Discussion 7

I/O

03/22/24

Staff

Announcements

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
	Spring Break	Spring Break	Spring Break	Spring Break	Spring Break	
	Homework 4 Due		Project 2 Due			

I/O

I/O

Access patterns

- **Character devices** access data as a character stream (i.e. data not addressable).
- **Block devices** access data in fixed-sized blocks (i.e. data is addressable).
- **Network devices** have a separate interface for networking purposes.

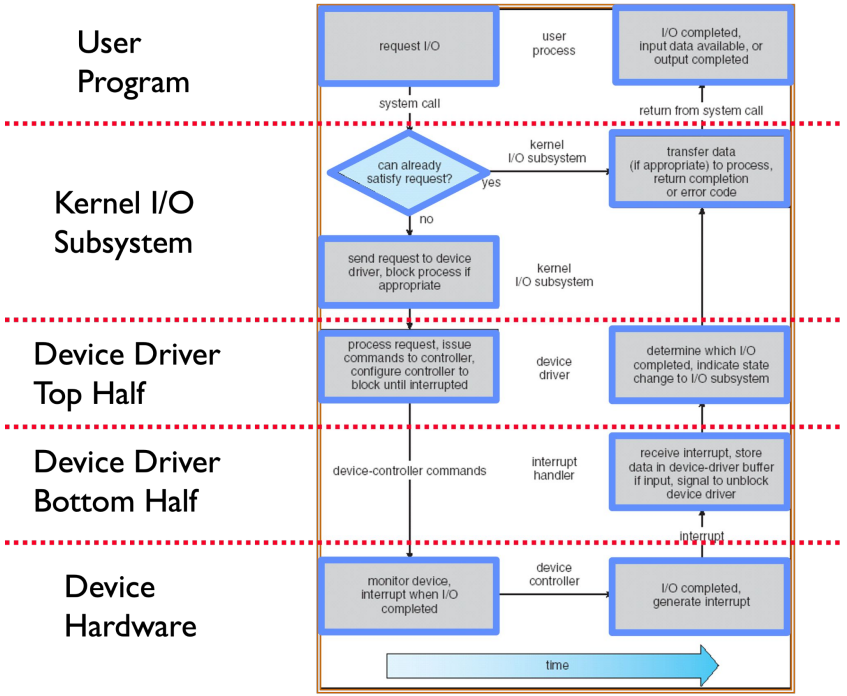
Access timing

- **Synchronous/blocking** interfaces wait until an I/O request is fulfilled.
- **Non-blocking** interfaces return quickly from a request without waiting.
- **Asynchronous** interfaces allow for other processing to continue without waiting.
- Asynchronous is not the same as non-blocking!

I/O

Device drivers connect high-level abstractions implemented by the OS and hardware specific details of I/O devices.

- Allows kernel to use a standard interface for I/O devices.
- **Top half** is used by the kernel to start I/O operations.
- **Bottom half** is used to service interrupts from the device.
- In Linux, top and bottom are flipped!



Device Access

Processors talk to I/O devices through **device controllers**

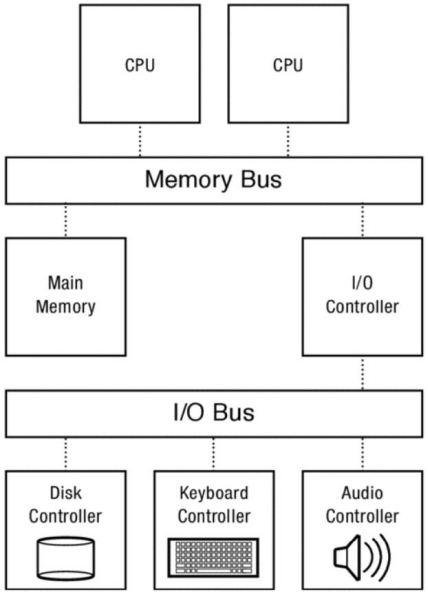
- Contain a set of registers that can be read from/written to.

