But first ...

- Continuation of discussion on I/O devices
Some example I/O devices

- Timers
- Terminals
- Graphical user interfaces
- Network terminals
Clocks & Timers
Programmable clocks

- **One-shot mode:**
  - Counter initialized then decremented until zero
  - At zero a single interrupt occurs

- **Square wave mode:**
  - At zero the counter is reinitialized with the same value
  - Periodic interrupts (called “clock ticks”) occur
Time

- **500 MHz Crystal** (oscillates every 2 nanoseconds)
- **32 bit register overflows in 8.6 seconds**
  - So how can we remember what the time is?

- **Backup clock**
  - Similar to digital watch
  - Low-power circuitry, battery-powered
  - Periodically reset from the internet
  - **UTC**: Universal Coordinated Time
  - **Unix**: Seconds since Jan. 1, 1970
  - **Windows**: Seconds since Jan. 1, 1980
Goals of clock software

- **Maintain time of day**
  - Must update the time-of-day every tick
- **Prevent processes from running too long**
- **Account for CPU usage**
  - Separate timer for every process
  - Charge each tick to the current process
- **Handling the “Alarm” syscall**
  - User programs ask to be sent a signal at a given time
- **Providing watchdog timers for the OS itself**
  - E.g., when to spin down the disk
- **Doing profiling, monitoring, and statistics gathering**
Software timers

- A process can ask for notification (alarm) at time T
  - At time T, the OS will signal the process

- Processes can “go to sleep until time T”

- Several processes can have active timers

- The CPU has only one clock
  - Must service the “alarms” in the right order

- Keep a sorted list of all timers
  - Each entry tells when the alarm goes off and what to do then
Software timers

- Alarms set for 4203, 4207, 4213, 4215 and 4216.
- Each entry tells how many ticks past the previous entry.
- On each tick, decrement the "NextSignal".
- When it gets to 0, then signal the process.
Block vs character devices

- Clocks just interrupt, there’s no data to input or output
- Other devices input/output data
  - Character (byte) at a time (i.e. terminals)
  - Block at a time (i.e. disks)
Terminals
Terminal hardware

- Serial line unit (SLU) for asynchronous bit-serial communication
  - RS-232 / Serial interface / Modem / tty / COM
  - UART: Universal Asynchronous Receiver Transmitter
  - byte → serialize bits → wire → collect bits → byte
  - Registers for control, status, input and output of data

Diagram:
- CPU
- Memory
- RS-232 interface
- UART
- Bus
- Transmit
- Receive
Terminals

- 56,000 baud = 56,000 bits per second = 7000 bytes / sec
  - Each byte is an ASCII character code

- Dumb CRTs / teletypes
  - Very few control characters
    - newline, return, backspace

- Intelligent CRTs
  - Also accept “escape sequences”
  - Reposition the cursor, clear the screen, insert lines, etc.
  - The standard “terminal interface” for computers
    - Example programs: vi, emacs
Terminal drivers

- Two interfaces
  - User interface (to communicate with user)
  - Process I/O (to interact with process)
Input software

- **Character processing**
  - User types “hella←o”
  - Computer echoes as: “hella←_←o”
  - Program will see “hello”

- **Raw mode**
  - The driver delivers all characters to application process
  - No modifications, no echoes
  - vi, emacs, the BLITZ emulator, password entry

- **Cooked mode**
  - The driver does echoing and processing of special chars.
  - “Canonical mode”
Cooked mode

- The terminal driver must...
  - Buffer an entire line before returning to application
  - Process special control characters
    - Control-C, Backspace, line-erase, tabs
  - Echo the character just typed
  - Accommodate type-ahead
    - Ie., it needs an internal buffer!
    - Example of producer consumer problem

- Approach 1 (for computers with many terminals)
  - Have a pool of buffers to use as necessary

- Approach 2 (for single-user computer)
  - Have one buffer (e.g., 500 bytes) per terminal
Central buffer pool vs. dedicated buffers

Terminal data structure

(a)

Central buffer pool

Terminal

0

1

2

3

Terminal data structure

(b)

