**Engineer**

# Kernel

the core of an operating system

## What it is

A kernel is the program at the centre of an operating system that runs with the processor's full privileges and controls access to the hardware. It decides which program runs next, hands out memory and keeps each program's memory apart from the rest, reads and writes disks through file systems, moves packets through the network stack, and enforces who may touch which file, port or device. Everything else runs in user space with fewer privileges and asks the kernel to act for it.

Those requests are system calls, which the Linux manual calls "the fundamental interface between an application and the Linux kernel". Programs rarely make them by hand: they call wrappers in a C library such as glibc, which pass the arguments on and turn the kernel's answer into a return value and an error code. When a program stops with "Permission denied", that wording is usually the kernel's answer to one of these calls, translated by the library.

Kernels differ in how much they keep inside. Linux keeps device drivers, file systems and the network stack in the kernel itself and can load more as modules while it runs, a design usually called monolithic. Every driver then runs with the kernel's full privileges. A microkernel keeps only the minimum. seL4 provides mechanisms for controlling access to the physical address space, interrupts and processor time, and runs drivers, file systems and network stacks as ordinary user-mode programs; its verified 64-bit RISC-V version is about 10,000 lines of code, with a formal proof that the C implementation does what its specification says. The macOS kernel, XNU, is a hybrid that combines the Mach microkernel with parts of FreeBSD.

A kernel update normally takes effect only when the machine boots into it, and the kernel's version number, such as 6.12, is separate from the release number of the distribution around it.

Use the concept in a decision

Put "What is a kernel?" to work in a real decision. We can help relate the concept to the system you are building or buying. Day-to-day care of servers, mail, accounts and updates, so the things everybody relies on stay boring.

**Explore our services** Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. Ran 10 PostgreSQL and MS SQL servers on Ubuntu — 10 database servers deployed and kept healthy

**Ran 10 PostgreSQL and MS SQL servers on Ubuntu** Get in touch →

## Why it matters

For a business, the kernel settles three questions: how far apart its workloads really are, what a security fix costs in downtime, and where the truth about a machine is kept.

Isolation first. A container is a set of ordinary processes that the kernel has placed in namespaces, which give each group its own view of process IDs, network interfaces, mounts and users, and in control groups, which share out processor time, memory and disk input and output. Every container on a host runs on the host's one kernel, so a flaw in that kernel is a flaw under all of them at once, and an image that brings its own libraries still relies on whatever kernel features the host has. A virtual machine boots a kernel of its own. For one team's services, containers are a reasonable boundary; for customers who do not trust each other, give each tenant its own kernel.

Fixes come next. A patched kernel package on disk changes nothing until the machine boots into it, and on a server that restart is the cost. Livepatching exists because, in the kernel documentation's words, users are "reluctant to reboot": it redirects individual functions to fixed versions while the system runs. Only functions the kernel can trace can be patched that way, so livepatching buys time for an urgent fix rather than replacing reboots. A plan with a regular, rehearsed reboot costs less than one that assumes the reboot never has to happen.

Last, the kernel holds the live state. A firewall file describes rules; the rules in the kernel's packet filter decide a packet's fate. A file in `/etc/sysctl.d` describes a setting; the value under `/proc/sys` is the one in force. A service can report itself

healthy whether or not its action ever reached the kernel. Checks that read the kernel's own state catch failures that a review of configuration files would pass.

## What it looks like in practice

A hardening review of a production host in August 2026 asked whether its SSH brute-force protections worked. Each was configured and appeared in every report, so the review checked them against what the kernel actually does with a packet rather than against the files describing what should happen. Seen from the kernel, none of the three had acted once since the host was built. The jail's bans named port 22, where no socket listened after SSH had moved to port 1986. The rate limit did reach the packet filter, which evaluates rules top down and stops at the first match, so a broader rule above it took every packet before the limit was consulted. The ban action shelled out to a packet-filter program that its package lists as recommended rather than required; on a host without it, the jail started, counted and decided, then failed at the one moment it had to hand a rule to the kernel. Two rules came out of the review: a security control is proved by an assertion, which a comment cannot do, and a firewall is checked by the order in which the kernel holds its rules, rather than by whether a rule exists.

