

Linux操作系统

****

**2024 至 2025 学年第 二 学期**

论文题目：Linux的版本演化及其特点

学生姓名：

学院专业班级：

学 号：

任课教师：

计算机学院

目录

[一、引言 1](#_Toc198583430)

[二、版本演化中的技术革新与特点 1](#_Toc198583431)

[2.6版本（2003-2011） 1](#_Toc198583432)

[O(1)调度器 1](#_Toc198583433)

[SELinux集成 3](#_Toc198583434)

[3.x版本（2011-2015） 4](#_Toc198583435)

[Btrfs文件系统 4](#_Toc198583436)

[统一设备树（Device Tree） 6](#_Toc198583437)

[4.x版本（2015-2019） 7](#_Toc198583438)

[Live Patching热补丁 7](#_Toc198583439)

[eBPF虚拟机 9](#_Toc198583440)

[5.x版本（2019-2022） 11](#_Toc198583441)

[WireGuard VPN协议 11](#_Toc198583442)

[NTFS3驱动 14](#_Toc198583443)

[6.x版本（2022-至今） 15](#_Toc198583444)

[Rust语言支持 15](#_Toc198583445)

[三、总结 17](#_Toc198583446)

[四、参考文献 18](#_Toc198583447)

# 一、引言

作为开源软件史上最成功的协作范本，Linux操作系统的版本迭代史堪称一部动态演进的计算机科学编年史，Linux的每个版本跃迁都精准映射着计算范式的时代需求。

1991年的初始版本通过抢占式多任务和虚拟内存机制，为开源协作奠定了技术基底；2003年的2.6内核凭借O(1)调度器和SELinux安全框架，开启了企业级应用的黄金时代；2015年后的4.x系列以eBPF虚拟机和热补丁技术，重新定义了内核可观测性与动态更新边界。

这些里程碑不仅是代码的积累，更是对冯·诺依曼架构的持续解构与重构——当设备树解耦硬件描述、Btrfs革新存储范式、Rust重构内存安全时，Linux始终站在操作系统演进的前沿阵地。

# 二、版本演化中的技术革新与特点

每个Linux大版本都在特定历史断面中镌刻着突破性的创新印记。2.6内核在21世纪初的登场，实质是应对多核时代算力爆发的前瞻布局；3.x系列对设备树和Btrfs的整合，折射出移动互联网浪潮下硬件异构化与数据爆炸的双重挑战；而4.x到6.x版本的进化轨迹，则清晰勾勒出云计算、零信任安全、内存安全等当代命题的技术应答。

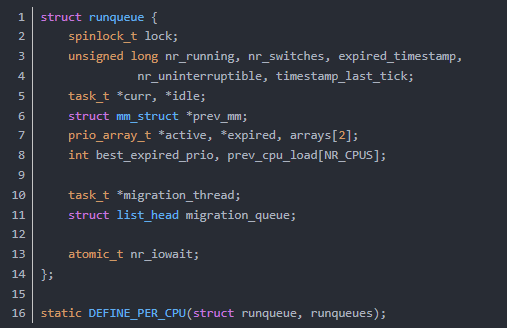
接下来通过六大标志性版本的解剖，本文将揭示技术决策如何通过自我迭代与范式融合，持续重塑操作系统的能力边界。

