Lecture 4 Handout

Juni Kim

January 26, 2026

Contents

§1 Userspace Configuration

At this point in the course, the kernel is capable of booting and executing an `initramfs`. This handout focuses on configuring the *persistent* userspace environment that lives on disk and takes over after `switch_root`.

Everything in this section assumes that:

- the kernel has successfully booted
- `/sbin/init` is executed as PID 1
- BusyBox is providing the init implementation

§1.1 (Important) Information Regarding Dhcpcd

`dhcpcd` is a DHCP client responsible for acquiring an IP address on modern networks. The kernel does *not* configure networking automatically; without a userspace DHCP client, your system will have no internet access.

Unlike many simple utilities, `dhcpcd` assumes the existence of:

- a dedicated `dhcpcd` user
- a dedicated `dhcpcd` group
- a writable runtime directory under `/var/run`

In particular, the directory:

```
/var/run/dhcpcd
```

must:

- exist in the root filesystem image
- be owned by `dhcpcd:dhcpcd`
- be writable by that user

Because your root filesystem image is constructed offline, these ownership and permission adjustments must be made using filesystem debugging tools rather than normal runtime commands. Refer to the **Disk Tools** section below for details on how to use `debugfs` to accomplish this.

If these permissions are incorrect, `dhcpcd` will fail silently or exit early during boot.

§1.2 A Known `eudev` Build Footgun

Eudev uses a build system that pulls in libtool for linking. As a result, simply passing `-static` to your linker flags will not suffice, and you will need to pass in another `-all-static` flag to force eudev to be linked statically.

For more information, you can see the corresponding GitHub issue at https://github.com/eudev-project/eudev/issues/230

If `eudev` builds successfully but fails at runtime, inspect the installed binaries before debugging the kernel or filesystem.

```
readelf -l /usr/bin/udevd | grep interpreter
```

§1.3 Configuring user and group files

Linux systems describe users and groups using plain text configuration files under `/etc`. These files are read by libc and userspace programs; the kernel itself does not interpret them.

You must configure the following files:

- `/etc/passwd`
- `/etc/shadow`
- `/etc/group`

`/etc/passwd` contains basic user metadata:

```
username:x:uid:gid:comment:home:shell
```

`/etc/group` contains group metadata:

```
groupname:x:gid:user1,user2
```

`/etc/shadow` stores password hashes and must be readable only by root:

```
username:hashed-password:...
```

For this course, you minimally need:

- a `root` user (UID 0)
- a `dhcpcd` user and group

The permissions on `/etc/shadow` are *security critical*. If this file is world-readable, your system is misconfigured. Because these files live inside an ext4 image, permissions must be verified and corrected using `debugfs` after filesystem creation.

§1.4 Busybox inittab

BusyBox `init` determines system behavior by reading `/etc/inittab`. This file describes *what* programs should be run and *under what conditions*.

An `inittab` entry conceptually consists of:

```
<id>:<runlevels>:<action>:<command>
```

You will encounter entries equivalent in structure to the following:

```
::sysinit:<startup-script>
console::respawn:<login-program>
::ctrlaltdel:<reboot-command>
::shutdown:<cleanup-command>
```

Important ideas:

- `sysinit` entries run very early, once
- `respawn` entries are restarted if they exit
- `init` is responsible for keeping essential services alive

Paths and arguments in `inittab` are *policy*. If you copy them without understanding:

- what program is being executed
- why it needs to persist
- what happens if it exits

then you will have a system that is difficult to debug.

§1.5 Startup script

Rather than placing all logic directly into `inittab`, it is customary to delegate early boot tasks to a startup script.

This script is invoked by `init` during the `sysinit` stage and is responsible for:

- mounting kernel-provided filesystems
- preparing runtime directories
- launching essential daemons

A conceptual outline of such a script might look like:

```
#!/bin/sh

# mount kernel interfaces
mount kernel-exposed-filesystem A
mount kernel-exposed-filesystem B

# prepare runtime state
create volatile directories

# start long-running services
start service X
start service Y
```

The exact programs and mount options matter less than understanding that:

- this script runs as PID 1 initially
- failure here usually prevents login entirely
- ordering is significant

§1.6 Nameserver Configuration (Optional)

Name resolution is configured via `/etc/resolv.conf`. This file tells userspace programs where to send DNS queries.

A minimal example might specify one or more nameservers:

```
nameserver 1.1.1.1
nameserver 8.8.8.8
```

With a valid `resolv.conf`, programs that rely on DNS (such as `wget`) can resolve hostnames.

Note that this is *not sufficient* for HTTPS. TLS requires:

- cryptographic libraries
- a trusted certificate store

Those topics are intentionally deferred to later lectures.

§2 Disk Tools

This section describes the tools used to construct and inspect disk images. These tools operate on *files*, not block devices, due to container constraints.

§2.1 Generic

`dd` is a low-level byte copying tool. In this course, it is primarily used to create zero-initialized disk images.

Conceptually:

- the output file represents a disk
- the size determines the disk capacity

`sgdisk` is used to create and modify GPT partition tables. It operates directly on disk images and allows you to:

- define partition boundaries
- assign partition types
- label partitions

Because Docker does not permit loopback devices, partitions must be stitched together manually by copying filesystem images into specific offsets of a larger disk image. Understanding sector size and offsets is essential; blindly copying commands without reasoning about these values will lead to corrupted images.

§2.2 Ext4

`ext4` is the primary Linux filesystem used for the root filesystem.

Relevant tools include:

- `mkfs.ext4` / `mke2fs`: create ext4 filesystems
- `debugfs`: inspect and modify ext4 images offline

`debugfs` is particularly important in this course. It allows you to:

- adjust file permissions
- change ownership (note you must change user and group owners by UID and GID, not name)
- fix mistakes without rebuilding everything

Its interface is interactive and non-obvious. Typical usage involves opening an image, issuing commands, and explicitly exiting. This tool is required to:

- restrict access to `/etc/shadow`
- configure ownership for `dhcpcd` runtime directories

§2.3 VFat

The EFI System Partition must use a FAT-based filesystem, typically FAT32.

Tools from `dosfstools` are used:

- `mkfs.vfat`: create the filesystem
- `mmd`: create directories inside the image
- `mcopy`: copy files into the image

EFI firmware does not search arbitrarily. The kernel must reside at a well-defined path within the ESP, and the exact filename depends on the target architecture.

This filesystem does not support permissions, symlinks, or Unix ownership. It exists solely as a firmware-readable transport mechanism.