



Introduction to Linux Operating System

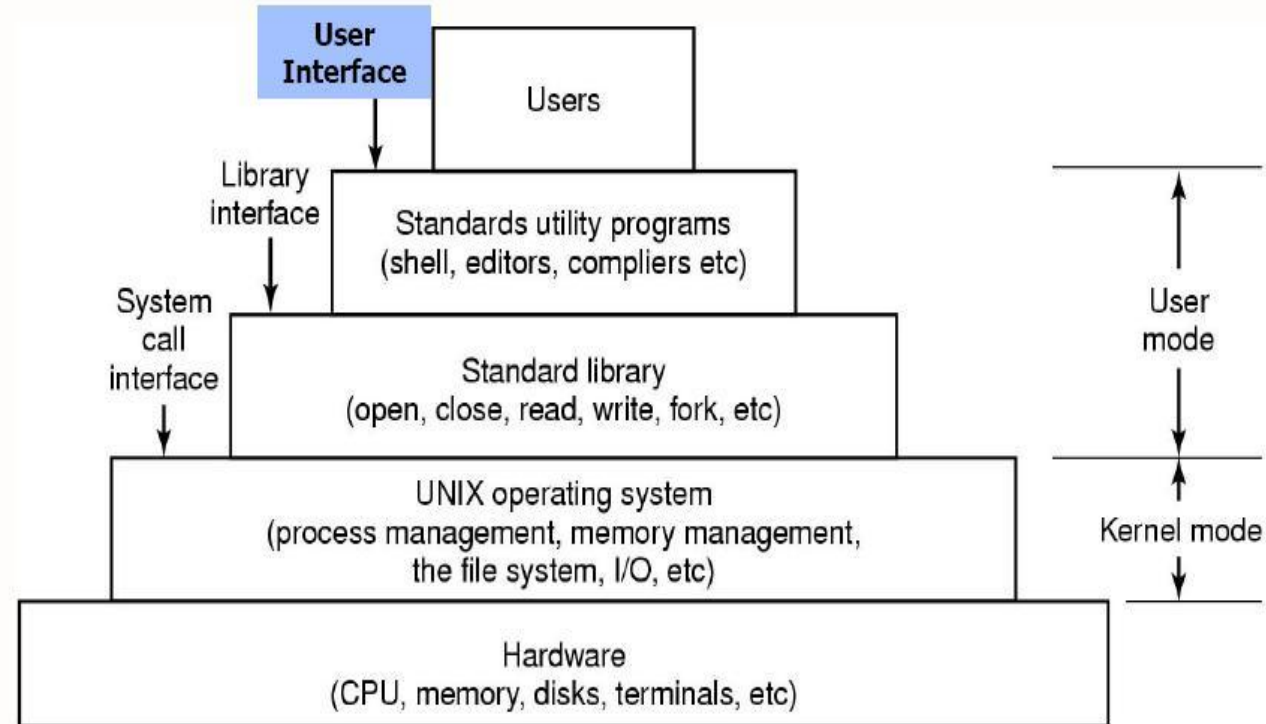


Table of contents

- Operating system tasks
- UNIX history, Linux history
- Linux distributions
- Linux basic features
- Building OS kernels
- Linux kernel modules
- eBPF
- Kernel reports – what is going on in the kernel
- Linux structure and kernel functions
- Basic concepts



Computer system layers (source: Stallings, Operating Systems)



Operating System is a program that mediates between the user and the computer hardware.

- **Hides hardware details** of the computer system by creating abstractions (virtual machines).
- **Manages resources**: memory, processor (CPU), input/output, communication ports
- **Other activities**: security, job accounting, error detecting tools, etc.



UNIX history

- Created in **1969**; authors: Ken Thompson, Denis Ritchie from Bell Laboratories, machine: old PDP-7; had many features of MULTICS.

(Brian Kernighan participated in the creation of Unix, he is co-author of the first book about C).



[Ken Thompson](#)

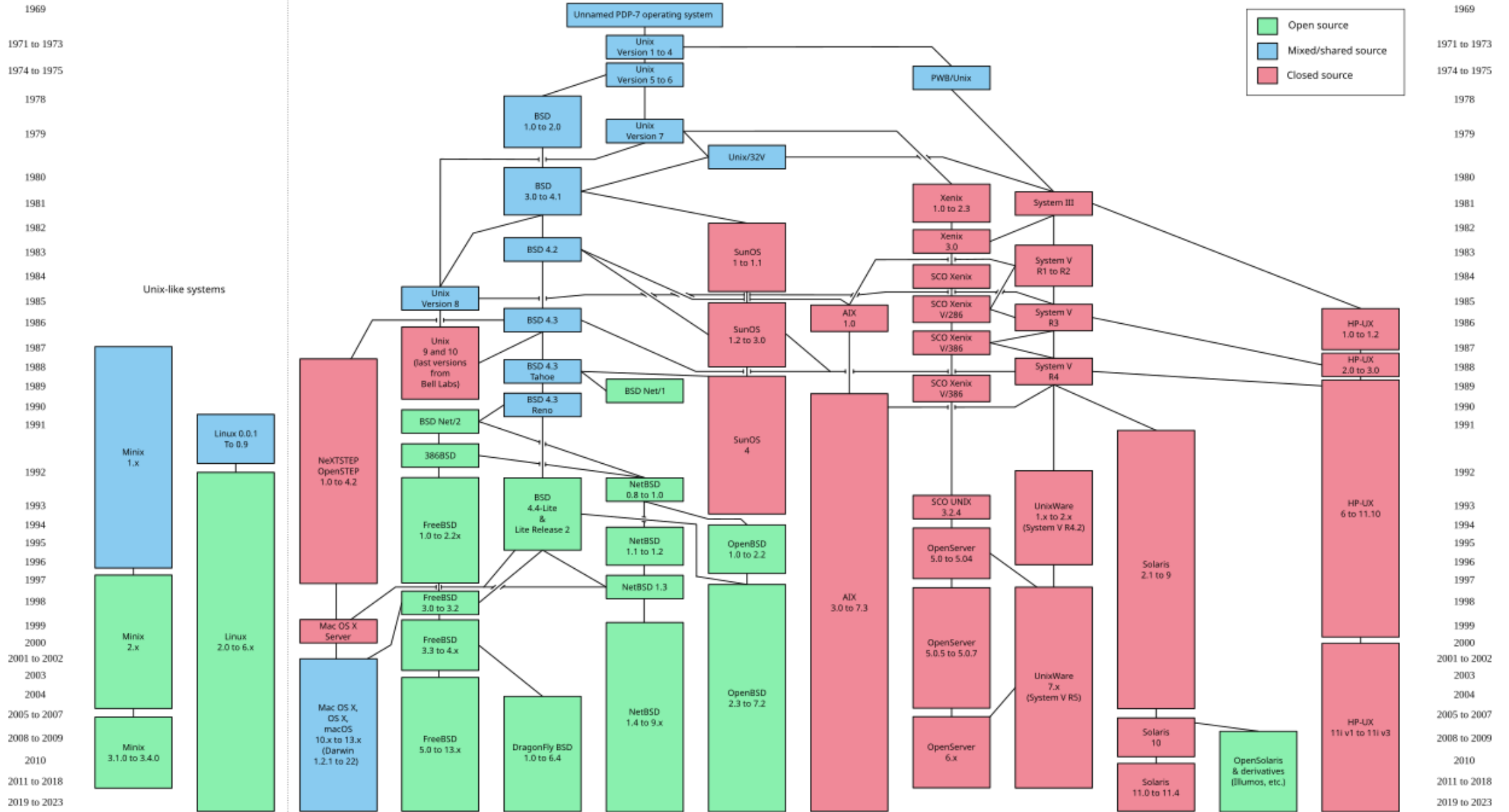


[Denis Ritchie](#)
died 12.10.2011



[Brian Kernighan](#)

- 1973: UNIX rewritten in C (language designed specifically for this purpose).
- 1974: presented on ACM Symposium on Operating Systems and in CACM, quickly gaining popularity.
- For hobbyists: [Unix history](#), [Unix, Linux, and variant history](#).
- [The early days of Unix at Bell Labs](#), Brian Kernighan (LCA 2022 online).
- [Ken Thompson interviewed by Brian Kernighan](#) at VCF East 2019.



Unix History Diagram - short version (source: [Wikipedia](https://en.wikipedia.org/wiki/Unix))



Linus Torvalds, August 1991



Hello everybody out there using minix -

I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing since april, and is starting to get ready. I'd like any feedback on things people like/dislike in minix, as my OS resembles it somewhat (same physical layout of the file-system (due to practical reasons) among other things).

I've currently ported bash(1.08) and gcc(1.40), and things seem to work. This implies that I'll get something practical within a few months, and I'd like to know what features most people would want. Any suggestions are welcome, but I won't promise I'll implement them :-)

Linus (torv...@kruuna.helsinki.fi)

PS. Yes - it's free of any minix code, and it has a multi-threaded fs. It is NOT protable (uses 386 task switching etc), and it probably never will support anything other than AT-harddisks, as that's all I have :-).



Linux history

[Linus Torvalds](#), Finland, born in the same year as UNIX, i.e. 1969, creator of the Linux kernel and the Git version control system.



*Linus Torvalds
announcing Linux
1.0, 30.03.1994*



*Linus
Torvalds in
2024
[in conversation
with Dirk Hohndel
at OSS Vienna](#)*

*Currently hired by the [Linux
Foundation](#)*

[Richard Stallman](#), founder of the GNU project and the Free Software Foundation, co-creator of the GNU GPL license, creator of the Emacs editor, GCC compiler, GDB debugger.