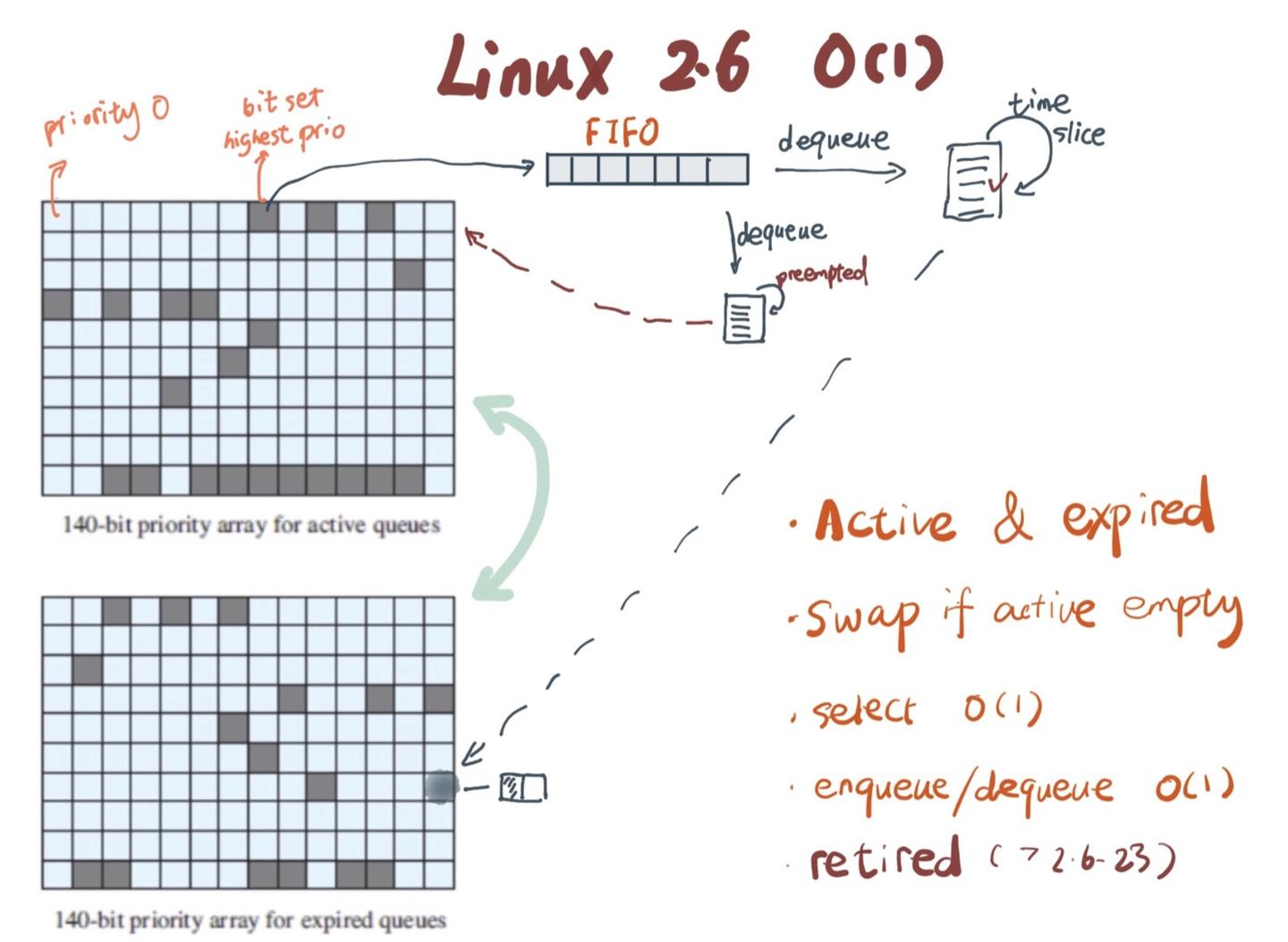
## 2.6版本（2003-2011）

### O(1)调度器

在Linux 2.6之前的版本中，调度器采用O(n)复杂度算法，其性能随进程数量增长而线性下降。例如，进程数量较多时，调度器需遍历整个就绪队列以选择下一个运行进程，导致高负载场景下系统性能急剧降低。

此外，多核处理器（SMP）架构中全局运行队列的锁竞争问题进一步加剧了性能瓶颈。为解决这些问题，2.6内核引入了O(1)调度器，其核心目标是通过常数时间复杂度的算法提升调度效率。

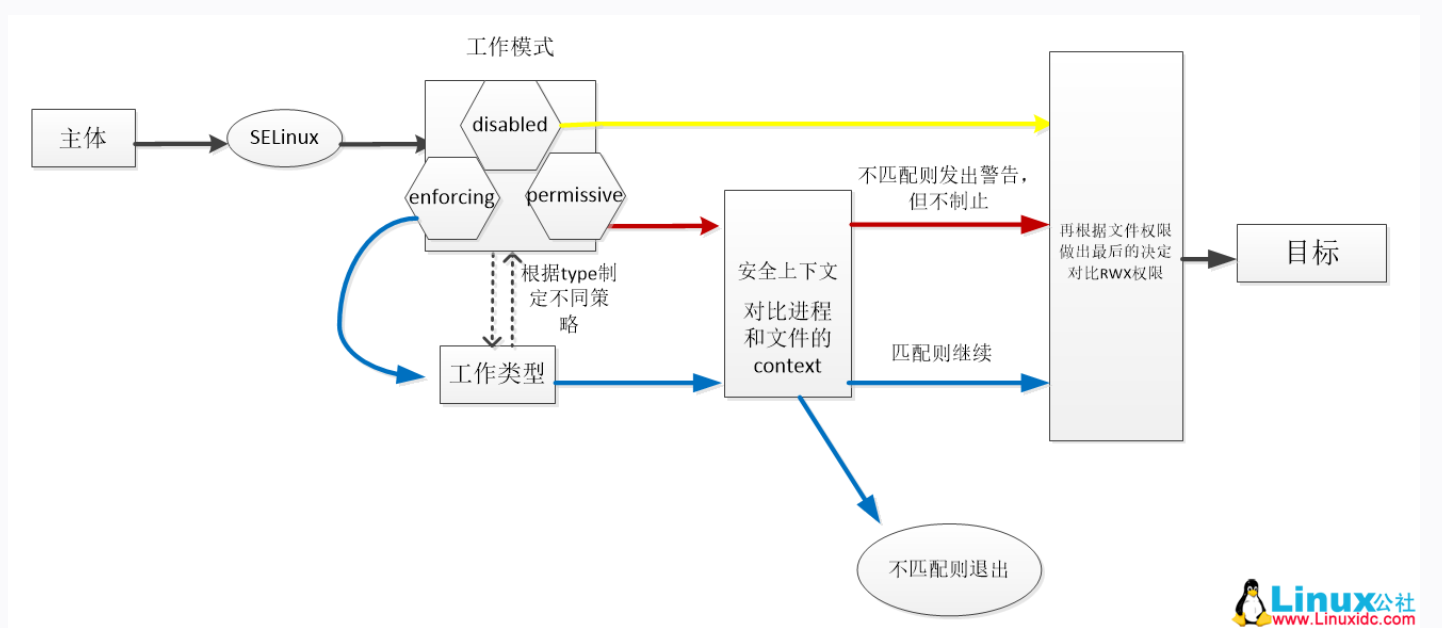
Linux内核中O(1)调度器的相关源码：  


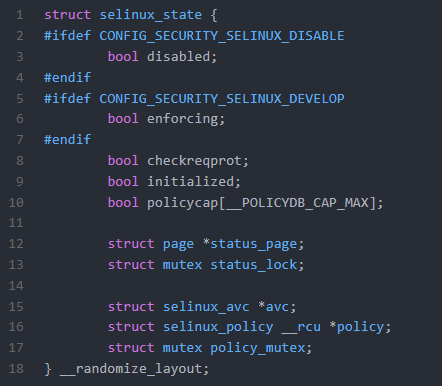
下图展示了Linux内核中O(1)调度器的工作原理：  
  
 2.6的内核中有 140 种优先级，所以我们就用长度为140的array去记录优先级。每个优先级下面用一个FIFO queue管理这个优先级下的process。新来的插到队尾，先进先出。在这种情况下，insert / deletio都是 O(1)。

在2.6版本的调度器里，聪明的先贤们采用bitarray。它为每种优先级分配一个 bit，如果这个优先级队列下面有 process，那么就对相应的bit染色，置为1，否则置为0。这样，问题就简化成寻找一个bitarray里面最高位是1的bit（left-most bit），这基本上是一条CPU指令的事(fls)。也就实现了O(1)的调度器。  
 虽然O(1)调度器目前已经被性能略输一筹，同时更加强调公平性的 CFS（Completely Fair Scheduler）取代，但其以独特的设计，简单的算法，依旧影响了很多系统的设计。

### SELinux集成

SELinux(Secure Enhanced Linux)安全增强的Linux是由美国国家安全局NSA针对计算机基础结构安全开发的一个全新的Linux安全策略机制。SELinux可以允许系统管理员更加灵活的来定义安全策略。 SELinux是一个内核级别的安全机制，从Linux2.6内核之后就将SELinux集成在了内核当中，因为SELinux是内核级别的，所以我们对于其配置文件的修改都是需要重新启动操作系统才能生效的。 现在主流发现的Linux版本里面都集成了SELinux机制，CentOS/RHEL都会默认开启SELinux机制。

