Mass Storage and I/O

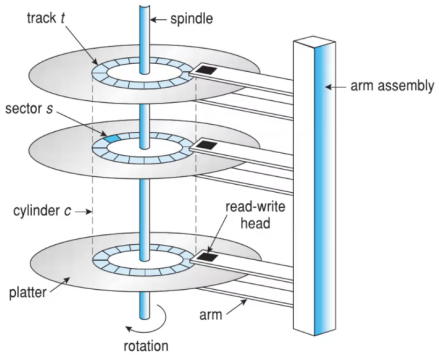
> ***Premessa***
>
> *La lezione **Mass Storage and I/O** è abbastanza semplice e di carattere generale, inoltre non stavo bene quando l'ho seguita, dunque non ho preso particolarmente appunti dato che le slides sono molto semplici, autoesplicative e sufficienti per capire i concetti spiegati.*

Overview of Mass Storage Structure

- The main mass-storage system in modern computers is secondary storage, which is usually provided by hard disk drivers and nonvolatile memory devices.
- The secondary storage for modern computers is **hard disk drives** (HDDs) and **nonvolatile memory** (NVM) devices.
- HDDs spin platters of magnetically-coated material under moving read-write heads.
  - Drives rotate at 60 to 250 times per second
  - Transfer rate is the rate at which data flow between the drive and the computer
  - Positioning time (random-access time) is the time to move the disk arm to the desired cylinder (seek time) and the time for the desired sector to rotate under the disk head (rotational latency)
  - Head crash results from disk head making contact with the disk surface -- That's bad
- Disks can be removable

Hard Disk Drives

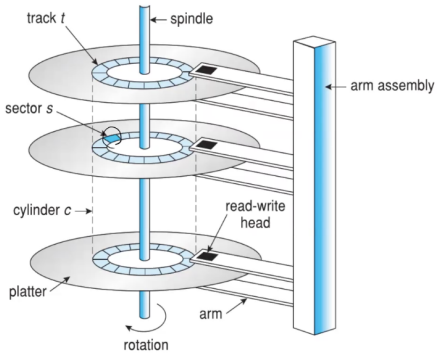
- HDDs are disk platters, with the two surfaces covered with a magnetic material.
- Storing information by recording it magnetically on the platters and reading information by detecting the magnetic pattern on the platters.
  - The surface of the platter is divided into circular **tracks**, which are divided into **sectors**.
  - The set of tracks at a given arm position is called a **cylinder**.



Hard Disk Drives

- HDDs are disk platters, with the two surfaces covered with a magnetic material.
- Storing information by recording it magnetically on the platters, and reading information by detecting the magnetic pattern on the platters.

- The **transfer rate** is the rate at which data flow between the driver and the computer.
- The **positioning time** or the **random-access time** consists of two times:
- The **seek time**, the necessity to move the disk arm to the desired cylinder
- The **rotational latency**, the time necessary for the desired sector to rotate to the disk head. Average latency = ½ latency
- Drivers rotate at 60 to 250 times per second.



Hard Disk Performance

- **Access Latency = Average access time** = average seek time + average latency
  - For fastest disk 3ms + 2ms = 5ms
  - For slow disk 9ms + 5.56ms = 14.56ms
- Average I/O time = average access time + (amount to transfer / transfer rate) + controller overhead

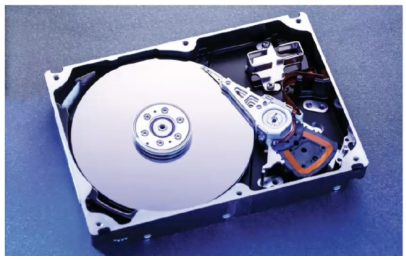
Tempo necessario per un'operazione di I/O dipende non solo dall'Avarage access time, ma anche dal tempo che impieghiamo a spostare i dati dal disco alla CPU. Vediamo quindi che sono coinvolti anche altri due tempi:

- `amount to transfer / transfer rate` → Rapporto tra *quanti dati devo trasferire e velocità di trasferimento*
- `controller overhead` → I dispositivi di I/O non possono parlare direttamente con la CPU. Il controller è un un circuito integrato che permette al dispositivo di parlare con il SO. Il *controller overhead* è il tempo che impiega il controller a gestire le operazioni.