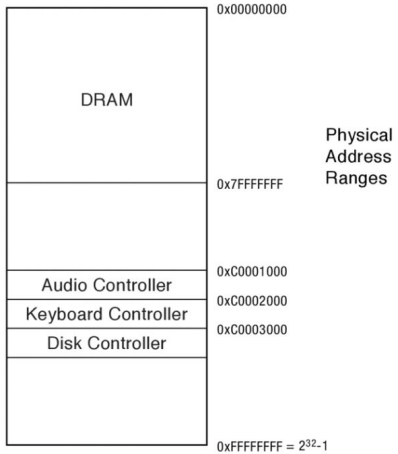
Programmed I/O (PIO) involves the processor in every byte transfer.

- **Port-mapped I/O (PMIO)**
  - Uses special memory instruction (e.g. in, out) → complicates CPU logic with special instructions.
  - Uses distinct memory address space from physical memory → useful on older/smaller devices with limited physical address space.
- **Memory-mapped I/O (MMIO)**
  - Uses standard memory instructions (e.g. load, store) → simplifies CPU logic by putting the responsibility on the device controller
  - Shares memory address with physical memory.

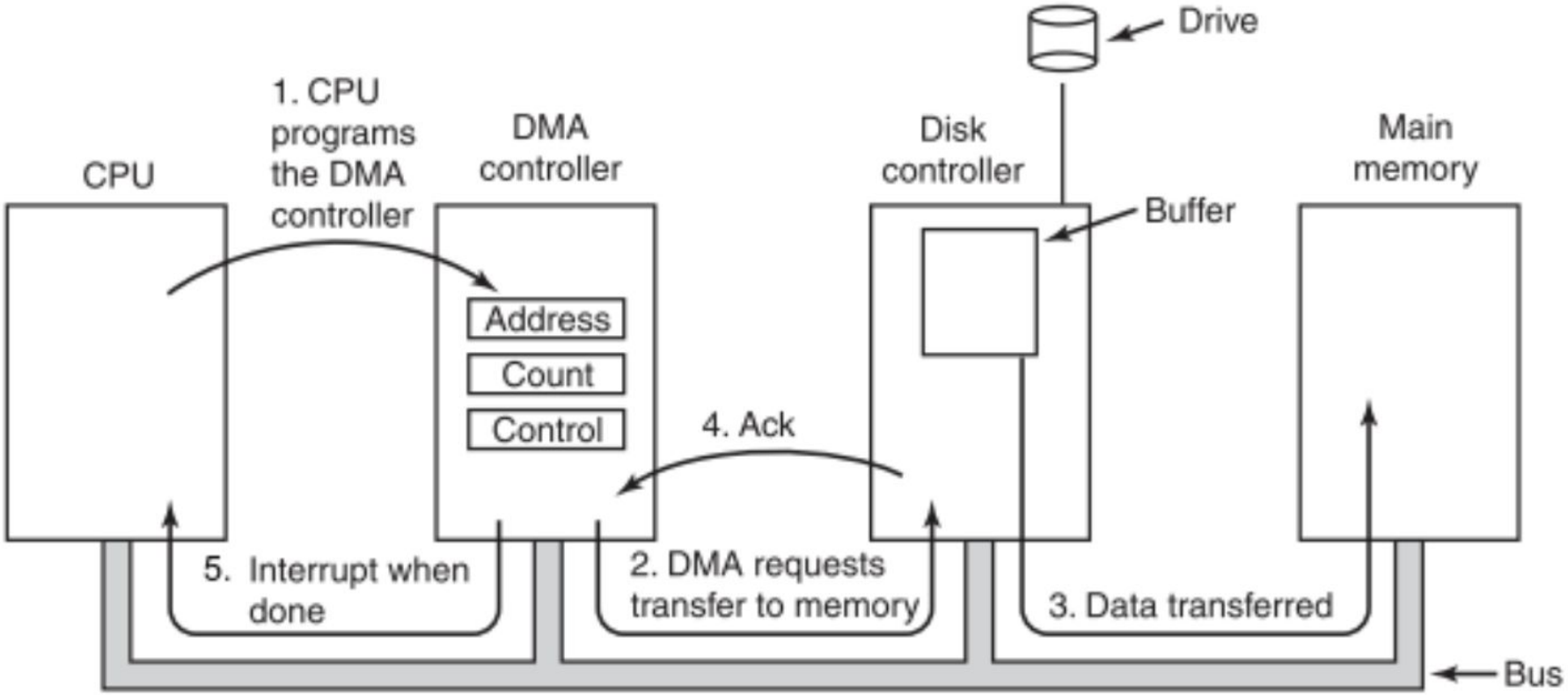
Direct memory access (DMA) allows device to write to main memory without CPU intervention.