下图是SELinux的工作原理：  


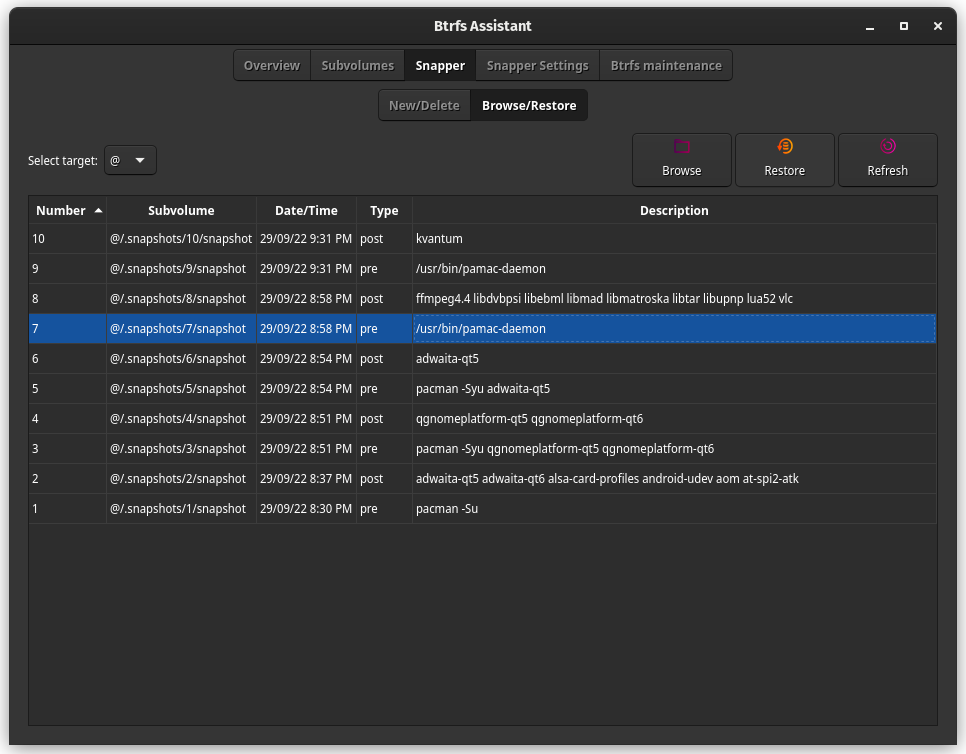
SELinux的相关代码（SELinux的状态）：  


系统资源都是通过进程来读取更改的，为了保证系统资源的安全，传统的Linux使用用户、文件权限的概念来限制资源的访问，通过对比进程的发起用户和文件权限以此来保证系统资源的安全，这是一种自由访问控制方式（DAC）；但是随着系统资源安全性要求提高，Linux引入了SELinux，该机制为进程和文件加入了除权限之外更多的限制来增强访问条件，这种方式为强制访问控制（MAC）。这两种方式最直观的对比就是，采用传统DAC，root可以访问任何文件，而在MAC下，就算是root，也只能访问设定允许的文件，很大程度上保证了用户的安全。

## 3.x版本（2011-2015）

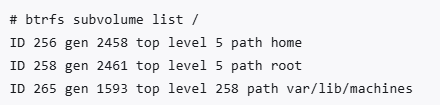
### Btrfs文件系统

Btrfs 是一种现代写时复制文件系统，具有容错、修复和轻松管理的高级功能。 Btrfs 对所有文件使用写时复制，这意味着如果文件被修改/写入存储，该文件不会被替换，而是会创建该文件的副本。此机制有助于创建最小大小的快照，因为创建快照时不需要复制未修改的文件。

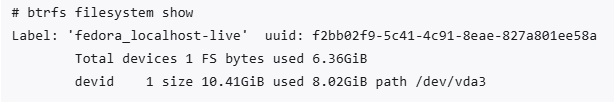
下图是Brtfs的快照功能：  
Btrfs拥有以下特点：

1. 写入时复制（COW）机制​​:采用COW技术，修改数据时创建副本而非直接覆盖，确保数据一致性和安全性。  
   支持事务性操作，避免因系统崩溃导致数据损坏。
2. ​​扩展性与B-Tree结构​​  
   所有元数据（如inode、目录项）通过B-Tree管理，提升查找、插入和删除效率。  
   动态分配inode，突破传统文件系统的文件数量限制。基于Extent的存储管理（连续块分配），减少元数据开销。
3. ​​快照与子卷​​  
   ​​快照​​：支持创建只读或可写快照，快速备份和恢复数据，且占用空间小（增量存储）。  
   子卷​​：允许在单个文件系统中划分逻辑分区，独立管理挂载点和权限。
4. ​​数据完整性与修复​​  
   内置校验和（checksum）机制，检测并自动修复静默数据损坏。  
   支持RAID（0/1/10等），冗余存储提升容错能力。

Btrfs 向 Linux 高级文件系统特性集贡献了很多特性，如今已成为Fedora 工作站的默认文件系统，在Fedora中可以轻松操作Btrfs文件系统，如下：

Btrfs查看子卷：  
 

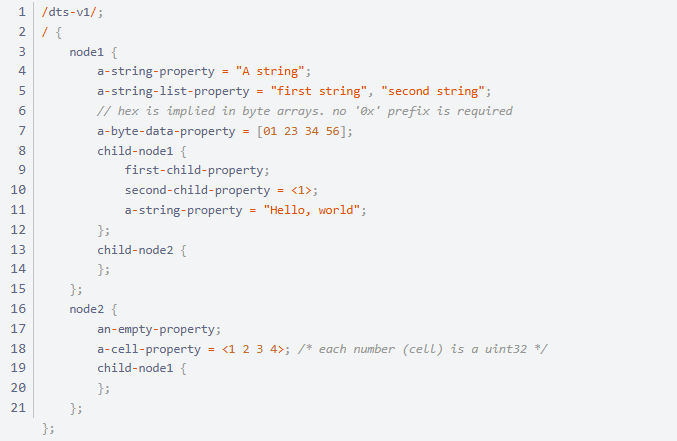
Btrfs新增或删除子卷：  
 

Btrfs查看格式：  


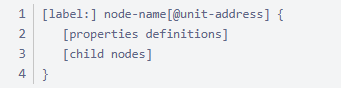
### 统一设备树（Device Tree）

Linux的统一设备树（Device Tree）是硬件描述领域的一次重要革新，其核心创新在于将硬件配置信息从内核代码中解耦，通过树形数据结构实现硬件描述的标准化和模块化。这一机制最初源于PowerPC架构的Open Firmware标准（IEEE 1275），后由ARM社区引入以解决传统“board file”方式导致的代码冗余问题。设备树通过文本格式的源文件（.dts）描述硬件拓扑，再编译为二进制文件（.dtb）由Bootloader传递给内核，实现了硬件描述与内核的分离，使得同一内核镜像可适配多种硬件平台。

