Devices and I/O

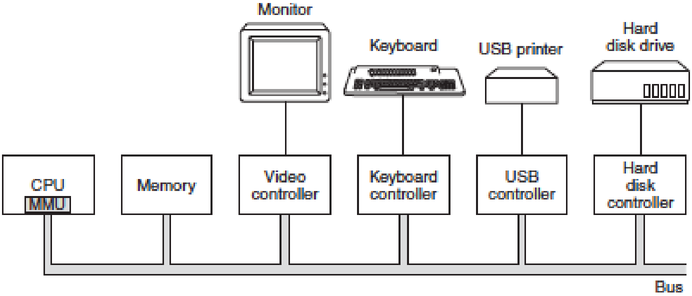
key concepts: device registers, device drivers, program-controlled I/O, DMA, polling, disk drives, disk head scheduling

Zille Huma Kamal

David R. Cheriton School of Computer Science
University of Waterloo

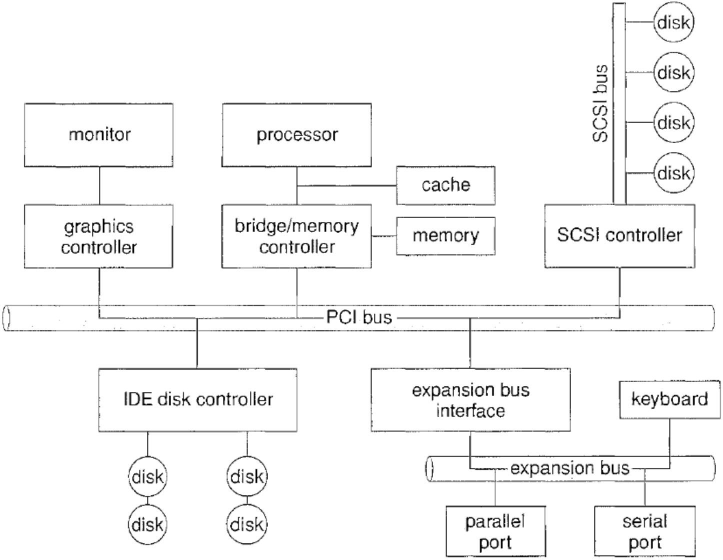
Spring 2022

Simple Architecture



Tanenbaum, A. S. & Bos, H. (2014), Modern Operating Systems , Pearson , Boston, MA .

Modern Architecture



Silberschatz, A. , Galvin, P.B.,Gagne G., (2018), Operating Systems Concepts. Wiley

Terminology

- **device:** how a processor receives input and produces output
  - a **keyboard** is an input device
  - a **printer** is an output device
  - a **touch screen** is both input and output
- **storage device:** a device that stores data for processor
- **bus:** a communication pathway between various devices in a computer
  - **internal bus:** memory bus or front side bus is for communication between the CPU and RAM. It is fast and close to the CPU(s).
  - **peripheral:** or expansion bus, allows devices in the computer to communicate
- **bridge:** connects two different buses

keyboards, mice, printers, graphics cards, USB fans, joysticks, CD drives, sound cards, ... are all devices

Persistent Storage Devices

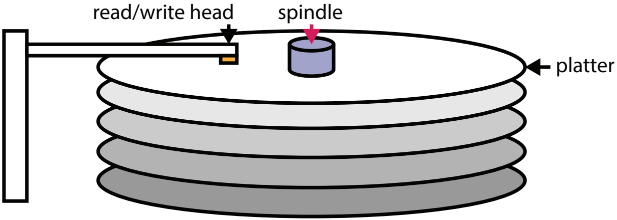
- **persistant** storage is any device where data persists even when the device is without power
  - physical memory is not persistant
  - a hard disk is persistant
  - also referred to as **non-volatile**
- persistant storage comes in many forms
  - punched cards of metal or paper (1700s-1970s)
  - magnetic drums (1930s-1960s), tapes (1920s)
  - floppy (1970s-2000s) and hard disks (1950s)
  - CDs (1980s), DVDs (1990s), Blu-ray (2000s)
  - solid state memory (1970s, 1990s)
  - ReRam (resistive RAM) (2000s)

The earliest form of persistant storage was punched metal cards which held the "programs" for Jacquard weaving looms in the 1700s.

Magnetic tapes are still in active use today!