*Richard
Stallman in
2019*

May 1991, version 0.01: no support for the network, limited number of device drivers, one file system (Minix), processes with protected address spaces

The Linux Kernel Archives – <https://www.kernel.org/>

- **2025-02-21**, latest stable version **6.13.4**
- **2025-02-16**, latest mainline **6.14-rc3**

Numbering of the kernel versions – see [Wikipedia](#)



[Andrew Tanenbaum in 2023](#)



Linux statistics and facts

- **100%** of the world's top **500 supercomputers** run on Linux. (Blackdown)
- Linux powers **53.1% of websites** whose operating system is known. ([W3Techs](#))
- **96.3%** of the world's top **one million web servers** run on Linux. (ZDNet)
- **90%** of all **cloud infrastructure** operates on Linux, and practically all the best cloud hosts use it.
- **90%** of Hollywood's special effects are made on Linux. (Wertsba)
- **47%** of professional developers use Linux-based operating systems. ([Statista](#))

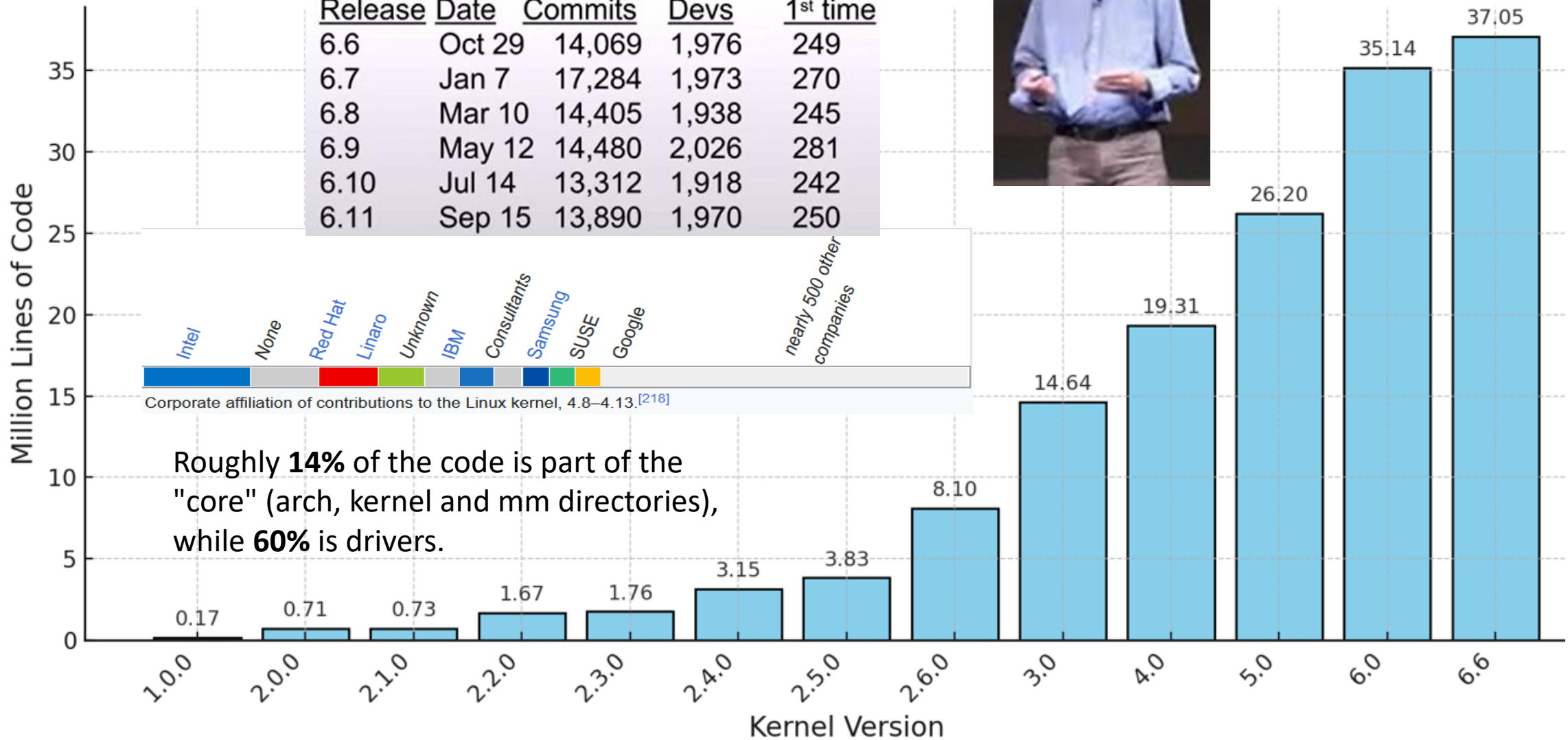
- In July 2022, **2.76% of all desktop operating systems worldwide ran on Linux.**
- In June 2024, Linux held a market share of **4.03%** of the global desktop market. ([ZDNet](#))
- Linux, the third most popular desktop OS, has a market share **of 2.09%**. (Statista)
- Linux powers **85%** of smartphones. (Hayden James)

<https://truelist.co/blog/linux-statistics/>

Jonathan Corbet in Oct. 2024 [Kernel Report](#):

Recent release history

Release	Date	Commits	Devs	1 st time
6.6	Oct 29	14,069	1,976	249
6.7	Jan 7	17,284	1,973	270
6.8	Mar 10	14,405	1,938	245
6.9	May 12	14,480	2,026	281
6.10	Jul 14	13,312	1,918	242
6.11	Sep 15	13,890	1,970	250



Linux kernel versions (source: [Wikipedia](#))



Linux distributions

A set of ready-to-install, precompiled packages; tools for package installation and uninstallation (RPM: Red Hat Package Manager); kernel, but also many service programs; tools for file systems management, creation and maintenance of user accounts, network management etc.

Page Hit Ranking		
Data span:		
<input type="text" value="Last 6 months"/>		<input type="button" value="Go"/>
Rank	Distribution	HPD*
1	Mint	2688▲
2	MX Linux	2158▼
3	EndeavourOS	1707▲
4	CachyOS	1526▲
5	Debian	1363▲
6	Pop!_OS	1182▲
7	Manjaro	1143▼
8	Ubuntu	1091▲
9	Fedora	989-
10	openSUSE	726-

[DistroWatch](#) is a website which provides news, popularity rankings, and other general information about Linux distributions as well as other free software/open source Unix-like operating systems.

← [Debian](#) used in labs in MIM UW

2025-02-23 (10 top out of 270)



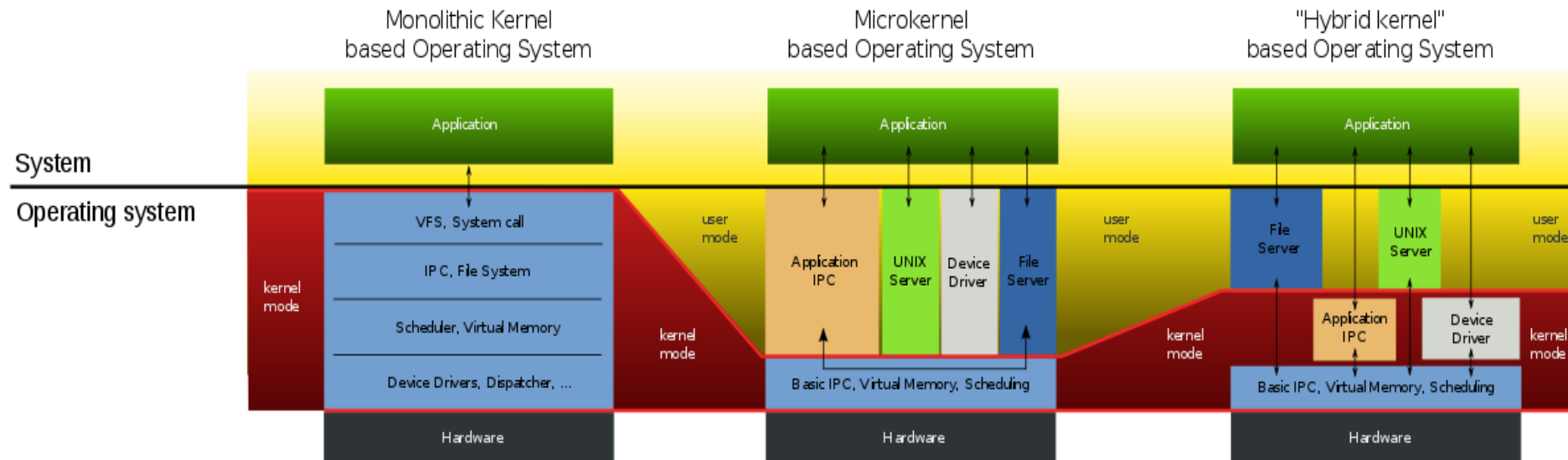
Linux basic features

- Multi-access system (with time sharing) and multi-tasking.
- Multiprocess system, simple mechanisms to create hierarchy of processes, kernel preemption.
- Available for many architectures.
- Simple standard user interface that can be easily replaced (shell → command interpreter).
- Hierarchical file systems.
- Files are seen as strings of bytes (easy to write filters).
- Loading programs on demand (fork with copy on write).
- Virtual memory with paging.
- Dynamic hard disk cache.
- Shared libraries, loaded into memory dynamically (one code used simultaneously by many processes).
- Compliance with the POSIX 1003.1 standard.
- Different formats of executable files.



Building OS kernels

- **Monolithic kernel** (the only solution until the 1980s) – Linux belongs to this category.
 - the whole kernel runs in a single address space,
 - communication via direct function invocation.
- **Microkernel** (e.g. Mach, MINIX).
 - functionality of the kernel is broken down into separate processes (servers),
 - some servers run in kernel mode, but some in user mode – all servers have own address spaces,
 - communication is handled via message passing,
 - modularity – failure in one server does not bring down another, one server may be swapped out for another,
 - context switch and communication generate extra overhead so currently user mode servers are rarely used.
- **Macrokernel** or „**Hybrid kernel**“ (e.g. the Microsoft Windows NT kernel that powers all operating systems in the Windows NT family, up to and including Windows 11 and Windows Server 2022, and powers Windows Phone 8, Windows Phone 8.1, Windows 10 Mobile, and the Xbox One and Xbox Series consoles).



Structure of monolithic kernel, microkernel and hybrid kernel-based operating systems (source: [Wikipedia](https://en.wikipedia.org/wiki/Kernel_(operating_system)))

Linus Torvalds :

“As to the whole ‘hybrid kernel’ thing - it’s just marketing. It’s ‘oh, those microkernels had good PR, how can we try to get good PR for our working kernel? Oh, I know, let’s use a cool name and try to imply that it has all the PR advantages that that other system has’.”

