CMPE 220

Class 14

Servers (and client/server applications)

What is an Embedded System?

Wikipedia: An **embedded system** is a computer system—a combination of a computer processor, computer memory, and input/output peripheral devices—that has a dedicated function within a larger mechanical or electrical system.

Embedded systems control many devices in common use today. Ninety-eight percent of all microprocessors manufactured are used in embedded systems.

Cars	Aircraft	Boats & Ships	Drones	Factory Equipment
Cash Registers	ATMs	Kitchen Appliances	Medical Devices	Kiosks
Home Security Systems	Home Thermostats & Control Systems	Computer Peripherals	TVs & Media Players	Toys

Complex Circuits as Embedded Systems

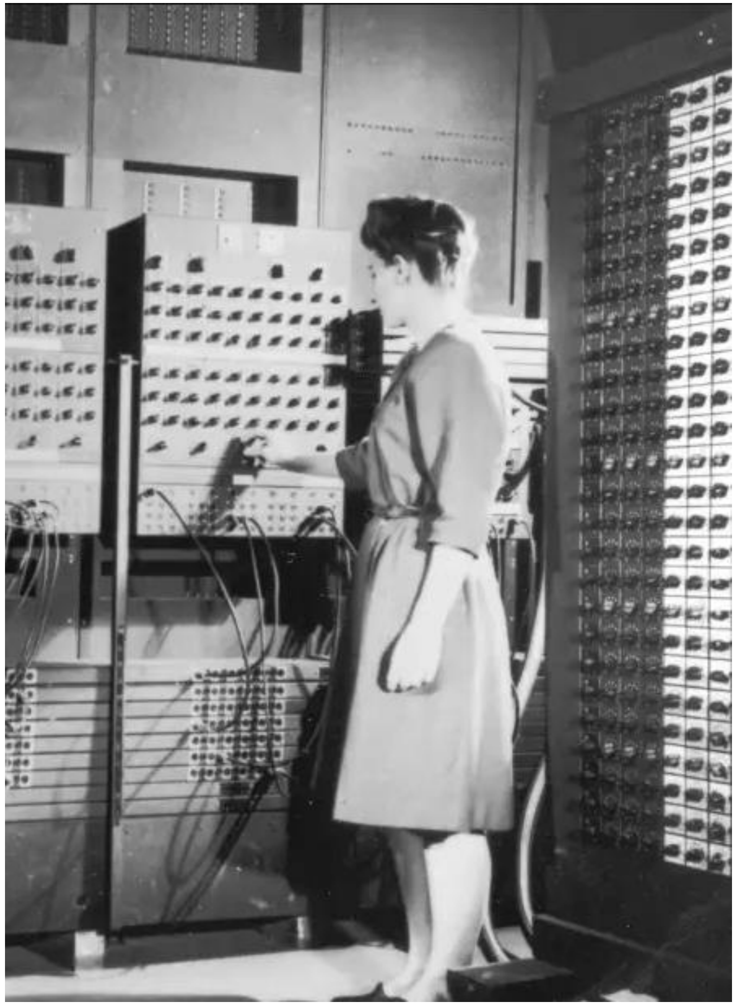
- An “embedded system” may be a very complex circuit – typically designed using a Hardware Definition Language (HDL) – which is capable of performance advanced algorithmic actions.
- *For purposes of this class, that is not what we are discussing.*
    - By embedded system, we mean a recognizable “computer,” with a processor, memory, and I/O capabilities, executing software that resides in memory.

Why Use Embedded Systems?

- Faster and cheaper to develop (standard computers & software, rather than custom hardware).
- More flexible: problems can be fixed and features added with software updates.
- *May* be cheaper to manufacture (mass-produced, off-the-shelf computer chips rather than low-volume custom circuits).

