Operating Systems

Input/Output and Disks

Thomas Ropars

thomas.ropars@imag.fr

Équipe ERODS – LIG/IM2AG/UJF

2015

References

The content of these lectures is inspired by:

- The lecture notes of Prof. David Mazières.
- *Operating Systems: Three Easy Pieces* by R. Arpaci-Dusseau and A. Arpaci-Dusseau

Other references:

- *Modern Operating Systems* by A. Tanenbaum
- *Operating System Concepts* by A. Silberschatz et al.

Agenda

Introduction

Interacting with an I/O device

Drivers

Basic Geometry of a disk

Scheduling disk I/O

Agenda

Introduction

Interacting with an I/O device

Drivers

Basic Geometry of a disk

Scheduling disk I/O

I/O: an important topic

Motivation

- Without I/O, computing is useless.
- It is the main purpose of most programs. (eg, editing a file, browsing a web page)

All kind of I/O devices

- mouse/keyboard
- disk/cdrom/usb stick
- network card
- screen/printer

A hardware/software infrastructure is required to interact with all these devices.

The I/O Bus

A bus is a communication system interconnecting several devices.

A hierarchical architecture

- A general I/O bus (PCI).
  - Connects the processor-memory subsystem to higher performance devices (video card, network card, etc.)
- One or several peripheral buses to connect other devices (USB, SATA)
  - Connects to disks, keyboard/mouse, etc.

Why hierarchical?

The I/O Bus

A bus is a communication system interconnecting several devices.

A hierarchical architecture

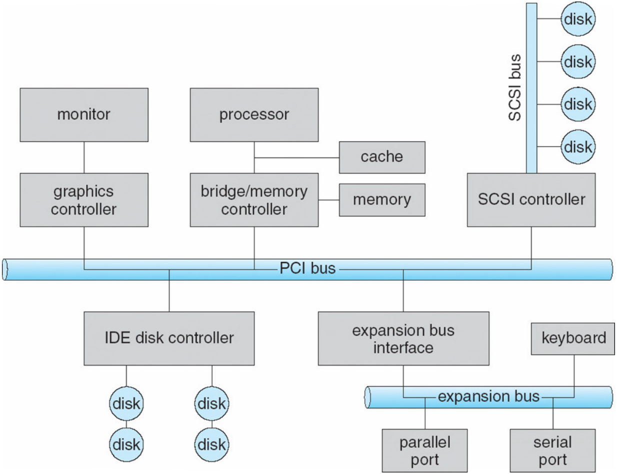
- A general I/O bus (PCI).
  - Connects the processor-memory subsystem to higher performance devices (video card, network card, etc.)
- One or several peripheral buses to connect other devices (USB, SATA)
  - Connects to disks, keyboard/mouse, etc.

Why hierarchical?

- Performance: performance decreases with the length of the bus
- Cost: designing a highly efficient bus is costly (and not useful to all devices)

The I/O Bus

Figure by Silberschatz et al



Controller= collection of electronics that operates a bus or a peripheral device

Agenda

Introduction

Interacting with an I/O device

Drivers

Basic Geometry of a disk

Scheduling disk I/O

A canonical device

Registers | Status | Command | Data

Micro-controller (CPU)
Memory
Specific hardware chips

Device hardware interface

The processor can access a set of registers:

- Status: Read to get current device status
- Command: Write to tell the device to perform a task
- Data: Read or write data

Polling

Sequence of actions (to be repeated for each byte)

1. The OS repeatedly reads the status register until it's not *BUSY*.
2. The OS writes a byte into the data register.
3. The OS sets the command register.
4. When the controller notices that a command is set, it sets its status to *BUSY*.
5. The OS repeatedly reads the status register to know when the command has been executed.
6. The controller reads the command register and the data register, and executes the command to the device.
7. The controller clears the command and resets its *BUSY* status once the command has been executed successfully. It can set its status to *ERROR* in case an error occurred.

Polling

Drawbacks

Polling

Drawbacks

- Wastes CPU cycles – especially when the device takes time to execute the operation.
- Hard to schedule polling in the future.

Polling

Drawbacks

- Wastes CPU cycles – especially when the device takes time to execute the operation.
- Hard to schedule polling in the future.