Readings

1. Tanenbaum – Torvalds debate on kernel architecture (MINIX vs Linux)
 - [Wikipedia](https://en.wikipedia.org/wiki/Tanenbaum-Torvalds_debate)
 - [O’Reilly](https://www.no-starch.com/)
2. [Comparing Linux and Minix](https://lwn.net/Articles/24220), February 5, 2007, Jonathan Corbet



Linux kernel modules

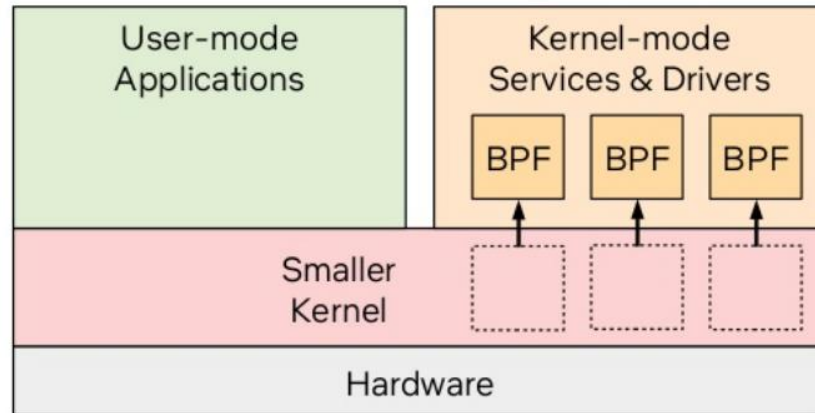
- Linux borrows much of the good from microkernels: modular design, capability to preempt itself, support for kernel threads, capability to dynamically load separate binaries (kernel modules).
- **Modules** – separately compiled, loaded into memory on demand and deleted when they are no longer needed.
- Examples: a device driver, a file system, an executable file format.
- Advantages: saving memory (occupies memory only when it is needed), the error in the module does not suspend the system, but only removes the module from the memory, one can use conflicting drivers without the need to restart the system, etc.
- Disadvantages ???
 - *name of the module*
 - *memory size of the module, in bytes*
 - *how many instances of the module are currently loaded*
 - *if the module depends upon another module(s)*

cat /proc/modules


```
Module          Size  Used by
af_packet_diag  16384  0
netlink_diag    16384  0
udp_diag        16384  0
raw_diag        16384  0
btrfs           1589248 0
blake2b_generic 20480  0
xor             24576  1 btrfs
raid6_pq        122880  1 btrfs
ufs             94208  0
gpx4            16384  0
hfsplus         126976  0
hfs             73728  0
cdrom           73728  2 hfsplus,hfs
minix           45056  0
vfat            20480  0
msdos           20480  0
fat             86016  2 msdos,vfat
jfs             212992  0
ext4            942080  0
```

But – eBPF makes a change ...

Modern Linux is becoming Microkernel-ish



The word "microkernel" has already been invoked by Jonathan Corbet, Thomas Graf, Greg Kroah-Hartman, ...

 **eBPF** is turning the Linux kernel into a microkernel.

- An increasing amount of new kernel functionality is implemented with eBPF.
- 100% modular and composable.
- New additions can evolve at a rapid pace. Much quicker than normal kernel development.

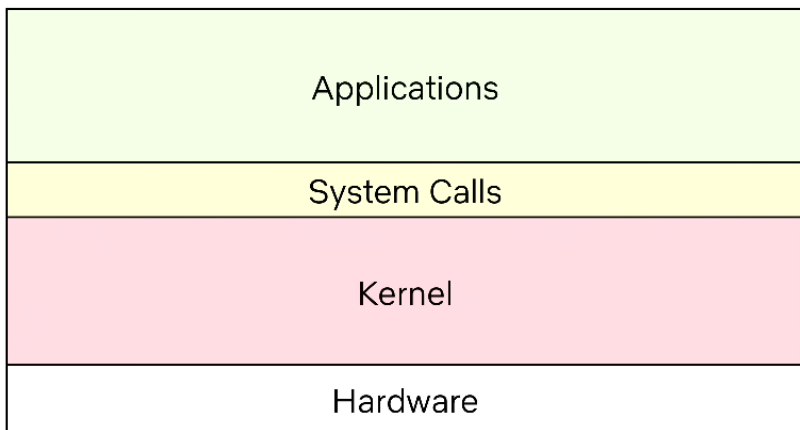
Example: The linux kernel is not aware of containers and microservices (it only knows about namespaces).  Cilium is making the Linux kernel container and  Kubernetes aware.

Extended BPF: A New Type of Software, Brendan Gregg at Ubuntu Masters Conf 2019
([presentation](#), [slides](#))

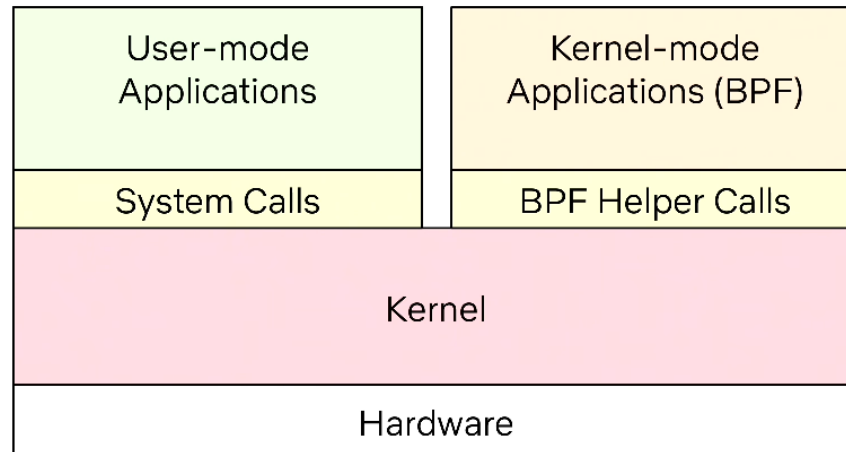
eBPF – Rethinking the Linux Kernel, Thomas Graf, QCon 2020
([presentation](#), [transcript](#))

Thomas Graf: *With BPF, we're starting to implement a **microkernel model** where we can now dynamically load programs, we can dynamically replace logic in a safe way, we can make logic composable. We're going away from the requirement that every single Linux kernel change requires full consensus across the entire industry or across the entire development community and instead, you can define your own logic, you can define your own modules and load them safely and with the necessary efficiency.*

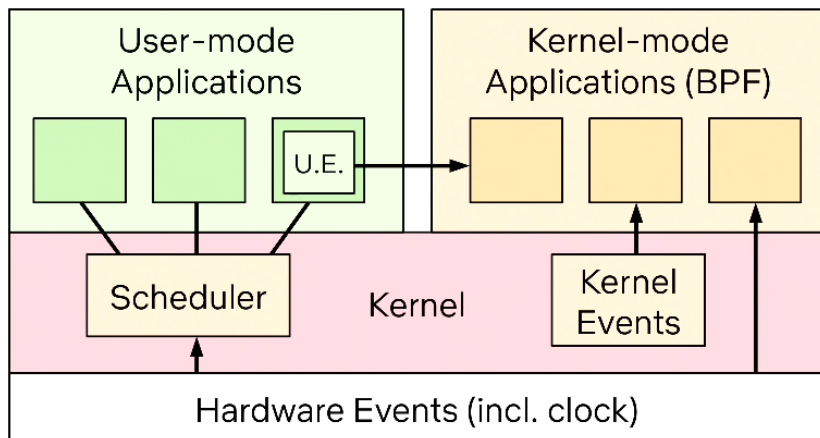
50 Years, one (dominant) OS model



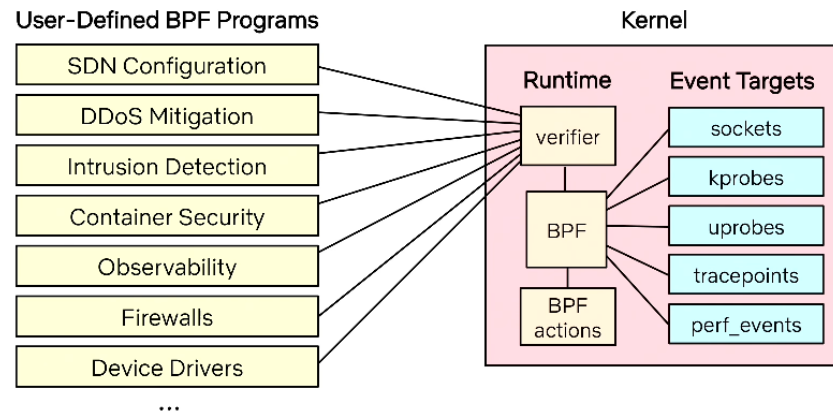
Modern Linux: A new OS model



Modern Linux: Event-based Applications

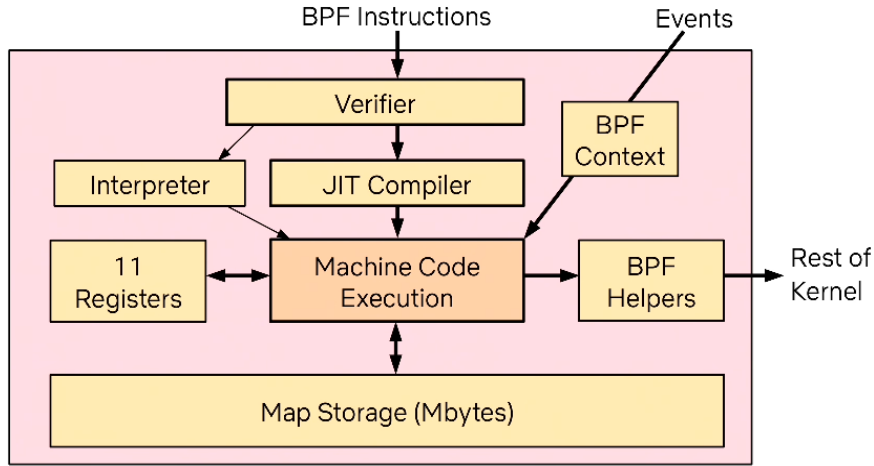


BPF 2019



Extended BPF: A New Type of Software, Brendan Gregg at Ubuntu Masters Conf 2019
[\(presentation, slides\)](#)