- Uses memory addresses within physical memory region, not just the physical address space.





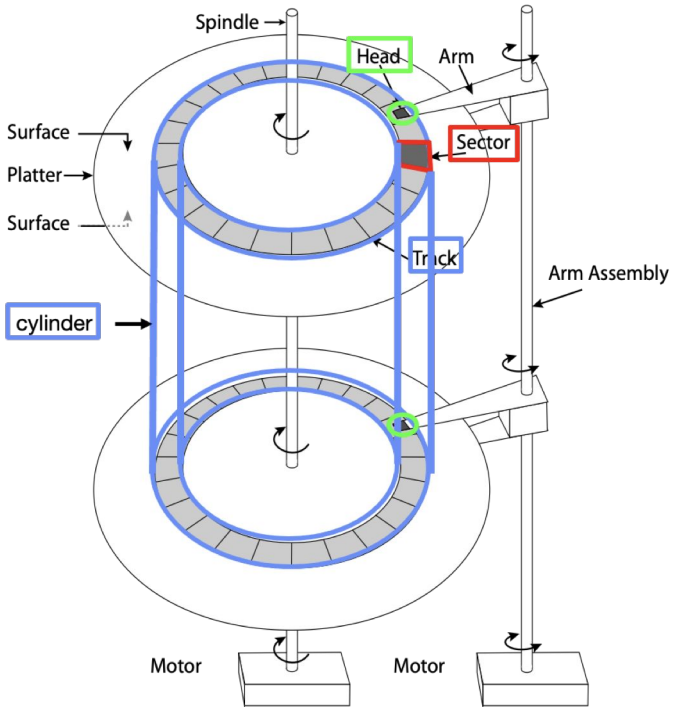
DMA Walkthrough



Storage Devices (HDD)

Magnetic disks use magnetization to read/write data.

- **Platter** is a single thin round plate that stores information.
- Each platter spins on a **spindle**, and each platter has two **surfaces**.
- Data is read and written with the **head**.
- Each head attaches to an **arm**, and each **arm** attaches to the **arm assembly.**
- Data is stored in fixed size units called **sectors**, the minimum unit of transfer.
- Circle of sectors make up a **track**.
- Track length varies from inside to outside of the disk.
- Lots of moving parts → physical repair issues.
- Read/write times slower compared to flash.
- Much cheaper than flash.



Storage Devices

Flash memory uses electrical circuits (e.g. NOR, NAND) for persistent storage.

- Focus on NAND flash which reads/writes in fixed size units called **pages** (typically 2 KB to 4 KB).
- Much less moving parts → less prone to issues.
- Read/write speeds much faster.
- Writing to a cell requires erasing it first, in large fixed size units called **erasure blocks**.
- Each page has a finite lifetime, can only be erased and rewritten a fixed number of times (e.g. 10K)

Flash translation layer (FTL) maps logical flash pages to different physical pages.

- Allows device to relocate data without the OS knowing/worrying.
- Can write data by writing to a new location, updating mapping in FTL, then erasing old page in the background.
- Ensure all physical pages are used equally using **wear levelling**.

Storage Devices

Time to access data is composed of

- **Queueing time** refers to how long a data request spends in the OS queue before being passed to the device controller.
- **Controller time** refers to how long the device controller spends processing the request.
- **Access time** refers to how long it takes to access data from the device.

