INF-2201
01- Booting

John Markus Bjørndalen
2024

Note: these lectures are "work in progress"

- The general pattern will be:
  - Identify problem with solution so far
  - What can we do to solve it?
  - How can we build that solution?
  - Identify abstractions
- Then iterate towards a modern computer and operating system

Starting from scratch

Before we define what we mean by an operating system, let's look at what leads to building one and why we need it

- Slightly rewriting history to show how modern computers and operating systems can be developed from following simple systems, problems and solutions to these problems.
- Will contain references to real computers (work in progress)

We will skip som part of history

- Mainframes and minicomputers: we mainly use microcomputers as examples here
- CPU instructions, the idea of computing, ... : we assume some basic computer architecture background
- Some historical side notes (rabbit holes):
  - Jaquard loom (interchangable punch cards inspiring early computing and binary)
    - https://www.youtube.com/watch?v=pzYucg3Tmho
  - Babbage and the differential engine (see also Ada Lovelace)
    - https://www.youtube.com/watch?v=KBuJqUfO4-w
  - Turing Machine (think about this when we run from paper tapes)
    - https://www.youtube.com/watch?v=DILF8usqp7M&list=PLzH6n4zXuckrEzV0CB1xXbSdsP_a7VUoK&index=3
    - The above is a one video from a playlist

Running your first program

- Simple computer (CPU and memory)
- Turn on computer. What does it do?
  - Needs instructions somewhere the CPU can fetch
  - Needs a clock that can tick to drive the internals of the CPU (memory fetches, Program Counter (PC) updates, ...)
  - How do we get instructions and data into the computer?

Running your first program

How do we get instructions and data into the computer?

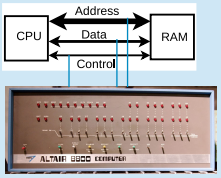
- One method: add a device to read and write directly into computer memory
- Here: Altair 8800 front panel

https://en.wikipedia.org/wiki/Altair_8800

Running your first program

First attempt: write program into memory before starting clock

- Altair 8800 or IMSAI 8080 front panel, switches.
- Punch in address with switches (binary code), press "store to location register"
- Punch in 1 byte, press store (possibly automatically updating the location register)
- Continue until you have put your data into memory



https://en.wikipedia.org/wiki/Altair_8800

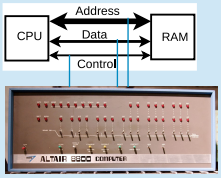
Running your first program

Issues:

- Manually punch in
- Takes time
- Have to convert data and programs to binary
- Easy to make mistakes

A good thing:

- You get a single-step debugger for free!

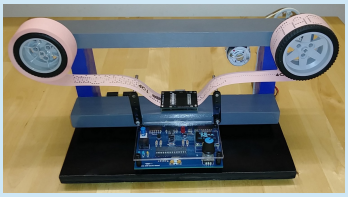


https://en.wikipedia.org/wiki/Altair_8800

Enter the paper tape reader

Problem: too cumbersome to punch in everything, and very easy to make mistakes

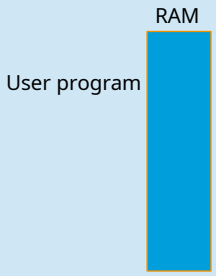
- Solution: wire up the data input to a paper tape reader
  - Can read the bytes much faster
  - Reuses the "address counter" that was used previously
- A paper tape writer can be used to store programs



https://github.com/dhansel/PaperTapeReader

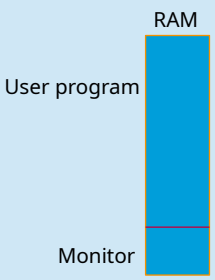
Portability and simpler programming

- Problem: portability.
  - New revisions of computers change hardware slightly or introduce new features
  - Every program needs common routines (print, input, I/O devices, self-check of computer, ...)



Portability and simpler programming

- Problem: portability.
  - New revisions of computers change hardware slightly or introduce new features
  - Every program needs common routines (print, input, I/O devices, self-check of computer, ...)
- Solution: provide a compatibility layer
  - Load a small program first in a fixed location in memory that takes care of interfacing with hardware
  - Then load your actual programs from a separate tape.
    - These can then call functions in fixed memory locations / entry points to use the functions from the first tape
  - Eearly computers would call this first program a "monitor"
  - Think about this as an early Hardware Abstraction Layer (HAL)



Historical sidenote

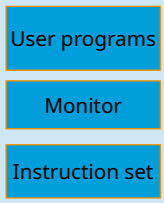
- Altair 8800 (1974)
  - Intel 8080
  - Serial interface for console (typically teletype or terminal)
  - 1KByte or 4KByte memory cards
  - Various cards for I/O, BASIC (programming language) etc.
  - Origin of the S-100 bus
- IMSAI 8080
  - Altair clone
  - Used in the movie "War Games" (1983)

Source: https://en.wikipedia.org/wiki/Altair_8800

Source: https://en.wikipedia.org/wiki/IMSAI_8080

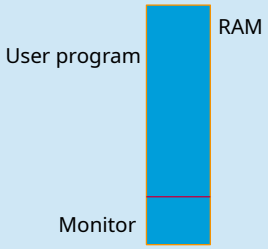
A note about abstractions

- Terms to consider
  - Abstraction
  - Layer of indirection (you can often use this when you implement an abstraction)
- Some abstractions up to this point
  - Instruction set: (abstracts away the construction of the CPU to a "virtual CPU")
    - Side note: look at retro computing and emulation (both CPU and systems)
  - Monitor: abstracts away some of the functionality of a computer
    - Serial port for I/O
    - Other I/O devices
    - One way to look at the Monitor: it creates a "virtual machine"
      - Not to be confused with Virtual Machines (VM) that will be introduced later.



