# Introduction to Linux

W4118 Operating Systems I

columbia-os.github.io

Credits to Jae and Jason

# Booting Linux

Is the OS the first program to run?

# Booting before Linux

Is the OS the first program to run? – **NO**

**STEP 0.1:** Firmware code (BIOS/UEFI) performs a power-on self-test, detects the available RAM, and pre-initializes the CPU and hardware.

# Booting before Linux

Is the OS the first program to run? – **NO**

**STEP 0.1:** Firmware code (BIOS/UEFI) performs a power-on self-test, detects the available RAM, and pre-initializes the CPU and hardware.

**STEP 0.2:** The firmware checks the **M**aster **B**oot **R**ecord of hard disks (and peripherals) in some order to identify a bootable partition.

# Booting before Linux

Is the OS the first program to run? – **NO**

**STEP 0.1:** Firmware code (BIOS/UEFI) performs a power-on self-test, detects the available RAM, and pre-initializes the CPU and hardware.

**STEP 0.2:** The firmware checks the **M**aster **B**oot **R**ecord of hard disks (and peripherals) in some order to identify a bootable partition.

**STEP 0.3:** The bootloader in the MBR reads some configuration files, possibly presents a menu to the user, loads the kernel from the disk, and launches it.

# Booting Linux

1. *kernel program starts*
2. *configure hardware, including virtual addressing*
3. *start the 0th process*
4. *initialize the scheduler*
5. *0th process "forks" 1st process to run kernel_init*
6. *0th process become the idle task, calls schedule*
7. *schedule context switches to some other process from runqueue*
8. *1st process will run and execve to "/init" or "/sbin/init" (process 1)*
9. *init is a regular program, e.g., systemd*
10. *at some point we return to the kernel again*

# Running Linux

Run # **ps afx** to see the process tree

1. *Check what is running – both kernel and user processes*
2. *Check the parents of the user and the kernel processes*
3. *Check the differences between ssh login and terminal*

# Parents and Children

Run **shell1.c**

1. What happens if the parent process terminates before its children?

When a parent process dies before a child process, the kernel knows that it's not going to get a wait call, so instead it makes these processes "orphans" and puts them under the care of init (remember mother of all processes). Init will eventually perform the wait system call for these orphans so they avoid becoming zombies.

2. What happens if a child process has terminated, but the parent never calls **`waitpid()`**?

Zombie process, there is still state around.

# Signals

Run **signal0.c**

How do you terminate it?

kill -SIGTERM <pid>

Run **signal1.c**

How do you terminate it?

kill -SIGKILL <pid> – SIGKILL cannot be caught

# Kernel Modules

```
insmod module.ko
rmmod module
```

How do we cause kernel panic?

- Division by zero, segfault

Why didn’t the system die?

- Linux kernel can handle single faults
- Linux kernel can handle double faults – x86 architecture has a specific double fault exception (vector number 8)
- Triple faults are fatal