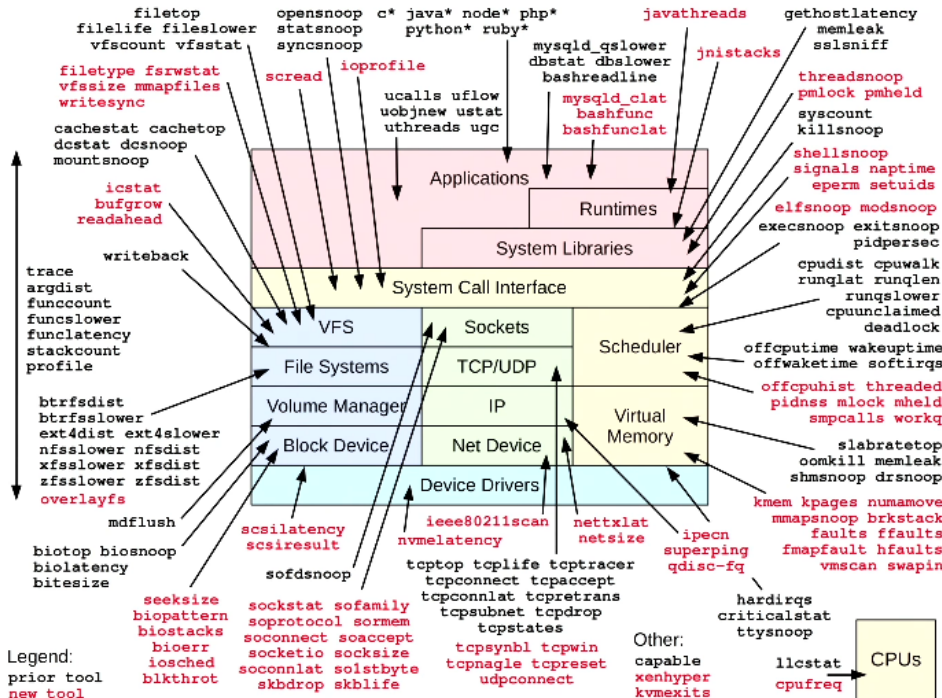
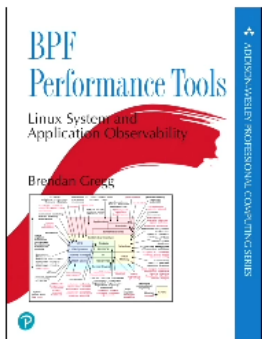
BPF Internals



A New Type of Software

	Execution model	User defined	Compilation	Security	Failure mode	Resource access
User	task	yes	any	user based	abort	syscall, fault
Kernel	task	no	static	none	panic	direct
BPF	event	yes	JIT, CO-RE	verified, JIT	error message	restricted helpers

BPF Perf Tools



Extended BPF: A New Type of Software, Brendan Gregg at Ubuntu Masters Conf 2019 ([presentation](#), [slides](#))





The Good:

- Open and transparent process
- Excellent code quality
- Stability
- Available everywhere
- Almost entirely vendor neutral

The Bad:

- Hard to change
- Shouting is involved (getting better)
- Large and complicated codebase
- Upstreaming code is hard, consensus has to be found.
- Upstreaming is time consuming
- Depending on the Linux distribution, merged code can take years to become generally available
- Everybody maintains forks with 100-1000s backports

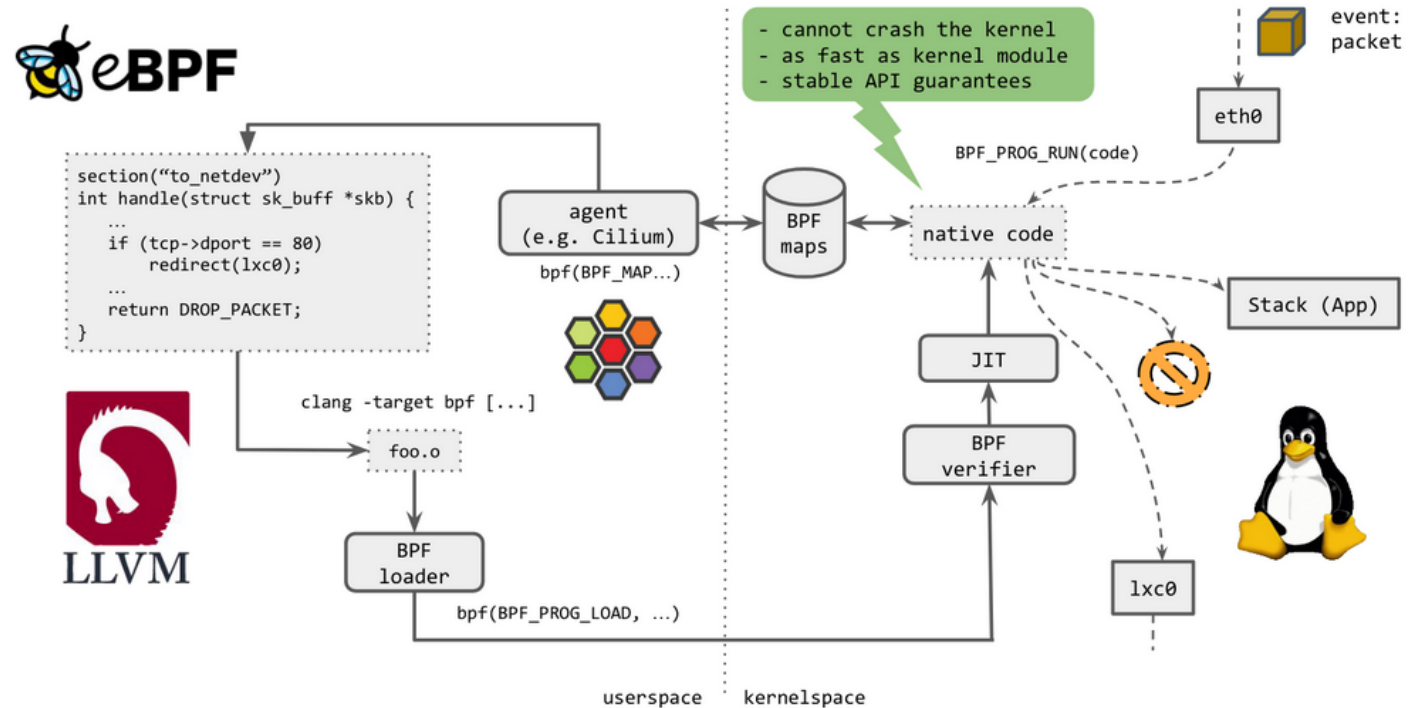
2018, new Code of Conduct

Linux Development

What is BPF?

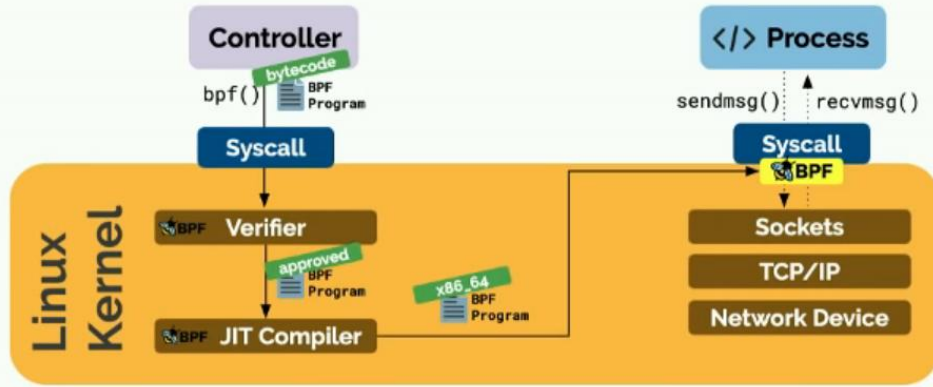
Highly efficient sandboxed virtual machine in the Linux kernel making the Linux kernel programmable at native execution speed.

Programmability allows to continuously adapt to changing requirements and innovate quickly.



How to Make Linux Microservice-Aware with Cilium and eBPF, Thomas Graf, QCon 2018, ([presentation](#), [transcript](#))

eBPF Runtime



Safety & Security

The verifier will reject any unsafe program and provides a sandbox.



Continuous Delivery

Programs can be exchanged without disrupting workloads.



Performance

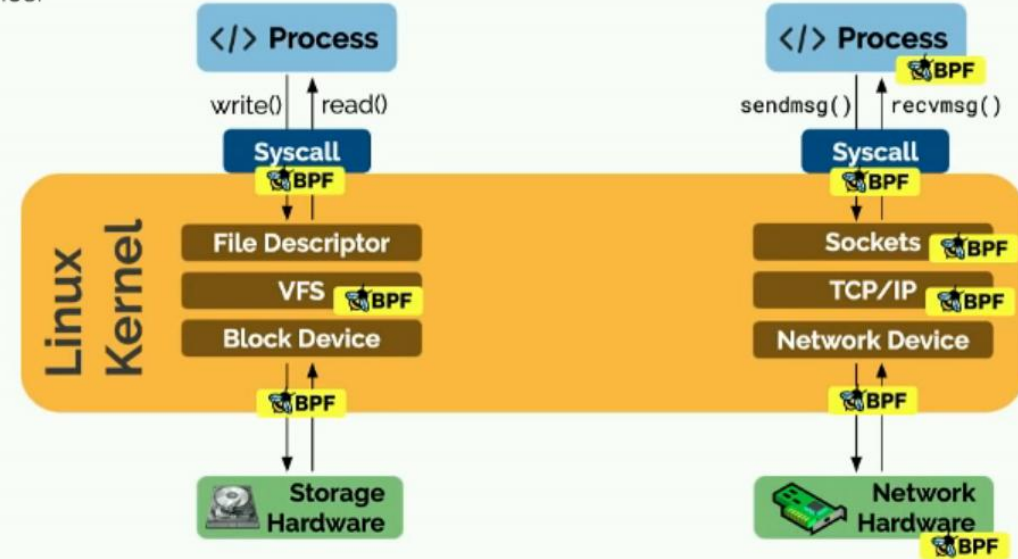
The JIT compiler ensures native execution performance.



The runtime accepts bytecode, verifies it, just-in-time compiles it, and runs it at the requested hook point.