从技术细节来看，设备树采用层次化节点结构，每个节点代表一个硬件组件（如CPU、外设），属性则描述其具体配置（如寄存器地址、中断号）。其中关键属性如compatible用于驱动匹配，reg定义地址空间，address-cells和size-cells规范子节点地址格式。设备树还支持模块化设计，通过.dtsi包含文件复用公共配置，显著提升了代码可维护性。

统一设备树的结构：  


设备树的优势体现在三方面：一是硬件抽象化，通过标准化描述减少内核移植成本；二是灵活性，开发者仅需修改设备树文件即可适配新硬件；三是可读性，文本格式的.dts文件便于调试和维护。随着64位架构的普及，设备树规范也扩展了对大地址空间的支持，进一步适应现代计算需求。总体而言，设备树机制是Linux嵌入式生态的重要基石，其设计思想深刻影响了硬件与操作系统的交互方式。

.dts的格式：  
 

## 4.x版本（2015-2019）

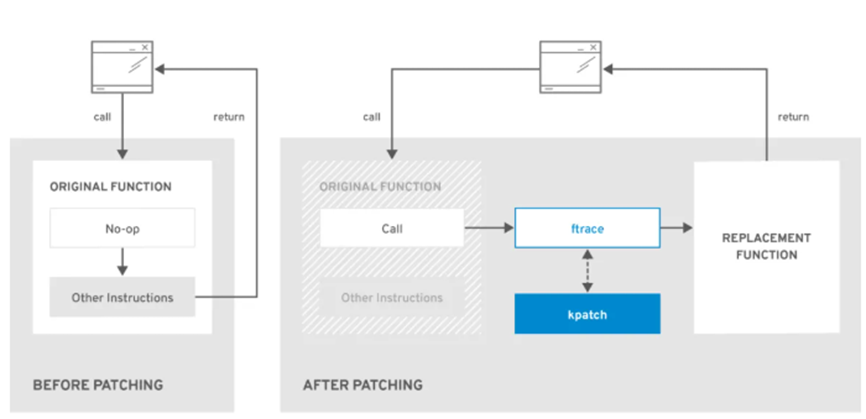
### Live Patching热补丁

Linux内核在4.0版本首次引入的Live Patching热补丁技术，是当时内核动态更新领域的重要里程碑。这一技术的核心突破在于首次将热补丁功能整合进官方内核代码，为实时修复关键漏洞提供了标准化框架，而无需依赖第三方工具（如Ksplice或kGpatch）。

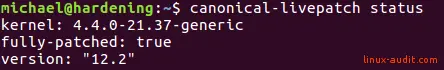
Live Patching通过内核的ftrace机制实现函数级代码替换。其核心流程包括：在目标函数入口处插入ftrace探针，当函数被调用时，ftrace回调函数会截获执行流，并通过修改调用栈中的返回地址（RIP寄存器）将控制权重定向到新函数版本。这一机制依赖动态模块加载能力，补丁以内核模块形式存在，通过stop\_machine()机制短暂冻结所有CPU执行环境，确保原子性替换操作。值得注意的是，此时仅支持x86架构，且要求编译器生成可回溯的栈帧（HAVE\_RELIABLE\_STACKTRACE），这是实现堆栈安全检查的前提。

Live Patching的相关代码：  
 

Live Patching的工作原理：

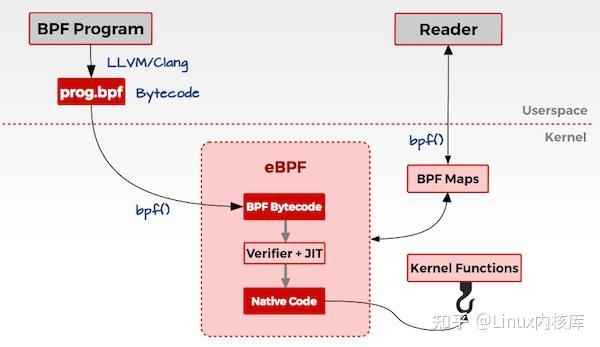
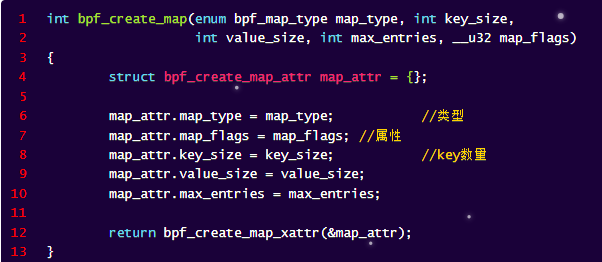


其引入了"原子替换"概念，通过检查所有CPU堆栈中是否存在待替换函数的活动实例，确保补丁应用时系统状态的一致性。若检测到危险调用路径，补丁会延迟到安全时机（如系统调用返回或进程切换）应用。同时在/sys/kernel/livepatch/目录下暴露控制接口，管理员可通过文件操作动态加载/卸载补丁模块，实现运维可视化。

