I/O Devices

Glenn Bruns
CSUMB

Lecture Objectives

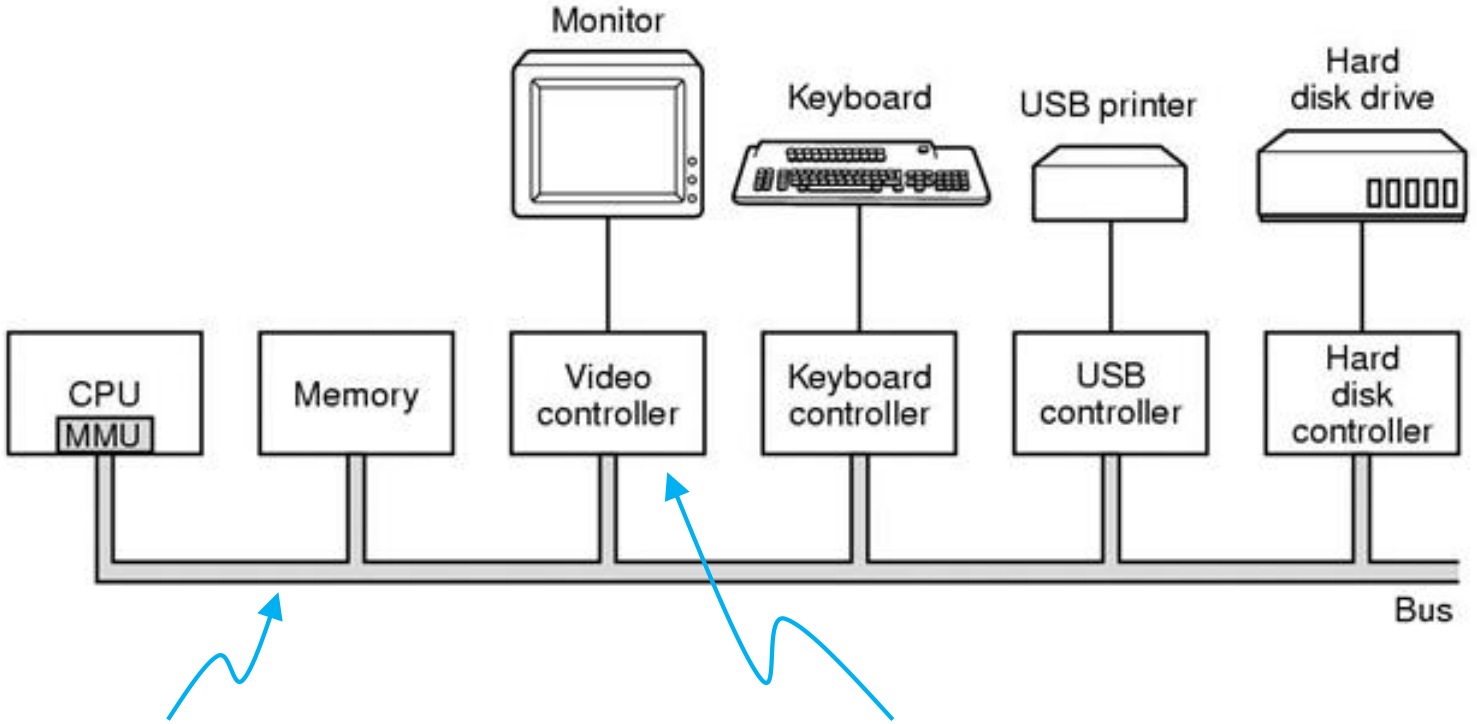
After this lecture, you should be able to:

- list the characteristics of block and character devices
- describe the hardware interfaces of I/O devices
- explain how the OS interacts with I/O devices

Motivation

- How does the OS actually talk to devices?
  - what CPU operations are involved?
  - how are devices addressed?
- How does the OS abstract away from the details of specific devices?
  - what is the software architecture?