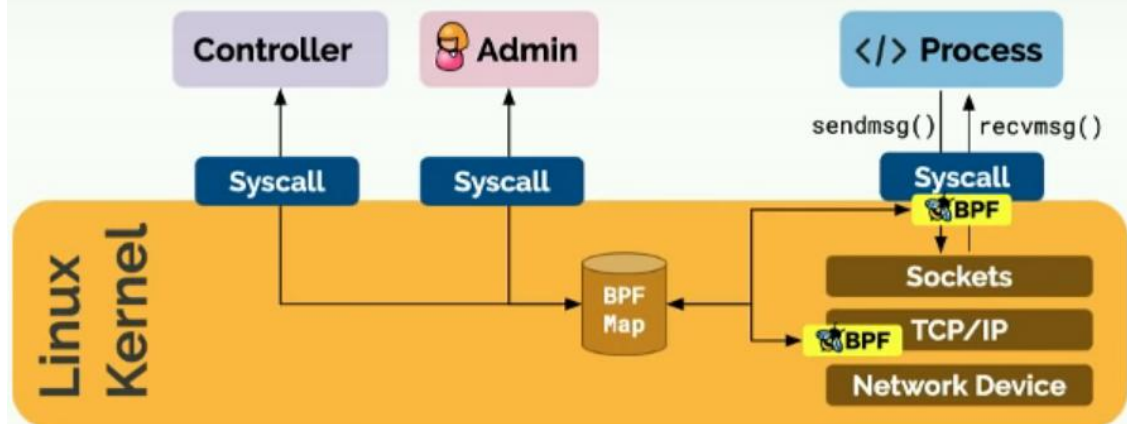
eBPF – Rethinking the Linux Kernel, Thomas Graf, QCon 2020
([presentation](#), [transcript](#))

eBPF Hooks



Where can you hook? kernel functions (kprobes), userspace functions (uprobes), system calls, fentry/fexit, tracepoints, network devices (tc/xdp), network routes, TCP congestion algorithms, sockets (data level)

eBPF Maps



Map Types:

- Hash tables, Arrays
- LRU (Least Recently Used)
- Ring Buffer
- Stack Trace
- LPM (Longest Prefix match)

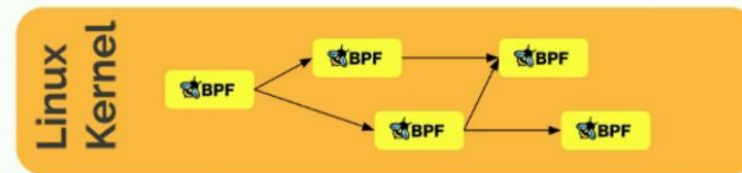
What are Maps used for?

- Program state
- Program configuration
- Share data between programs
- Share state, metrics, and statistics with user space

Accessed via file descriptors

eBPF – Rethinking the Linux Kernel, Thomas Graf, QCon 2020
([presentation](#), [transcript](#))

eBPF Tail and Function Calls



What are Tail Calls used for?

- Chain programs together
- Split programs into independent logical components
- Make BPF programs composable

What are Functions Calls used for?

- Reuse functionality inside of a program
- Reduce program size (avoid inlining)

eBPF Helpers



What helpers exist?

- Random numbers
- Get current time
- Map access
- Get process/cgroup context
- Manipulate network packets and forwarding
- Access socket data
- Perform tail call
- Access process stack
- Access syscall arguments
- ...



eBPF – summary

“It seems that the Linux kernel continues its march towards becoming BPF runtime-powered microkernel.”
— Toke Høiland-Jørgensen, December 2019



Alexei Starovoitov

- eBPF is a **programming language** and **runtime** to extend operating systems.
- eBPF is like JavaScript but for kernel developers.
- In-kernel *just-in-time compiler*.
- eBPF features
 - Secure (runtime verification, sandbox concept, program signing),
 - Efficient (JIT compiler, embedded in OS, per-CPU data structures),
 - Portable (generic bytecode, data type discovery, stable API to OS).
- Many places to **attach programs**: packet filters, tracepoints, security policies, ...
- Enable the **addition of new functionality** – no kernel hacking required.
- Highly flexible kernel configuration.
- eBPF is controlled by the **eBPF Foundation** (Meta, Google, Isovalent, Microsoft, RedHat, Netflix, ...)

[The Beginner’s Guide to eBPF](#), Liza Rice (live programming + source code)

[What is eBPF?](#) – eBPF portal

[BPF at Facebook](#), Performance Summit 2019, Alexei Starovoitov

[BPF at Facebook](#), ([slides](#)) Kernel Recipes 2019, Alexei Starovoitov

[A thorough introduction to eBPF](#) (four articles in lwn.net), Matt Fleming, December 2017.

[BPF compiler collection](#) (BCC - Tools for BPF-based Linux IO analysis, networking, monitoring, and more)



eBPF – official documentary

https://www.youtube.com/watch?v=Wb_vD3XZYOA (30 min)

In **2014**, a group of engineers at Plumgrid needed to find an innovative and cost-effective solution to handle network traffic in SDN environments. What they created was a landmark in the industry known as the extended **Berkeley Packet Filter** (or **eBPF**). This vital technology allows user-level code execution inside the Linux Kernel, transforming network traffic handling for SDN environments. Whether these engineers knew it or not, they had just revolutionized the Linux Kernel.

- Growth of Linux and SDN
- PLUMgrid
- Initial Patch Submission
- eBPF Merged into the Linux Kernel
- Hyperscalers Adopt eBPF
- Cilium Bring eBPF to End Users
- DockerCon 2017 eBPF Takes Off
- eBPF Expands to Security
- eBPF on Windows
- eBPF Everywhere

Thomas Graf
Daniel Borkmann
Chris Wright
Liz Rice
Purvi Desai (Google)
David Miller (network kernel maintainer)
Alexei Starovoitov
Brendan Gregg
Dave Thaler (Microsoft)



<https://ebpfdocumentary.com/>



What is going on in the kernel – kernel reports

This talk will review recent events in the kernel development community, discuss the current state of the kernel and the challenges it faces, and look forward to how the kernel may address those challenges.

- [The Kernel Report, Jonathan Corbet, Open Source Summit 2024](#) Japan (October 2024, 21 min)
- [The Kernel Report, Jonathan Corbet, Open Source Summit 2024](#) North America (April 2024, 42 min)
- [The Kernel Report, Jonathan Corbet, Open Source Summit 2023](#)
- [The Kernel Report, Jonathan Corbet, Open Source Summit 2022](#)
- [The Kernel Report, Jonathan Corbet, Linux Plumbers Conference 2021](#) (starting from 6:45)
- [The Kernel Report](#), Jonathan Corbet, LPC 2020, 2020 edition.
- [The Kernel Report](#), Jonathan Corbet, linux.conf.au 2019 edition.
- [Linux Weekly News](#)
 - [Kernel index](#)
 - [Conference index](#)



Jonathan Corbet, Executive Editor, LWN.net & Kernel Documentation Maintainer



The Kernel Report 2024 – what to expect in 2025?



- Rust ([Committing to Rust in the kernel](#))
 - Memory-safe systems programming, eliminates whole classes of bugs.
 - First merged in October 2022 for 6.1 (as an experiment), slow development.
 - Conclusion from the Kernel Maintainers Summit in 2024: Rust in the kernel is viable.
 - Rust concerns – not easy to learn, language stability, Rust/C API correspondence, getting abstractions upstream – if they are not used yet.
 - What might be merged (lots of infrastructure, Nova – NVIDIA GPU driver, Apple GPU driver).
- CPU scheduler creativity
 - For decades there could be only one CPU scheduler.
 - [Sched_ext](#) – a new scheduling class, based on BPF, merged in November 2024 for 6.12.
 - Anybody can write a CPU scheduler.
 - Quick and safe iteration.
 - Focus on one use case.
- Security
 - The problems with [CVE](#) (Common Vulnerabilities and Exposure) numbers
 - Many vulnerabilities never get CVEs.
 - The kernel is now a CNA – Certificate Numbering Authority.

www.youtube.com › watch

Asahi Linux Lead Developer Resigns: Why



While the reluctance to accept Rust into the Kernel is understandable, the degree of animosity and disdain towards Rust developers is ...

YouTube · Nicco Loves Linux · 5 dni temu

www.youtube.com › watch

The Linux Rust Drama Reaches a NEW High.. More Step Down



The Drama continues after Linus Torvalds got involved and the Asahi Linux project lead is now not only leaving the kernel dev but their own ...

YouTube · SavvyNik · tydzień temu

www.youtube.com › watch

The Rust vs C Linux Divide.. Another Rust Maintainer Steps ...



Linus the creator of Linux has to step in about the latest Rust vs C Linux Drama. Leading to the step down of another Rust for Linux ...

YouTube · SavvyNik · 2 tygodnie temu

www.youtube.com › watch

Asahi Dev Rage Quits Linux kernel Maintenance over Rust ...



Asahi #linux #kernel #rust #maintainer #quit #Ad: laptops & more @Amazon: ...

YouTube · Code Therapy w/ René Rebe · 2 tygodnie temu



The Kernel Report 2024 – what to expect in 2025?



- Better tools
 - The kernel community has been slow to adopt new tools.
 - Patch management ([b4](#) continues to improve).
 - Bug tracking (bugzilla, bugspray – being developed, regression tracking – currently unfunded).
 - Testing (kernel selftests growing, KernelCI, stable update testing).
- Dark areas in the kernel
 - Documentation.
 - Build system.
 - Many core-kernel areas.
 - Drivers for older hardware.
 - Maintainers (lack support, burned-out, frustrated, ...)

Summary

Releases: every 9-10 weeks, as always
Rust: it's happening, if slowly
Sched_ext: an exciting new community
CVEs: they will keep coming
Tools: getting better



The Kernel Report 2024



- BPF

- BPF Tokens (fine-grained capabilities for loading BPF programs) – more security.
- BPF Arena – mechanism for sharing memory between BPF programs running in the kernel and user space programs.
- Fuse-BPF.
- BPF network device.
- Challenges (complexity, resistance like sched_ext).

- Confidential computing

- Security in a hostile environment.
- How to re-establish trust?
- A secure, verified boot chain.
- CPU attestation.
- Encrypted memory.
- Vendors working on supporting this idea (AMD SEV+SNP, Intel TDX, ARM CCA).

No time for

EEVDF scheduler	Realtime preemption
Tasklets	Anonymous folios
IOCost	Deadline servers
Kernel hardening	Stable kernel mgmt
Memory tiering	Code tagging
mseal()	composefs
Kernel-text replication	Software interrupts
Shadow stacks	...

Slides: <https://lwn.net/talks/2024/kr-ossna.pdf>



2024 Kernel Maintainers Summit group photo





The Kernel Report 2023



- **Rust**

- Has a lot to offer (a stronger type system, no undefined behavior, attractive to newer developers).
- Why **not** Rust in the kernel (a new language adds complexity, the language is still evolving – quickly, maintainers will need to learn Rust, a difficult learning curve, lots of glue code, some things are hard to do in Rust, conservatism).
- [Initial Rust infrastructure has been merge into Linux 6.1](#) (October 2022).
- More support code in subsequent kernels (access to existing types and functions ... but safer).
- [A pair od Rust kernel modules](#) (NVM Express driver, 9P filesystem server)
- Nothing in a production kernel yet, nothing that anybody is actually using.
- Rust support was merged as an **experiment**.
- The Rust **decision** point is coming soon.

