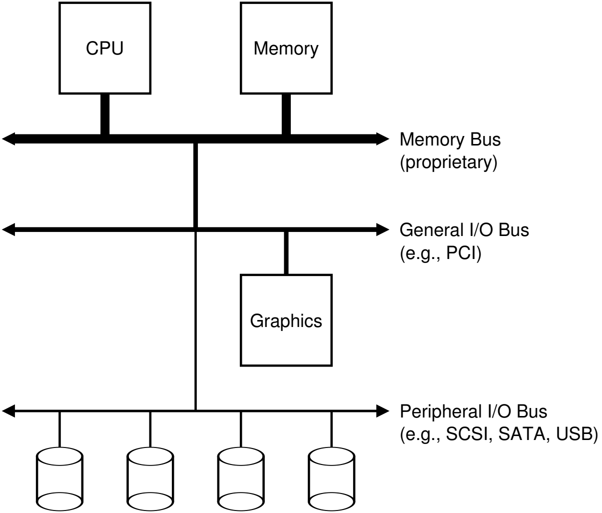
Tel-Aviv Academic College

I/O Devices

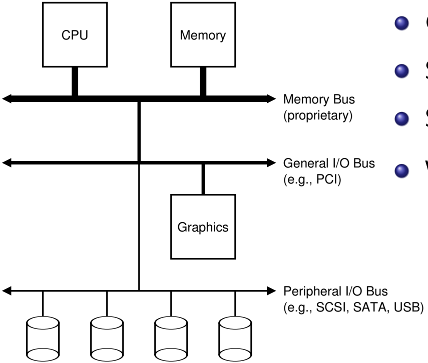
- New part: **persistence**
- But first: **input/output (I/O) devices**
  - Critical to computer systems

How should I/O be integrated into systems?

System Architecture



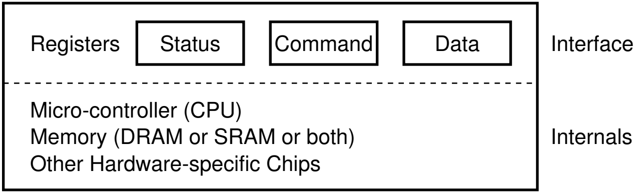
System Architecture



- CPU attached to memory via **memory bus**
- Some devices via **general I/O bus**
- Slow devices via **peripheral bus**
- Why hierarchical?
  - Physics and cost
  - Faster bus → shorter
  - Lower performance → further

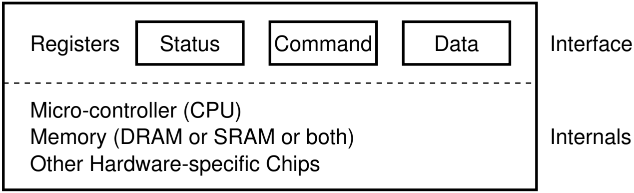
Canonical Device

- A canonical device has two important components:
  - **Hardware interface**: allows the system to control its operation
  - **Internal structure**: implementation specific to the device



Canonical Protocol

- Device interface comprised of three registers
  - **Status**: current status of device
  - **Command**: tell device to perform a task
  - **Data**: pass data to device or get data from it



- Control device behavior by reading and writing these registers

Canonical Protocol

- Typical interaction of OS with the device:

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

Canonical Protocol

1. **Polling**
   - Repeatedly reading status register
2. OS sends some data
   - Multiple writes may be needed
   - CPU involved with data movement: **programmed I/O (PIO)**
3. Write command
   - Lets device know that data is present
4. **Polling**
   - OS waits for device to finish

Canonical Protocol

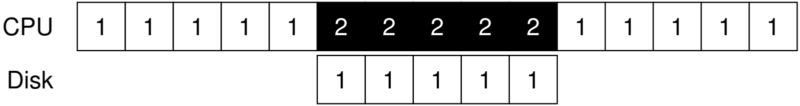
- Polling is inefficient
  - Wastes CPU time waiting for device
  - Switch to another process: better utilize CPU

CPU	1	1	1	1	1	p	p	p	p	p	1	1	1	1	1
Disk						1	1	1	1	1					

How can the OS check device status without polling?

