

Interrupts and I/O

Hui Chen ^a

^aCUNY Brooklyn College

February 11, 2021

Outline

- 1 Overview of Computer Architecture
- 2 Overview of I/O devices
- 3 Input/Output
 - Addressing Device Memories
 - I/O Schemes
 - Polling
 - Interrupted I/O
 - DMA
- 4 I/O Software
- 5 Simple Character Device Driver in Linux

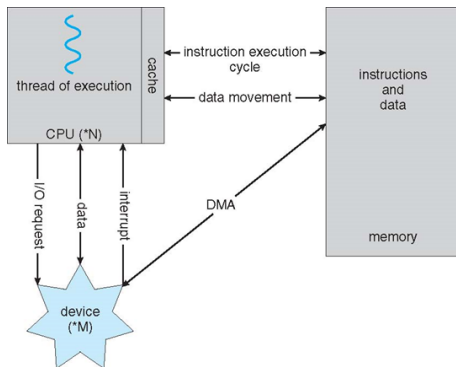
Architecture, OS, and Programming

- ▶ Architecture underpins design of OS and programming
- ▶ How we wrote our boot sector code?
- ▶ How we write a program in high-level programming languages like C/C++, Java, and Python?

von Neumann Computers

Process and memory connected by a bus (Von Neumann, 1945)

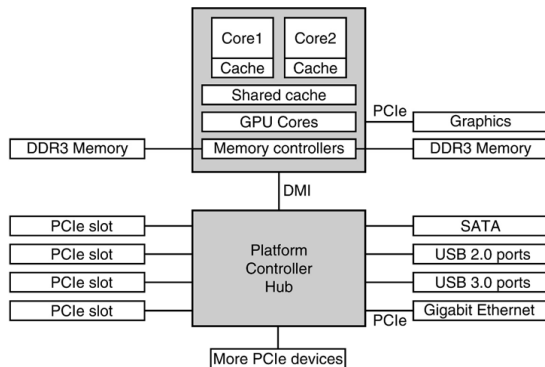
Modern von Neumann Computers



Source: Figure 1.7 in Silberschatz et al., 2018¹

¹Abraham Silberschatz, Peter B. Galvin, and Greg Gagne. *Operating system concepts*. 10th edition. John Wiley & Sons, 2018.

An x86 Realization



Source: Figure 1-12 in Tanenbaum and Bos, 2014²

²Andrew S. Tanenbaum and Herbert Bos. *Modern Operating Systems*. 4th. USA: Prentice Hall Press, 2014. ISBN: 013359162X.

von Neumann Bottleneck

"I propose to call this tube the von Neumann bottleneck. The task of a program is to change the contents of the store in some major way; when one considers that this task must be accomplished entirely by pumping single words back and forth through the von Neumann bottleneck, the reason for its name becomes clear."
– John Backus, 1977

Discussion: Examples of I/O Devices

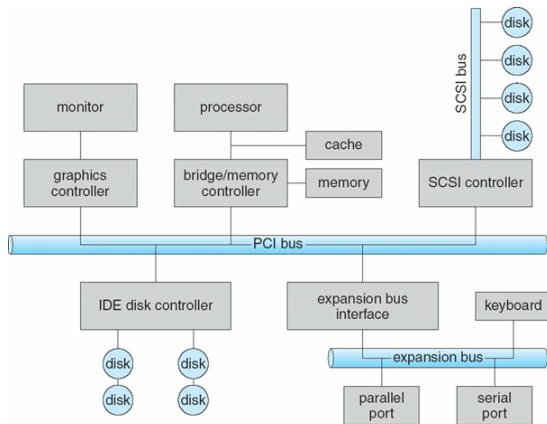
- ▶ What are the examples of computer I/O devices?
- ▶ How do we categorize them? Why do we categorize them?
- ▶ How do we connect an I/O device to a computer?
- ▶ How does an I/O device communicate with a computer?

I/O Bus

I/O devices communicate with a computer via a connection point

- ▶ (Physical) port, e.g., USB port, serial port, parallel port
- ▶ I/O Bus (or Expansion Bus), e.g., PCI bus, SCSI bus

Typical PCB Bus



Source: Figure 12-1 in Silberschatz et al., 2018³

³Abraham Silberschatz, Peter B. Galvin, and Greg Gagne. *Operating system concepts*. 10th edition. John Wiley & Sons, 2018.

