ECE 531 – Advanced Operating Systems
Lecture 4

Vince Weaver

https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

10 September 2025

Announcements

- Don't forget HW#1
- Pis will hopefully be ready Friday

Booting a System

- Why is it called booting?

- Most likely source is the idea of “Pulling oneself up by ones bootstraps”, i.e., getting somewhere by starting with nothing

Simple Booting

- Simplest systems have code in ROM.
  The CPU initializes, points the Program Counter to a known location, and starts executing.

- The STM32L boards in ECE271 do something similar; code is in flash, reset vector (at offset 0) points at code to start. press reset, runs reset vector, up to you to do everything else.

Boot Firmware

Provides booting, configuration/setup, sometimes provides rudimentary hardware access routines.
Kernel developers like to complain about firmware authors.
Often mysterious bugs, only tested under Windows, etc.

- BIOS – legacy 16-bit interface on x86 machines
- UEFI – Unified Extensible Firmware Interface
  ia64, x86, ARM. From Intel. Replaces BIOS
- OpenFirmware – old macs, SPARC
- LinuxBIOS

Boot Firmware Aside

- Tell story of doing spread spectrum on Progear webpad but early in BIOS so doing i2c in assembly language with no RAM configured yet

Firmware Boot Process

- Low-level code (often written in assembly language) that initializes the system.
- Often in ROM/EEPROM/FLASH
- Boot firmware initializes system.

Firmware Boot Initialization

- Init RAM? Set it up (often over i2c), clear out random or old contents (if a soft reboot). This part operates without memory or stack to use, tricky.
- Init other hardware. I/O, serial ports, keyboard, display, etc.
- Load code to boot.

Boot Code Location

- Hard-disk (SSD or spinning rust)
- Removable: floppy-disk, CD/DVD, USB key, SD-card
- network (PXE)
- Old days: tape, paper-tape, console switches?

Other Boot Firmware Concerns

- Might have other interfaces: boot selection/configuration screen?
- Some firmware provides routines for hardware to use, for things like accessing disks, writing to screen, reading keyboard, initializing security, etc.
- Firmware development is hard. Not all corner-cases well tested (can it boot Windows? Ship it). Kernel and Firmware devels have antagonistic relationship.

Booting on x86

- BIOS original firmware. 16-bit. Dates back to CP/M days. Provided booting and a library for accessing I/O. (MS-DOS a thin layer over BIOS).
- These days EFI and uEFI replacing it, 32/64-bit. Written in higher level language.
- Firmware provides other interfaces, like power management, ACPI, device enumeration, etc.
- x86 firmware can use SMM mode which allows secret/hidden code running behind the scenes for things

like hardware emulation (USB keyboards) and power management.

Booting on x86 / Historical

- Firmware traditionally loaded a 512Byte bootsector from floppy/hard disk (last two bytes 0x55 0xAA) to 0x7c00 and jumped to it. This “first stage” then had enough code to load a more complex “second stage”
- Demoscene: people write demos or entire games that fit in one bootsector

x86 Bootloader

- The bootloader (GRUB is common on x86 Linux) loads OS
- Linux has had many, many other bootloaders over years
- Other OSes like Windows and OSX also have them, more subtle
- Can Provide nice graphical interface often (to select images)
- Can have console for command line arguments and browsing kernel images.

Loading Linux

- Linux is usually on disk, sometimes a separate boot partition. Complicated because blocks might not be contiguous on disk.
- Some Linux images can be loaded directly, without need of bootloader.
- Linux image itself can be complex

Linux Image

- "vmlinux" (why called that? historical, unix, vm unix)
- decompresser and compressed image (zImage, bzImage, uIMage, etc)
- When building, the kernel image is taken, stripped, compressed. piggy "piggyback" code put on, as well as decompresser. Originally floppy boot code stuck on beginning as well.
- Different entry points. On x86 BIOS boots into 16-bit. EFI and bootloaders can jump into 32/64

- So optionally boots in 16-bit mode. Switches to 32-bit mode. If 64-bit, optionally switch to 64-bit Decompressed kernel to 0x10 0000 (might have to move decompress code). (above 1MB. Why? 640k) What about initrd?
- Jump to startup_32 / startup_64 function
- 16-bit code handles various stuff, gets memory size from BIOS, etc
- 32/64 relies more on boot loader. Has specification for how registers set up, etc.
- relocates decompression code if needed. Sets up stack,

clears BSS, Decompresses.

- relocate if needed. why? randomization is one.
- Memory map. Virtual mem. First 896M of physical mem mirrored in top of 32-bit. Why? So kernel can easily copy to/from. Can convert kernel virt to phys with just subtraction. Complicated if more than that much RAM, have to copy around. HIGHMEM.
- space above for vmalloc
- somewhat more complicated 64-bit
- kernel just an ELF executable

Linux Userspace Transition

- Starts Userspace program “init” (old days simple program and shell scripts, these days “systemd”)
- Sometimes an “initrd” is included too that has enough drivers to get Linux going and a very minimal filesystem to help with booting before disks/filesystem ready.

Disk Partitions

- Master Boot Record, Boot Sector
- Followed by partition table
- Way to virtually split up disk.
- DOS – old way, in MBR. Start/stop sectors, type
- Types: Linux, swap, DOS, etc
- Had 4 primary and then more secondary
- Lots of different schemes (each OS has own, Linux supports many).
- UEFI/GPT (GUID) more flexible, greater than 2TB

Bootloaders on ARM

- The most common is uBoot
- uBoot – Universal Bootloader, for ARM and other embedded systems
- Almost like minimal OS
- More of a challenge to write a bootloader for a widely nonstandardized architecture like ARM. (Why is ARM so nonstandardized?)

Uboot Booting

- Most non-Pi ARM devices, ARM chip runs first-stage boot loader (often MLO) and second-stage (uboot)
- FAT partition
  Why FAT? (Simple, Low-memory, Works on most machines, In theory no patents despite MS's best attempts (see exfat))
  The boot firmware (burned into the CPU/ROM) is smart enough to mount a FAT partition

Booting on typical ARM/uboot

- vmlinux. strip. compress. piggy / piggyback. convert to zImage. mkuimage converts to uimage suitable for booting with uboot
- No bios really. Bootloader provides all info.
- Device Enumeration: Device Tree provides config info for hardware (memory size, interrupts, what hardware is there). This allows kernel that will run on many ARM boards (PI, beaglebone, pandaboard, etc) rather than having to have a different hard-coded kernel for each

possible platform.