Review: computer architecture

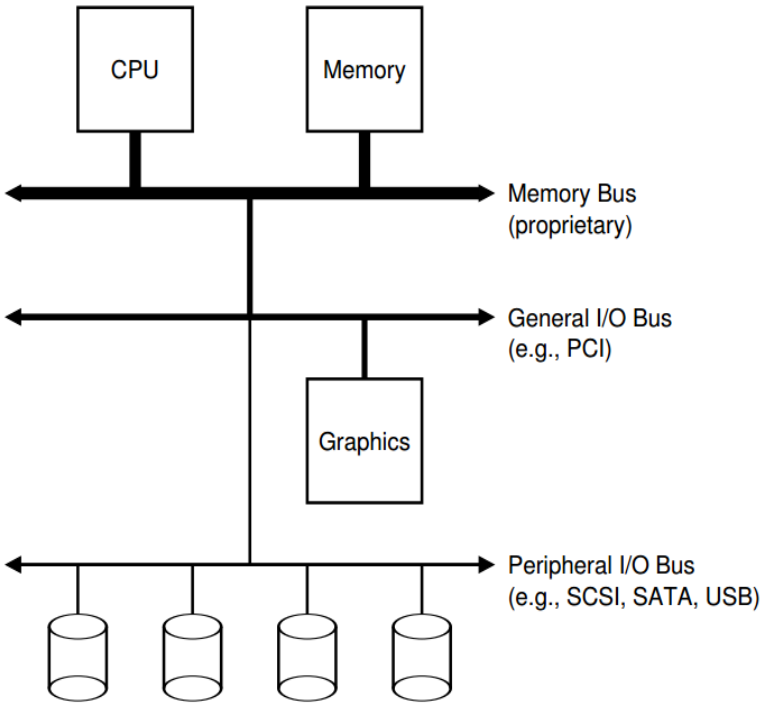


the bus carries bytes between components

a controller is hardware; it can be located in the computer or integrated into a device

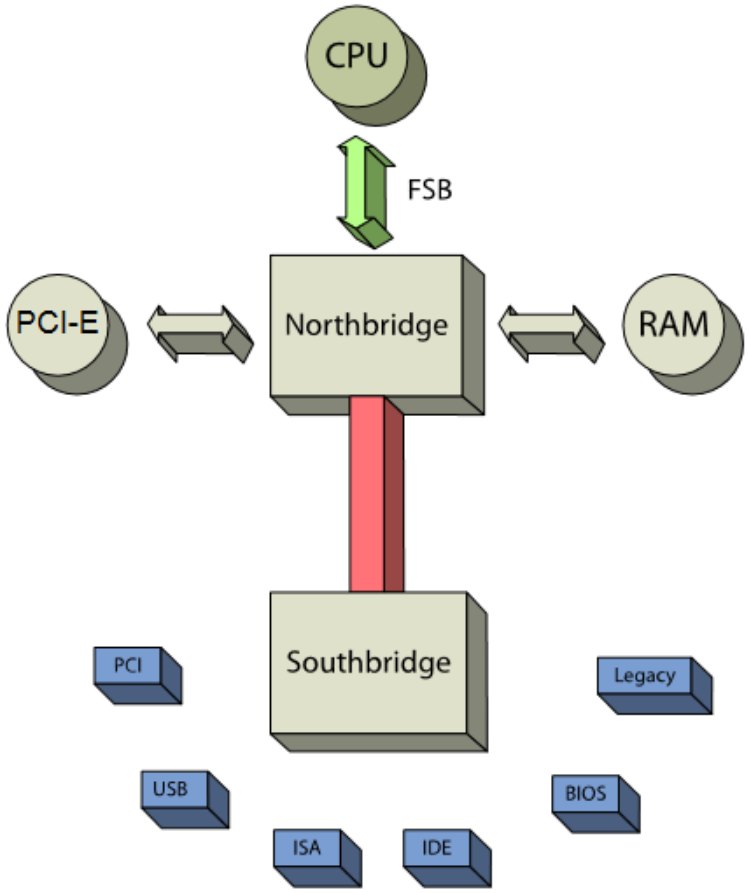
(Figure from Tanenbaum, *Modern Operating Systems*, 3rd edition)

Memory and I/O buses



(Source: Arpacci-Dusseau, *OSTEP,* chapter 36)

PC architecture



(Source: By Fred the Oyster, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=35241077)