Advantages

- Efficient if the device is ready very rapidly
- (Only a few cycles are needed for one polling)

Programmed I/O

When the main processor is involved in the data movement related to I/O, it is called Programmed I/O (PIO).

Execution with polling

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

Interrupts

Execution with polling

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

Execution with interrupts

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

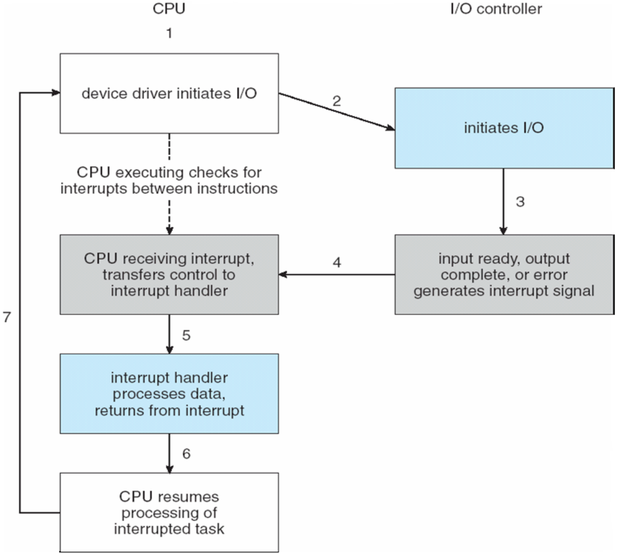
- Using interrupts allow putting process 1 to *sleep* until the IO is completed.
- The scheduler can schedule another process on the CPU.

How do interrupts work?

- The controller raises an interrupt
  - The CPU hardware has a wire called the interrupt-request line
  - The CPU senses it after executing every instruction
- The CPU catches the interrupt and dispatches it to the interrupt handler
  - The CPU performs a state save and jumps to the interrupt handler routine at a fixed address in memory.
- The handler clears the interrupt by servicing the device
  - The interrupt handler determines the cause of the interrupt and performs the necessary processing
  - It then performs a state restore, and executes a return from interrupt instruction to return the CPU to the execution state prior to the interrupt.

How do interrupts work?

Figure by Silberschatz et al



How to select the proper interrupt handler?

Basic solution

Check all devices to find which one is ready.

How to select the proper interrupt handler?

Basic solution

Check all devices to find which one is ready.

- Problem: there can be many devices to check.

How to select the proper interrupt handler?

Basic solution

Check all devices to find which one is ready.

- Problem: there can be many devices to check.

Interrupt dispatching

The interrupt accepts an integer as input.

- It is an offset in a table called the interrupt vector
  - Each entry in the vector contains a pointer to an interrupt handler
- Problem: The host might include more devices than the number of entries in the vector

How to select the proper interrupt handler?

Basic solution

Check all devices to find which one is ready.

- Problem: there can be many devices to check.

Interrupt dispatching

The interrupt accepts an integer as input.

- It is an offset in a table called the interrupt vector
  - Each entry in the vector contains a pointer to an interrupt handler
- Problem: The host might include more devices than the number of entries in the vector
  - Use interrupt chaining (ie, each entry points to a list of handlers)

More on interrupts

Masking and priorities

- Some interrupts are maskable (handling can be deferred), some are not (eg, errors).
- Priorities between interrupts can be defined
  - A high-priority interrupt can preempt the execution of a low-priority interrupt

Interrupts are not always better than polling

Hybrid approach

- Handling an interrupt is costly (hundreds of cycles)
- What if the device is ready almost immediately?
- Hybrid approach: The best of both world
  - Start by polling
  - If the device is not ready, put calling process to wait and schedule another process

Interrupts are not always better than polling

Livelock

- The processor receives so many interrupts that it only processes interrupts and never allows a user-level process to run
  - Problem with too many messages received on a network interface.
- Better use polling
- Interrupt coalescing: wait before sending interrupts until several requests have been completed

Improving data transfer performance

Execution with interrupts and PIO

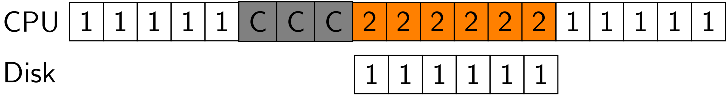
(For a single word)

CPU	1	1	1	1	1	C	C	C	2	2	2	2	2	2	1	1	1	1	1
Disk									1	1	1	1	1	1					

Improving data transfer performance

Execution with interrupts and PIO

(For a single word)