Inoltre, poichè il settore che dobbiamo leggere può essere "appena passato", e quindi impiegare un giro completo prima di essere letto, oppure può essere che il primo settore che sta passando sia proprio quello che dobbiamo leggere, poniamo come ***Avarage latency = latency / 2.***

Hard Disks

- Platters range from .85" to 14" (historically)
  - Commonly 3.5", 2.5", and 1.8"
- Range from 30GB to 10TB per drive
- Performance
  - **Transfer Rate – theoretical – 6 Gb/sec**
  - Effective Transfer Rate – real – 1Gb/sec
  - **Seek time from 3ms to 12ms** – 9ms common for desktop drives
  - Average seek time measured or calculated based on 1/3 of tracks
  - Latency based on spindle speed
    - 1 / (RPM / 60) = 60 / RPM
  - Average latency = ½ latency



Hard Disk Performance

- For example, to transfer a **4KB block** on a **7200 RPM** disk with a **5ms** average **seek time**, 1Gb/sec transfer rate with a .1ms controller overhead =
  - 5ms + 4.17ms + 0.1ms + transfer time
  - Time per rotation = 60 seconds / 7200 RP = 8.333 ms
  - Transfer time = (1Gb/sec) / 0.032768 = 0.031 ms
  - 4KB = 4*8*1024 = 32768bit = 0.032768Gb
  - Transfer time = 4KB / 1Gb/s * 8Gb / GB * 1GB / 1024^2KB = 32 / (1024^2) = 0.031 ms
  - Average I/O time for 4KB block = 9.27ms + .031ms = 9.301ms

Nonvolatile Memory Devices

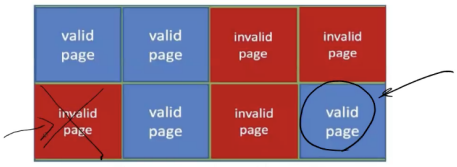
- Nonvolatile Memory (NVM) devices are electrical rather than mechanical.
- NVM is used in a disk-drive-like container called a **solid-state disk** (SSD)
  - NVM can be **more reliable** than HDDs because they have no moving part
  - They can be faster because they have no seek time or rotational latency.
  - They consume less power.
  - More expensive per MB
  - Maybe have a shorter life span – need careful management
  - Less capacity
  - But much faster
  - Over time, NVM devices' capacity has increased faster than HDD capacity and their price dropped so their use is increasing dramatically.

Nonvolatile Memory Devices

- **Challenges:**
- They can be Read and written in “page” increments (think sector) but can’t be overwritten in place
  - The NAND cells have to be erased first
  - The erasure which occurs in a block increment that is several pages in size, take much more time than a read or write.
  - Can only be erased a limited number of times before worn out – ~ 100,000
  - Life span measured in **Drive Writes Per Day (DWPD)** that measures how many times the driver capacity can be written per day before it fails.
    - A 1TB NAND drive with a rating of 5DWPD is expected to have 5TB per day written within the warrantee period without failing

NAND Flash Controller Algorithms

- Since NAND semiconductors cannot be overwritten once written, they are usually pages containing invalid data.
- With no overwrite, pages end up with a mix of valid and invalid data.
- To track which logical blocks are valid, the controller maintains a **flash translation layer (FTL)** table
- This table maps which physical pages contain currently valid logical blocks.
- Also implements **garbage collection** to free invalid page space (valid data could be copied to other locations, freeing up blocks that could be erased and receive the write.



NAND block with valid and invalid pages

Volatile Memory

- DRAM frequently used as a mass-storage device
  - Not technically secondary storage because volatile, but can have file systems, and be used like very fast secondary storage
- **RAM drives** (with many names, including RAM disks) present as raw block devices, commonly file system formatted
- Computers have buffering, caching via RAM, so why RAM drives?
  - Caches / buffers allocated / managed by programmer, operating system, hardware
  - RAM drives under user control
  - Found in all major operating systems
    - Linux /dev/ram, macOS diskutil to create them, Linux /tmp of file system type tmpfs
- Used as high-speed temporary storage
  - Programs could share bulk date, quickly, by reading/writing to RAM drive

Disk Attachment

- A secondary storage device is attached to a computer by the system bus or an I/O bus.
- Several kind of buses are available including **Advanced Technology Attachment (ATA), Serial ATA (SATA), eSATA, Serial Attached SCSI (SAS), Universal Serial Bus (USB), and Fiber Channel (FC)**.
- The most common one is SATA
- Since NVM devices are faster than HDDs, the industry created a special, fast interface for NVM devices called **NVM express (NVMe)**, connecting directly the device to the system PCI bus.
- Data transfers on a bus carried out by special electronic processors called **controllers** (or **host-bus adapters, HBAs**).
  - Host controller on the computer end of the bus, device controller on device end
  - Computer places command on host controller, using memory-mapped I/O ports
    - Host controller sends messages to device controller
    - Data transferred via DMA between device and computer DRAM

HDD Scheduling

- The operating system is responsible for using hardware efficiently — for the disk drives, this means **minimizing access time** and **maximizing data transfer bandwidth**.
- Access time has two major components: Seek time and rotational latency
  - Minimize seek time
  - **Seek time ≈ seek distance**
- Disk **bandwidth** is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer.
- We can improve both the access time and the bandwidth by managing the order in which storage I/O requests are serviced.