Terminal

0

Buffer area for terminal 0

1

Buffer area for terminal 1
The end-of-line problem

- **NL** “newline” (ASCII 0x0A, \n)
  - Move cursor down one line (no horizontal movement)
- **CR** “return” (ASCII 0x0D, \r)
  - Move cursor to column 1 (no vertical movement)
- “ENTER key”
  - Behavior depends on the terminal specs
    - May send CR, may send NL, may send both
    - Software must be device independent

- **Unix, Macintosh:**
  - Each line (in a file) ends with a NL
- **Windows:**
  - Each line (in a file) ends with CR & NL
### Special control characters (in “cooked mode”)  

<table>
<thead>
<tr>
<th>Character</th>
<th>POSIX name</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTRL-H</td>
<td>ERASE</td>
<td>Backspace one character</td>
</tr>
<tr>
<td>CTRL-U</td>
<td>KILL</td>
<td>Erase entire line being typed</td>
</tr>
<tr>
<td>CTRL-V</td>
<td>LNEXT</td>
<td>Interpret next character literally</td>
</tr>
<tr>
<td>CTRL-S</td>
<td>STOP</td>
<td>Stop output</td>
</tr>
<tr>
<td>CTRL-Q</td>
<td>START</td>
<td>Start output</td>
</tr>
<tr>
<td>DEL</td>
<td>INTR</td>
<td>Interrupt process (SIGINT)</td>
</tr>
<tr>
<td>CTRL-\</td>
<td>QUIT</td>
<td>Force core dump (SIGQUIT)</td>
</tr>
<tr>
<td>CTRL-D</td>
<td>EOF</td>
<td>End of file</td>
</tr>
<tr>
<td>CTRL-M</td>
<td>CR</td>
<td>Carriage return (unchangeable)</td>
</tr>
<tr>
<td>CTRL-J</td>
<td>NL</td>
<td>Linefeed (unchangeable)</td>
</tr>
</tbody>
</table>
Control-D: EOF

- Typing Control-D ("End of file") causes the read request to be satisfied immediately
  - Do not wait for "enter key"
  - Do not wait for any characters at all
  - May return 0 characters

- Within the user program

  count = Read (fd, buffer, buffSize)
  if count == 0
    -- Assume end-of-file reached...
Outputting to a terminal

- The terminal accepts an “escape sequence”
- Tells it to do something special

**Example:**

```
Tells it to do something special
```

```
Each terminal manufacturer had a slightly different specification

- Makes device independent software difficult
- Unix “termcap” file
  - Database of different terminals and their behaviors.
```
# ANSI escape sequence standard

<table>
<thead>
<tr>
<th>Escape sequence</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESC [ nA</td>
<td>Move up ( n ) lines</td>
</tr>
<tr>
<td>ESC [ nB</td>
<td>Move down ( n ) lines</td>
</tr>
<tr>
<td>ESC [ nC</td>
<td>Move right ( n ) spaces</td>
</tr>
<tr>
<td>ESC [ nD</td>
<td>Move left ( n ) spaces</td>
</tr>
<tr>
<td>ESC [ m; nH</td>
<td>Move cursor to ((m,n))</td>
</tr>
<tr>
<td>ESC [ sJ</td>
<td>Clear screen from cursor (( 0 ) to end, ( 1 ) from start, ( 2 ) all)</td>
</tr>
<tr>
<td>ESC [ sK</td>
<td>Clear line from cursor (( 0 ) to end, ( 1 ) from start, ( 2 ) all)</td>
</tr>
<tr>
<td>ESC [ nL</td>
<td>Insert ( n ) lines at cursor</td>
</tr>
<tr>
<td>ESC [ nM</td>
<td>Delete ( n ) lines at cursor</td>
</tr>
<tr>
<td>ESC [ nP</td>
<td>Delete ( n ) chars at cursor</td>
</tr>
<tr>
<td>ESC [ n@</td>
<td>Insert ( n ) chars at cursor</td>
</tr>
<tr>
<td>ESC [ nm</td>
<td>Enable rendition ( n ) (( 0 )=normal, ( 4 )=bold, ( 5 )=blinking, ( 7 )=reverse)</td>
</tr>
<tr>
<td>ESC M</td>
<td>Scroll the screen backward if the cursor is on the top line</td>
</tr>
</tbody>
</table>
Graphical User Interfaces
Graphical user interfaces (GUIs)