Problem

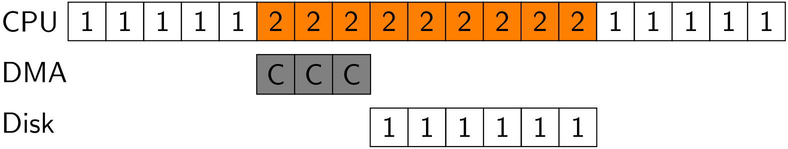
- The processor wastes CPU cycles for every word
- What if a large amount of data has to be output to the device?

Direct Memory Access (DMA)

Direct Memory Access engine

A DMA engine is a specific device that orchestrate data transfer between memory and I/O devices without CPU intervention.

- The OS writes a command to the DMA engine with the source address, the destination address and the amount of data to transfer.
- The DMA engine sends an interrupt to the CPU when the transfer is done.



Interacting with a device

How does the OS actually communicates with a device?

I/O instructions

- Specific instructions (*in* and *out* on x86)
- Allow to send data to specific device registers

Memory-mapped I/O

- The device-control registers are mapped into the address space of the processor.
- The processor can issue reads and writes to those specific addresses.

Agenda

Introduction

Interacting with an I/O device

Drivers

Basic Geometry of a disk

Scheduling disk I/O

The problem

Context

- We would like the OS to be as general as possible (work on any hardware)
- Each device can have a very specific interface

An example: a file system

We would like to open a file but it could be stored on different I/0 devices:

- A disk (different kinds)
- A USB stick
- A CD

Drivers

Keywords

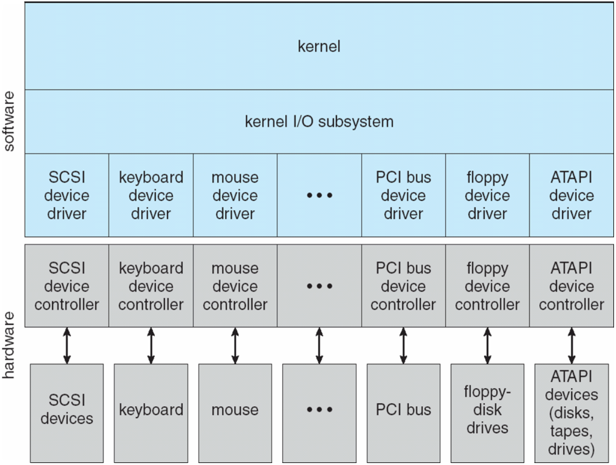
- Abstraction
- Encapsulation
- Software layering

A piece of software must know in detail how a device works: this is the Device Driver.

- The driver exposes a generic interface to the rest of the OS.
- Any new device should come with a driver that implements (at least part of) the standard I/0 interface to be usable.

Drivers

Figure by Silberschatz et al



About drivers

Drawbacks

- The generic approach might prevent from taking advantage of advanced features of the hardware
- Example: SCSI devices provide reach error reporting. The Linux I/O interface only reports generic I/O errors.

In the kernel

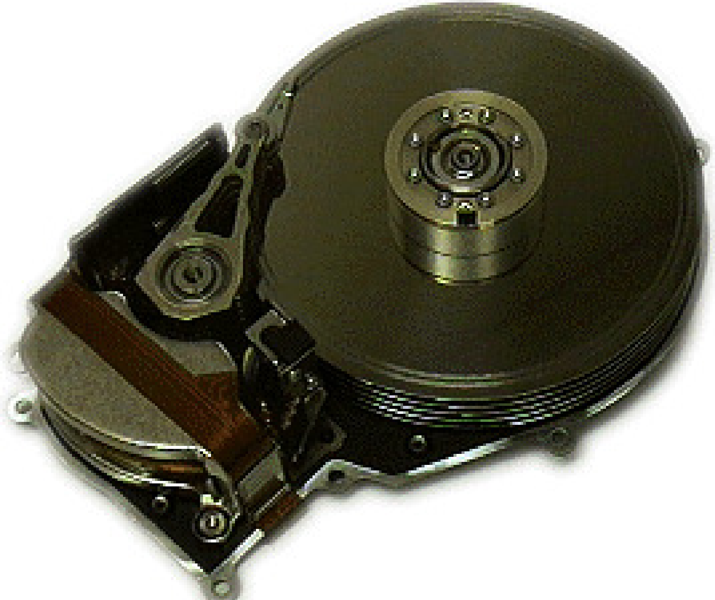
- In 2001, drivers were accounting for 70% of the kernel code
- Of course it is not all active at the same time
- Many bugs are in the drivers