Disk Scheduling

- There are many sources of disk I/O request
  - OS
  - System processes
  - User processes
- When a process needs I/O to or from the drive, it issues a system call to the operating system which request for information such as:
  - Whether this operation is input or output
  - What memory address for the transfer is
  - The amount of data to transfer
- If the desired driver and controller are available, the request can be services immediately.
- If they are busy, any new requests for service will be placed in the queue of pending request.
  - Optimization algorithms only make sense when a queue exists

Disk Scheduling

- Note that driver controllers have small buffers and can manage a queue of I/O requests (of varying "depth")
- Several algorithms exist to schedule the servicing of disk I/O requests
- The analysis is true for one or many platters
- We illustrate scheduling algorithms with a request queue (0-199)

number of cylinder: 98, 183, 37, 122, 14, 124, 65, 67

Head pointer 53

FCFS

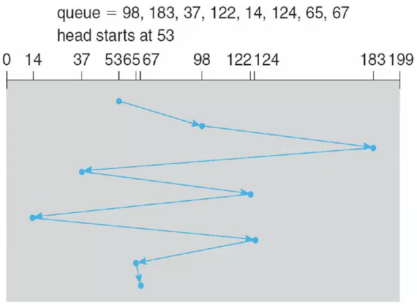
- The simplest for of disk scheduling is First Come First Served which is fair but not always fastest.
- We illustrate scheduling algorithms with a request queue (0-199)

98, 183, 37, 122, 14, 124, 65, 67

Head pointer 53

- The total head movement of 640 cylinders.
- The swing from 122 to 14 and then back to 124 illustrated the problem.
- If the requests for cylinder 37 and 14 could be serviced together, before or after the requests for 122 and 124, the total head movement could be decreased, improving the performance.

Illustration shows total head movement of 640 cylinders

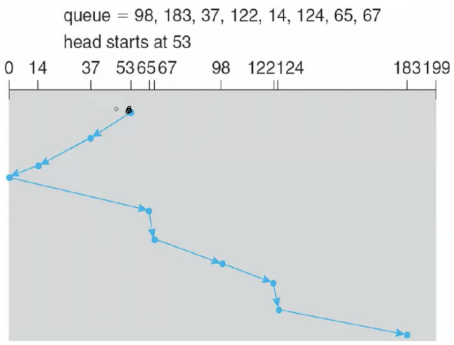


SCAN

- The disk arm starts at one end of the disk, and moves toward the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed, and servicing continues.
- **SCAN algorithm** sometimes called the **elevator algorithm**
- Illustration shows total head movement of 236 cylinders
- But note that if requests are uniformly dense, the largest density is at another end of disk and those wait the longest

SCAN

- We need to know the direction of head movement in addition to the head's current position.
- Assuming the disc arm is moving toward o and that the initial head position is again 53, the head will serve 37, 14, 65, 67, 98, 122, 124 and 183.

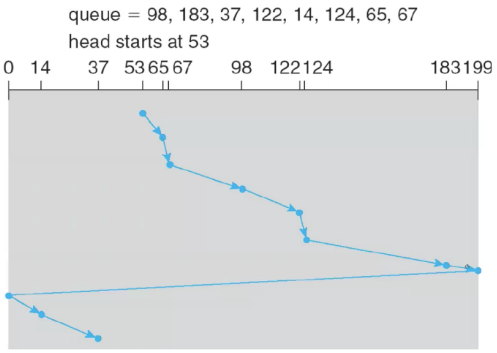


C-SCAN

- Provides a more uniform wait time than SCAN
- The head moves from one end of the disk to the other, servicing requests as it goes
- When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip
- Treats the cylinders as a circular list that wraps around from the last cylinder to the first one

C-SCAN

- We need to know the direction of head movement in addition to the head's current position.
- Assuming the disc arm is moving 0 to 199 and that the initial head position is again 53, the head will serve as:



Selecting a Disk-Scheduling Algorithm

- FCFS is common and has a natural appeal
- SCAN and C-SCAN perform better for systems that place a heavy load on the disk
  - Since they cause Less starvation, but still possible
- To avoid starvation Linux implements **deadline scheduler**
  - Maintains separate read and write queues, gives read priority
    - Because processes more likely to block on read than write
  - Deadline Implements four queues: 2 x read and 2 x write
    - 1 read and 1 write queue implementing C-SCAN
    - 1 read and 1 write queue sorted in FCFS order
    - All I/O requests sent in batch sorted in that queue's order
    - After each batch, checks if any requests in FCFS older than configured age (default 500ms)
      - If so, LBA queue containing that request is selected for next batch of I/O
- The deadline I/O scheduler is the default in the Linux RedHat 7 distribution, but RHEL 7 also includes two others: **NOOP** and **completely fair queueing** scheduler **(CFQ)** also available, defaults vary by storage device

NVM Scheduling

- No disk heads or rotational latency but still room for optimization
- In RHEL 7 **NOOP** (no scheduling) is used but adjacent LBA requests are combined
  - NVM best at random I/O, HDD at sequential
  - Throughput can be similar
  - **Input/Output operations per second (IOPS)** much higher with NVM (hundreds of thousands vs hundreds)
  - But **write amplification** (one write, causing garbage collection and many read/writes) can decrease the performance advantage