Magnetic disks' access time involves

- **Seek time**: move the arm over the desired track.
- **Rotation time**: wait for target sector to rotate under head.
- **Transfer time**: transfer data to/from buffer for read/write
- Aim to minimize seek and rotation since transfer is fixed.

Use disk intelligence to improve performance.

- **Track skewing** staggers logical sectors of the same number on each track by the time it takes to move across one track.
- **Sector sparing** remaps bad sectors to spare sectors.
- Use **buffer memory** to store sectors read by disk head but not requested by the OS

Concept check

1. If a particular I/O device implements a blocking interface, do you need multiple threads to have concurrent operations which use that device?

2. For I/O devices which receive new data very frequently, is it more efficient to interrupt the CPU than to have the CPU poll the device?

3. When using SSDs, which between reading or writing data is complex and slow?

4. Why might you choose to use DMA instead of MMIO? Give a specific example where one is more appropriate than the other.

Concept check

1. If a particular I/O device implements a blocking interface, do you need multiple threads to have concurrent operations which use that device?
   Yes. Only with non-blocking IO can you have concurrency without multiple threads.

2. For I/O devices which receive new data very frequently, is it more efficient to interrupt the CPU than to have the CPU poll the device?

3. When using SSDs, which between reading or writing data is complex and slow?

4. Why might you choose to use DMA instead of MMIO? Give a specific example where one is more appropriate than the other.

Concept check

1. If a particular I/O device implements a blocking interface, do you need multiple threads to have concurrent operations which use that device?
   Yes. Only with non-blocking IO can you have concurrency without multiple threads.

2. For I/O devices which receive new data very frequently, is it more efficient to interrupt the CPU than to have the CPU poll the device?
   No. It is more efficient to poll, since the CPU will get overwhelmed with interrupts.

3. When using SSDs, which between reading or writing data is complex and slow?

4. Why might you choose to use DMA instead of MMIO? Give a specific example where one is more appropriate than the other.

Concept check

1. If a particular I/O device implements a blocking interface, do you need multiple threads to have concurrent operations which use that device?
   Yes. Only with non-blocking IO can you have concurrency without multiple threads.

2. For I/O devices which receive new data very frequently, is it more efficient to interrupt the CPU than to have the CPU poll the device?
   No. It is more efficient to poll, since the CPU will get overwhelmed with interrupts.

3. When using SSDs, which between reading or writing data is complex and slow?
   SSDs have complex and slower writes because their memory can't easily be mutated.

4. Why might you choose to use DMA instead of MMIO? Give a specific example where one is more appropriate than the other.

Concept check

1. If a particular I/O device implements a blocking interface, do you need multiple threads to have concurrent operations which use that device?
   Yes. Only with non-blocking IO can you have concurrency without multiple threads.

2. For I/O devices which receive new data very frequently, is it more efficient to interrupt the CPU than to have the CPU poll the device?
   No. It is more efficient to poll, since the CPU will get overwhelmed with interrupts.

3. When using SSDs, which between reading or writing data is complex and slow?
   SSDs have complex and slower writes because their memory can't easily be mutated.

4. Why might you choose to use DMA instead of MMIO? Give a specific example where one is more appropriate than the other.
   Transferring large amounts of data which take a long time to not involve the CPU.

Concept Check

5. Usually, the OS deals with bad or corrupted sectors. However, some disk controllers magically hide failing sectors and re-map to back-up locations on disk when a sector fails.
    a. If you had to choose where to lay out these “back-up” sectors on disk - where would you put them? Why?

    b. How do you think that the disk controller can check whether a sector has gone bad?

    c. Can you think of any drawbacks of hiding errors like this from the OS?

6. When writing data to disk, how can the buffer memory be used to increase the perceived write speed from the OS viewpoint?

Concept Check

5. Usually, the OS deals with bad or corrupted sectors. However, some disk controllers magically hide failing sectors and re-map to back-up locations on disk when a sector fails.
    a. If you had to choose where to lay out these “back-up” sectors on disk - where would you put them? Why?
    Should spread them out evenly, so when you replace an arbitrary sector your find one that is close by.

    b. How do you think that the disk controller can check whether a sector has gone bad?

    c. Can you think of any drawbacks of hiding errors like this from the OS?