Hard Disks

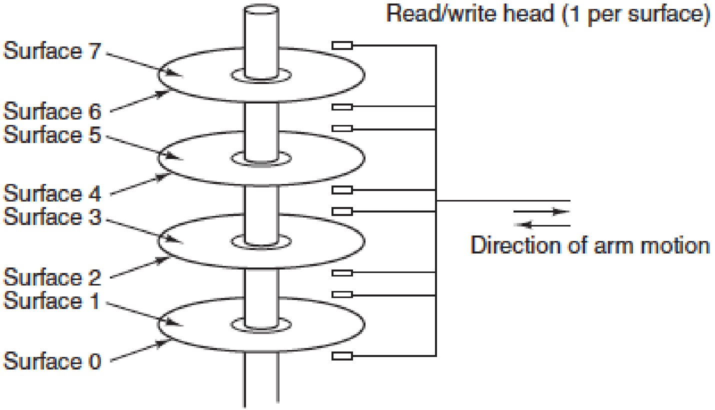
- commonly used persistant storage device
- a number of spinning, ferromagnetic-coated platters read/written by a moving R/W head



Platters are typially made from glass or porcelain. Hence, they are exceptionally fragile.

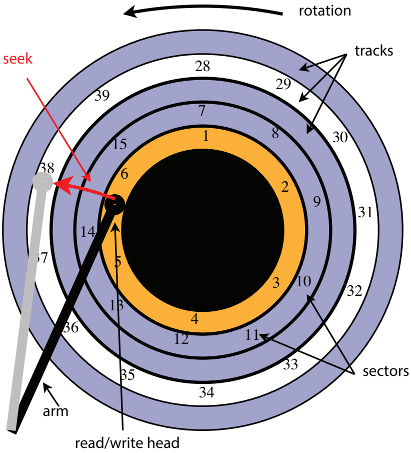
Often called mechanical disks, both patter and read/write head must move to perform a read or write operation. This motion is costly.

Another look at hard disks



Tanenbaum, A. S. & Bos, H. (2014), Modern Operating Systems , Pearson , Boston, MA .

Logical View of a Disk Drive



- disk is an array of numbered blocks (or sectors)
- each block is the same size (e.g., 512 bytes)
- blocks are the unit of transfer between the disk and memory
    - typically, one or more contiguous blocks can be transferred in a single operation
- assume, for simplicity, that each track contains the same number of sectors

Cost Model for Disk I/O

- moving data to/from a disk involves:
  1. **seek time:** move the read/write heads to the appropriate track
     - depends on **seek distance**, the distance (in tracks) between previous and current request
     - value: 0 milliseconds to cost of max seek distance
  2. **rotational latency:** wait until the desired sectors spin to the read/write heads
     - depends on rotational speed of disk
     - disk is always spinning
     - value: 0 milliseconds to cost of single rotation
  3. **transfer time:** wait while the desired sectors spin past the read/write heads
     - depends on the rotational speed of the disk and the amount of data accessed
- Request Service Time = Seek Time + Rotational Latency + Transfer Time

Request Service Time Example

A disk has a total capacity of 2^{32} bytes. The disk has a single platter with 2^{20} tracks. Each track has 2^8 sectors. The disk operates at 10000RPM and has a maximum seek time of 20 milliseconds.

1. How many bytes are in a track?
   $BytesPerTrack = DiskCapacity/NumTracks$
2. How many bytes are in a sector?
   $BytesPerSector = BytesPerTrack/NumSectorsPerTrack$
3. What is the maximum rotational latency?
   $MaxLatency = 60/RPM$
4. What is the average seek time and rotational latency?
   $AverageSeek = MaxSeek/2$
   $AverageLatency = MaxLatency/2$
5. What is the cost to transfer 1 sector?
   $SectorLatency = MaxLatency/NumSectorsPerTrack$
6. What is the expected cost to read 10 consecutive sectors from this disk?
   $RequestServiceTime = Seek + RotationalLatency + TransferTime$

Request Service Time Example

A disk has a total capacity of 2^{32} bytes. The disk has a single platter with 2^{20} tracks. Each track has 2^8 sectors. The disk operates at 10000RPM and has a maximum seek time of 20 milliseconds.