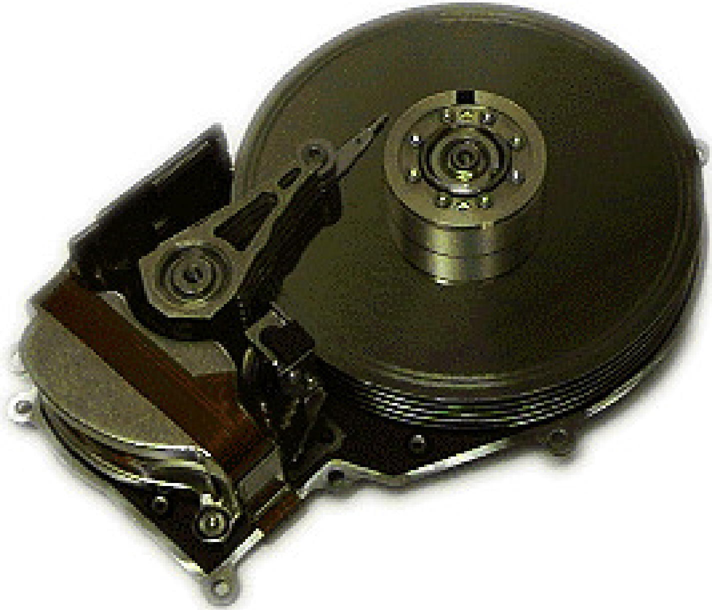
Example: an IDE disk driver

The full example is to be read from *Operating Systems: Three Easy Pieces* (chapter 36)

- 4 types of registers: Control, Command block, Status, Error
- accessed using *in* and *out* instructions on x86.
- Tasks of the driver:
  - Wait for the drive to be ready
  - Write parameters to command register
  - Start the I/O (write READ or WRITE to the command register)
  - Data transfer (wait for DRQ status – drive request for data – and write data to data port)
  - Handle interrupts
  - Error handling

The case of Hard Disk Drives





Agenda

Introduction

Interacting with an I/O device

Drivers

Basic Geometry of a disk

Scheduling disk I/O

Storage on a magnetic platter

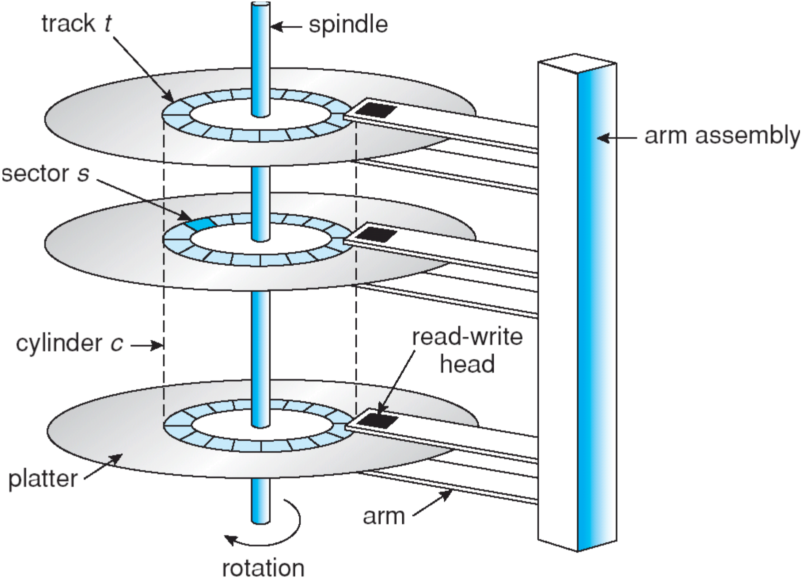
- Platter: a circular hard surface on which data is stored persistently by inducing magnetic changes to it.
  - A disk may have one or multiple platters.
- Surface: One side of a platter
  - Data is encoded on each surface
- Tracks: A surface divided into concentric tracks.
  - Many thousands of tracks on a surface

Storage on a magnetic platter

- Head/Arm: Reading or writing is accomplished by a disk head attached to a disk arm.
  - One head per surface
  - Heads record and sense data along cylinders
  - Generally only one head is active at a time
- Sector: A cylinder into 512-byte blocks called sectors
  - Sectors are numbered from 0 to $n-1$ (n-sector disk)
  - Multi-sectors operations are possible (eg, update 4 Mb at a time)
  - A sector is the granularity for atomic operations.