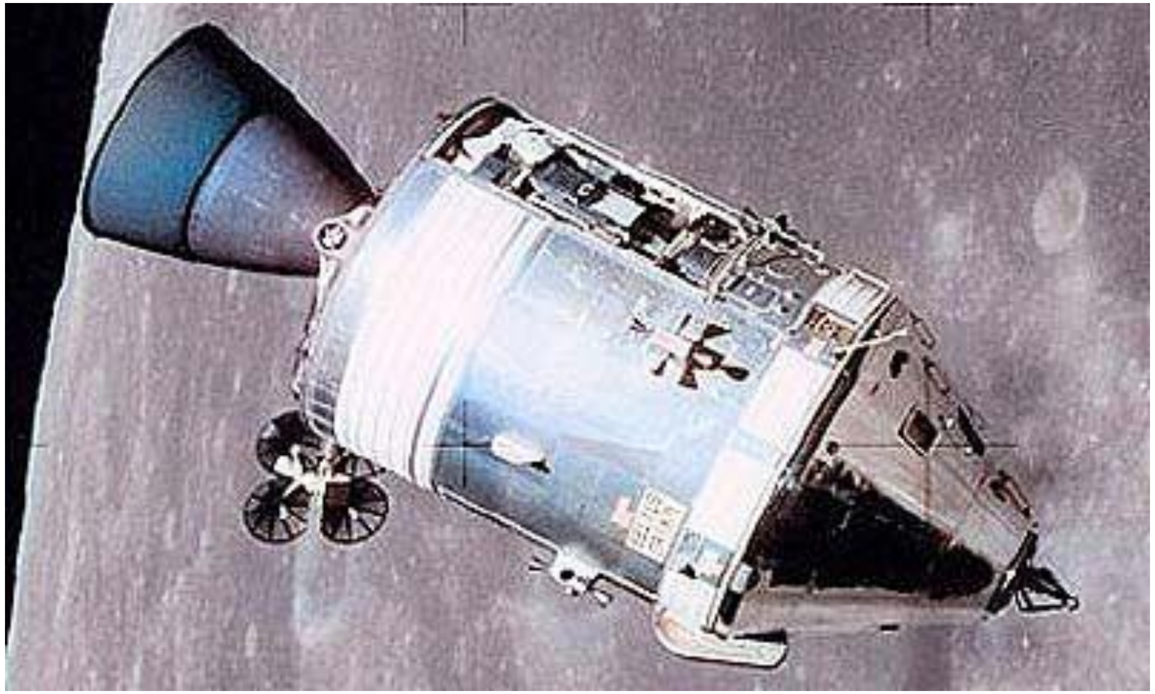
History

- One of the earliest mass-produced embedded systems was the Autonetics D-17 guidance computer for the Minuteman missile, released in 1961.
- Just as the *ENIAC* in 1945, which was delivered to the US Army to compute artillery trajectories, a significant advance in computing was driven by military requirements.



History

- A major milestone embedded system: the Apollo Guidance Computer (1965).
- Flew both the Apollo Command Module, and the Lunar Excursion Module (1969)
- *The Apollo space program and moon landing would not have been possible without the development of integrated circuits (ICs) and embedded systems*.

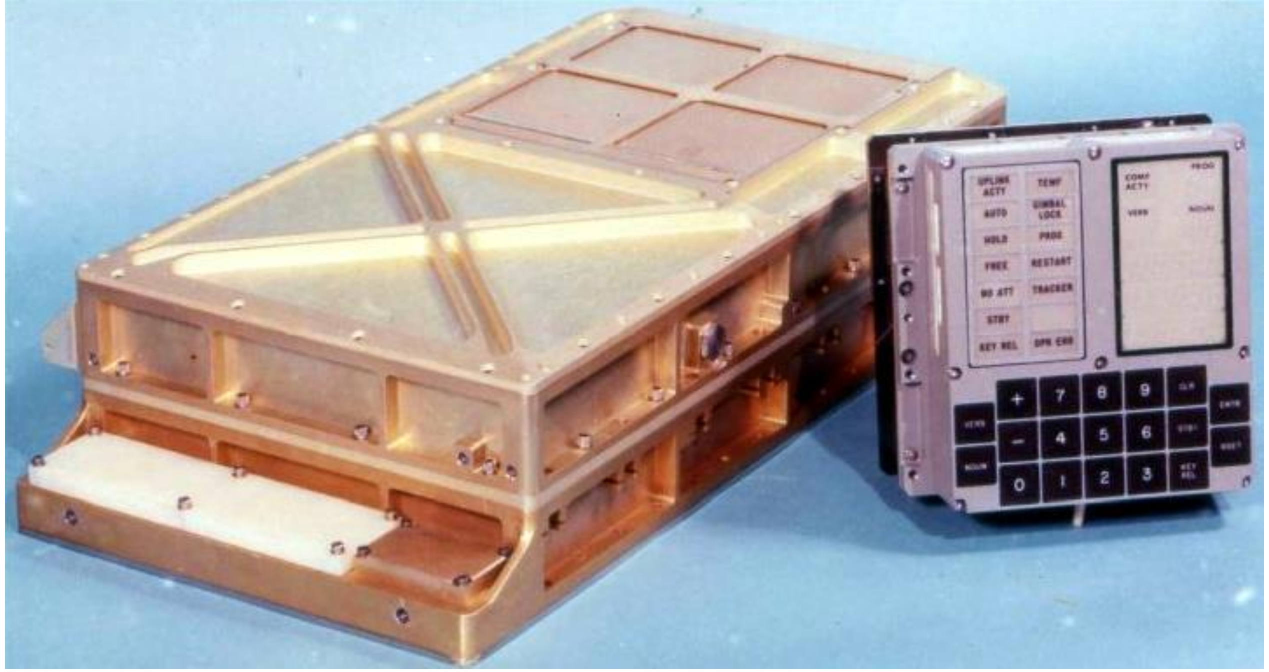


IBM System/360

- First released 1964.
- Built with discrete Transistor / Transistor Logic (TTL)
- “Modern” operating system

Apollo Guidance Computer

- *Integrated Circuits*
- 16-bit words
- 72 KB of ROM for programs
- 4 KB of RAM
- Keypad control interface
- Functions:
  - Displays System Status
  - Navigates
  - Flies the Apollo Command Module
  - Lands the LEM on the moon



More History

- In 1968, the first embedded system for a car was released. The Volkswagen 1600 used a microprocessor to control its electronic fuel injection system.
- The first microcontroller was developed by Texas Instruments in 1971. The TMS 1000 series, which became commercially available in 1974, contained a 4-bit processor.
- In 1987, the first embedded operating system, the real-time **VxWorks**, was released by Wind River Systems.