Error Detection and Correction

- Fundamental aspect of many parts of computing (memory, networking, storage)
- **Error detection** determines if a problem has occurred (for example a bit flipping)
  - For example, a bit in DRAM changed from a 0 to a 1.
  - If detected, can halt the operation before it is propagated
  - Detection is frequently done via parity bit that records whether the number of bits in the byte set to 1 is even or odd.
- Parity is one form of **checksum** – uses modular arithmetic to compute, store, and compare values of fixed-length words
  - Another error-detection method common in networking is **Cyclic Redundancy Check (CRC)** Which uses hash function to detect multiple-bit errors
- **Error-correction code (ECC)** not only detects, but can correct some errors
  - Soft errors correctable, hard errors detected but not corrected

Storage Device Management

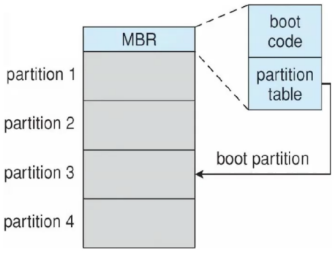
- A new storage device is a blank slate; it is a set of uninitialized semiconductor storage cells.
- Before a storage device can store data, it must be divided into sectors that the controller can read and write.
- This process is called **Low-level formatting**, or **physical formatting** — Dividing a disk into sectors that the disk controller can read and write
  - Each sector can hold header information, data area, trailor, plus error correction code (**ECC**)
  - Usually 512 bytes of data but can be selectable

Storage Device Management

- To use a disk to hold files, the operating system still needs to record its own data structures on the disk
  - **Partition** the disk into one or more groups of blocks or pages. The partition information is written in a fixed format at a fixed location on the storage device.
  - **Logical formatting** or "making a file system" in which the operating system stores the initial file-system data structures onto the device. The data structure may include maps of free and allocated space and an initial empty directory.

Storage Device Management

- For a computer to start running, it must have an initial program to run.
- This initial **bootstrap loader** is stored in NVM flash memory firmware on the system motherboard and mapped to a known memory location.
- It initializes all aspects of the system, from CPU registers to device controllers and the content of the main memory.



Booting from secondary storage in Windows

RAID Structure

- Storage devices have continued to get smaller and cheaper, so it is economic to attach many drivers to computer systems.
- Having a large number of drivers in a system presents opportunities for improving the rate at which data can be read or written as well as improving the reliability of data storage
  - Redundant information can be stored on multiple drives.
- A variety of disk organization techniques called **redundant arrays of independent disks (RAIDs)**.

RAID Structure

- **RAID – redundant array of inexpensive independent disks**
  - multiple disk drives provides reliability via **redundancy**
- Increases the **Mean Time To Failure (MTTF)**
  - Suppose that Mean Time To Failure (MTTF) of a single disk is 100,000 hours, then the MTTF of some disk in an array of 100 disk will be 100,000/100 = 1,000 hours or 41.66 days.
  - Storing one copy of data >> each disk failure will result in loss of a significant amount of data.
- The solution to the problem of reliability is to introduce **Redundancy**.
  - Storing the disk failure to rebuild the lost information that is not normally needed but can be used in the event of disk failure to rebuild the lost information.
- The simplest approach to introduce redundancy is duplication of every drive – **Mirroring**.
  - A logical disk consists of two physical drives, and every write is carried out on both drives – **mirrored volume**.

RAID Structure

- The MTBF of a mirrored volume depends on two factors:
  - MTBF of individual drives
  - **Mean Time To Repair (MTTR):** the time it takes to replace a failed drive and restore the data on it.
  - Suppose that the failures of the two drives are independent, if the MTBF of a single drive is 100,000 hours, and the MTTR is 10 hours, the Mean Time to Data Loss of a mirrored drive system is $100{,}000^2/(2*10) = 500*10^6$ hours or 57,000 years.

RAID Structure

- **RAID – redundant array of inexpensive independent disks**
  - multiple disk drives provides reliability via **redundancy**
- Increases the **mean time to failure (MTTF)**
- **Mean time to repair** – exposure time when another failure could cause data loss
- **Mean time to data loss** based on above factors
- If mirrored disks fail independently, consider disk with 100,000 **mean time to failure**
- and 10 hour mean time to repair
  - Mean time to data loss is 100, 0002 / (2 ∗ 10) = 500 ∗ 106 hours, or 57,000 years!
- Frequently combined with **NVRAM** to improve write performance
- Several improvements in disk-use techniques involve the use of multiple disks working cooperatively

RAID Structure

- RAID 2 disks, considering a disk with 100,000 Mean Time To Failure and 10 hour Mean Time To Repair, Mean Time to Data Loss?