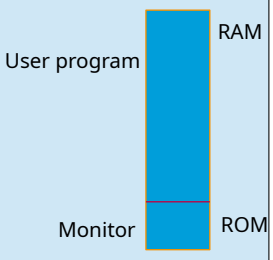
Next question: how to load monitor

- From tape: need two tapes (monitor + program) every time you want to run a program



Next question: how to load monitor

- From tape: need two tapes (monitor + program) every time you want to run a program
- Solution:
  - Use a read-only memory that does not lose information when powered off (ROM vs RAM).
  - Map the ROM into a fixed location in memory (instead of RAM)
  - Computer starts executing (8080 and Z80 : at addr 0) from ROM, sets up the computer
  - Monitor can provide a simple menu
  - Monitor could also support loading from external devices (Floppy disk, casette tape, external tape readers, ...)
    - No longer need to always load from tape reader



Side note: game consoles and game cartridges

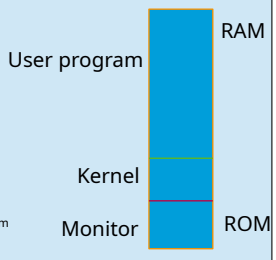
Image: Fairchild Channel F
https://www.fastcompany.com/3040889/the-untold-story-of-the-invention-of-the-game-cartridge

- A game cartridge can be a basic ROM chip that is mapped into the address space of the CPU when it is slotted into the game console
  - It could also provide flash storage and extra RAM
- Think about it as a replaceable ROM
- Some history about the Atari 2600
  - https://spectrum.ieee.org/atari-2600

Image: https://spectrum.ieee.org/atari-2600

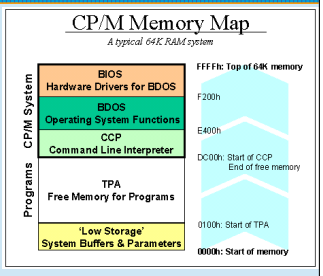
Next question: is the monitor enough?

- Monitor can load programs, but hard to update
- Solution:
  - CPU starts executing from a small monitor in ROM
    - Basic functionality for I/O and hardware abstractions
    - In later systems, the monitor is replaced with a BIOS (Basic Input/Output System)
  - Monitor then loads a part 2 with more functionality for dealing with hardware, I/O, resources etc
    - Can load from any storage system
    - Easily updated to new versions
    - This is basically what we later will develop into an Operating System Kernel.



Historical note: CP/M and BIOS

- Basic Input/Output System (BIOS)
  - The portability layer of CP/M
  - Porting to new computers: in principle, only the BIOS would need to be specific for the computer.
- Complexity
  - The BIOS must be mapped into addr 0 from the start (CPU starts executing there)
  - The operating system stores information from addr 0
  - => need to move the BIOS before executing the operating system
    - One option: let the BIOS
      - boot computer
      - load CP/M.
      - Then the relevant part of the BIOS can be copied to higher addresses before switching out the BIOS ROM.



https://obsolescence.wixsite.com/obsolescence/cpm-internals

Booting on an old PC

- Computer turns on in **16-bit mode** and starts running BIOS
- BIOS loads a "Boot Loader" from a designated storage device (typically your first hard disk)
  - The boot loader is a tiny piece of code that has one task: find and load the operating system kernel
  - After the kernel is loaded, the boot loader jumps to a predefined start location in the kernel and the operating system is running
  - The operating system can choose to use the BIOS to handle I/O, but can also choose to interface directly with hardware
- This is similar to CP/M, but adds the boot loader
- To get to 64-bit mode, the operating system has to

Booting a newer PC through legacy mode

Here be dragons:

- Load and start 16-bit operating system using BIOS
  - Real mode
  - Limited address space
    - Segmented memory: 16 bit addr => 64KB segments. Combine with segment register to create 20-bit addrs (1MB).
- Kernel switches to 32-bit mode. Rough description:
  - Protected mode (introduced with 80286)
  - Historical baggage: need to fiddle with keyboard driver to enable pin 20 on the address bus
  - Set up memory mapping (more about this later in the course): global descriptor table (GDT).
  - Enable 32-bit by setting Protection Enable bit in control register 0 (CR0).
  - Execute "long jump" (ljmp)
- Kernel can now switch to 64-bit mode:
  - Long mode
  - https://wiki.osdev.org/Setting_Up_Long_Mode
- Problems:
  - Complicated due to old baggage (backwards compatibility) – it's actually much more convoluted than the description above
  - Modern computers are dropping support for this (can no longer boot like this on many computers)
    - Booting directly using UEFI instead of BIOS
    - Future: dropping support for 16 and 32-bit mode:
      - https://www.intel.com/content/www/us/en/developer/articles/technical/envisioning-future-simplified-architecture.html

Booting a modern PC - implications

Implications for our OS kernel

- P1 has to change – it no longer boots on modern hardware
- UEFI takes care of initialisation: our kernel can start directly from 64-bit mode
  - Security mechanisms may cause issues
- Newer hardware is more complicated and may require more drivers (ACPI, ...)
- May need to consider switch to simpler architectures