The ban path then got a test that runs every time the configuration is applied. Each run bans an address from a documentation-reserved block, reads the resulting rule back from the kernel's packet filter and then lifts the ban in a cleanup step that runs either way. The read-back is what makes it a test: the banning service's own status would report success for a ban that produced no rule. A reserved address belongs to no real network, so a ban left behind by a failed run locks nobody out. The cost: a few seconds per run, and a firewall rule written to production and withdrawn each time.

A 464 MB host with no room for a metrics stack got seven read-only roles that render the live machine to Markdown; its facts snapshot computes processor utilisation from two samples of the kernel's own counter. The security report prints the firewall rules in the order they are matched, beside the live ban rules, and each of the seven roles prints only figures its own run read, the kernel's counters among them, never an estimate to fill a gap. Those reports turned up two security controls that were dead and a listening port that nobody had declared.

# Where it goes wrong

**A kernel setting changed by hand reverts after the next reboot.** `sysctl -w` writes the value into the running kernel only, so it lasts until restart. The opposite slip: a file under `/etc/sysctl.d` is read at boot, so the change waits, unannounced, for some later restart. Write the setting to a file, load it now, and read the live value back:

```
echo 'vm.swappiness = 10' | sudo tee /etc/sysctl.d/90-local.conf
sudo sysctl --system
sysctl vm.swappiness
```

**A CPU graph stays flat through a load spike.** The `cpu` line in `/proc/stat` counts time since boot, in units of USER_HZ (hundredths of a second on most architectures), so a single reading gives the average since boot, which barely moves on a machine up for months. Read it twice and divide the differences:

```
head -1 /proc/stat; sleep 5; head -1 /proc/stat
```

Busy time is the change in user, nice, system, irq and softirq, idle time is the change in idle and iowait, and utilisation is busy over busy plus idle. The manual calls iowait unreliable, so treat it as a hint rather than an alert.

**A crash report comes back with a request to reproduce it on an untainted kernel.** The kernel marks itself tainted when something happens that may matter to a later investigation, such as loading a proprietary, out-of-tree or unsigned module, and its documentation warns that developers often ignore bug reports from tainted kernels. The mark stays until reboot, even after the module is unloaded. Read it before filing:

```
cat /proc/sys/kernel/tainted
```

Zero means untainted; any other number is a sum of flags, among them `1` for a proprietary module, `4096` for an out-of-tree module and `8192` for an unsigned one. Reproduce the fault after booting without the module that set the flag.

**A new network card shows up in the hardware list and the system offers no interface for it.** Drivers are part of the kernel, and a kernel older than the hardware may lack one. `lspci -k` shows each device's driver in use and the modules able to handle it; a device without a driver-in-use line has none bound. Install a newer kernel series from the distribution rather than building the vendor's module by hand: an out-of-tree module must be rebuilt for every new kernel, and loading it taints the kernel.

## Further reading

- The Linux kernel user's and administrator's guide — the kernel project's own manual for running Linux.
- syscalls(2) — every Linux system call, with the kernel version that introduced it.
- namespaces(7) — the kinds of namespace and how containers are built from them.
- proc_stat(5) — each field of `/proc/stat`, including the iowait warning.
- Tainted kernels — each taint flag and how to decode the number.
- seL4 FAQ — what a microkernel leaves out, and the assumptions behind seL4's proof.

Glossary

Choose for your actual constraints

Bring the deployment path, the operational risks and the people on call. We can scope a change with checks, a recovery route and clear ownership.

**See what it costs** Get in touch →

# Make the terminology practical

Ask how “What is a kernel?” applies to your organisation. A concrete example from your current system is enough to start the discussion.

**Explore our services**

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/kernel/