6. When writing data to disk, how can the buffer memory be used to increase the perceived write speed from the OS viewpoint?

Concept Check

5. Usually, the OS deals with bad or corrupted sectors. However, some disk controllers magically hide failing sectors and re-map to back-up locations on disk when a sector fails.
    a. If you had to choose where to lay out these “back-up” sectors on disk - where would you put them? Why?
    Should spread them out evenly, so when you replace an arbitrary sector your find one that is close by.

    b. How do you think that the disk controller can check whether a sector has gone bad?
    Using a checksum, this can be efficiently checked in hardware during disk access.

    c. Can you think of any drawbacks of hiding errors like this from the OS?

6. When writing data to disk, how can the buffer memory be used to increase the perceived write speed from the OS viewpoint?

Concept Check

5. Usually, the OS deals with bad or corrupted sectors. However, some disk controllers magically hide failing sectors and re-map to back-up locations on disk when a sector fails.
    a. If you had to choose where to lay out these “back-up” sectors on disk - where would you put them? Why?
    Should spread them out evenly, so when you replace an arbitrary sector your find one that is close by.

    b. How do you think that the disk controller can check whether a sector has gone bad?
    Using a checksum, this can be efficiently checked in hardware during disk access.

    c. Can you think of any drawbacks of hiding errors like this from the OS?
    Excessive sector failures are warning signs that a disk is beginning to fail.

6. When writing data to disk, how can the buffer memory be used to increase the perceived write speed from the OS viewpoint?

Concept Check

5. Usually, the OS deals with bad or corrupted sectors. However, some disk controllers magically hide failing sectors and re-map to back-up locations on disk when a sector fails.
    a. If you had to choose where to lay out these “back-up” sectors on disk - where would you put them? Why?
    Should spread them out evenly, so when you replace an arbitrary sector your find one that is close by.

    b. How do you think that the disk controller can check whether a sector has gone bad?
    Using a checksum, this can be efficiently checked in hardware during disk access.

    c. Can you think of any drawbacks of hiding errors like this from the OS?
    Excessive sector failures are warning signs that a disk is beginning to fail.

6. When writing data to disk, how can the buffer memory be used to increase the perceived write speed from the OS viewpoint?
    Acknowledge to OS that write has completed after it’s written to buffer, not the platters. Write to platter in the background.

Hard Drive Performance

Assume we have a hard drive with the following specifications.

- An average seek time of 8 ms
- A rotational speed of 7200 revolutions per minute (RPM)
- A controller that can transfer data at a maximum rate of 50 MiB/s

We will ignore the effects of queueing delay for this problem.

1. What is the expected throughput of the hard drive when reading 4 KiB sectors from a random location on disk?

Hard Drive Performance

Assume we have a hard drive with the following specifications.

- An average seek time of 8 ms
- A rotational speed of 7200 revolutions per minute (RPM)
- A controller that can transfer data at a maximum rate of 50 MiB/s

We will ignore the effects of queueing delay for this problem.

1. What is the expected throughput of the hard drive when reading 4 KiB sectors from a random location on disk?

Break down into seek time, rotation time, transfer time.

Already given seek time of 8 ms.

Hard Drive Performance

Assume we have a hard drive with the following specifications.

- An average seek time of 8 ms
- A rotational speed of 7200 revolutions per minute (RPM)
- A controller that can transfer data at a maximum rate of 50 MiB/s

We will ignore the effects of queueing delay for this problem.

1. What is the expected throughput of the hard drive when reading 4 KiB sectors from a random location on disk?

Break down into seek time, rotation time, transfer time.

Already given seek time of 8 ms.

On average, disk must complete ½ revolution to wait for the correct sector to rotate under head. Rotation time is

$$1/2 \times 1 / 7200 \text{ RPM} \cong 4.17 \text{ ms}$$

Hard Drive Performance

Assume we have a hard drive with the following specifications.