Embedded Systems Today

- The **Internet of Things** (IoT) describes the network of physical objects—“things”—that are embedded with sensors, software, and other technologies for the purpose of connecting and exchanging data with other devices and systems *over the internet*.
- These devices range from ordinary household objects to sophisticated industrial tools.
- There are an estimated 10 billion devices connected to the Internet today.
- A primary reason for IP v6

Embedded System Differences

”Traditional” Computers	Embedded Systems
General Purpose (GP)	Dedicated (Single Purpose)

Different Hardware Architecture & System Characteristics

Different Development Tools

Difference Operating Systems

Different System Software Services (Servers)

Different Architecture / System Characteristics

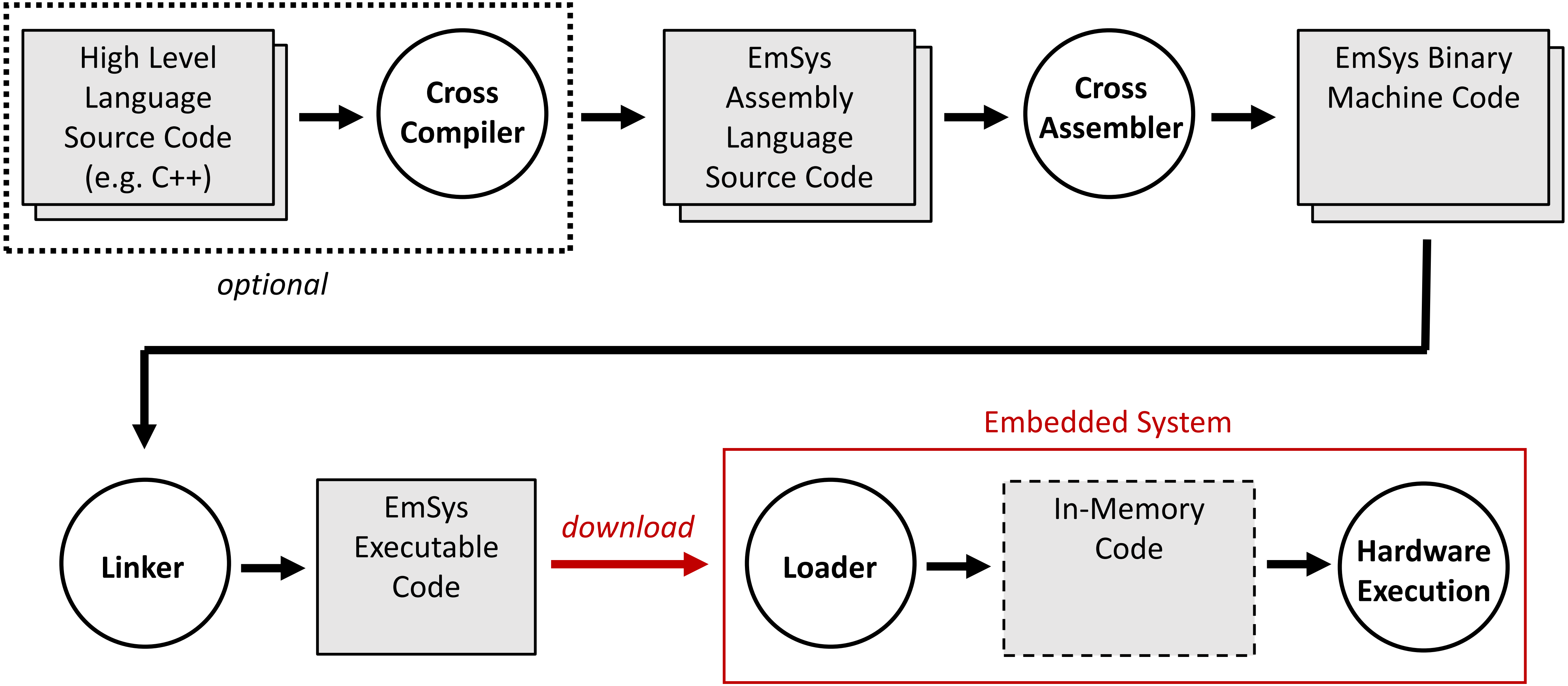
- Specialized Processors
  - Cheap (mass produced, limited processing power)
  - Small (simple architecture)
  - Low power consumption, low heat radiation (slow)
- Limited memory
  - No Memory Management Unit (MMU)
- No Disk
  - OS and Application are loaded from ROM/EPROM (non-volatile memory)
- No “traditional” I/O devices
  - No terminal, card reader, printer
  - Specialized I/O devices: sensors & controllers (A-to-D and D-to-A)
  - May have network capabilities (IoT)

Different Development Tools

Cross Development

- Cross-Compiler – Runs on a General Purpose (GP) computer; generates assembly code for an embedded computer
- Cross-Assembler – Runs on a GP computer, assembles source code for an embedded computer; emits binary object code for that computer
- Cross-Linker – Runs on a GP computer; links object files for an embedded computer
- Absolute Loader – Runs on embedded computer; loads binary code that was built on a GP computer

Building Embedded System Software



Different Development Tools - Debugging

Expanded embedded system

- A version of the embedded system with expanded capabilities (more memory, terminal interface, disk drive, etc), used for software development
- May support a full debugger, and even other development tools, which run natively on the embedded system