Cylinders, tracks, & sectors

Figure by Silberschatz et al



Accessing some sectors: Seeks

A seek is the action of moving the head from its current track to the track containing the target sector.

4 phases

- Acceleration: accelerate arm to max speed or half way point
- Coasting: move at max speed (for long seeks)
- Slowdown: stops arm near destination
- Settle: adjusts head to actual desired track
  - Is a costly operation (0.5 to 2 ms)
  - The drive must be certain to find the right track!

Accessing some sectors

Other times

- Rotational delay: Time for the target sector to pass under the disk head.
  - Rotating speed of modern disks: 7,200 RPM to 15,000 RPM (RPM= rotations per minute)
- Transfer time: Time for I/O to take place.

$$\text{I/O Time} = \text{Seek time} + \text{Rotational delay} + \text{Transfer time}$$

About performance

Comments about performance

- Accessing sectors that are close is faster
- Accessing contiguous sectors is faster than random access

Cache

Disks may use a cache to improve observed performance

- Read and cache consecutive sectors
- Caching writes can be dangerous (break atomicity)

Agenda

Introduction

Interacting with an I/O device

Drivers

Basic Geometry of a disk

Scheduling disk I/O

Context

The OS should decide in which order to execute I/O on the disk to optimize performance

- Contiguous accesses are better
- Try to avoid long seeks.

Difference with process scheduling

- It is possible to estimate seek time and rotational delay (the future).
- A strategy similar to SJF can be applied!

First Come First Served (FCFS)

Process disk requests in the order they are received

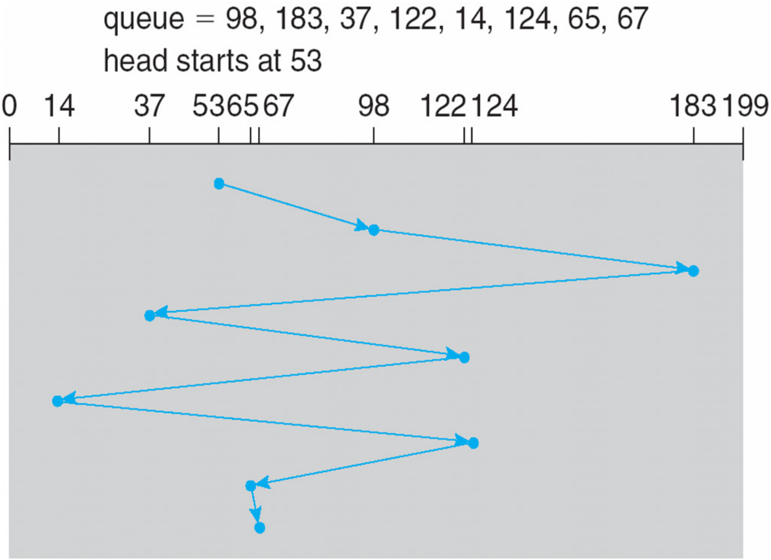
Advantages

- Easy to implement
- Good fairness

Disadvantages

- Cannot exploit request locality
- Increases average latency, decreasing throughput

FCFS example



Shortest seek time first (SSTF)

Always pick request with shortest seek time

Advantages

- Exploits locality of disk requests
- Higher throughput

Disadvantages

Shortest seek time first (SSTF)

