Genesis:
From Raw Hardware to Processes

Sarah Diesburg
Operating Systems
CS 3430
How does it all begin?

- How we go from nothing to the operating system
- How the operating system starts up processes (services)
Booting Sequence

- The address of the first instruction is fixed
- It is stored in read-only-memory (ROM)
  - Why ROM instead of RAM?
Booting Procedure

- ROM stores a **Basic Input/Output System (BIOS)**
  - BIOS contains information on how to access storage devices
BIOS Code

- Performs Power-On Self Test (POST)
  - Checks memory and devices for their presence and correct operations
  - During this time, you will hear memory counting, which consists of noises from the floppy and hard drive, followed by a final beep
After the POST

- The *master boot record (MBR)* is loaded from the *boot device* (configured in BIOS)
- The MBR is stored at the first logical sector of the boot device (e.g., a hard drive) that
  - Fits into a single 512-byte disk sector (*boot sector*)
  - Describes the physical layout of the disk (e.g., number of tracks)
After Getting the Info on the Boot Device

- BIOS loads a more sophisticated loader from other sectors on disk
- The more sophisticated loader loads the operating system
Operating System Loaders

- **GRUB** *(GRand Unified Bootloader)*

```
GNU GRUB  version 0.97  (638K lower / 2095040K upper memory)

Debian GNU/Linux, kernel 2.6.26-2-686
Debian GNU/Linux, kernel 2.6.26-2-686 (single-user mode)
```

Use the ↑ and ↓ keys to select which entry is highlighted. Press enter to boot the selected OS, 'e' to edit the commands before booting, or 'c' for a command-line.

The highlighted entry will be booted automatically in 4 seconds.
More on OS Loaders

- Is partly stored in MBR with the disk partition table
  - A user can specify which disk partition and OS image to boot
  - Windows loader assumes only one bootable disk partition
- After loading the kernel image, OS loader sets the kernel mode and jumps to the entry point of an operating system
Kernel Mode?

- Two hardware modes: kernel mode and user mode
  - Implemented as a single bit
  - Some privileged instructions can only be run in kernel mode to protect OS from errant users
  - Operating system must run in kernel mode
Booting Sequence in Brief

- A CPU jumps to a fixed address in ROM,
- Loads the BIOS,
- Performs POST,
- Loads MBR from the boot device,
- Loads an OS loader,
- Loads the kernel image,
- Sets the kernel mode, and
- Jumps to the OS entry point.
Booting Sequence Visualized
Linux Initialization

Set up a number of things:
- Trap table
- Interrupt handlers
- Scheduler
- Clock
- Kernel modules (hardware and software drivers)
- …
- Process manager
Process 1

- Is instantiated from the *init* program
- Is the ancestor of all processes
- Controls transitions between *runlevels*
-Executes startup and shutdown scripts for each runlevel
Runlevels

- Level 0: shutdown
- Level 1: single-user (command-line only)
- Level 2 - 5: the GUI (called “X” in Linux)
  - These levels are typically duplicated
- Level 6: reboot

*These runlevels map to /etc/rc\text{x}.d, where \text{x} is 0-6 or S for “Single User”*
Runlevels

- SysV (“System 5”) runlevels meant that you would process them this way:
  - Booting: start with 1, go up each run level to default stop level, executing scripts that start with “S” for “start”
  - Shutdown: start at your current runlevel, go down one at a time until you reach 0, executing scripts that start with “K” for “kill”
Runlevels

- Systemd is the newer system, although a lot of the old runlevel stuff is still preserved.
- Instead of thinking about numbers, you think about labels mapped to numbers
  - Easier? Hmmm....
    - Run level 0 is matched by poweroff.target (and runlevel0.target is a symbolic link to poweroff.target).
    - Run level 1 is matched by rescue.target (and runlevel1.target is a symbolic link to rescue.target).
    - Run level 3 is emulated by multi-user.target (and runlevel3.target is a symbolic link to multi-user.target).
    - Run level 5 is emulated by graphical.target (and runlevel5.target is a symbolic link to graphical.target).
    - Run level 6 is emulated by reboot.target (and runlevel6.target is a symbolic link to reboot.target).
    - Emergency is matched by emergency.target.

https://www.tecmint.com/change-runlevels-targets-in-systemd/
Runlevels

- You can start and stop services with the systemctl command

```
# systemctl start [name.service]
# systemctl stop [name.service]
# systemctl restart [name.service]
# systemctl reload [name.service]
$ systemctl status [name.service]
# systemctl is-active [name.service]
$ systemctl list-units --type service --all
```

https://www.linux.com/learn/understanding-and-using-systemd
Windows?
Process Creation

- How does the init process create all these other processes (services) that run independently?

- Via the *fork* system call family

Before we discuss process creation, a few words on system calls…
System Calls

- **System calls** allow processes running at the *user mode* to access kernel functions that run under the *kernel mode*.

- Prevent processes from doing bad things, such as:
  - Halting the entire operating system
  - Modifying the MBR
UNIX System Calls

- Implemented through the *trap* instruction
- Causes an interrupt and allows the OS to switch to kernel mode
- From there, it looks up system call and runs it
More on Fork

- **Fork** is a system call to create a new process
  - What does each process have (two things)?

- Two processes may be bulky
  - Can create multiple *threads* instead
  - For now, we will concentrate on processes