Different Development Tools - Debugging

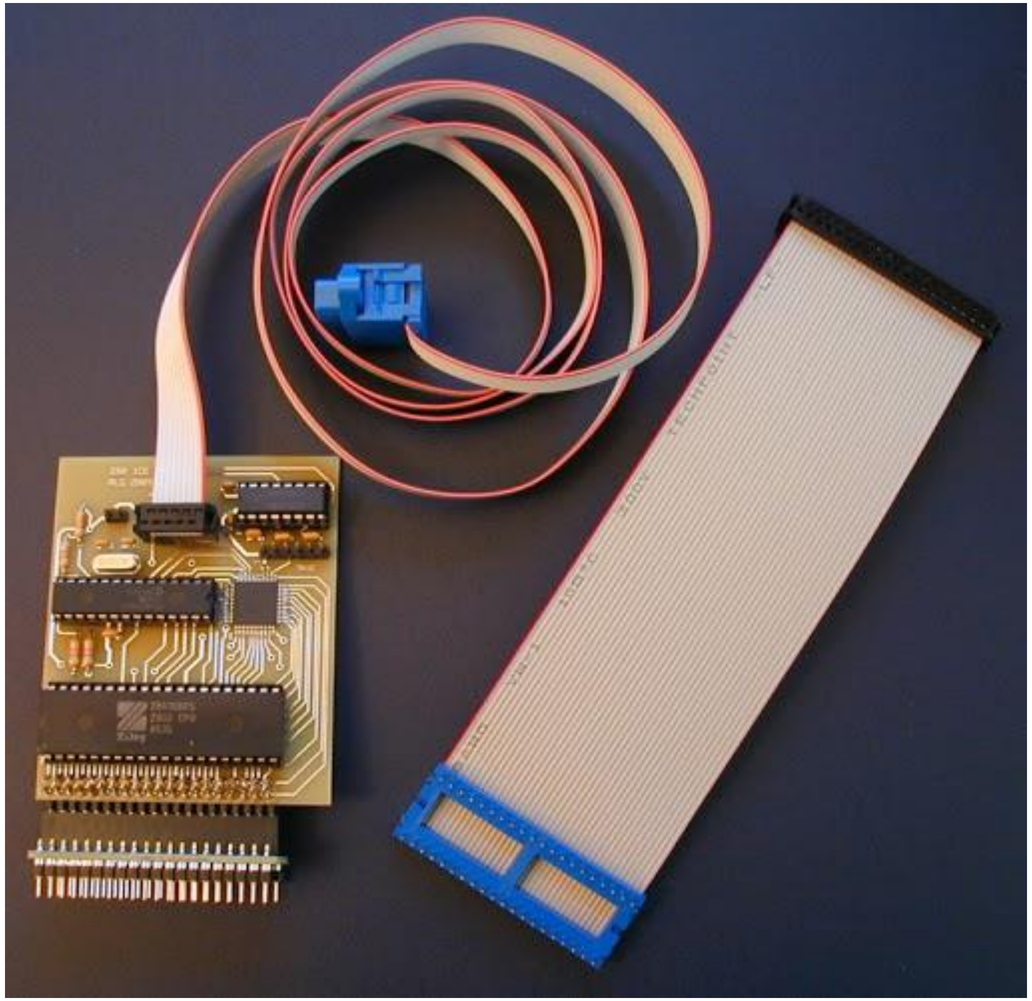
Remote Debugging

- Debugging tools run on GP computer.
- Debugger talks to a very simply “monitor” program on the embedded computer in order to examine and set memory, and set breakpoints.

Different Development Tools - Debugging

<u>In-Circuit Emulator (ICE)</u>

- A hardware interface connected to a GP computer “plugs in” to a circuit or device, completely replacing the embedded system.
- The GP computer supports a full development environment, and *emulates* the embedded system at a circuit level.



Different Development Tools - Debugging

Joint Test Action Group (JTAG)

- Interface capabilities built in to the embedded processor support external access to memory and control signals for purposes of debugging and testing.
- Interfaces can be accessed and controlled by a GP computer.
- A standard codified by the IEEE in 1990.

Different Development Tools - Debugging

Simulation

- A *simulator* running on a GP computer can execute programs written for the embedded computer.
- Interprets the instruction set.
- For useful development of embedded systems, may need to simulate I/O devices (sensors, controllers) as well.
- Typically used in very "big budget" organizations, such as car companies, aircraft companies, etc.

Languages for Embedded Systems

- Assembly Languages
- Traditional Languages: C, C++, Python, Ada (DoD)
- Specialized Languages: Rust, Go / Golang (Google)

Different Operating Systems

Traditional / General Purpose	Embedded
POSIX (Unix, Linux)	Embedded Linux
Mac OS	Windows 10 IoT
Windows	VxWorks (Wind River Systems)
Android	TinyOS (UC Berkeley, Intel)
IOS	QNX (Quantum Software Systems)
	Literally dozens of others

Is a Smartphone an Embedded System?

Low Power Consumption / Low Heat	✓
No Disk	✓
Cross-Development Tools	✓
Limited Memory / No MMU	X
No Traditional I/O Devices	X
Single Purpose	X

Not an embedded system, based primarily on the general-purpose nature of the system.

Embedded Operating Systems

1. Simplified version of General Purpose Operating System (GPOS)
   E.g. Embedded Linux, Windows 10 IoT
   - Familiar
   - Allows use of existing development tools

2. Written from the ground up for embedded systems
   - Modular (an OS framework; add and deleted features as needed)
   - Efficient
   - Supports specific needs (e.g. **Real Time**)

What Does an (Embedded) OS Do?

1	Process Management	Maybe
2	Input / Output (I/O) Management	Yes
3	Memory Management	No
4	File System Management	No
5	System Functions and Kernel Mode	Limited
6	User Interaction	No
7	Network Communications	Maybe

Security

- **Traditional:** Since applications are "friendly," there are fewer security requirements. The system doesn't need to protect itself against malicious applications.
- **Today:** IoT opens up the possibility of remote hacking!
  - In 2015, security researchers demonstrated the ability to take control of a Jeep Cherokee while driving on a highway.
  - *There is no agency that regulates the security of embedded systems.*

Real-Time Operating Systems (RTOS)

- A real-time operating system (RTOS) is an operating system (OS) intended to serve real-time applications. Real-time applications must respond to *inputs* within a *specified time*.
  - Often a requirement for embedded systems
- GP operating systems usually cannot *guarantee* a response time, for a number of reasons:
  - Too many processes might be in the process queue
  - Interrupt processing and process switching may be slow operations
  - Important code may be “swapped out” on disk
  - *However, there are specialized real-time operating systems for traditional general purpose computers; each of these problems needs to be addressed and overcome by the operating system*

Types of RTOS

- **Asynchronous** (Event Response)
  Responds to asynchronous events. For example, if a proximity alert occurs on a self-driving car, evasive action may be triggered.
  - Powerful and flexible
  - Difficult to prove & guarantee real-time response
- **Continuous** (Closed Loop)
  Constantly monitors inputs and adjusts outputs. For example, monitors temperature sensors and control fans.
  - Can prove maximum response time by computing longest code path

Further RTOS Classifications

How Rigid are the Real Time Requirements?