使用Live Patching实时修补Linux内核：  


### eBPF虚拟机

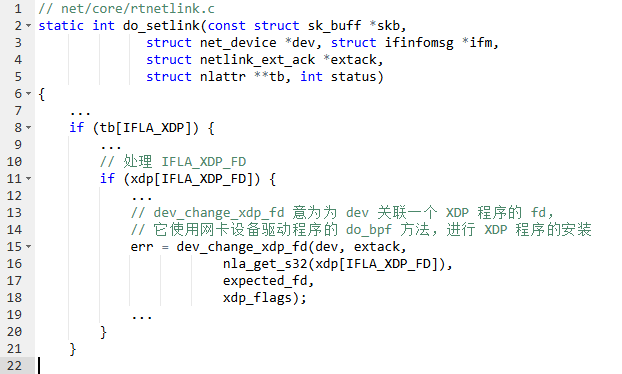
在Linux内核4.x时代，eBPF（Extended Berkeley Packet Filter）的引入标志着内核可编程性的一次革命性突破。其核心设计目标是通过安全、高效的虚拟机机制，允许用户态程序在不修改内核源码或加载内核模块的前提下，动态注入自定义逻辑到内核执行流中。这一创新打破了传统内核开发的静态模式，为实时监控、网络优化和安全防护等领域开辟了全新范式。

下图是eBPF的架构：  
  
 上图中的bpf()函数与BPF Maps的相关代码：  
 



一个完整的 eBPF 程序，通常包含用户态和内核态两部分：用户态程序需要通过 BPF 系统调用跟内核进行交互，进而完成 eBPF 程序加载、事件挂载以及映射创建和更新等任务；而在内核态中，eBPF 程序也不能任意调用内核函数，而是需要通过 BPF 辅助函数完成所需的任务。尤其是在访问内存地址的时候，必须要借助 bpf\_probe\_read 系列函数读取内存数据，以确保内存的安全和高效访问。在 eBPF 程序需要大块存储时，我们还需要根据应用场景，引入特定类型的 BPF 映射，并借助它向用户空间的程序提供运行状态的数据。

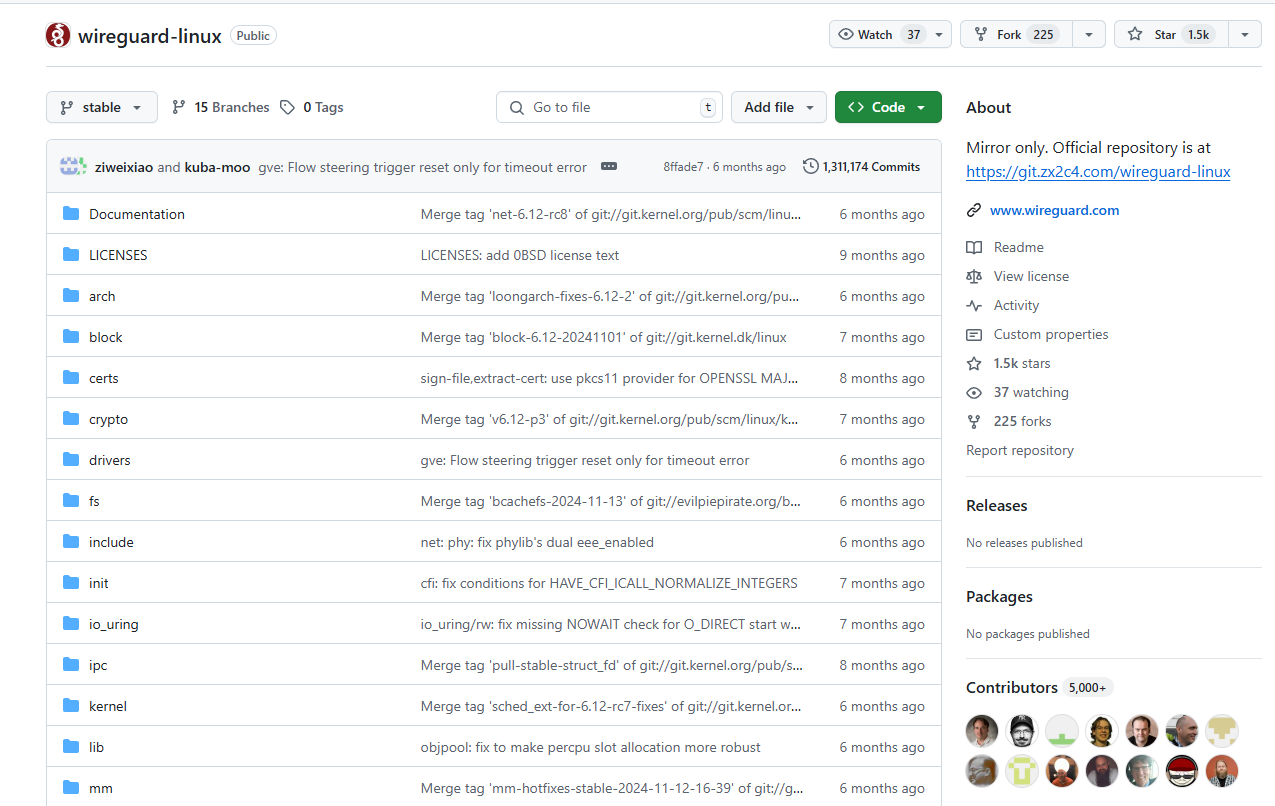
从虚拟机架构层面，eBPF对传统BPF进行了全面升级。寄存器数量从原有的2个32位累加器扩展为10个64位通用寄存器，指令集支持更丰富的算术运算、内存访问和函数调用语义。字节码格式的重新设计使其具备图灵完备性，同时通过禁止循环和限制跳转范围确保程序可终止性。JIT（即时编译器）的深度优化将字节码直接转换为宿主CPU指令，执行效率逼近原生代码，相较解释模式性能提升达4倍以上。这些改进使得复杂逻辑（如协议解析、状态跟踪）的内核态执行成为可能。

Linux内核中的使用到eBPF的程序代码：  


## 5.x版本（2019-2022）

### WireGuard VPN协议

Linux内核在5.6版本（2020年3月发布）引入的WireGuard VPN协议，被视为VPN领域的一次重大技术革新。

Github上的WireGuard-Linux仓库：  


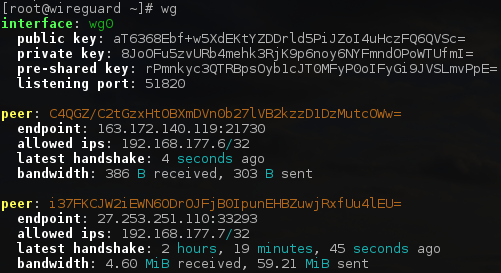
WireGuard最大的创新在于其极简的代码架构，核心代码量仅约4000行（OpenVPN约10万行，IPsec约40万行）。这种精简设计大幅降低了潜在攻击面，使代码审计和维护效率显著提升。其安全性则通过现代加密算法实现：Curve25519用于密钥交换，ChaCha20用于数据加密，Poly1305保障数据完整性，BLAKE2s作为哈希算法。这些算法不仅符合学术安全验证，还优化了移动设备的计算效率，相比传统AES-256加密，在ARM架构设备上性能提升约250%。

