Lecture 3 Handout

Juni Kim

January 23, 2026

Contents

§1 Kernel Configuration

To begin configuration, you should create a scratch directory (we highly recommend this is a subdirectory of `$DIST` since that is also mounted in the external filesystem)

```
cd /scratch/directory
make -C "/path/to/kernel/src" menuconfig O="/scratch/directory"
```

§1.1 Essential Kernel Configuration Options

For this course, you will build a *monolithic* kernel with a statically embedded initramfs. This simplifies early boot debugging and avoids reliance on kernel modules or an external root filesystem during bring-up.

¶ Disabling Kernel Modules We recommend disabling loadable kernel modules entirely. This ensures all drivers required for boot are built directly into the kernel image.

```
CONFIG_MODULES=n
```

With this option disabled, the kernel will refuse to load modules at runtime, and all required drivers must be enabled as built-ins.

¶ Embedding a Custom Initramfs Your initramfs archive should be embedded directly into the kernel image at build time. This archive is typically produced using `cpio` and `gzip`.

```
CONFIG_INITRAMFS_SOURCE="/path/to/archive.cpio.gz"
```

When this option is set, the kernel will unpack the archive into a temporary root filesystem and execute `/init` as PID 1.

¶ Kernel Command Line The kernel command line may be supplied in one of three ways:

- explicitly via QEMU using `-append` when booting with `-kernel`
- by firmware (EFI) when booting from disk
- as a built-in default via kernel configuration

Since QEMU or firmware does not provide command line arguments by default, we recommend supplying a default command line.

```
# please do not cargo cult this
CONFIG_CMDLINE="console=ttyYourChoice otherkey=option"
```

On x86 only, you must also enable:

```
CONFIG_CMDLINE_BOOL=y
```

This option enables the presence of a built-in command line on x86 systems, even if it is empty.

¶ Debugging the Command Line Assuming you have a working console but are suspicious about what parameters are actually being passed to the kernel, you can always call

```
cat /proc/cmdline
```

This assumes you have the `/proc` pseudofilesystem mounted on your system.

¶ Default Host Name We prefer you configure the default hostname of your system to some name that clearly displays your kerb.

```
CONFIG_DEFAULT_HOSTNAME="6s913-system"
```

Generally, you would configure hostnames via `/etc/hostname`, but this is one way you can personalize your kernel.

¶ EFI Support To support booting via UEFI firmware, the kernel must include the EFI stub and EFI runtime support.

```
CONFIG_EFI=y
CONFIG_EFI_STUB=y
```

With these options enabled, the kernel can be loaded directly by UEFI firmware from disk without an intermediate bootloader.

§1.2 Required Serial Console Drivers

Your kernel must include built-in drivers for the serial consoles used by the virtual machines in this course. Disabling the wrong serial driver will result in a silent boot with no visible output.

The following options must *not* be disabled:

```
CONFIG_SERIAL_8250=y
CONFIG_SERIAL_8250_CONSOLE=y
```

The above options provide the legacy PC serial port used by x86 virtual machines, which appears as `/dev/ttyS0`.

```
CONFIG_SERIAL_AMBA_PL011=y
CONFIG_SERIAL_AMBA_PL011_CONSOLE=y
```

The above options provide the ARM PL011 UART used by aarch64 virtual machines, which appears as `/dev/ttyAMA0`.

§1.3 Virtio Device Support

All virtual machines in this course use `virtio` devices for storage and networking. You must ensure that virtio support is enabled in the kernel.

At a minimum, the following options must remain enabled:

```
CONFIG_VIRTIO=y
CONFIG_VIRTIO_PCI=y
CONFIG_VIRTIO_NET=y
```

Disabling these options will cause virtual disks or network interfaces to silently disappear, even though QEMU successfully emulates the hardware.

§1.4 Console Devices and `/dev/console`

The specific serial device used by QEMU depends on the architecture:

- aarch64 virtual machines use `/dev/ttyAMA0`
- x86_64 virtual machines use `/dev/ttyS0`

Rather than hard-coding one of these device names in userspace, your `/init` script should bind standard input, output, and error to `/dev/console`.

```
exec 0</dev/console 1>/dev/console 2>/dev/console
```

The kernel dynamically links `/dev/console` to the active serial console specified by the kernel command line. This allows the same initramfs and userspace to work across architectures without modification.

§2 QEMU Invocation and Architecture Differences

QEMU emulates hardware, not an operating system. As a result, different CPU architectures require different machine models, firmware, and device options. The commands below reflect the minimal sets of options required to boot systems in this course.

If you are on x86, you should use `qemu-system-x86_64`, and if you are on arm you should use `qemu-system-aarch64`.

§2.1 Exiting out of QEMU

Press ctrl-a, followed by x while inside the qemu window. This will automatically quit qemu. Once you have a real filesystem, you should probably use `/bin/poweroff` instead.

§2.2 Booting Directly with the Kernel

In early stages of development, it is often convenient to boot QEMU directly into a kernel without using firmware or a disk image.

¶ Common Options

- `-m 1024` sets the amount of guest memory (in MB).
- `-kernel <path>` specifies the kernel binary to boot directly.
- `-nographic` disables graphical output and routes the console to the terminal.
- `-no-reboot` causes QEMU to exit instead of restarting on failure.
- `-netdev user,id=net0` and `-device virtio-net-pci` configure a virtual network interface.

¶ aarch64-Specific Options

- `-machine virt` selects a generic virtual ARM machine model.
- `-cpu cortex-a72` selects a CPU model supported by the kernel.
- `-append "console=ttyAMA0"` directs kernel output to the serial console used by `-nographic`

¶ x86_64-Specific Options

- `-machine q35` selects a modern PCIe-based x86 machine model.
- `-append "console=ttyS0"` directs kernel output to the serial console used by `-nographic`

When booting directly with `-kernel`, the kernel is responsible for initializing the system without firmware involvement.

§2.3 Booting from Disk Using Firmware

Once a full disk image has been constructed, QEMU should be run using firmware to more closely resemble a real system boot.

¶ Firmware Options

- `-drive if=pflash,readonly=on,file=...CODE...` provides the read-only firmware image.
- `-drive if=pflash,file=...VARS...` provides writable firmware state (EFI variables).

The firmware binaries differ by architecture:

- aarch64 uses `AAVMF_CODE.fd` and `AAVMF_VARS.fd`.
- x86_64 uses `OVMF_CODE_4M.fd` and `OVMF_VARS_4M.fd`.

¶ Disk and Console Options

- `-drive if=virtio,format=raw,file=<image>` attaches the bootable disk image.
- `-serial mon:stdio` multiplexes the serial console and QEMU monitor onto standard I/O.
- `-nographic` disables graphical output.

When booting via firmware, the kernel is loaded from disk by the firmware rather than passed directly by QEMU.