- **Hard:** Guaranteed response time.
  - Flight control systems, robots, drones.
- **Firm:** Range of response times acceptable. Failure to meet the desired response time is undesirable, but not catastrophic.
  - Assembly line automation.
- **Soft:** Failure to meet desired response times degrades system performance, but consequences are minimal.
  - Human-facing applications.

RTOS Adaptations

- Prioritized Scheduling
- Minimized Interrupt Latency
  - No “blocking” code
- No User/Kernel mode switches

Myth: Real Time Systems must be fast (high performance)

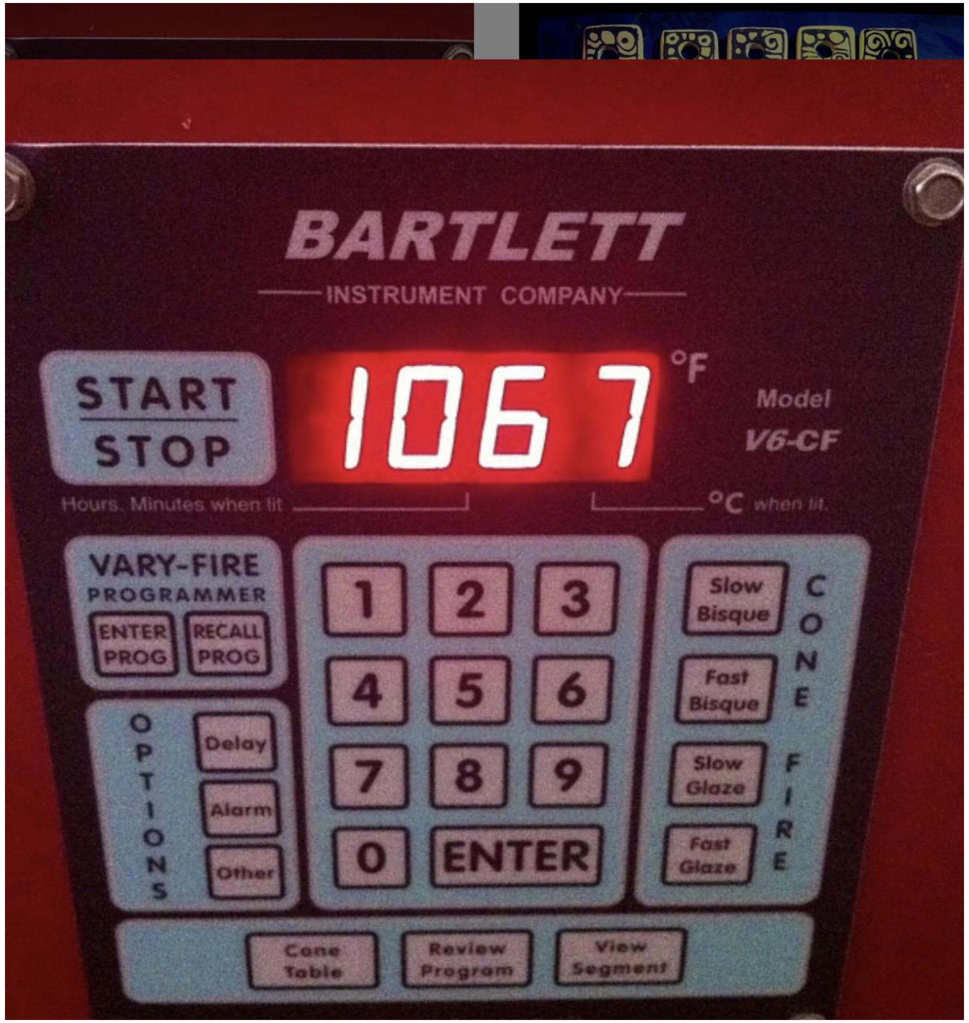
Automated assembly line: required response times may be in tens of milliseconds – but must be guaranteed.

Different System Software Services

- Tradition, general-purpose computers may run various *servers*: FTP, email, database, http/web, etc.
- Embedded systems *may* run a “network OS” and support standard network protocols
- Surprisingly, many embedded systems run a very light-weight web server
  - Limited capabilities
  - Supports http/https protocols
  - Used to provide a user interface to the system