Device and Device Controller

An I/O device typically packages two major components.

- ▶ (Mechanical) Device, e.g., hard disk drives have motors, magnetic headers, and disks
- ▶ Controller, a collection of electronics that operate a port, a bus, or a device (some contain small embedded computer), e.g., a SATA controller, a USB controller
- ▶ CPU communicates with the device via the controller
 - ▶ Accept and act on commands from the OS
 - ▶ Present a simpler interface to the OS

Discussion: How to do I/O

Absent an OS or in an OS, how do we write a program to read or write to an I/O device?

Device Controller Memories

- ▶ Typically have 4 registers or more
 - ▶ Data-in register. Read by the CPU
 - ▶ Data-out register. Written by the CPU.
 - ▶ Status register. Ready by the CPU, a number of bits indicating the status of the device (e.g., busy, error)
 - ▶ Control register. Written by the CPU, a number of bits indicating the mode of the device
- ▶ May have a data buffer, e.g., a video adapter (video memory)

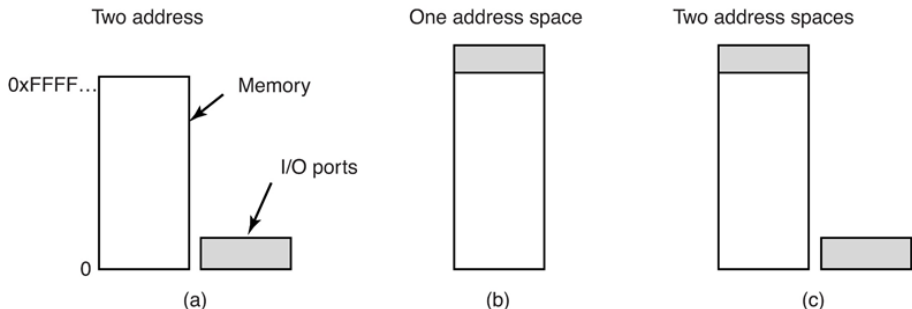
Device I/O is essentially to read or write to these memories in the device controller.

- ▶ How do we address the device (controller) memories?
- ▶ In what programming pattern do we write to or read from the memories?

Addressing Device (Controller) Memories

- ▶ Port-mapped I/O
- ▶ Memory-mapped I/O
- ▶ Hybrid of the two

Addressing Device (Controller) Memories



(a) Port-mapped I/O; (b) Memory-mapped I/O; (c) Hybrid (Figure 5-2 in Tanenbaum and Bos, 2014⁴)

⁴Andrew S. Tanenbaum and Herbert Bos. *Modern Operating Systems*. 4th. USA: Prentice Hall Press, 2014. ISBN: 013359162X.

Example I/O Port Allocation on PCs

Take a look at the VM

Hybrid Scheme

- ▶ Memory-mapped I/O (addressing) for data buffer, i.e., data buffers are mapped to memory address
- ▶ Port-mapped I/O (addressing) for control registers, i.e., control registers have dedicated I/O ports

Port-Mapped I/O vs. Memory-Mapped I/O Addressing

- ▶ Memory-mapped I/O (addressing) is easier to program, easier to protect, faster to access (addressing it as if it were main memory and do not require special instructions)
- ▶ Port-mapped I/O typically requires special instructions, like `in`, `out` in x86 instruction set.
- ▶ However, memory-mapped (addressing) is more complex to design cache, more complex to design bus as the two types addresses logically identical, but physically different

I/O Schemes

In what programming pattern do we write to or read from the memories?