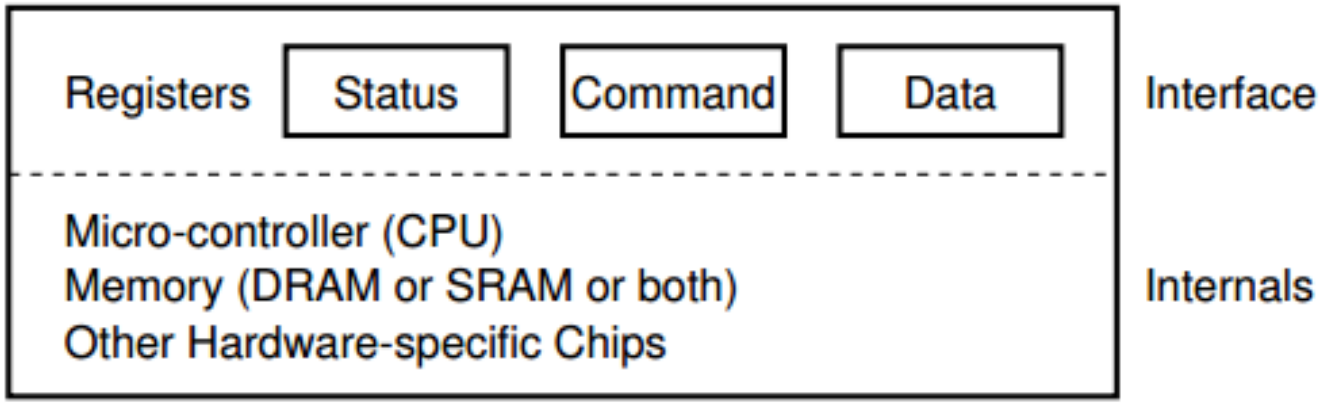
I/O Devices

- Block devices
  - example: hard drive
  - random access of fixed-size “blocks” of data
  - each block has an address
  - block size typically 512 bytes to 32 KB
  - transfers are in units of entire blocks
- Character devices
  - examples: printers, mouses, keyboards
  - processes a stream of bytes, one after another
  - character devices are not addressable

I/O Device structure

What's in a device?

- the same things you might find in any computer
  - CPU, memory, special purpose chips
- a hardware interface
  - typically a set of **registers**
  - status register: read it to see device status
  - command register: write to it to tell device to do something
  - data register – used to pass data to/from device



How CPU interacts with device

```
while (status == busy)
   ;
write data to data register
write command to command register
while (status == busy)
   ;
```

executes command

This is called "programmed I/O" – the CPU is running code to talk to the device.

Note use of polling.

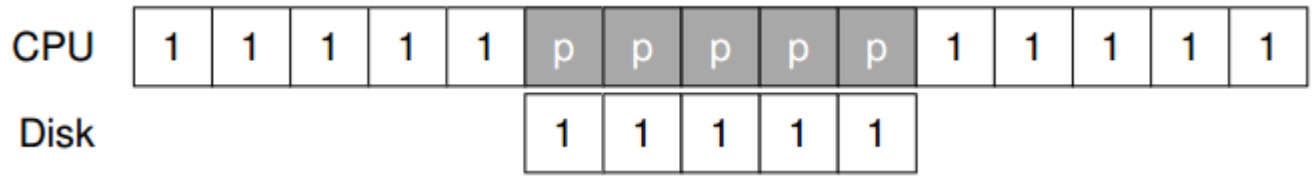


figure source: OSTEP

Option 2: Interrupt-driven I/O

To avoid polling, I/O can be interrupt-driven:

- OS issues an I/O request
- OS puts calling process to sleep
- when I/O device is done, it raises a hardware interrupt
- interrupt handler finishes processing of request; wakes the waiting process