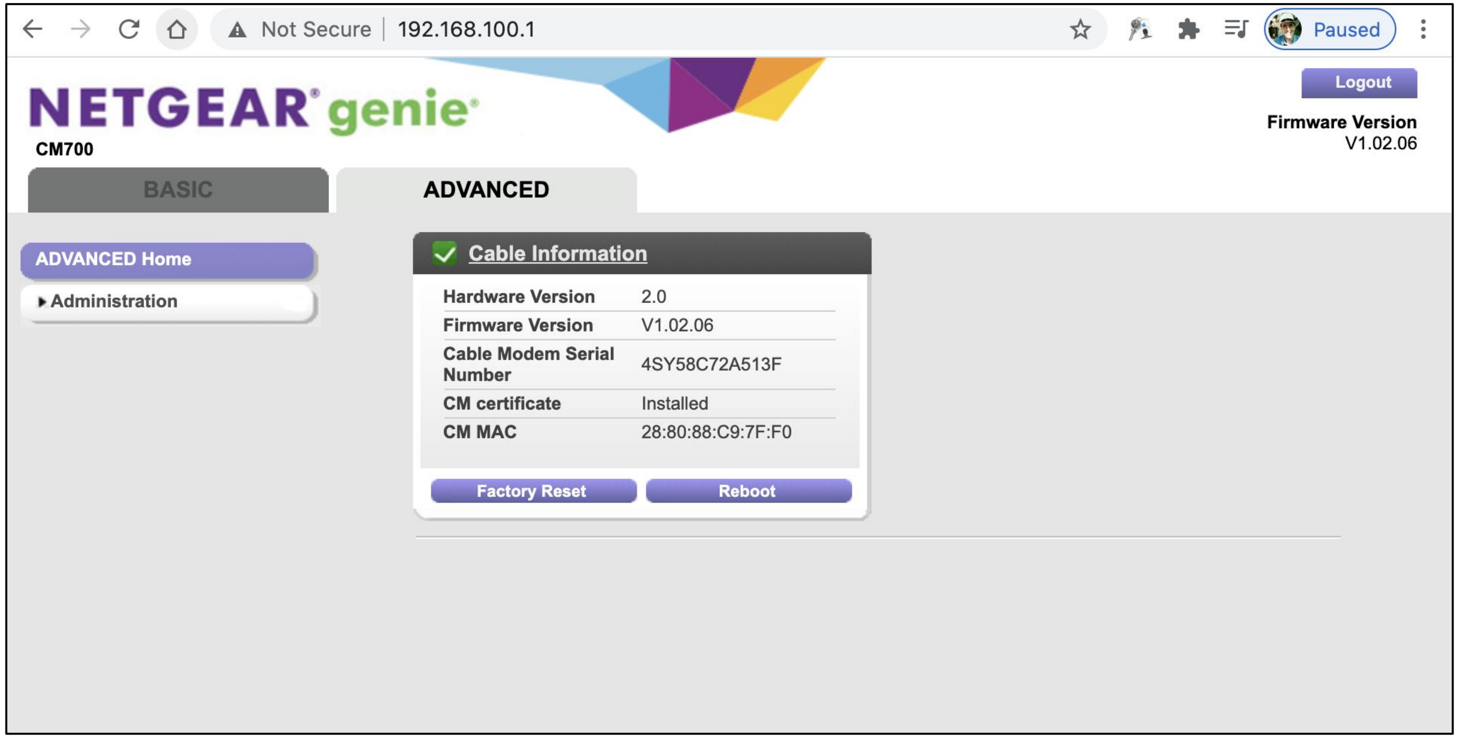
Embedded System Example: Kiln Controller

- Allows the user to program a firing cycle: when it starts, how quickly the temperature rises, the max temperature, etc.
- A specialized device made by one company (Bartlett) that is found on virtually every kiln sold in the United States, from $1,500 hobbyist models to $200,000 commercial kilns.
- Local / manual interface



Embedded System Example: Cable Modem

- A *cable modem & router* uses an embedded computer system
- Most cable modems & routers run their own web server, so you can manage and configure the router using a web browser