1. How many bytes are in a track?
   $= 2^{32}/2^{20} = 2^{12}$ bytes per track
2. How many bytes are in a sector?
   $= 2^{12}/2^8 = 2^4$ bytes per sector
3. What is the maximum rotational latency?
   $= 60/10000 = 0.006$ or 6 milliseconds
4. What is the average seek time and rotational latency?
   $= 20/2 = 10$ milliseconds average seek time
   $= 6/2 = 3$ milliseconds average rotational latency
5. What is the cost to transfer 1 sector?
   $= 6/2^8 = 6/256 = 0.0234375$ milliseconds per sector
6. What is the expected cost to read 10 consecutive sectors from this disk?
   $= 10 + 3 + 10(0.0234375) = 13.234375$ milliseconds
   Note that since we do not know the position of the head, or the platter, we use the average seek and average rotational latency.

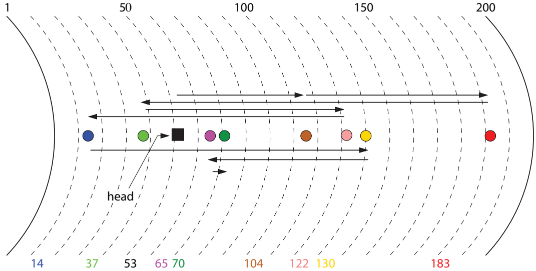
Performance Implications of Disk Characteristics

- larger transfers to/from a disk device are **more efficient** than smaller ones. That is, the cost (time) per byte is smaller for larger transfers. (Why?)
- sequential I/O is faster than non-sequential I/O
  - sequential I/O operations eliminate the need for (most) seeks
- while sequential I/O is not always possible, we can group requests to try and reduce average request time

Historically, seek time is the dominating cost. High-end drives can have maximum seek times around 4 milliseconds. Consumer grade drives more commonly have seek times between 9 and 12 milliseconds.

Disk Head Scheduling

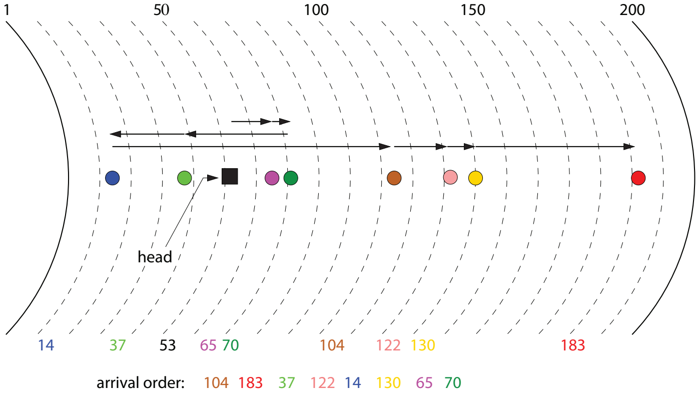
- goal: reduce seek times by controlling the order in which requests are serviced
- disk head scheduling may be performed by the device, by the operating system, or both
- for disk head scheduling to be effective, there must be a queue of outstanding disk requests (otherwise there is nothing to reorder)
- **first-come, first served** is fair and simple, but offers no optimization for seek times



arrival order: 104 183 37 122 14 130 65 70

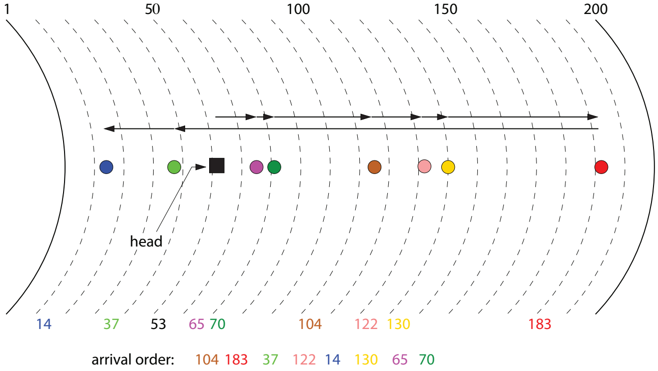
Shortest Seek Time First (SSTF)

- choose closest request (a greedy approach)
- seek times are reduced, but requests may starve



Elevator Algorithms (SCAN)

- Under SCAN, aka the elevator algorithm, the disk head moves in one direction until there are no more requests in front of it, then reverses direction.
- there are many variations on this idea
- SCAN reduces seek times (relative to FCFS), while avoiding starvation