Interrupts

- Instead of pulling, issue a request
  - Put calling process to sleep
  - Context switch to another task
- Device finished: hardware interrupt
  - CPU jumps into OS **interrupt handler**
  - (Also: **interrupt service routing (ISR)**)
  - Handler will finish the request and wake waiting process



Interrupts

- Not always the best solution
  - Device performs very quickly
  - Interrupts will slow down the system
  - Switching back and forth is expensive
- **Hybrid** (**two-phased** approach)
  - Poll for a little while
  - If not finished, use interrupts

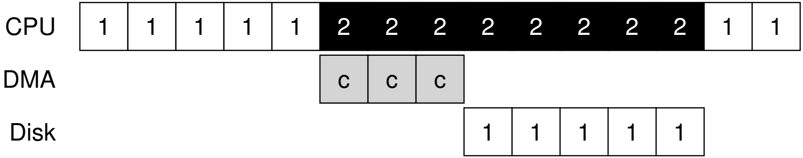
DMA

- Programmed I/O:
  - CPU transfers a large chunk of data to a device
  - CPU overburdened with a trivial task

CPU	1	1	1	1	1	c	c	c	2	2	2	2	2	1	1
Disk									1	1	1	1	1		

DMA

- Solution: **Direct Memory Access (DMA)**
  - DMA controller (a device) handles copying of data
  - OS programs DMA:
    - Where the data lives in memory
    - How much data to copy
    - Which device to send to/read from



Device Interaction

- How does the OS/CPU communicate with devices?
- **I/O instructions**
  - OS sends data to specific device registers
  - For example, `in` and `out` **privileged** instructions on x86
- **Memory-mapped I/O**
  - Device registers available as if they were memory locations
  - OS issues `load` or `store` to address
  - Hardware routes to device instead of main memory

Device Driver

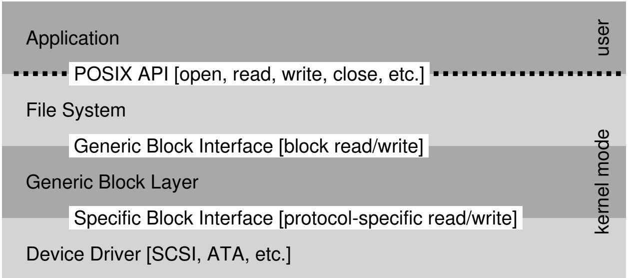
- Devices have specific interfaces
  - Keep OS as general as possible
  - e.g., build file system that works on SCSI disks, IDE disks, USB drives, etc.
- Solution?

Device Driver

- Devices have specific interfaces
  - Keep OS as general as possible
  - e.g., build file system that works on SCSI disks, IDE disks, USB drives, etc.
- Solution? **abstraction**
  - Software that knows device specifics: **device driver**
  - Over 70% of OS code in Linux
  - Primary contributor to **kernel crashes**

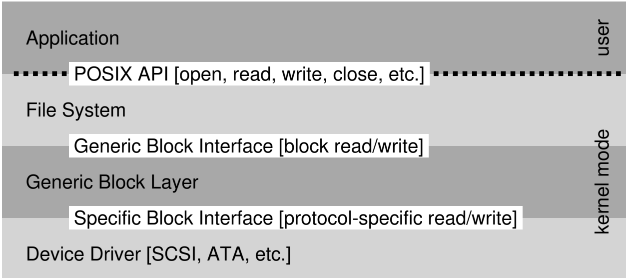
Device Driver

- The Linux file system software stack:



Device Driver

- The Linux file system software stack:



- Also available: **raw interface**
  - Enables special applications to directly read and write blocks
  - e.g., file-system checker, disk defragmentation tool

Summary (I/O Devices)

- Canonical device: registers, HW interface, internal structure
- Canonical protocol: polling, data, command, polling
- Interrupts: instead of polling, issue a request
  - Device finished: hardware interrupt
  - **Hybrid** approach: poll for a little while, then use interrupts
- **DMA** controller: handles copying of data
- Device interaction: **I/O instructions** or **memory-mapped I/O**
- **Device driver**: software abstraction that knows device specifics