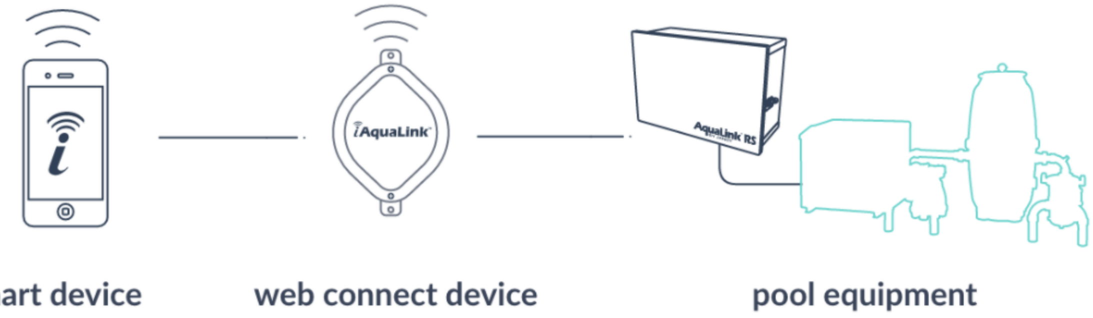
Firmware Version
V1.02.06

✓ Cable Information

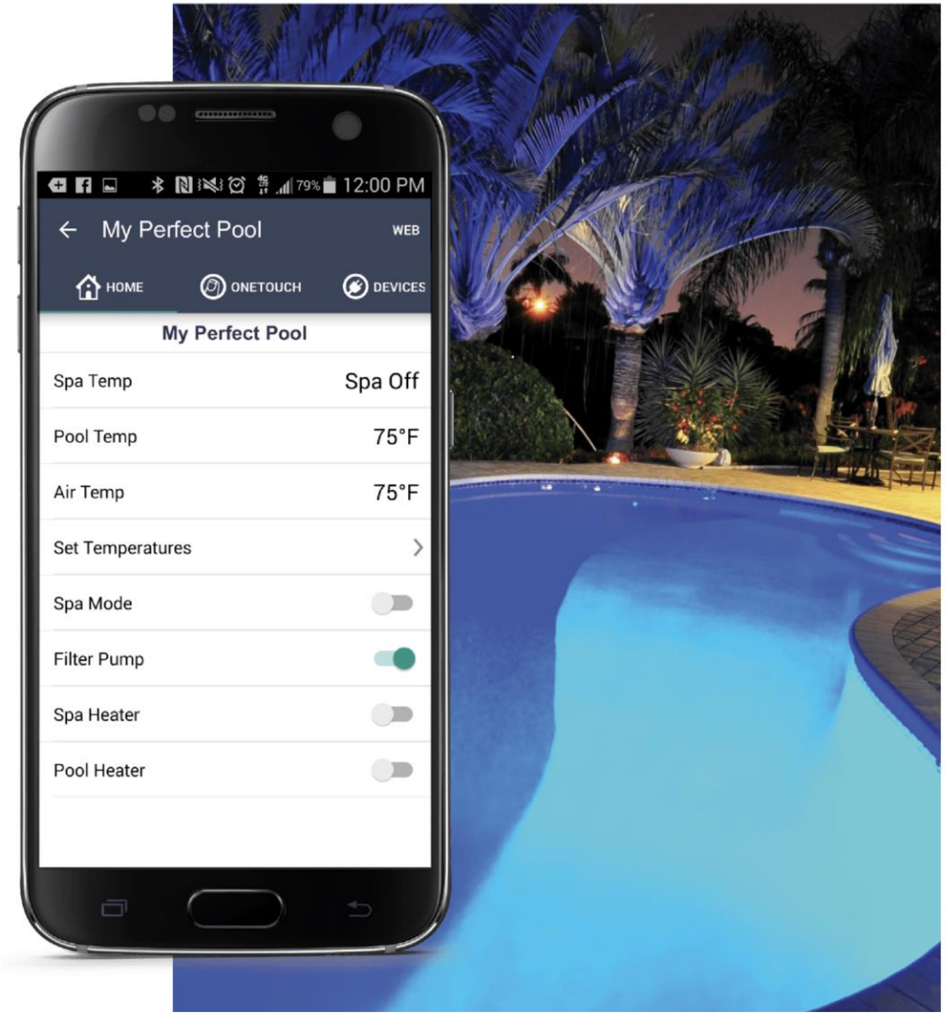
Hardware Version	2.0
Firmware Version	V1.02.06
Cable Modem Serial Number	4SY58C72A513F
CM certificate	Installed
CM MAC	28:80:88:C9:7F:F0

Embedded System Example: Pool Controller

- Uses smartphone app or browser to control pool functions remotely



- Embedded “network OS,” client-server application, and web server



My Perfect Pool

My Perfect Pool

Spa Temp	Spa Off
Pool Temp	75°F
Air Temp	75°F
Set Temperatures	›
Spa Mode	○
Filter Pump	●
Spa Heater	○
Pool Heater	○

Embedded Systems in My House

- TV(s)
- Printer(s)
- Cable Modem / Router
- Thermostat
- Air Purifier
- Roomba vacuum
- Oven
- Microwave
- Refrigerator
- Dishwasher
- Clothes Washer
- Clothes Dryer
- Bathroom Scale
- Pool Controller
- Kiln Controller
- Sprinkler System Controller
- Security System
- Video Cameras

Writing Embedded System Applications

- Roughly 3-5% of software developers work on embedded systems.
- The development process is more cumbersome.
- Libraries and system functions are limited.
- Programmers need to focus on *performance* and *efficient use of memory*.
- Programmers may need a deep understanding of specific hardware capabilities.

Arduino

- Arduino is a micro-controller* on a board, used for educational purposes and limited commercial development
- Developed in 2005 at the Interaction Design Institute Ivrea (IDII) in Ivrea, Italy
- No independent OS; library functions are linked into the application program
- Inexpensive; can be built into commercial products ($4-25)

* Micro-controller: a very simple, self-contained computer-on-a-chip

Raspberry Pi

- Raspberry Pi is a computer on a board, often used in educational environments
- Developed in 2012 by Raspberry Pi Foundation and Broadcom
- Supports monitor, mouse, and keyboard
- Can be interfaced to external sensors and devices to teach some basics of embedded systems programming
- A full, general purpose processor and operating system (Linux)
- Significant RAM (256MB to 8GB)
- No disk
- Too expensive to embed in many real-world applications ($35-75)

Alternatives

Many single-board microcontrollers & computers are available for education, development, and embedded applications.

- Teensy 3.6
- Launchpad MSP430
- Netduino N3 Wi-Fi
- SparkFun RedBoard Artemis
- Silicon Labs Wonder Gecko
- RockPi4 Model C
- NanoPi NEO3
- PocketBeagle
- Odroid-XU4
- Banana Pi M3

Following a Trend…

- Just as with every other technology we've look at this semester, embedded systems – and embedded system applications – are becoming more powerful and more complex
- At the same time, the systems are becoming smaller, cheaper, and more widespread – meaning more jobs for software developers
- Better tools!

For Next Week

- Log in to Canvas and complete Assignment 7

Midterm Next Monday

- Open Book, Open Notes
- We will **not** use the lockdown browser
- A mix of multiple choice, fill-in-the-blanks, and short-answer questions
- **Bring your laptops!**

Next Week: Midterm (Intro)

- Historical figures: Grace Hopper, Kathleen Booth, David Wheeler, Dennis Ritchie, Ken Thompson, Edsger Dijkstra, David Patterson
- Key dates: first assembler, punched cards, first binary computer, first linker, first command-line shell, make, ASCII / UTF characters, first IDE, structured programming, etc.
- What are Unix / Linux / POSIX systems”

Command Line Interfaces

- Basic shell commands (cat, ls, cd, pwd) and concept (pipes, I/O redirection)
- File permissions and ownership
- Makefiles and make rules