- ▶ Busy waiting (polling)
while (busy) wait; do I/O;
- ▶ Interrupted I/O
do something else; when (interrupted) do I/O;
- ▶ Direct memory access (DMA)
initialize DMA; do something else; I/O done when interrupted;

Implementing Polling

- ▶ CPU and Device Controller work together

- ▶ CPU does

do

 read the busy-bit in the device status register

while (busy)

 set the write-bit in the control register

 write a byte into the data-out register

 set the command-ready bit in the control register

- ▶ Device controller does

do

 read the command-ready bit

while (not set)

 set the busy bit

 read the byte in the data-out register

 write the byte to the device

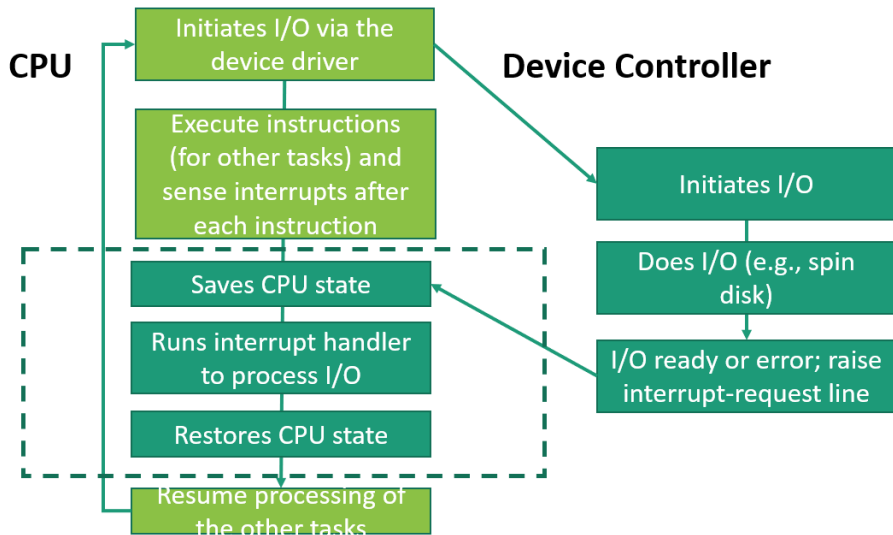
 if (success) clear the command-ready bit and the busy bit

 else set the error bit

(CPU) Interrupt

- ▶ Interrupt transfers control *asynchronously* to the interrupt service routine
- ▶ Two sources of interrupts
 - ▶ External (hardware-generated) interrupts: interrupts are generally caused by hardware
 - ▶ Software generated interrupts: a trap or exception is a software-generated interrupt caused either by an error or a user request
- ▶ Related concepts
 - ▶ Interrupt vector (interrupt descriptor by Intel)
 - ▶ Interrupt service routine: interrupt handler, a program processes the interrupt
 - ▶ Interrupt vector table: consists of interrupt vectors
 - ▶ Interrupt vector: the address of an interrupt handler
 - ▶ Interrupt architecture must save the address of the interrupted instruction

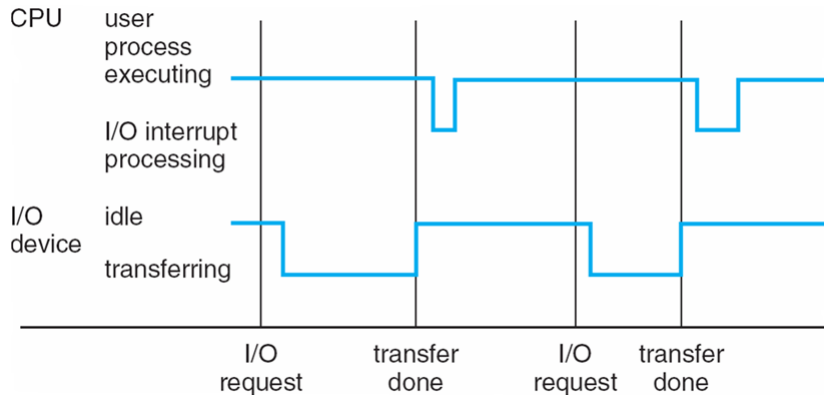
Implementing Interrupted I/O



Polling vs. Interrupted I/O

What are the advantages and disadvantages?