Mirrored RAID: 2 disks

$MTTF_{(1st\_fail)} = MTTF/2 = 100{,}000/2 = 50{,}000$ hours

$Prob_{(2nd\_fail\_during\_repair)} = MTTR/MTTF = 10/100{,}000 = 10^{-4}$

Mean time to data loss = $MTTF_{(1st\_fail+2nd\_fail\_during\_repair)}$

$MTTF_{(fail+fail\_during\_repair)} = MTTF_{(1st\_fail)} / Prob_{(2nd\_fail\_during\_repair)}$

$= MTTF^2/(2*MTTR) = 10^{10}/(2*10)$

$= 5*10^8$ hours

RAID Structure

- RAID 2 disks, considering a disk with 100,000 Mean Time To Failure and 10 hour Mean Time To Repair, Mean Time to Data Loss?

- MTTF= 100,000 hours
- Probability of failure of one disk= 1 /MTTF = 1/ 100,000 = 10^{-5}

- MTTdl = 1/probability of data loss(dl)
- Pdl = (P(f1) + P(f2)) * (p(f1orf2) * MTTR) = (10^{-5} + 10^{-5}) * (10^{-5} * 10) = 2* 10^{-9}
- MTTdl = 1/ 2* 10^{-9} = 5 * 10^8 h

Exam Exercise:

- Sia dato un disco organizzato con struttura DAIR. Che cosa si intende, in relazione a tale disco e alla probablità di guasto e/o tempo medio tra Guasti (MMTF), con intermini seguenti?
- Mean Time to Repair (MTTR) – Mean Time to Data Loss (MTTDL)
- Si consdieri una struttura DAID con 2 dischi in configurazione "mirrored". Se ognuno dei dischi può guastarsi in modo independente dall'altro, con MTTF = 5000 ore e MTTR = 20 ore, quanto sarà il MTTDL?

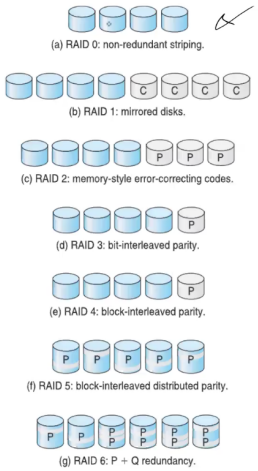
- MTTDL = $MTTF^2$ /(2*MTTR) = 50000^2/(2*20) = 25 /4*10^7 ore = 6.25*10^7

RAID for improvement in performance

- With mirroring, the rate at which read requests can be handled is doubles, since read requests can be sent to either drive.
  - The transfer rate of each read is the same as in a single-drive system but the number of reads per unit time has doubles.
- Improving the transfer rate by **stripping data** across the drives.
  - **Bit-level stripping**: splitting the bits of each byte across multiple drives.
    - If we have an array of eight drives, we write but iof each byte to drive i.
  - **Block-level stripping**: blocks of files are stripped across multiple drives.
    - With n drives, block I of a file goes to drive (i mod n) + 1.
- Benefits of parallelism in a storage system:
  - Increasing the throughput of multiple small accesses
  - Reducing the response time of large accesses.

RAID Levels

- Mirroring provides high reliability, but it is expensive.
- Stripping provides high data-transfer rate, but it doesn't improve reliability.
- Data stripping combined with redundancy : **RAID Levels**
  - **RAID Level 0**
  - **RAID Level 1**
  - **RAID Level 4**
  - **RAID Level 5**
  - **RAID Level 6**
  - **Multidimensional RAID Level 6**
  - **RAID Levels 0 + 1 and 1 + 0**



I/O Systems

Overview

- I/O management is a major concern of operating system design and operation
  - Important aspect of computer operation
  - I/O devices vary greatly >> various methods are required to control them
  - Performance management
  - New types of devices frequent
- Ports, busses, device controllers connect to various devices
- To encapsulate the details and oddities of different devices, the kernel of an operating system is structured to use **device driver** modules.
  - Device drivers present uniform device-access interface to I/O subsystem

I/O Hardware

- A device communicates with a computer system by sending signals.
  - **Port** - connection point for a device for example serial port
  - **Bus** - used in most computers today, such as PCI, a set of wires and rigidly defined protocol that specify a set of messages.
  - Bus - daisy chain or shared direct access
    - PCIe bus, connects the processor-memory subsystem to fast devices.
    - expansion bus connects relatively slow devices such as keyboard and USB ports.

I/O Hardware

- A device communicates with a computer system by sending signals.
  - **Controller** (**host adapter**) – is a collection of electronics that operate a port, a bus, a device that controls the signals.
    - Sometimes integrated
    - Sometimes separate circuit board (host bus adapter - HBA) that connects to the bus in the computer.
    - It contains processor, microcode, private memory, bus controller, etc
      - Some talk to per-device controller with bus controller, microcode, memory, etc

I/O Hardware

- How does the processor give commands and data to a controller to accomplish an I/O transfer?
  - The controller has one or more registers for data and control signals. The processor communicates with the controller by reading and writing bit patterns in these registers.