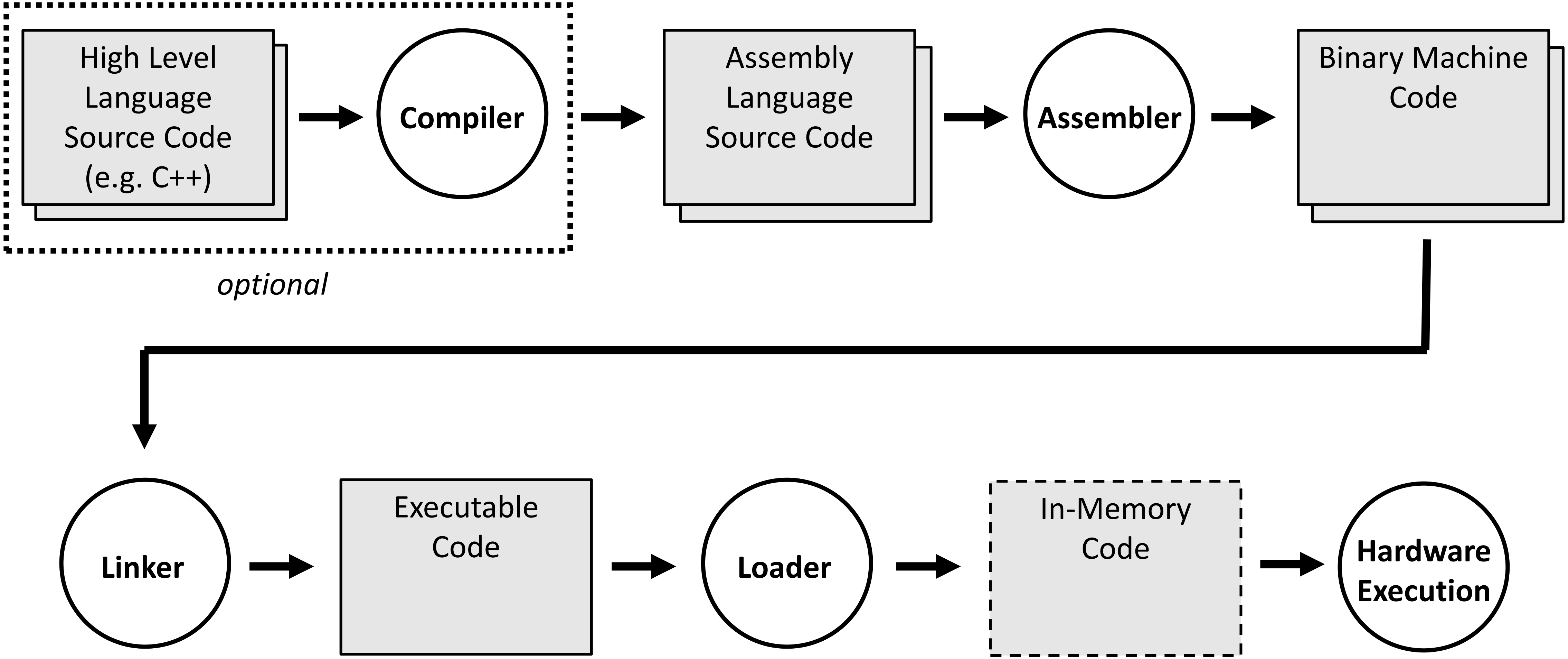
Midterm (Architecture)

- BCD arithmetic
- Character set representations (ASCII, EBCDIC, Unicode, UTF-8, UTF-16)
- Floating point representations
- Addressing modes (immediate, displacement, indirect, register, stack)
- RISC versus CISC
- Pipelining
- Microprogramming
- The SIC and SIC/XE instruction set
    - Short programs

Machine (Architectural Directions)

- Integrating functions onto processor chip
- Shifting function to MMU
- Quantum computers

Midterm (The Software Build Cycle)



Midterm (The Software Development Cycle)

- Two pass and single pass assemblers
- Relocating linkers
- Dynamic libraries
- Absolute versus relocating loaders
- IDEs
  - Smart editors
  - Version control
  - Debuggers & breakpoints
- Macro Languages and Macro-Processors

Software Development Concepts

- Structured Programming
    - Sequence
    - Selection
    - Iteration
- Pseudocode

Compilers

- Scanning (lexical processing)
  - Finite State Machines
- Parsing (syntactic processing)
- Code Generation
  - Optimization

Operating Systems

1. Process Management
    - Interprocess Communications
2. Input / Output (I/O) Management
3. Memory Management
4. File System Management
5. System Functions and Kernel Mode
6. User Interaction – (maybe)
7. Network Management

OS Details

1. Process Management
   - Interprocess Communications
   - Process Control Blocks (PCBs)
   - Schedulers
   - Dispatchers
2. Input / Output (I/O) Management
   - I/O Control Blocks
   - I/O Wait States
3. Memory Management
   - Virtual Memory
   - Memory Management Units (MMUs)

Operating Systems

4. File System Management
   - Directories
   - File system cleanup
5. System Functions and Kernel Mode
   1. User mode versus kernel mode
   2. Context switching
6. User Interaction – (maybe)
   - Shells
   - Windowing systems

Operating Systems

7. Network Management
    - Protocols
    - Ports

Types of Operating Systems

- Distributed Operating Systems
- Network Operating Systems
- Cloud Operating Systems
- Object Oriented Operating Systems

Client-Server Applications

- OSI versus TCP/IP Stack
- Protocols
- IP addressing
- IP v4 versus IP v6
- Ports
- UDP, TCP, HTTP, HTTPS
- Client-Server Applications
    - FTP
    - DNS lookup
    - Web Servers
    - Web Services and Applications
    - Email