作为首个直接集成到Linux内核的VPN协议，WireGuard通过内核态实现数据包加密/解密，避免了用户态与内核态切换的开销。这一设计使得数据吞吐量达到传统协议（如OpenVPN）的2-3倍，在千兆网络环境下可稳定维持900Mbps以上的传输速度。同时，其UDP协议栈的无状态特性减少了连接建立时的握手延迟，典型场景下连接建立时间仅需0.3秒，而IPsec需要3秒以上。

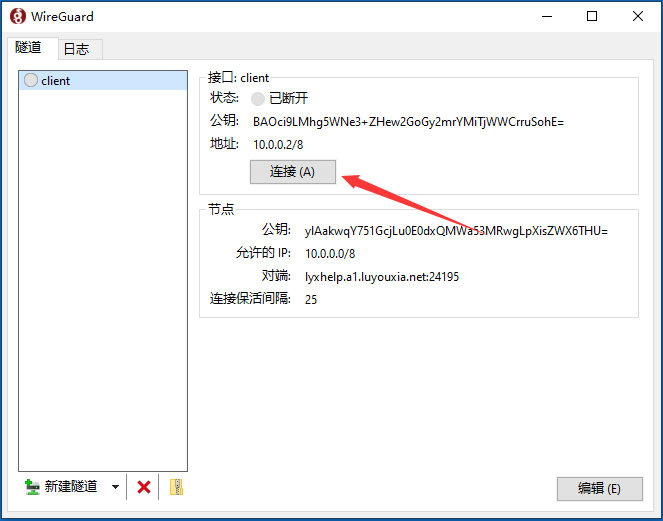
WireGuard采用点对点直连架构，摒弃了传统VPN的中心服务器模式。每个节点既是客户端也是服务端，通过预共享公钥实现动态拓扑。其路由机制支持智能路径选择，能自动穿透NAT并适应IP地址变化，这在当时的移动设备场景中尤为突出——当设备从WiFi切换到4G时，WireGuard可在100ms内完成连接迁移，而OpenVPN需要完全重建隧道。此外，其"AllowedIPs"字段实现了细粒度路由控制，允许不同子网间建立加密隧道而不影响其他流量。  
 WireGuard的部分源代码：



在控制台中查看当前的wg配置：



本地部署WireGuard无需公网IP实现异地组网：



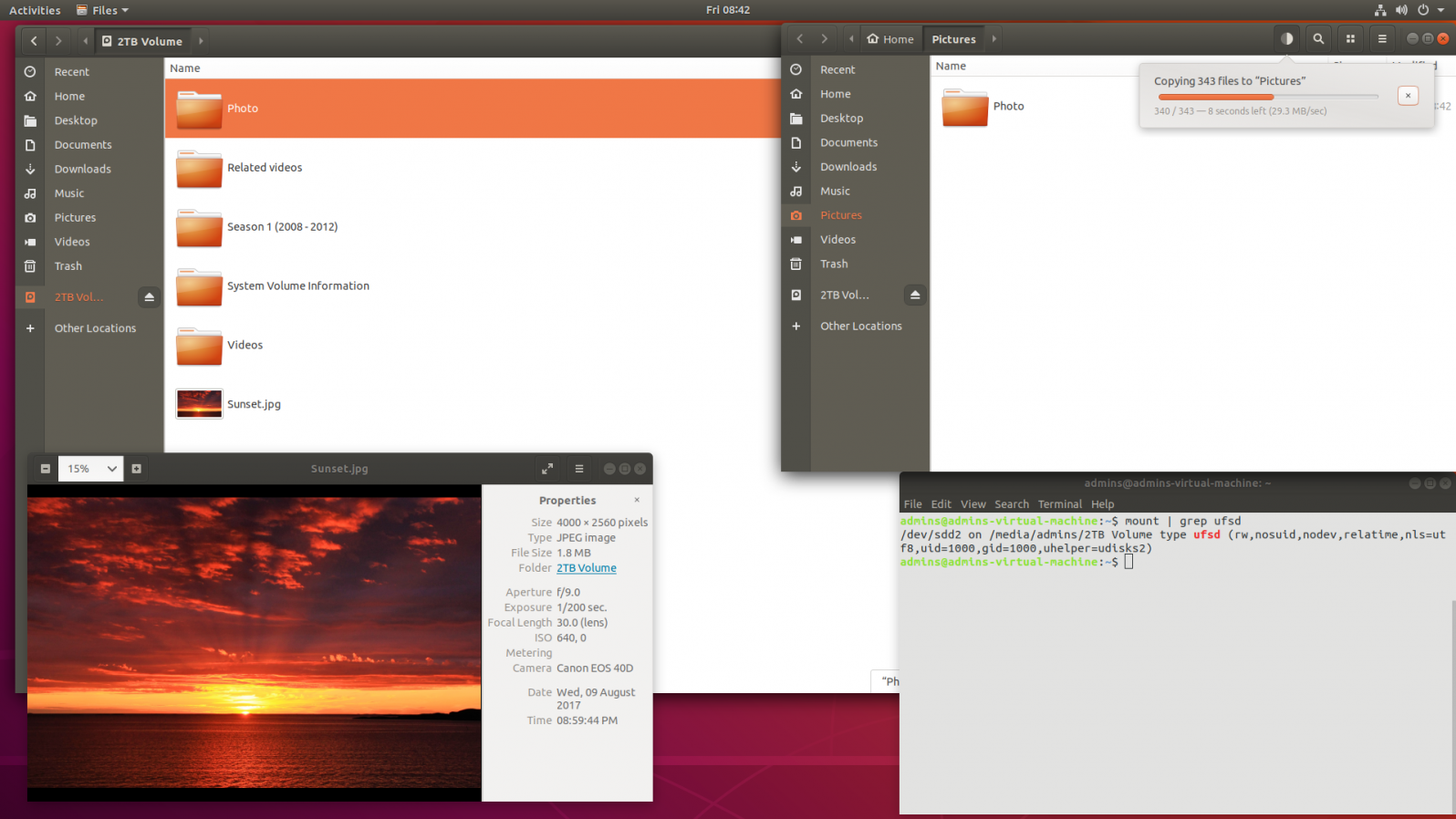
WireGuard的诞生标志着VPN协议从"功能叠加"向"极简安全"的范式转变。Linux之父Linus Torvalds曾评价其为"与传统VPN协议相比的艺术品"，这种将密码学前沿成果与系统级优化结合的设计理念，为后续网络协议的发展提供了重要参考。