- Memory-mapped displays “bit-mapped graphics”
- Video driver moves bits into special memory region
  - Changes appear on the screen
  - Video controller constantly scans video ram
- Black and white displays
  - 1 bit = 1 pixel
- Color
  - 24 bits = 3 bytes = 1 pixels
    - red (0–255)
    - green (0–255)
    - blue (0–255)

\[1280 \times 854 \times 3 = 3 \text{ MB}\]
Graphical user interfaces (GUIs)
X Window System

- **Client – Server architectures**
  - Based on Remote Procedure Calls (RPC)
    - Client makes a call.
    - Server is awakened; the procedure is executed.

- **Intelligent terminals ("X terminals")**
  - The display side is the **server**.
  - The application side is the **client**.
  - The application (client) makes requests to the display server.
  - Client and server are separate processes
    - (May be on the same or different machines)
**X window system**

Remote host

User space

- Window manager
- Motif
- Intrinsics
- Xlib
- X client

Kernel space

- UNIX
- Hardware

X terminal

X server

X protocol

Network
X window system

- **X-Server**
  - Display text and geometric shapes, move bits
  - Collect mouse and keyboard status

- **X-Client**
  - Xlib
    - library procedures; low-level access to X-Server
  - Intrinsics
    - provide "widgets"
    - buttons, scroll bars, frames, menus, etc.
  - Motif
    - provide a "look-and-feel" / style
  - Window Manager
    - Application independent functionality
    - Create & move windows
Disks
Disk geometry

- Disk head, surfaces, tracks, sectors ...
# Comparison of (old) disk technology

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IBM 360-KB floppy disk</th>
<th>WD 18300 hard disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of cylinders</td>
<td>40</td>
<td>10601</td>
</tr>
<tr>
<td>Tracks per cylinder</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>Sectors per track</td>
<td>9</td>
<td>281 (avg)</td>
</tr>
<tr>
<td>Sectors per disk</td>
<td>720</td>
<td>35742000</td>
</tr>
<tr>
<td>Bytes per sector</td>
<td>512</td>
<td>512</td>
</tr>
<tr>
<td>Disk capacity</td>
<td>360 KB</td>
<td>18.3 GB</td>
</tr>
<tr>
<td>Seek time (adjacent cylinders)</td>
<td>6 msec</td>
<td>0.8 msec</td>
</tr>
<tr>
<td>Seek time (average case)</td>
<td>77 msec</td>
<td>6.9 msec</td>
</tr>
<tr>
<td>Rotation time</td>
<td>200 msec</td>
<td>8.33 msec</td>
</tr>
<tr>
<td>Motor stop/start time</td>
<td>250 msec</td>
<td>20 sec</td>
</tr>
<tr>
<td>Time to transfer 1 sector</td>
<td>22 msec</td>
<td>17 µsec</td>
</tr>
</tbody>
</table>
Disk zones

**Constant rotation speed**
- Want constant bit density

**Inner tracks:**
- Fewer sectors per track

**Outer tracks:**
- More sectors per track
Disk geometry

- **Physical Geometry**
  - The actual layout of sectors on the disk may be complicated
  - The disk controller does the translation
  - The CPU sees a “virtual geometry”.
Disk geometry

physical geometry

virtual geometry

(192 sectors in each view)
Disk formatting

- A disk sector

<table>
<thead>
<tr>
<th>Preamble</th>
<th>Data</th>
<th>ECC</th>
</tr>
</thead>
</table>

- Typically
  - 512 bytes / sector
  - ECC = 16 bytes
Cylinder skew
Sector interleaving

No Interleaving

Single Interleaving

Double Interleaving
A reminder about terms/units

- **For communication...**
  - 1 Kbps = 1,000 bits per second \((10^3)\)
  - 1 Mbps = 1,000,000 bits per second \((10^6)\)
  - 1 Gbps = 1,000,000,000 bits per second \((10^9)\)