“There are possibly some well-designed and written parts which have not suffered a memory safety issue in many years. It’s insulting to present this as an improvement over what was achieved by those doing all this hard work.”
— a longtime kernel developer

Rust-for-Linux developer Miguel Ojeda





The Kernel Report 2023



- **The maintainership crisis**

- Increasing demands.
- Understaffing.
- Lack of employer support (many maintainers are not paid to maintain).
- Kernel fuzzers (bad quality bug reports).
- Maintainers.
 - An unparalleled depth of skills and experience.
 - But also resistance to change (e.g. Rust), lack of diversity, increasingly tired single points of failure.
 - Preparing for change (shared maintenance duties, documentation, investment in tools).
- Linux Kernel Contribution Maturity Model
<https://www.kernel.org/doc/html/latest/process/contribution-maturity-model.html>

Being maintainer feels like a punishment, and that cannot stand. We need help.
— Darrick Wong

Maintainers/longtime developers are burning out.
— Josef Bacik

Slides: <https://lwn.net/talks/2023/kr-osseu.pdf>



2023 Kernel Maintainers Summit group photo



Back row: Alexei Starovoitov, Dave Chinner, Konstantin Ryabitsev, Thomas Gleixner, Jakub Kicinski, Josef Bacik, Dan Williams, Wolfram Sang, Rafael Wysocki, Kees Cook, Greg Kroah-Hartman, Jiri Kosina.

Front row: Tejun Heo, Jens Axboe, Sasha Levin, Ted Ts'o, Jan Kara, Linus Torvalds, Martin Petersen, Christian Brauner, Steve Rostedt, Arnd Bergmann.

Floor: Miguel Ojeda.



The Kernel Report 2022

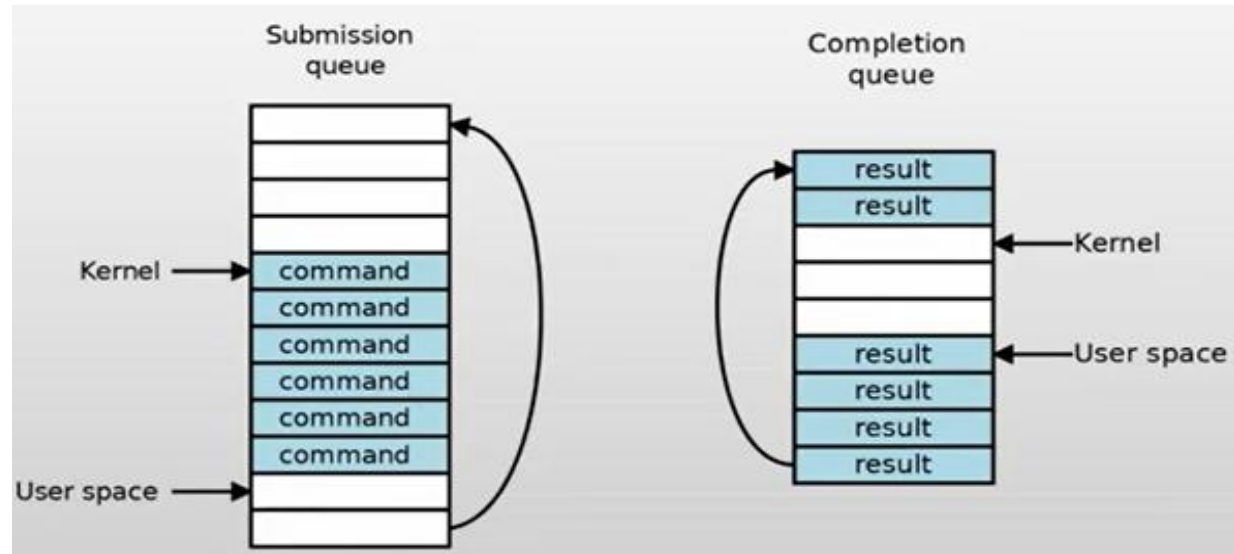


- **io_uring**

- System calls slow down your program.
- Shared memory area (user, kernel).
- What it brings
 - Asynchronous operations.
 - Submission/results without system calls.
 - Registered files and buffers.
 - A wide range of commands.
 - Chained operations.

When the kernel encounters a chain, it will only initiate the first operation; the next operation in the chain will only run after the first completes.

- [User-space block driver using io_uring \(ublk\)](#)
- Is io_uring the basis for future microkernel architecture?



io_uring is an alternative, high-performance API that runs within the kernel



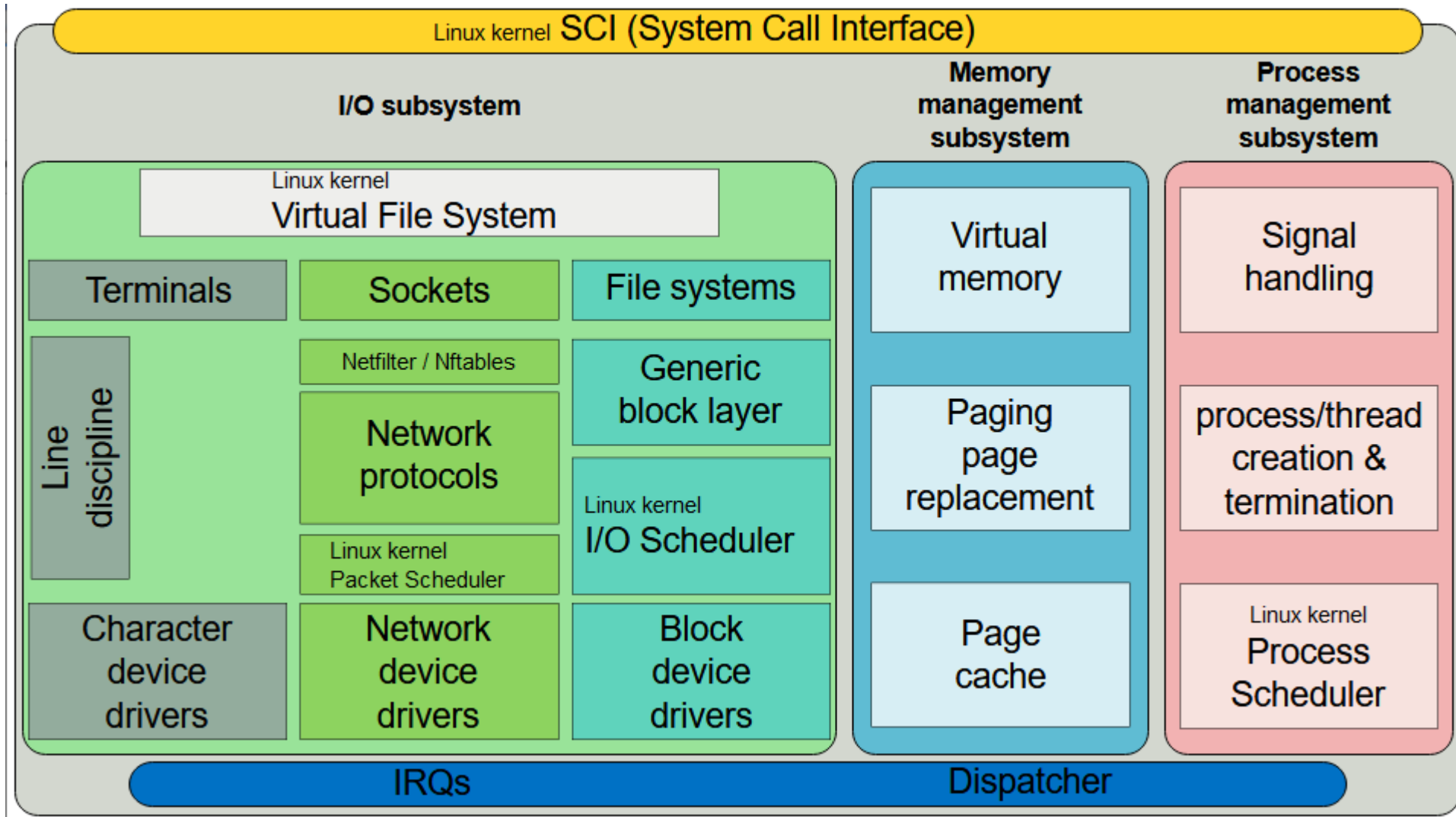
Linux structure and kernel functions

Basic concepts

- Linux structure and kernel functions
- Basic concepts – process, user mode and kernel mode, context switch, system calls, user stack and kernel stack, process state transitions



Linux – the structure and functions of the kernel



Source: [Wikipedia](https://en.wikipedia.org/wiki/Linux_kernel)



Process, address space, context

- **Process** is a program in execution; execution runs sequentially, according to the order of instructions in a **process address space**.
- **Process address space** is a collection of memory addresses, referenced by the process during execution.
- **Process context** is its operational environment. It includes contents of general and control registers of the processor, in particular:
 - program counter (PC),
 - stack pointer (SP),
 - processor status word (PSW),
 - memory management registers (allow access to code and data of a process).
- Linux is a **multiprogramming** system. The kernel dynamically allocates resources necessary for processes to operate and provides security.

For this purpose, it needs **hardware support**:

- processor executing in two modes: **user mode** and **system mode (kernel mode)**,
- privileged instructions and memory protection,
- interrupts and exceptions.



Kernel address space

System address space or **kernel space** comprises code and kernel data structures. Access to them is only possible in system mode. The kernel has direct access to the address space of the current process. Occasionally, it can reach up to address space of the other process than the current one.

Kernel thread is executed in kernel mode.