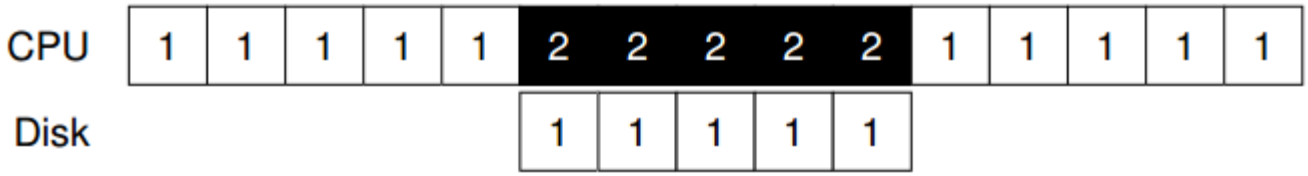


figure source: OSTEP

Problem with data movement

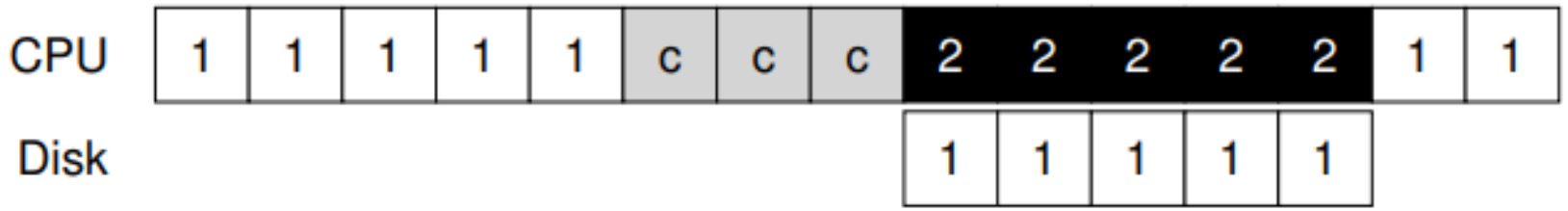


figure source: OSTEP

Option 3: Direct Memory Access (DMA)

DMA uses a special hardware device that transfers data between devices and memory without CPU help.

- CPU tells DMA about device, memory location, quantity to transfer,…
- DMA does the data transfer
- when done, DMA raises an interrupt

How does OS talk to devices?

Hardware interface to devices usually has registers.

How does CPU read/write those registers?

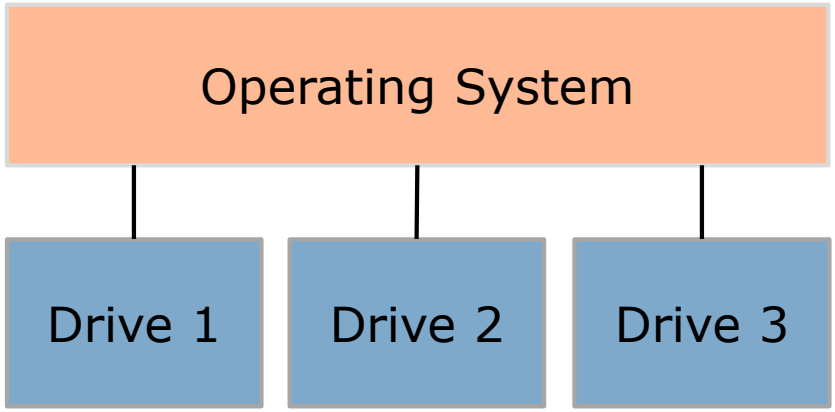
Approach 1:

- CPU has special instructions, such as ‘in’, ‘out’
- the arguments specify a CPU register, and a “port” that names a device

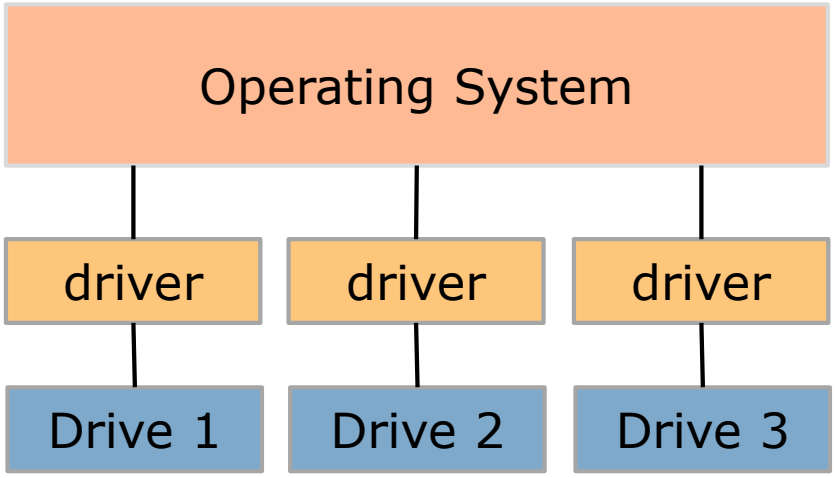
Approach 2:

- device registers are mapped to memory addresses (“memory-mapped I/O”)