- **For disks and memories...**
  - \(K = \text{kilo} = 2^{10} = 1024\)
  - \(M = \text{mega} = 2^{20} = 1024 \times 1024 = 1,048,576\)
  - \(G = \text{giga} = 2^{30} = 1024^3 = 1,073,741,824\)
  - 1 GB = 2^{30} bytes = 1,073,741,824 bytes
Disk scheduling algorithms

- Time required to read or write a disk block determined by 3 factors
  - Seek time
  - Rotational delay
  - Actual transfer time

- Seek time dominates
  - Schedule disk heads to minimize it
Disk scheduling algorithms

- First-come first serve
- Shortest seek time first
- Scan → back and forth to ends of disk
- C-Scan → only one direction
- Look → back and forth to last request
- C-Look → only one direction
Shortest seek first (SSF)

![Diagram showing initial position and pending requests with sequence of seeks]

- Initial position
- Pending requests
- Sequence of seeks
Shortest seek first (SSF)

- Cuts arm motion in half
- Fatal problem:
  - Starvation is possible!
The elevator algorithm

- Use one bit to track which direction the arm is moving
  - Up
  - Down
- Keep moving in that direction
- Service the next pending request in that direction
- When there are no more requests in the current direction, reverse direction
The elevator algorithm

Initial position

Sequence of seeks

Time

Cylinder
Other disk scheduling algorithms

- First-come first serve
- Shortest seek time first
- Scan → back and forth to ends of disk
- C-Scan → only one direction
- Look → back and forth to last request
- C-Look → only one direction
Errors on disks

- **Transient errors v. hard errors**
- **Manufacturing defects are unavoidable**
  - Some will be masked with the ECC (error correcting code) in each sector
- **Dealing with bad sectors**
  - Allocate several spare sectors per track
- **At the factory, some sectors are remapped to spares**
  - Errors may also occur during the disk lifetime
- **The sector must be remapped to a spare**
  - By the OS
  - By the device controller
Using spare sectors

- Substituting a new sector
- Shifting sectors
Handling bad sectors in the OS

- Add all bad sectors to a special file
  - The file is hidden; not in the file system
  - Users will never see the bad sectors
    - There is never an attempt to access the file

- Backups
  - Some backup programs copy entire tracks at a time
    - Efficient
  - Problem:
    - May try to copy every sector
    - Must be aware of bad sectors
Stable storage

- The model of possible errors:
  - Disk writes a block and reads it back for confirmation
  - If there is an error during a write...
    - It will probably be detected upon reading the block
  - Disk blocks can go bad spontaneously
    - But subsequent reads will detect the error
  - CPU can fail (just stops)
    - Disk writes in progress are detectable errors
  - Highly unlikely to lose the same block on two disks (on the same day)
Stable storage

- Use two disks for redundancy
- Each write is done twice
  - Each disk has N blocks.
  - Each disk contains exactly the same data.
- To read the data ...
  - you can read from either disk
- To perform a write ...
  - you must update the same block on both disks
- If one disk goes bad ...
  - You can recover from the other disk
Stable storage

- **Stable write**
  - Write block on disk #1
  - Read back to verify
  - If problems...
    - Try again several times to get the block written
    - Then declare the sector bad and remap the sector
    - Repeat until the write to disk #1 succeeds
  - Write same data to corresponding block on disk #2
    - Read back to verify
    - Retry until it also succeeds
Stable storage

- **Stable Read**
  - Read the block from disk #1
  - If problems...
    - Try again several times to get the block
  - If the block can not be read from disk #1...
    - Read the corresponding block from disk #2

- Our Assumption:
  - The same block will not simultaneously go bad on both disks
Stable storage

- **Crash Recovery**
- Scan both disks
- Compare corresponding blocks
- For each pair of blocks...
  - If both are good and have same data...
    - Do nothing; go on to next pair of blocks
  - If one is bad (failed ECC)...
    - Copy the block from the good disk
  - If both are good, but contain different data...
    - *(CPU must have crashed during a “Stable Write”)*
    - Copy the data from disk #1 to disk #2
Crashes during a stable write