- Devices usually have registers where device driver places commands, addresses, and data to write, or read data from registers after command execution
  - Data-in register: used by the host to get input
  - data-out register: written by the host to send output
  - status register: contains bits that can be read by the host which indicates states, such as whether the current command had been completed.
  - Control register: written by the host to start a command or to change the mode of a device
  - Typically 1-4 bytes, or FIFO buffer.

I/O Hardware

- Devices have addresses, used by
  1. Direct I/O instructions that specify the transfer of a byte or a word to an I/O port address.
  2. **Memory-mapped I/O**
     - Device control registers are mapped into the address space of the processor.
     - The CPU executes I/O requests using the standard data-transfer instructions to read and write the device-control registers at their mapped locations in physical memory.

Polling

- Assuming that 2 bits are used for coordinating the relationship between controller and host.
- For each byte of I/O
  1. The host repeatedly reads the **busy bit** until that bit becomes clear, Read busy bit from status register until 0.
  2. The host sets the write bit and writes a byte into the ***data-out* register**.
  3. The host sets ***command-ready*** bit.
  4. When the controller notices that the ***command-ready*** bit is set, it sets the *busy* bit.
  5. The controller reads the command register and sees the write command. It reads the ***data-out*** register to get the byte and does the I/O to the device.
  6. Controller clears ***command-ready*** bit, clears the *error bit* in the status register to indicate that the device I/O succeeded, and clears the *busy* bit to indicate that it is finished.
- In step 1, the host is **busy-waiting** or **poling**: it is in a loop, reading the status register over and over until the busy bit becomes clear.

Polling

- Step 1 is **busy-wait** cycle to wait for I/O from device
  - Reasonable if device is fast
  - But inefficient if device slow
  - CPU switches to other tasks?
    - But if miss a cycle data overwritten / lost

Interrupts

- In many computer architectures, three CPU-instruction cycles are sufficient to poll a device: read a device register, logic and to extract a status bit, and branch if not zero.
- Pooling is not efficient when it is attempted repeatedly but does not find a device ready for service.
- Therefore, it is more efficient to arrange for the hardware controller to notify the CPU when the device becomes ready for service, rather than to require the CPU to poll repeatedly for an I/O completion.
- The hardware mechanism that enables a device to notify the CPU is called an **interrupt**.

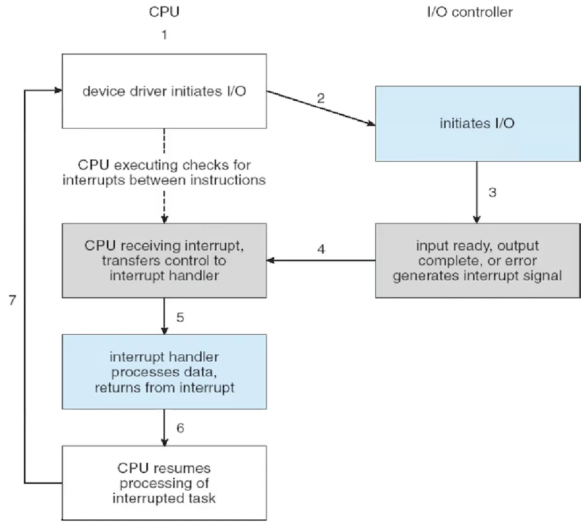
Interrupts

- The CPU hardware has a wire called the **interrupt-request line** that the CPU sense after executing every structure.
- CPU detecting that a controller has asserted a signal on this line, it performs a jump to **Interrupt handler.**
  - Determines the cause of the interrupt and performs the necessary processing.
- The device controller **raises** an interrupt by asserting a signal on the interrupt request line, the CPU **catches** the interrupt and dispatches it to the interrupt handler and the handler clear the interrupt by servicing the device.

Interrupts

- The interrupt mechanism accepts an address that selects a specific interrupt-handling routine from a small set. This address is an offset in a table called the **interrupt vector**.
  - **Interrupt vector** to dispatch interrupt to correct handler
  - Context switch at start and end
  - Based on priority
  - Some nonmaskable
  - Interrupt chaining if more than one device at same interrupt number

Interrupt-Driven I/O Cycle



Interrupts

- Interrupt mechanism also used for **exceptions**
  - Terminate process, crash system due to hardware error
- Page fault executes when memory access error
- System call executes via **trap** to trigger kernel to execute request
- Multi-CPU systems can process interrupts concurrently
  - If operating system designed to handle it
- Used for time-sensitive processing, frequent, must be fast

Interrupts

- Interrupts: An interrupt is a signal sent to the processor by an external device, such as a keyboard or mouse, to request its attention. When an interrupt occurs, the processor stops executing the current instruction and starts executing a special routine called an interrupt handler. The interrupt handler takes care of the event and then returns control to the interrupted program.
- **Exceptions**: An exception is an event that occurs within the processor itself, such as an arithmetic overflow or a segmentation fault. When an exception occurs, the processor stops executing the current instruction and transfers control to an exception handler, which is a special routine designed to handle the specific type of exception.
- **Traps**: A trap is a software-generated interrupt caused by a specific instruction in a program. When the instruction is executed, the processor generates a trap and transfers control to a trap handler, which is a special routine designed to handle the specific type of trap. Traps are used to implement system calls, which allow programs to request services from the operating system, such as reading or writing files.