Insulating OS code from hardware



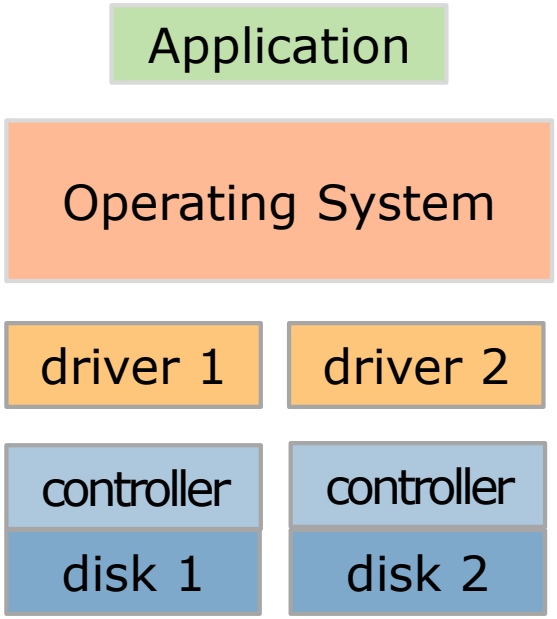
Bad: OS is aware of all types of driver interface



Good: OS is oblivious to differences between drives

Device drivers

- Each device type needs its own driver code
- There are lots of devices, so lots of drivers
- 70% of Linux is device driver code!
- Microsoft hates device drivers
  - many Windows crashes caused by bad device drivers
  - lots of money on software to automatically find problems with driver code



Hardware interface for IDE disk drive

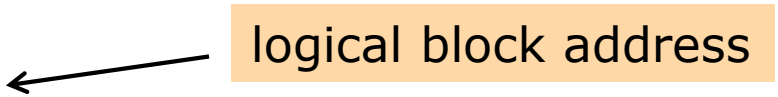
```
Control Register:
  Address 0x3F6 = 0x80 (0000 1RE0): R=reset, E=0 means "enable interrupt"

Command Block Registers:
  Address 0x1F0 = Data Port
  Address 0x1F1 = Error
  Address 0x1F2 = Sector Count
  Address 0x1F3 = LBA low byte
  Address 0x1F4 = LBA mid byte
  Address 0x1F5 = LBA hi  byte
  Address 0x1F6 = 1B1D TOP4LBA: B=LBA, D=drive
  Address 0x1F7 = Command/status

Status Register (Address 0x1F7):
    7      6     5      4     3     2     1     0
   BUSY  READY FAULT SEEK  DRQ  CORR IDDEX ERROR

Error Register (Address 0x1F1): (check when Status ERROR==1)
    7      6     5      4     3     2     1     0
   BBK    UNC   MC    IDNF  MCR  ABRT T0NF AMNF

   BBK  = Bad Block
   UNC  = Uncorrectable data error
   MC   = Media Changed
   IDNF = ID mark Not Found
   MCR  = Media Change Requested
   ABRT = Command aborted
   T0NF = Track 0 Not Found
   AMNF = Address Mark Not Found
```



Device driver for IDE disk drive

```
Control Register:
  Address 0x3F6 = 0x80 (0000 1RE0): R=reset, E=0 means

Command Block Registers:
  Address 0x1F0 = Data Port
  Address 0x1F1 = Error
  Address 0x1F2 = Sector Count
  Address 0x1F3 = LBA low byte
  Address 0x1F4 = LBA mid byte
  Address 0x1F5 = LBA hi  byte
  Address 0x1F6 = 1B1D TOP4LBA: B=LBA, D=drive
  Address 0x1F7 = Command/status

Status Register (Address 0x1F7):
    7      6     5      4     3     2     1     0
   BUSY  READY FAULT SEEK  DRQ  CORR IDDEX ERROR

Error Register (Address 0x1F1): (check when Status ERRO
    7      6     5      4     3     2     1     0
   BBK    UNC   MC    IDNF  MCR  ABRT T0NF AMNF

   BBK  = Bad Block
   UNC  = Uncorrectable data error
   MC   = Media Changed
   IDNF = ID mark Not Found
   MCR  = Media Change Requested
   ABRT = Command aborted
   T0NF = Track 0 Not Found
   AMNF = Address Mark Not Found
```

Basic idea for a write:

- wait for drive to be ready (status register)
- write LBAs, drive number to command registers
- start the I/O (command register 0x1f7)
- data transfer (wait for READY and DRQ; write data to data port)
- handle interrupts
- error handling (check status register)

Summary

- The hardware interface of I/O devices normally consists of some registers (status, data, command,…)
- The OS reads/writes these registers via either:
  - special CPU I/O instructions (in/out)
  - memory-mapped I/O
- Device drivers insulate the OS from details of individual devices