The transition to the execution of the kernel code can occur as a result of several events:

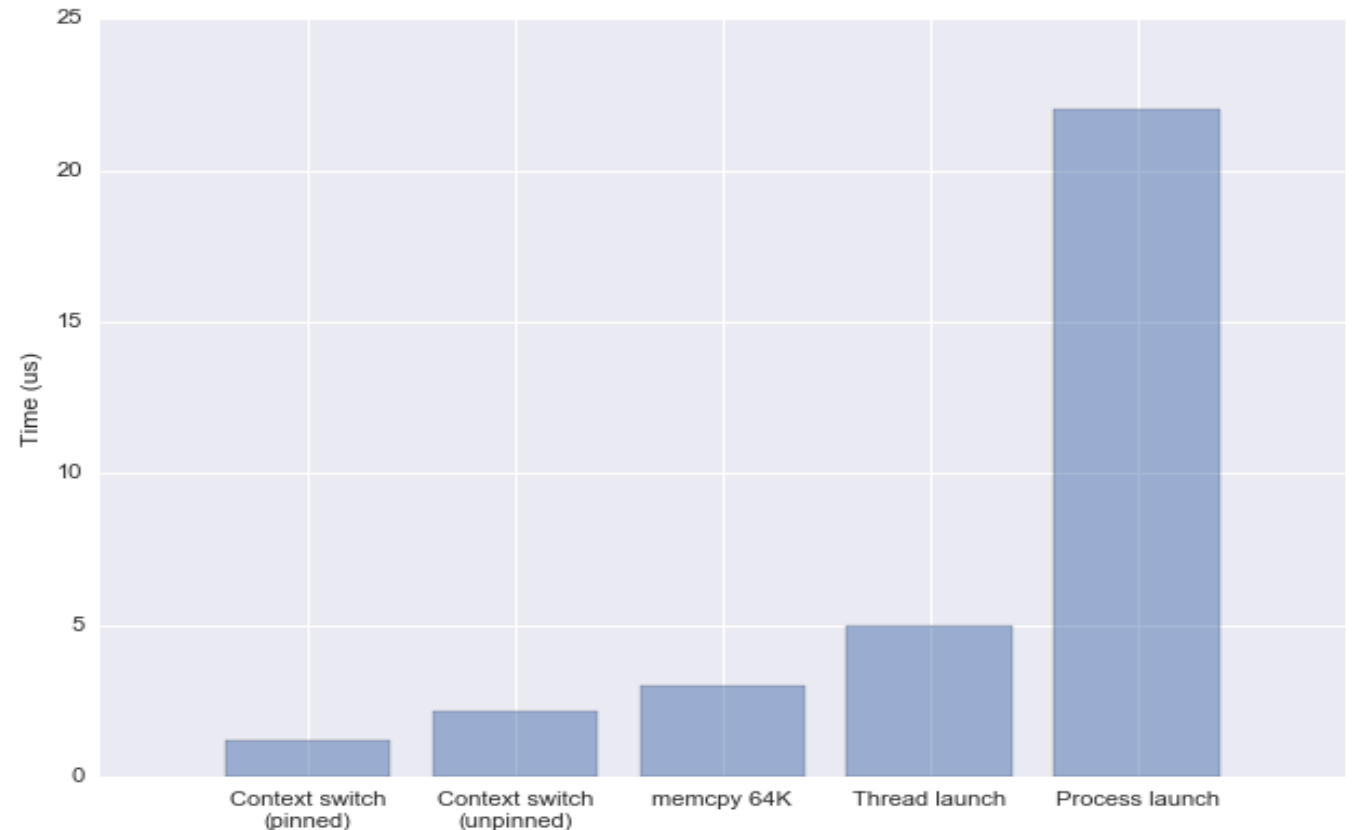
- The process calls the **system function** (*system call*). The user process instructs the kernel to perform certain actions (e.g. I/O operations) on its behalf.
- The processor reports **exception** while executing the process, e.g. a non-existent instruction. The kernel handles an exception on behalf of the process that caused it.
- An external device reports **an interrupt** to the CPU informing about the occurrence of an asynchronous event, e.g. completion of an input-output operation. Interrupt support is handled in the *interrupt handling routine*.



Context switching

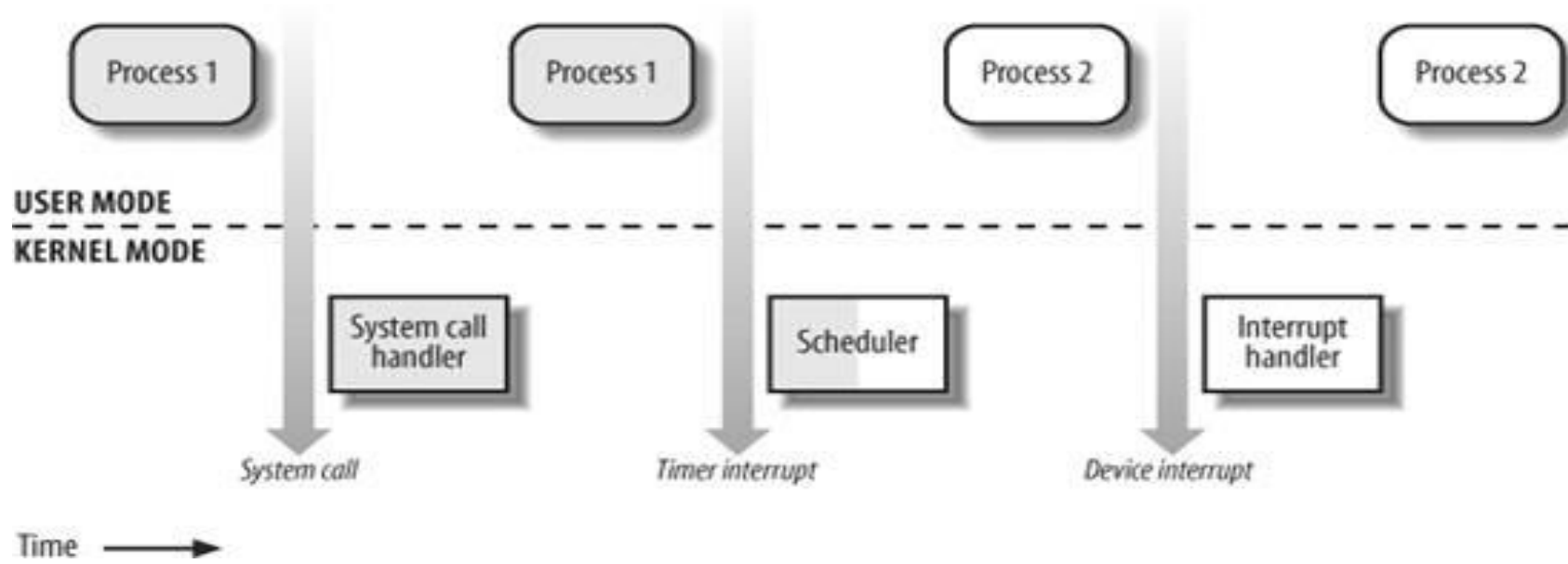
Context Switching – saving the context of the current process (in the structure that is part of the process address space) and loading the context of another process into the processor registers.

The context switch time is an overhead of the system and depends on hardware support (can take from a few **100 nanoseconds** to a **few microseconds**).

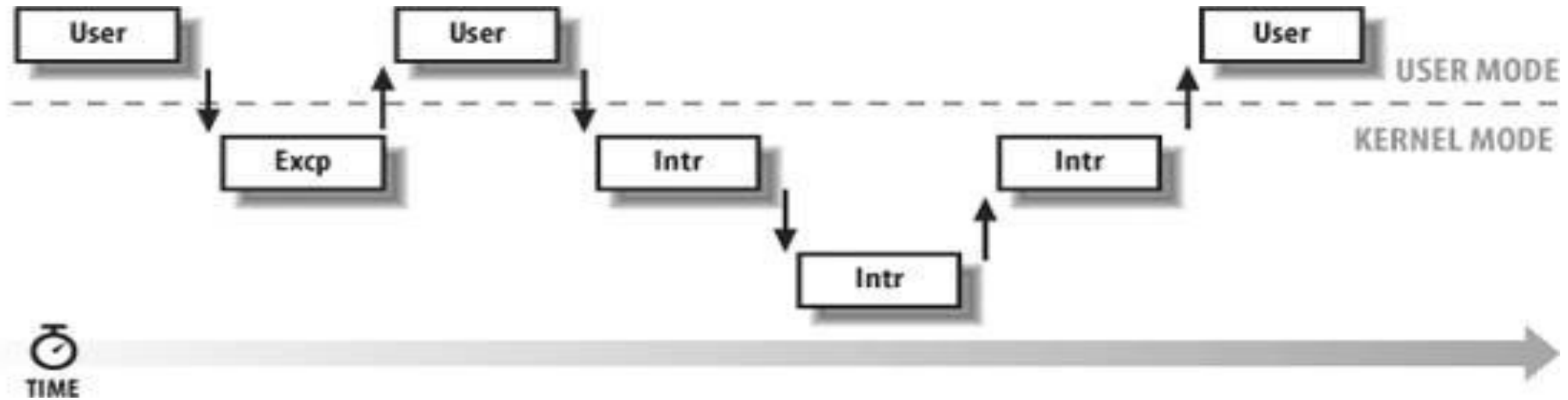


[Measuring context switching and memory overheads](#)

Context switching **itself has a cost in performance**, due to running the task scheduler, TLB flushes, and indirectly due to sharing the CPU cache between multiple tasks. L2 cache have substantial impact on the cost of context switch.



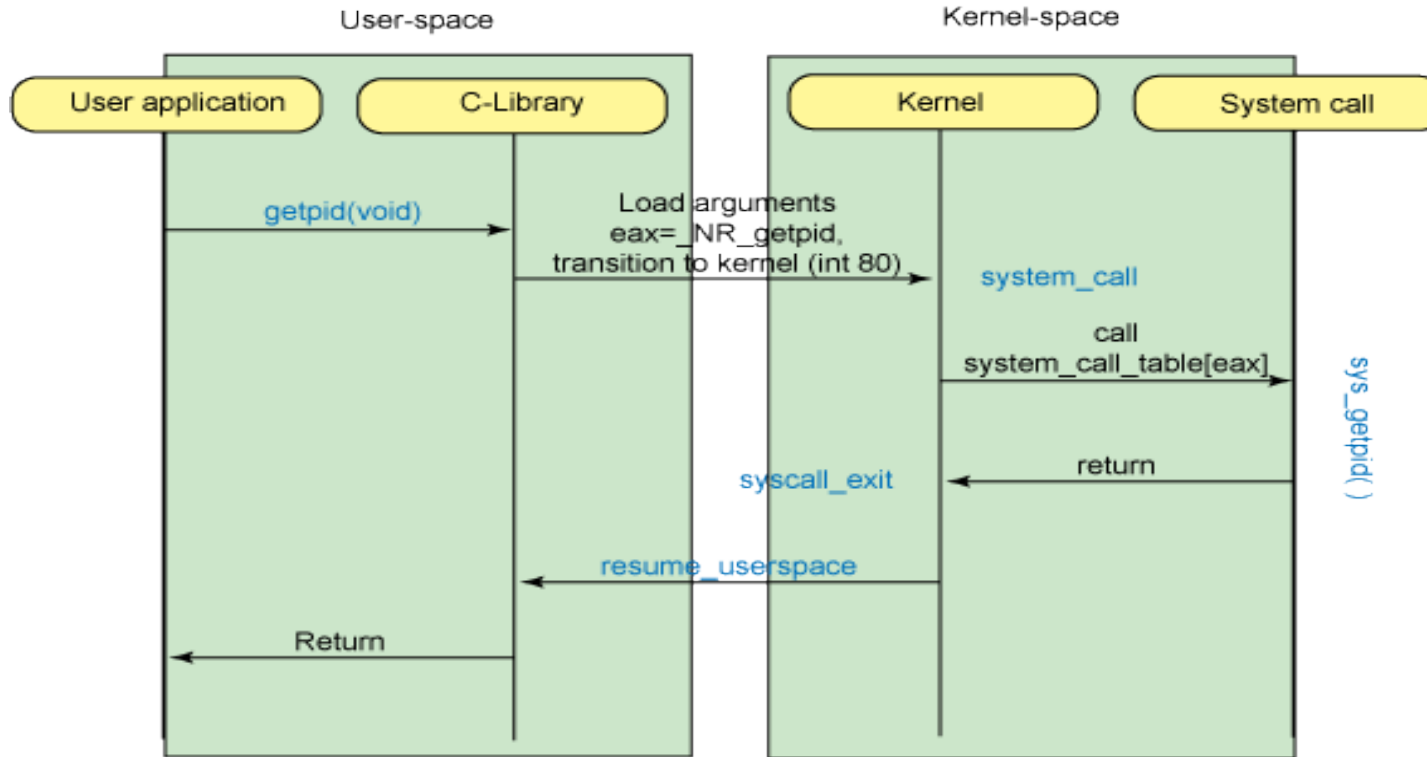
Transitions between user and kernel mode, source: Bovet, Cesati