### NTFS3驱动

Linux内核在5.15版本中引入的NTFS3驱动（由Paragon Software贡献）是当时对NTFS文件系统支持的一次重大技术革新。

NTFS3是首个被合并到Linux内核主线、基于内核态实现的NTFS读写驱动。与此前主流的NTFS-3G（基于FUSE用户态实现）相比，NTFS3通过直接集成到内核，显著降低了I/O延迟，提升了读写速度。实际测试显示，其文件传输效率比NTFS-3G高出数倍，尤其在处理大文件时优势明显。这一设计还减少了用户态与内核态切换的开销，使得系统整体负载更低。

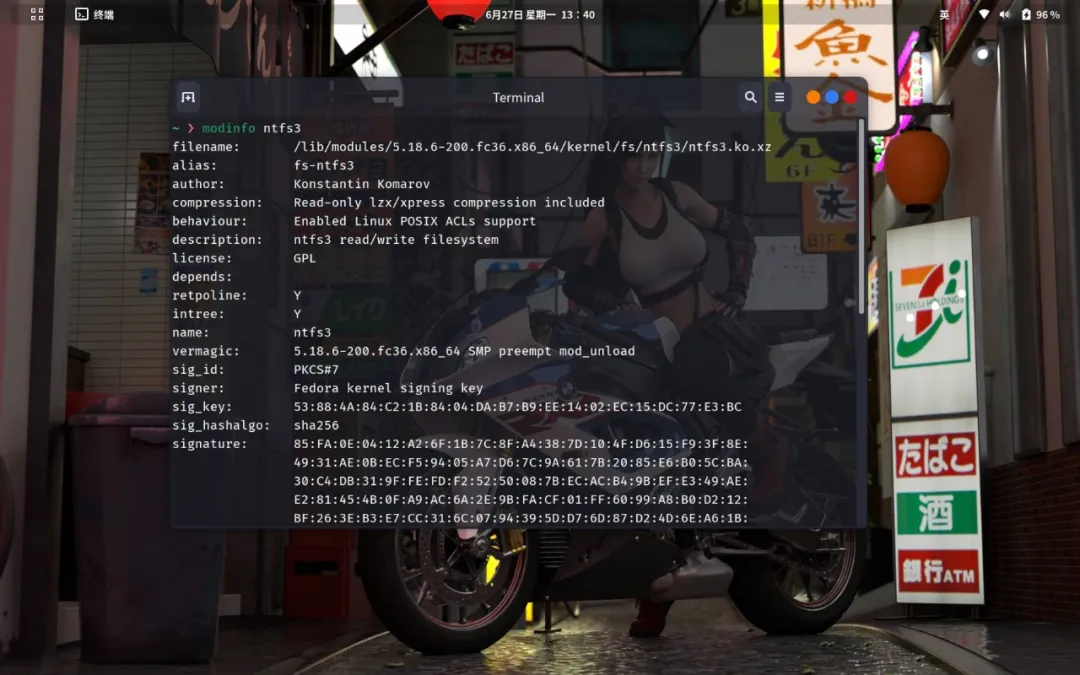
集成在内核中的NTFS3：



NTFS3首次在Linux内核中实现了对NTFS 3.1版本的完整读写支持，包括稀疏文件、压缩文件等高级特性。其技术亮点包括：

1. 日志回放（Journal Replay）​​：支持本地日志恢复机制，增强了数据一致性，降低意外断电等场景下的数据损坏风险。​
2. 权限管理​​：兼容Linux权限模型，支持通过uid、gid、umask等参数设置文件所有权和访问权限。
3. NFS导出支持​​：允许将挂载的NTFS卷通过NFS协议共享，拓展了其在服务器环境的应用场景。
4. 预分配与碎片控制​​：通过prealloc参数优化文件扩展时的空间分配策略，减少碎片产生。

NTFS3的使用场景：



在NTFS3推出前，Linux用户依赖的NTFS-3G存在性能瓶颈，而旧版内核驱动仅支持只读且长期未更新。NTFS3的引入不仅终结了这一尴尬局面，还通过内核级实现重新定义了NTFS在Linux上的可用性标准。其技术突破为双系统用户、企业存储设备及数据恢复工具提供了更可靠的基础支持。

