Lecture 4

Juni Kim

6.S913

January 26, 2026

From Previous Lectures

- Build shell scripts with predictable behavior in spite of:
  - Working Directory
  - Undeclared environment variables
  - State of the filesystem
- The kernel unpacks an initramfs, which is purely in memory. We mount our disk, then `switch_root`.

Pointers and Strategies

- Debugging things in this class can be incredibly difficult!
- Are your archives laid out correctly?
- Are all your set paths correct?
- Are your binaries not broken?
- Is your kernel not misconfigured (no networking, bad cmdline, ...)?
- Are your config files not misconfigured?
- Are your environment variables correct?
- Are your file permissions correct?

Always think:

- Where are the possible points of failure in system bring up?
- How could I try checking whether this is a point of failure?
- Is there something evil hiding in the logs? (When make or configure fails)

Note Grading Environment Permissions!

- Your entire filesystem will be locked down in grading.
- `/workspace` will be read-only
- Only `$DIST` will be readable.
- You should be doing out of tree builds.
- Your main build pipeline should not have interactive steps.
- There are scripts on the repo to help you test this! Consult the README for more details.

Final Note Re Integrity

Permitted:

- Discussing ideas, design choices, and debugging strategies
- Reading documentation, blog posts, and reference material
- Using AI tools to explain errors or suggest approaches

However, the assignment must reflect your own reasoning. Do not:

- copy another student's root filesystem, disk image, or VM
- reuse configuration files (`/etc/*`, init scripts, etc.) verbatim
- start from someone else's working system and modify it

This Lecture

- We close the story on userspace and filesystems.
- Part 1: User programs, root filesystem, and configuration
- Part 2: Partitions and Filesystems
- **Reminder: Do Attendance!**

Initramfs Rootfs Distinction

The Initramfs:

- Gets archived as a cpio.gz file and embedded into the kernel
- Unpacked by the kernel at startup and backed by RAM
- Only real task is to switch to root.

The real Rootfs:

- Backed up by the disk
- Contains the actual relevant files for your operating system
- What you will be focusing on in the next few labs

The FHS (Again)

The root filesystem needs to follow the filesystem hierarchy standard even more strictly than the initramfs.

You need these:

- `/usr/bin`, `/usr/share`, `/usr/include`
- `/lib`, `/bin`
- `/etc`
- `/run`, `/proc`, `/sys`, `/dev`

In effect, a near-superset of your initramfs tree.

/etc

- Important system configuration files will be put here.
- PID 1 configuration
- daemon configuration
- filesystem mount points (fstab)
- Package manager config files (not in this class)
- ... (more files in complex systems)

We recommend you inject these into your user filesystem once figured out!

PID 1

Usually, PID 1 will:

- Reap orphaned zombies (essential!)
- Mounts filesystems
- Spin up daemons
- Maybe other functionality too
- If PID 1 Fails, **The Kernel will panic!**

Examples of Init

- Busybox init (We use this for simplicity)
- Systemd (popular but controversial!)
- OpenRC, runit

Configuring PID 1

- Taken from conventions of System V UNIX (released 1983)
- This is *one* of many ways init systems can be configured
- `/etc/inittab` describes *what* should run and *when*
- Entries can be one-shot, respawned, or tied to boot stages (startup, shutdown, ...)
- Long-running services must be restarted if they die

Startup scripts (See next slide)

- Typically a script like `/etc/init.d/rcS`
- Responsible for mounting filesystems and starting daemons
- Invoked automatically by `init`

Boot Levels

BusyBox init runs programs in phases.

Important boot stages

- `sysinit` — run once, very early
  - Mount `/proc`, `/sys`, `/dev`
  - Prepare runtime directories
- `respawn` — long-running services
  - `getty`, login shells
  - Restarted automatically if they die
- `shutdown` — cleanup operations on poweroff/reboot

You should think about ordering very clearly.

Configuring Users and Groups

- The most basic way in which the operating system regulates access to privileged resources.
- User names and Group names are really aliases for integer UID's and GID's
- You only need to have `root` and `dhcpcd` (see handout for more info on dhcpcd)
- Root is all-powerful, generally avoid being root because you can break a lot of things!

Configuration files (passwd and group read by musl)

- `/etc/passwd`: Contains metadata about users
- `/etc/shadow`: Contains hashed passwords (needs very restrictive permissions)
- `/etc/group`: Contains metadata about groups

Important Userspace Daemons

Dhcpcd

- You basically need DHCP on modern networks for any kind of internet access
- The kernel will not do IP address configuration for you
- You can test working by `ping 1.1.1.1` (or some other raw ip address; DNS will be covered later)
- Dhcpcd requires its own user and group, along with a writable directory of its own! (See handout)

Other Essential Daemons (You must start them!)

getty

- Spawns login prompts on `/dev/tty*`
- One instance per terminal
- Reads `/etc/shadow` to allow access

eudev

- Userspace device manager
- Creates device nodes under `/dev`
- Triggers driver loading and hotplug events

chrony

- NTP client for wall-clock time
- Corrects clock drift after boot
- Required for TLS and sane logs

Filesystems

What is a Filesystem?

- A filesystem defines how bytes on disk are interpreted
- The filesystem interface is in the kernel (Linux is monolithic!)
- It maps names to data and metadata
- It defines persistence across reboots

Responsibilities

- File and directory naming
- Permissions and ownership
- Metadata (timestamps, size, links)
- Crash recovery and consistency guarantees

Filesystems we use

Ext4

- Standard Linux filesystem
- Journaling (crash recovery), some papers about this!
- Allows for permissions, other complex metadata

VFat

- Readable by EFI Firmware
- Much simpler, no permissions or even symlinks!
- We will put just the kernel in a special path in this filesystem.

There are obviously other filesystems out there, but these are most standard.

GPT

- You want partitions to have different filesystems on the same disk.
- GUID Partition Table (replaces MBR)
- A standardized partition table format
- Unique identifiers for each partition
- Well-defined partition types (not just numbers)

Regarding Firmware

- Firmware reads metadata at the front of the disk (sectors 1-33)
- Special partition types signal boot-related data
- The EFI System Partition (ESP) is identified this way
- For legacy BIOS, sector 0 is just executed. (+ checking last two bytes are `0x55AA`)

You should consult the handout for further info!

Before we keep moving

There are some toolchain constraints in the assignment!

- We will use filesystem debug tools and `sgdisk` to adjust metadata and partition layouts.
- LFS manuals will often get you to create a loopback device, which is impossible inside a docker container.
- If you attempt to copy files into filesystems via loopback devices, your build pipeline will fail.

Boot Path Summarized

(Cue blackboard)

1. Firmware starts
2. Firmware figures out where ESP is from GPT metadata
3. Launch bootloader (aka kernel) inside ESP (fixed path)
4. Kernel unpacks initramfs
5. initramfs `/init` mounts the real filesystem, then runs `switch_root`
6. `/sbin/init` in the disk takes over and launches programs.
7. login screen (hopefully)

Up Next

- Topics beyond the lab: Graphical Environments, Dbus, Certificates, DNS, ...
- Course Wrap-up, final lab hours