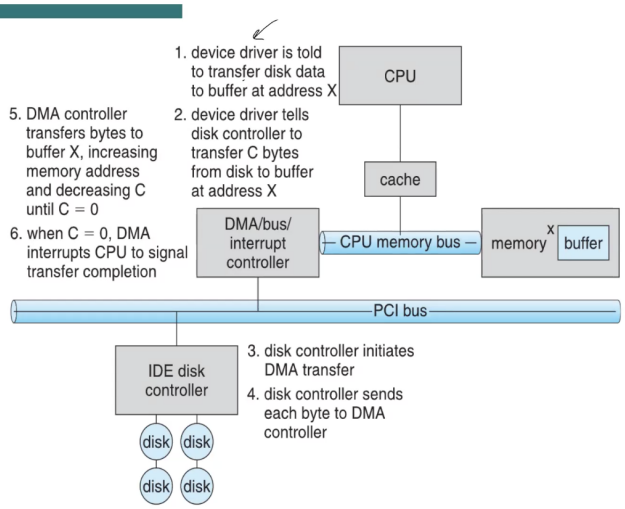
Interrupts

- The key differences between these mechanisms are:
  - **Source of the event**: Interrupts are generated by external devices, exceptions are generated by the processor itself, and traps are generated by specific instructions in a program.
  - **Triggering mechanism**: Interrupts are triggered by external signals, exceptions are triggered by specific processor conditions, and traps are triggered by specific program instructions.
  - **Handling routine**: Interrupts are handled by interrupt handlers, exceptions are handled by exception handlers, and traps are handled by trap handlers.
  - **Purpose**: Interrupts are used to handle external events, exceptions are used to handle internal errors, and traps are used to implement system calls.

Direct Memory Access

- Used to avoid **programmed I/O** (one byte at a time) for large data movement
- Requires **DMA** controller
- Bypasses CPU to transfer data directly between I/O device and memory
- OS writes DMA command block into memory
  - Source and destination addresses
  - Read or write mode
  - Count of bytes
  - Writes location of command block to DMA controller
  - Bus mastering of DMA controller – grabs bus from CPU
    - **Cycle stealing** from CPU but still much more efficient
  - When done, interrupts to signal completion
- Version that is aware of virtual addresses can be even more efficient - **DVMA**

Six Step Process to Perform DMA Transfer



Application I/O Interface

- I/O system calls encapsulate device behaviors in generic classes
- Device-driver layer hides differences among I/O controllers from kernel
- New device talking already-implemented protocols need no extra work
- Each OS has its own I/O subsystem structures and device driver frameworks

Application I/O Interface

- Devices vary in many dimensions
  - **Character-stream** or **block**
    - A character-stream device transfers bytes one by one, whereas block device transfer a block of bytes as a unit.
  - **Sequential** or **random-access**
    - A sequential device transfers data in a fixed order determined by the device, whereas the user of a random-access device can instruct the device to seek to any of the available data storage locations.
  - **Synchronous** or **asynchronous** (or both)
    - A synchronous device performs data transfers with predictable response times, in coordination with other aspects of the system. As asynchronous device exhibits irregular or unpredictable response times not coordinated with other computer events.
  - **Sharable** or **dedicated**
    - A sharable device can be used concurrently by several processes or threads, a dedicated device no.
  - **Speed of operation**
    - Device speed range from a few bytes per second to gigabytes per second.
  - **read-write**, **read only**, or **write only**
    - Some devices perform both input and output, but others support only one data transfer direction.

Nonblocking and Asynchronous I/O

- **Blocking** - process suspended until I/O completed
  - Easy to use and understand
  - Insufficient for some needs
- **Nonblocking** - I/O call returns as much as available
  - User interface, data copy (buffered I/O)
  - Implemented via multi-threading
  - Returns quickly with count of bytes read or written
  - `select()` to find if data ready then `read()` or `write()` to transfer
- **Asynchronous** - process runs while I/O executes
  - Difficult to use
  - I/O subsystem signals process when I/O completed

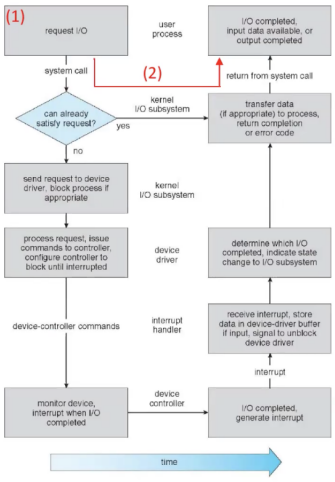
Kernel I/O Subsystem

- Kernels provide many services related to I/O: scheduling, buffering, caching, spooling, device reservation, and error handling.
  - **I/O Scheduling**: determining good order to execute I/O requests to improve performance and share devices fairly among processes.
    - Operating system developers implement scheduling by maintaining a wait queue of requests for each device.
    - When an application issues a I/O system call, the request is placed on the queue for that device.
    - The I/O scheduler rearranges the order of the queue to improve the overall system efficiency and average response time.