Solid State Drives(SSD)

- no mechanical parts; use integrated circuits for persistant storage instead of magnetic surfaces
- variety of implementations
  - DRAM: requires constant power to keep values
  - Flash Memory: traps electrons in quantum cage
- logically divided into blocks and pages
  - 2, 4 or 8KB pages
  - 32KB-4MB blocks
- reads/writes at page level
  - pages are initialized to 1s; can transition $1 \rightarrow 0$ at page level (i.e., write new page)
  - a high voltage is required to switch $0 \rightarrow 1$ (i.e., overwrite/delete page)
  - cannot apply high voltage at page level, only to blocks
    - overwriting/deleting data must be done at the block level

Writing and Deleting from Flash Memory

- **Naive Solution (slow):**
  - read whole block into memory
  - re-initialize block (all page bits back to 1s)
  - update block in memory; write back to SSD
- **SSD controller handles requests (faster):**
  - mark page to be deleted/overwritten as invalid
  - write to an unused page
  - update translation table
  - requires garbage collection

Each block of an SSD has a limited number of write cycles before it becomes read-only. SSD controllers perform **wear leveling**, distributing writes evenly across blocks, so that the blocks wear down at an even rate.
Hence, defragmentation, which takes files spread across multiple, non-sequential pages and makes them sequential, can be harmful to the lifespan of an SSD. Additionally, since there are no moving parts, defragmentation serves no performance advantage.

Persistent RAM

- values are persistant in the absence of power
  - ReRAM: resistive RAM
  - 3D XPoint, Intel Optane
- can be used to improve the performance of secondary storage
  - traditional CPU caches are small; not effective for caching many disk blocks
  - RAM can cache i-nodes and data blocks; but should be used for address spaces
  - use persistant RAM instead
    - i-nodes and data blocks silently cached to this special memory
    - Intel Optane, for example, modules are 16-32GB, so many blocks can be cached giving big performance improvements when mechanical disks are used

I/O Subsystem and OS

- **Design Goals for OS:**
  - simplest interface
  - maximum concurrency
- sys/161 example devices:
  - timer/clock - current time, timer, beep
  - disk drive - persistent storage
  - serial console - character input/output
  - text screen - character-oriented graphics
  - network interface - packet input/output
- variation in devices implies OS must provide a range of functionality

Device Register Example: Sys/161 timer/clock

- communication with devices carried out through **device registers**
- three primary types of device registers
  - **status**: tells you something about the device's current state. Typically, a status register is **read**.
  - **command**: issue a command to the device by writing a particular value to this register.
  - **data**: used to transfer larger blocks of data to/from the device.
- some device registers are combinations of primary types
  - a **status and command** register is read to discover the device's state and written to issue the device a command.

Device Register Example: Sys/161 timer/clock

Offset	Size	Type	Description
0	4	status	current time (seconds)
4	4	status	current time (nanoseconds)
8	4	command	restart-on-expiry
12	4	status and command	interrupt (reading clears)
16	4	status and command	countdown time (microseconds)
20	4	command	speaker (causes beeps)

The clock is used in preemptive scheduling.

Device Register Example: Serial Console

Offset	Size	Type	Description
0	4	command and data	character buffer
4	4	status	writeIRQ
8	4	status	readIRQ

If a write is in progress, the device exhibits **undefined** behaviour if another write is attempted.

Device Drivers

- a **device driver** is a part of the kernel that interacts with a device
- example: write character to serial output device

```
// only one writer at a time
P(output device write semaphore)
// trigger the write operation
write character to device data register
repeat {
    read writeIRQ register
} until status is ‘‘completed’’
// make the device ready again
write writeIRQ register to ack completion
V(output device write semaphore)
```

- this example illustrates **polling**: the kernel driver repeatedly checks device status

> Although the majority of device drivers are a (dynamically loadable) part of the kernel, some drivers exist in user-space.