Always pick request with shortest seek time

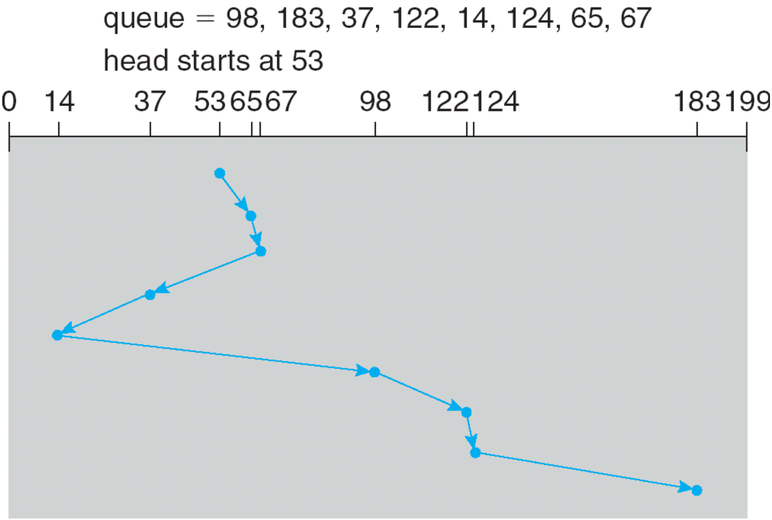
Advantages

- Exploits locality of disk requests
- Higher throughput

Disadvantages

- Starvation (some aging strategy could be used to fix the problem)
- Don't always know what request will be fastest

SSTF example



"Elevator" scheduling (SCAN)

Sweep across disk, servicing all requests passed

- Like SSTF, but next seek must be in same direction
- Different variants:
  - Switch directions only if no further requests (SCAN)
  - Back to first track when no further requests (Circular-SCAN)

Advantages

- Takes advantage of locality
- Bounded waiting

Disadvantages

"Elevator" scheduling (SCAN)

Sweep across disk, servicing all requests passed

- Like SSTF, but next seek must be in same direction
- Different variants:
  - Switch directions only if no further requests (SCAN)
  - Back to first track when no further requests (Circular-SCAN)

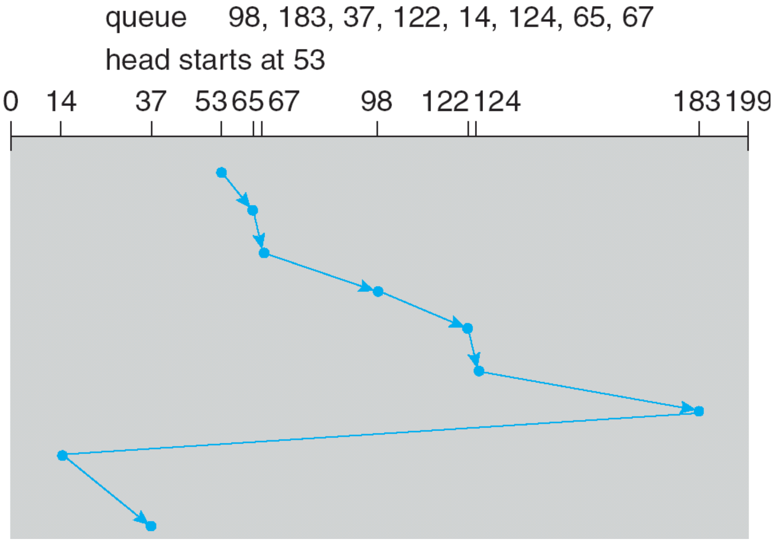
Advantages

- Takes advantage of locality
- Bounded waiting

Disadvantages

- Might miss locality SSTF could exploit

CSCAN example



More on scheduling

- Some strategies try to mix SSTF and SCAN
  - VSCAN(r): Apply SSTF but a *r* weight to account for the direction
- All presented strategies only take into seek time
  - Rotational delay might be as important as seek time
  - SPTF (Shortest Positioning Time First) tries do this
  - However rotational delay is hard to evaluate at the OS level

Scheduling with modern disks

Features of modern disks

- Disks can accommodate multiple outstanding requests
- Disks include sophisticated schedulers
  - They can implement SPTF accurately!
- Disks can also do I/O merging
  - Wait for multiple I/O requests to try to merge consecutive ones in a single multi-blocks request

Interactions with the OS

- The OS issues a few request (tries to select best from its point of view)
- The disk applies advanced scheduling to those requests

References for this lecture

- *Operating Systems: Three Easy Pieces* by R. Arpaci-Dusseau and A. Arpaci-Dusseau
  - Chapter 36: I/O devices
  - Chapter 37: Hard Disk Drives
- *Operating System Concepts* by A. Silberschatz et al.
  - Chapter 13: I/0 systems