- An average seek time of 8 ms
- A rotational speed of 7200 revolutions per minute (RPM)
- A controller that can transfer data at a maximum rate of 50 MiB/s

We will ignore the effects of queueing delay for this problem.

1. What is the expected throughput of the hard drive when reading 4 KiB sectors from a random location on disk?

Break down into seek time, rotation time, transfer time.

Already given seek time of 8 ms.

On average, disk must complete ½ revolution to wait for the correct sector to rotate under head. Rotation time is

$$1/2 \times 1 / 7200 \text{ RPM} \cong 4.17 \text{ ms}$$

Transfer time for 4 KiB is

$$4 \text{ KiB} \times 1 / 50 \text{ MiB} \cong 0.078125 \text{ ms}$$

Hard Drive Performance

Assume we have a hard drive with the following specifications.

- An average seek time of 8 ms
- A rotational speed of 7200 revolutions per minute (RPM)
- A controller that can transfer data at a maximum rate of 50 MiB/s

We will ignore the effects of queueing delay for this problem.

1. What is the expected throughput of the hard drive when reading 4 KiB sectors from a random location on disk?

Break down into seek time, rotation time, transfer time.

Already given seek time of 8 ms.

On average, disk must complete ½ revolution to wait for the correct sector to rotate under head. Rotation time is

$$1/2 \times 1 / 7200 \text{ RPM} \cong 4.17 \text{ ms}$$

Transfer time for 4 KiB is

$$4 \text{ KiB} \times 1 / 50 \text{ MiB} \cong 0.078125 \text{ ms}$$

Total time is 8 ms + 4.17 ms + 0.078125 ms ≅ 12.248 ms, giving a throughput of

$$4 \text{ KiB} / 12.248 \text{ ms} \cong 326.6 \text{ KiB/s}$$

Hard Drive Performance

Assume we have a hard drive with the following specifications.

- An average seek time of 8 ms
- A rotational speed of 7200 revolutions per minute (RPM)
- A controller that can transfer data at a maximum rate of 50 MiB/s

We will ignore the effects of queueing delay for this problem.

2. What is the expected throughput of the hard drive when reading 4 KiB sectors from the same track on disk (i.e. the read/write head is already positioned over the correct track when the operation starts)?

3. What is the expected throughput of the hard drive when reading the very next 4 KiB sector (i.e. the read/write head is immediately over the proper track and sector at the start of the operation)?

Hard Drive Performance

Assume we have a hard drive with the following specifications.

- An average seek time of 8 ms
- A rotational speed of 7200 revolutions per minute (RPM)
- A controller that can transfer data at a maximum rate of 50 MiB/s

We will ignore the effects of queueing delay for this problem.

2. What is the expected throughput of the hard drive when reading 4 KiB sectors from the same track on disk (i.e. the read/write head is already positioned over the correct track when the operation starts)?

Ignore seek time, giving a total time of 4.17 ms + 0.078125 ms ≅ 4.24 ms and throughput of

4 KiB / 4.24 ms ≅ 943.4 KiB/s

3. What is the expected throughput of the hard drive when reading the very next 4 KiB sector (i.e. the read/write head is immediately over the proper track and sector at the start of the operation)?

Hard Drive Performance

Assume we have a hard drive with the following specifications.

- An average seek time of 8 ms
- A rotational speed of 7200 revolutions per minute (RPM)
- A controller that can transfer data at a maximum rate of 50 MiB/s

We will ignore the effects of queueing delay for this problem.

2. What is the expected throughput of the hard drive when reading 4 KiB sectors from the same track on disk (i.e. the read/write head is already positioned over the correct track when the operation starts)?

Ignore seek time, giving a total time of 4.17 ms + 0.078125 ms ≅ 4.24 ms and throughput of

$$4 \text{ KiB} / 4.24 \text{ ms} \cong 943.4 \text{ KiB/s}$$

3. What is the expected throughput of the hard drive when reading the very next 4 KiB sector (i.e. the read/write head is immediately over the proper track and sector at the start of the operation)?

Ignore both rotational and seek times to take full advantage of the transfer rate of 50 MiB/s.