Interleaving of kernel control paths, source: Bovet, Cesati



System function call with int 0x80



source:
Anatomy of the
Linux kernel,
M.Tim Jones

The details of the system function call depend on the architecture (the figure illustrates i386). The register **eax** is used to transmit the number of the function being called. The machine instruction **int 0x80** calls the program interrupt 0x80 (decimal 128) – context switching and calling the kernel function **system_call**. The function transfers control to the proper system function (uses **system_call_table** with `eax` treated as an index).

After returning from the system function, the **syscall_exit** function is executed, the **resume_userspace** function call returns the control back to the user space.

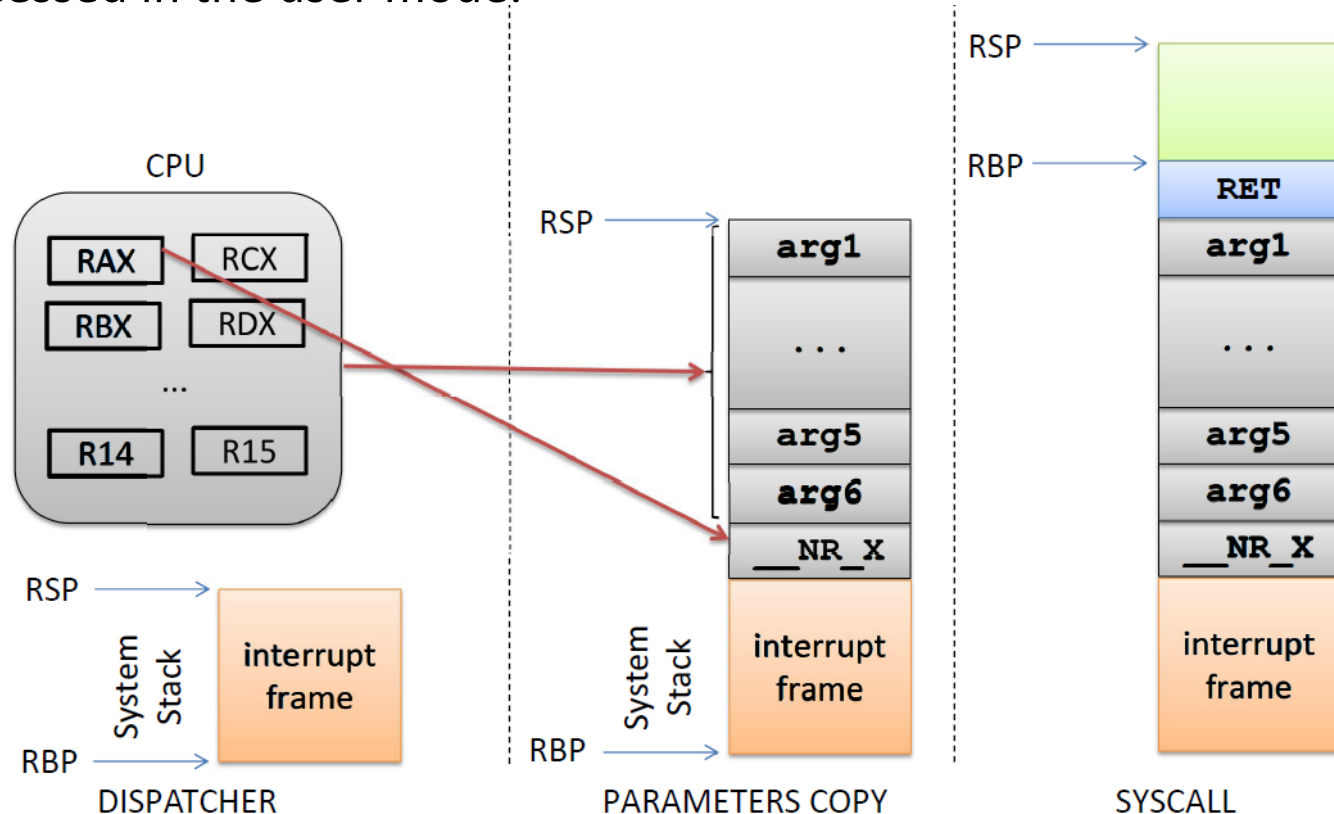


System call and process stacks

Each process uses **two stacks**:

- **user stack** – used in user mode (grows dynamically during program execution),
- **kernel stack** – in kernel mode (has a fixed, small size); is usually allocated in address space of the process, but it can not be accessed in the user mode.

system_call() starts by saving the registers in the kernel stack. After checking other things such as validating parameters, it will call the respective system call.





System call – sequence of steps

System calls: <https://linux-kernel-labs.github.io/refs/heads/master/lectures/syscalls.html>

This is what happens during a **system call**:

1. The application is setting up the **system call number** and **parameters** and it issues a trap instruction.
2. The execution mode switches from **user** to **kernel**; the CPU switches to a kernel stack; the user stack and the return address to user space is saved on the kernel stack.
3. The kernel entry point **saves registers** on the **kernel stack**.
4. The system call dispatcher **identifies the system call** function and **runs it**.
5. The user space **registers** are **restored** and **execution** is **switched back** to user (e.g. calling IRET).
6. The user space application **resumes**.

See also: [Shadow stacks for user space](#), Jonathan Corbet February 21, 2022

Whenever a function is called, the return address is pushed onto both the **regular stack** and the **shadow stack**. When that function returns, the return addresses are popped off both stacks and compared; if they fail to match, the system goes into red alert and kills the process involved.



System call conventions

Definition of the system function from the C level (file include/linux/syscalls.h):

```
asmlinkage long sys_exit (int error_code);
```

asmlinkage tells compiler to look on the kernel stack for the function parameters, instead of registers.

In architecture x86 the registers **ebx**, **ecx**, **edx**, **esi** and **edi** are used to pass the first five parameters. If there are more parameters, it is through one register that a pointer to the user's address space is transferred, where all parameters are placed.

The value passed from the system function is placed in the **eax** register.

Other registers are used in 64-bit architecture:

- [x64 Architecture, registers, calling conventions, addressing modes](#)
- [syscall numbers](#)

Copying data between the kernel space and the user space is done using **copy_to_user()** and **copy_from_user()**.

When executing the system function, the kernel works **in the context of the process** (the variable **current** points to the current process).

eax	return value
edx	dividend register
ecx	count register
ebx	local register variable
ebp	stack frame pointer (optional)
esi	local register variable
edi	local register variable
esp	stack pointer



Sysenter and sysexit

Machine instructions **sysenter** and **sysexit** were added to x86 processors (newer than Pentium II). They allow a **faster transition** (return) to the kernel mode to perform a system function than using the **int** statement. Support for this mechanism has been added to the Linux kernel ([Sysenter Based System Call Mechanism in Linux 2.6](#)).

Calling the x86 function

- 64-bit version – defined in the file [arch/x86/entry/entry_64.S](#)
- 32-bit version – defined in the file [arch/x86/entry/entry_32.S](#)

Content of the system function table

- 64-bit version – defined in the file [arch/x86/entry/syscalls/syscall_64.tbl](#)
- 32-bit version – defined in the file [arch/x86/entry/syscalls/syscall_32.tbl](#)

This is the beginning

```
# The abi is always "i386" for this file.
#
0      i386    restart_syscall    sys_restart_syscall
1      i386    exit                sys_exit
2      i386    fork                sys_fork
3      i386    read                sys_read
4      i386    write               sys_write
5      i386    open                sys_open                compat_sys_open
6      i386    close               sys_close
7      i386    waitpid            sys_waitpid
8      i386    creat               sys_creat
```

In other operating systems, there are many more functions than **435** in Linux 5.6 (32-bit).



Process and system context

Context of execution – summary:

- **user code** is executed in **user mode** and in **process context**, can only reach the address space of the process,
- **system functions and exceptions** (e.g. dividing by zero or violation of memory protection) are supported in **system mode**, but **in context of the process**, they have access to the **process and system address space**.

The kernel acts on behalf of the current process (e.g. by executing a system function), it can reference the address space of the process and the process stack. It can also block the current process if it has to wait for resources.

- **interrupts** are handled in **system mode** in the **context of the system** with access only to the **system address space**.

System-wide operations, such as recalculating priorities or handling an external interrupt. Not performed on behalf of any particular process and therefore take place in the context of the system. The kernel does not reach to the address space or the stack of the current process, also it can not block.



Process state transitions

The Linux kernel is **preemptable** and **re-entrant**, it can support different processes concurrently.

The process during execution changes **state**.

The basic states of the process are:

- **new**: the process has been created,
- **ready**: the process is waiting for the processor to be allocated,
- **executed** (more precisely: **executed in user mode** or **executed in system mode**): process instructions are executed,
- **waiting**: the process is waiting for an event to occur,
- **finished**: the process completed execution.