Life Cycle of An I/O Request

- An I/O operation requires a great many steps that together consume a tremendous number of CPU cycles:

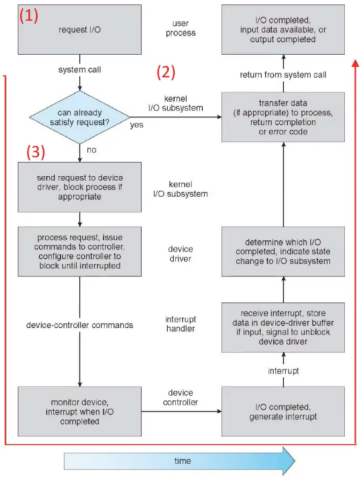
1. A process issues a blocking read() system call to a file descriptor of a file that has been opened previously.
2. The system call code in the kernel checks the parameters for correctness. In the case of input, if the data are already available in the buffer cache, the data are returned to the process, and the I/O request is complete.



Life Cycle of An I/O Request

- An I/O operation requires a great many steps that together consume a tremendous number of CPU cycles:

  3. otherwise, a physical I/O must be performed. The process is removed from the run queue and is place on the wait queue for the device, and the I/O request is scheduled. Eventually, the I/O subsystem sends the request to the device driver.

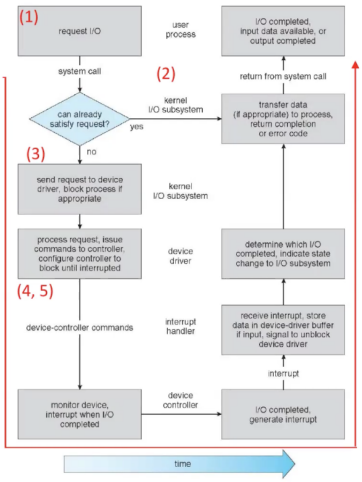


Life Cycle of An I/O Request

- An I/O operation requires a great many steps that together consume a tremendous number of CPU cycles:

  4. The device driver allocates kernel buffer space to receive the data and schedules the I/O. Eventually, the driver sends commands to the device controller by writing into the device-control registers.

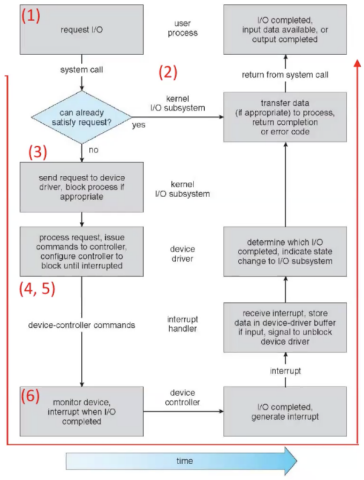
  5. The device controller operates the device hardware to perform the data transfer.



Life Cycle of An I/O Request

- An I/O operation requires a great many steps that together consume a tremendous number of CPU cycles:

  6. The driver may pool for status and data, or it may have set up to a DMA transfer into kernel memory. We assume that the transfer is managed by a DMA controller, which generates an interrupt when the transfer completes.

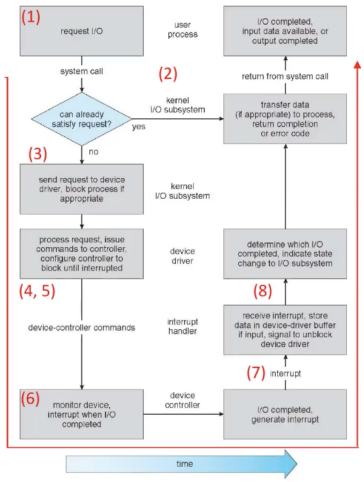


Life Cycle of An I/O Request

- An I/O operation requires a great many steps that together consume a tremendous number of CPU cycles:

  7. The correct interrupt handler receives the interrupt via the interrupt-vector table, stores any necessary data, signal the device driver and returns from the interrupt.

  8. The device driver receives the signal, determines which I/O request has completed, determines the request's status, and signal the kernel I/O subsystem that the request has been completed.



Life Cycle of An I/O Request

- An I/O operation requires a great many steps that together consume a tremendous number of CPU cycles:

9. The kernel transfers data or return codes to the address space of the requesting process and moves the process from the wait queue back to the ready queue.

10. Moving the process to the ready queue unblocks the process. When the scheduler assigns the process to the CPU, the process resumes execution at the completion of the system call.