I/O Interrupt Timeline



Source: Figure 1.3 in Silberschatz et al., 2018⁵

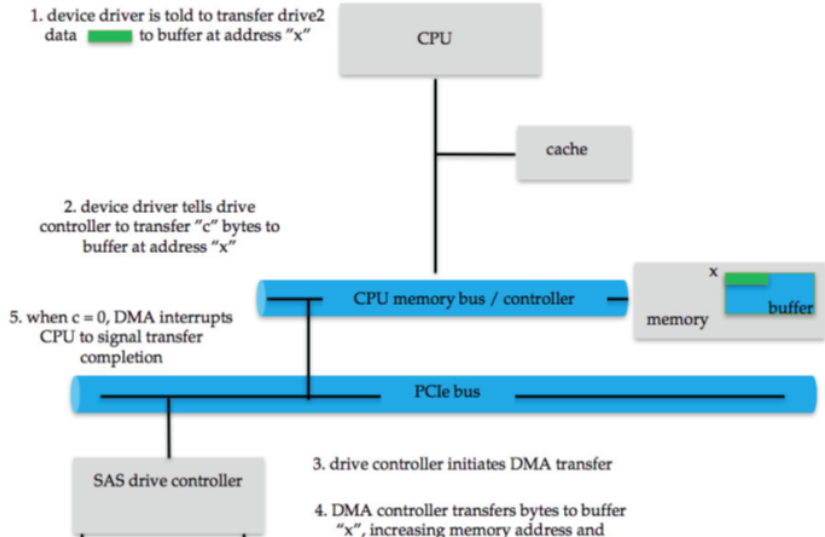
⁵Abraham Silberschatz, Peter B. Galvin, and Greg Gagne. *Operating system concepts*. 10th edition. John Wiley & Sons, 2018.

Implementing DMA

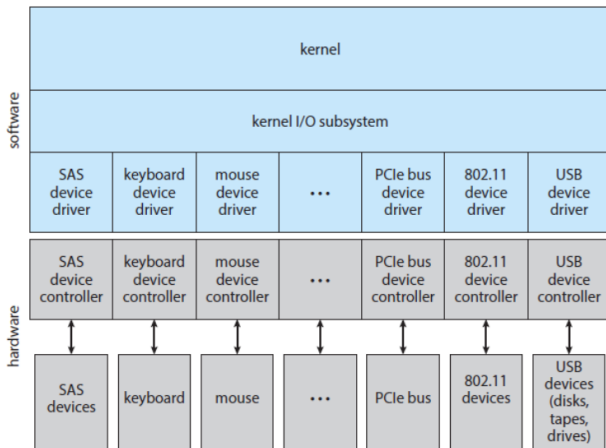
Aided by a special purpose processor called direct-memory-access (DMA) controller

- ▶ CPU writes a DMA command block into memory
 - ▶ Pointer to the source of transfer
 - ▶ Pointer to the destination of transfer
 - ▶ A count of the number of bytes to be transferred
- ▶ CPU writes the address of this block to the DMA controller
- ▶ The DMA controller does I/O by directly access devices and system bus
- ▶ CPU is interrupted when the DMA controller completes the transfer or encounters an error

Implementing DMA



Overview of OS I/O Software



Source: Figure 12.7 in Silberschatz et al., 2018⁷

⁷Abraham Silberschatz, Peter B. Galvin, and Greg Gagne. *Operating system concepts*. 10th edition. John Wiley & Sons, 2018.

Device Independency?

- ▶ To achieve device independent, categorize devices (device types) based on general characteristics.
- ▶ A few factors:
 - ▶ Size of transfer: Character-stream or block
 - ▶ Access order: sequential or random access
 - ▶ Predictability and responsiveness: Synchronous and asynchronous
 - ▶ Shared or dedicated
 - ▶ Speed of operation, e.g., latency, seek time, transfer rate
 - ▶ Read-write, read only, or write only

Device Type Examples

Block device vs. character device

Block Devices

- ▶ Read and write a block a time
- ▶ Essential behavior: `read()`, `write()`, and `seek()` for random-access block devices

Character Devices

- ▶ Read and write a character a time
- ▶ Essential behavior: `get()`, `put()`

Device Driver

Reduce complexity, increase uniformity and reliability

- ▶ OS provides an abstraction (service) for the essential behavior of the device.
- ▶ Device driver implements the logic for the abstraction
- ▶ User programs communicate to OS, device driver, and then the device controller

Lab. Implementing Device Driver

Implement a simple character device driver as a kernel module on Linux.