Using Interrupts to Avoid Polling

Device Driver Write Handler:

```
// only one writer at a time
P(output device write semaphore)
// trigger write operation
write character to device data register
```

Interrupt Handler for Serial Device:

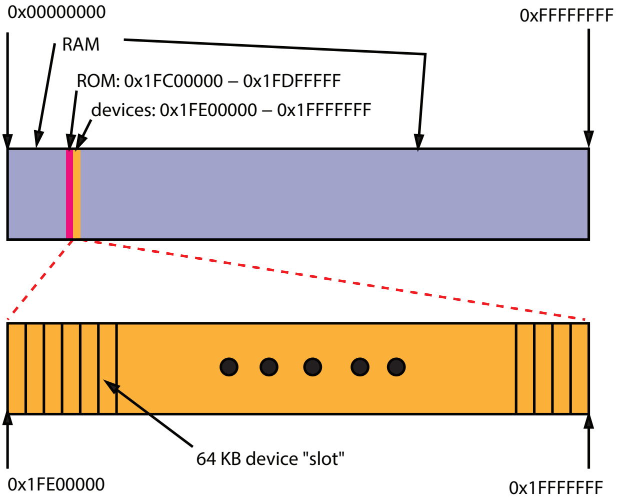
```
// make the device ready again
write writeIRQ register to ack completion
V(output device write semaphore)
```

Accessing Devices

- how can a device driver access device registers?
- Option 1: **Port-Mapped I/O** with special I/O instructions
  - device registers are assigned "port" numbers, which correspond to regions of memory in a separate, smaller address space
  - special I/O instructions, such as `in` and `out` instructions on x86 are used to transfer data between a specified port and a CPU register
- Option 2: **memory-mapped I/O**
  - each device register has a physical memory address
  - device drivers can read from or write to device registers using normal load and store instructions, as though accessing memory

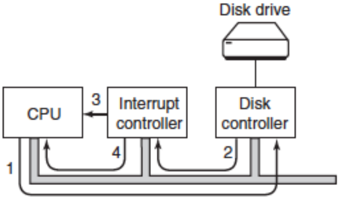
A system may use both port-mapped and memory-mapped I/O.

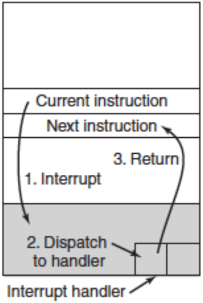
MIPS/OS161 Physical Address Space



Each device is assigned to one of 32 64KB device “slots”. A device’s registers and data buffers are memory-mapped into its assigned slot.

Accessing Devices





Tanenbaum, A. S. & Bos, H. (2014), Modern Operating Systems , Pearson , Boston, MA .

Large Data Transfer To/From Devices

In addition to port and memory mapped I/O, large data blocks can be transferred using other strategies.

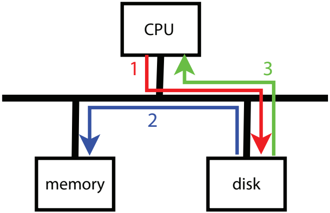
- **program-controlled I/O**

  The device driver moves the data between memory and a buffer on the device.

  The CPU is **busy** while the data is being transferred.

- **direct memory access (DMA)**

  The device itself is responsible for moving data to/from memory. CPU is **not busy** during this data transfer, and is free to do something else.



Sys/161 disks do program-controlled I/O.

Device Register Example: Sys/161 disk controller

Offset	Size	Type	Description
0	4	status	number of sectors
4	4	status and command	status
8	4	command	sector number
12	4	status	rotational speed (RPM)
32768	512	data	transfer buffer

Writing to a Sys/161 Disk

Device Driver Write Handler:

```
// only one disk request at a time
P(disk semaphore)
copy data from memory to device transfer buffer
write target sector number to disk sector number register
write ‘‘write’’ command to disk status register
// wait for request to complete
P(disk completion semaphore)
V(disk semaphore)
```

Interrupt Handler for Disk Device

```
// make the device ready again
write disk status register to ack completion
V(disk completion semaphore)
```

The thread that initiates the write should wait until that write is completed before continuing.

Reading From a Sys/161 Disk

Device Driver Read Handler:

```
// only one disk request at a time
P(disk semaphore)
write target sector number to disk sector number register
write ‘‘read’’ command to disk status register
// wait for request to complete
P(disk completion semaphore)
copy data from device transfer buffer to memory
V(disk semaphore)
```

Interrupt Handler for Disk Device

```
// make the device ready again
write disk status register to ack completion
V(disk completion semaphore)
```

The thread that initiates the read **must** wait until that read is completed before continuing.