(a) Disk 1 Old, Disk 2 Old
(b) Disk 1 Old, Disk 2 New
(c) Disk 1 New, Disk 2 Old
(d) Disk 1 New, Disk 2 New
(e) Disk 1 New, Disk 2 New

Crash

ECC error
Stable storage

- Disk blocks can spontaneously decay
  - Given enough time...
    - The same block on both disks may go bad
      - Data could be lost!
    - Must scan both disks to watch for bad blocks (e.g., every day)

- Many variants to improve performance
  - Goal: avoid scanning entire disk after a crash.
  - Goal: improve performance
    - Every stable write requires: 2 writes & 2 reads
    - Can do better...
RAID

- Redundant Array of Independent Disks
- Redundant Array of Inexpensive Disks

Goals:
- Increased reliability
- Increased performance
RAID

(a) Strip 0  Strip 1  Strip 2  Strip 3  RAID level 0
    Strip 4  Strip 5  Strip 6  Strip 7
    Strip 8  Strip 9  Strip 10  Strip 11

(b) Strip 0  Strip 1  Strip 2  Strip 3  RAID level 1
    Strip 4  Strip 5  Strip 6  Strip 7
    Strip 8  Strip 9  Strip 10  Strip 11

(c) Bit 1  Bit 2  Bit 3  Bit 4  RAID level 2
    Bit 5  Bit 6  Bit 7
Spare slides
CDs & CD-ROMs

- Spiral groove

- 2K block of user data

- Pit

- Land
**CD-ROMs**

- 32x CD-ROM = 5,000,000 Bytes/Sec
- SCSI-2 is twice as fast.
**CD-R (CD-Recordable)**

[Diagram of a CD-R structure with layers labeled: Protective lacquer, Reflective gold layer, Dye layer, Polycarbonate, Substrate, Lens, Prism, Infrared laser diode, Printed label, Direction of motion, Dark spot in the dye layer burned by laser when writing.]
Updating write-once media

- VTOC = Volume Table of Contents
- When writing, an entire track is written at once
- Each track has its own VTOC
Updating write-once media

- VTOC = Volume Table of Contents
- When writing, an entire track is written at once.
- Each track has its own VTOC.
- Upon inserting a CD-R,
  - Find the last track
  - Obtain the most recent VTOC
    - This can refer to data in earlier tracks
  - This tells which files are on the disk
  - Each VTOC supercedes the previous VTOC
Updating write-once media

- VTOC = Volume Table of Contents
- When writing, an entire track is written at once.
- Each track has its own VTOC.
- Upon inserting a CD-R,
  - Find the last track
  - Obtain the most recent VTOC
    - This can refer to data in earlier tracks
  - This tells which files are on the disk
  - Each VTOC supercedes the previous VTOC
- Deleting files?
CD-RW

- Uses a special alloy

- Alloy has two states, with different reflectivities
  - **Crystalline** (highly reflective) - Looks like “land”
  - **Amorphous** (low reflectivity) - Looks like a “pit”

- Laser has 3 powers
  - **Low power**: Sense the state without changing it
  - **High power**: Change to amorphous state
  - **Medium power**: Change to crystalline state
DVDs

- **“Digital Versatile Disk”**
  - Smaller Pits
  - Tighter Spiral
  - Laser with different frequency

- **Transfer speed**
  - 1X = 1.4MB/sec (about 10 times faster than CD)

- **Capacity**
  - 4.7 GB  Single-sided, single-layer (7 times a CD-ROM)
  - 8.5 GB  Single-sided, double-layer
  - 9.4 GB  Double-sided, single-layer
  - 17 GB  Double-sided, double-layer
DVDs

0.6mm Single-sided disk

Polycarbonate substrate 1

Semireflective layer

Aluminum reflector

Adhesive layer

Aluminum reflector

Semireflective layer

Polycarbonate substrate 2

0.6mm Single-sided disk