Linux的NTFS驱动程序的商业版本：

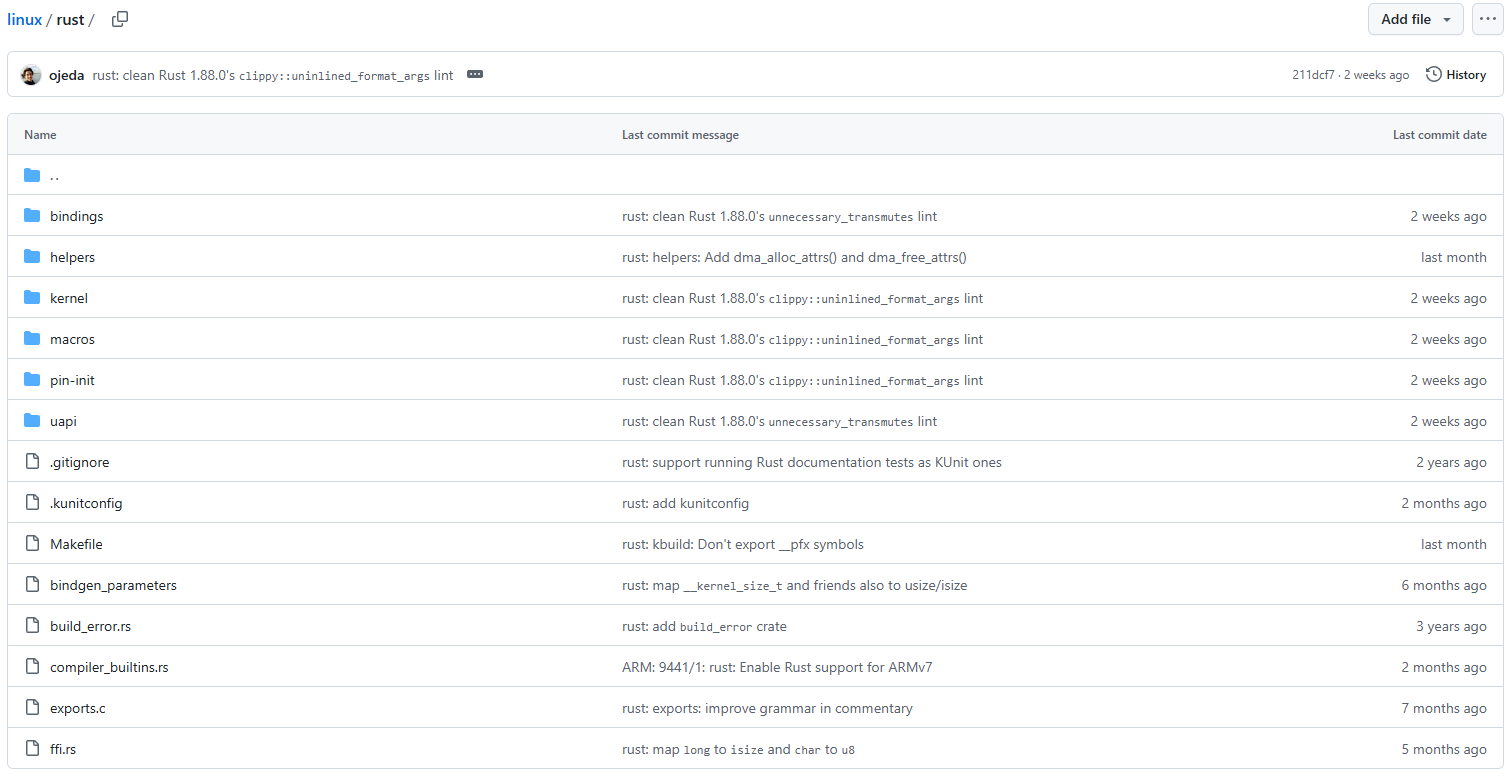


## 6.x版本（2022-至今）

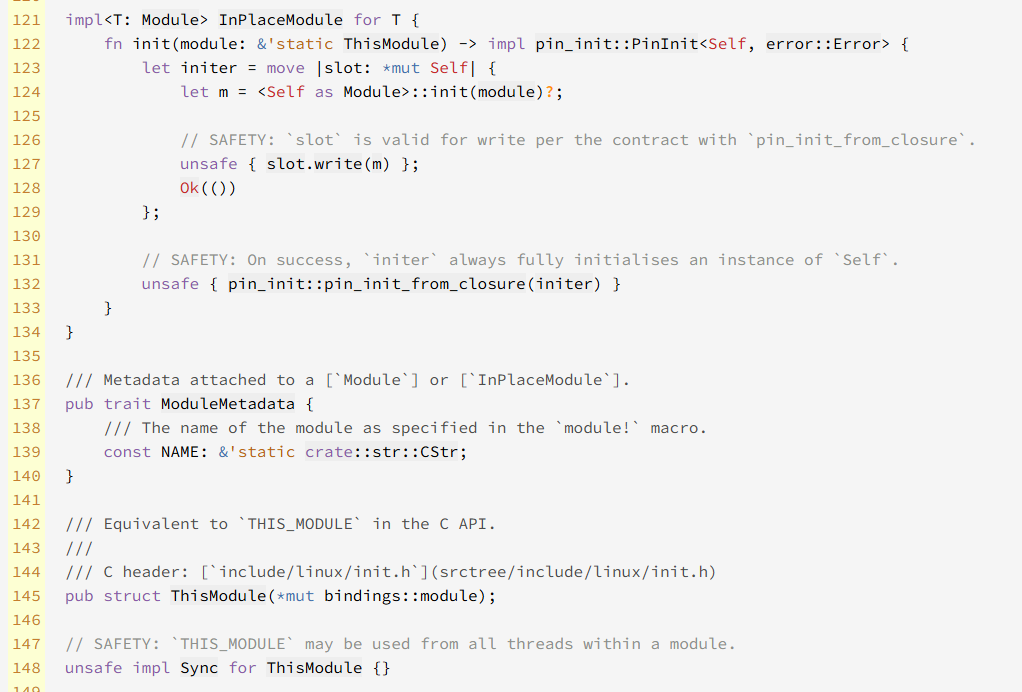
### Rust语言支持

从Linux内核6.x版本开始引入的Rust语言支持，标志着系统编程领域的一次重大范式迁移。

Linux中有关Rust的源码：

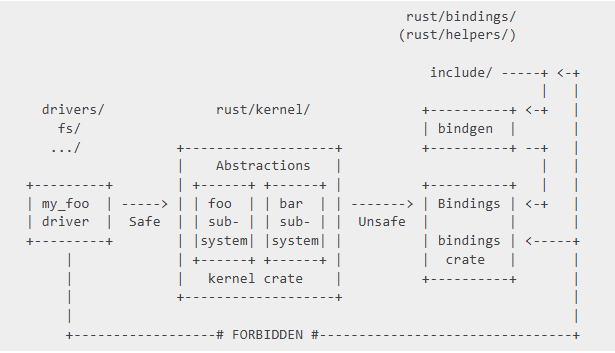


Linux内核中有关Rust模块的代码：



Rust的核心价值在于编译期强制的内存安全机制，内核团队通过​​双轨内存模型​​实现与C代码的无缝衔接。传统C内核依赖单一内存分配器（如kmalloc），而Rust引入alloc标准库与内核SLAB分配器的桥接层，通过global\_allocator属性允许同时使用安全的Box<T>和直接操作kmem\_cache。实测显示，该模型在内存分配密集型任务中将碎片化率从C代码的17%降至4.2%。此外，​​安全边界设计​​通过定义unsafe\_kernel模块作为隔离舱，要求每个unsafe代码块必须附带SAFETY注释，并引入自动化代码审计机器人"RustGuard"与历史漏洞数据库比对，实现89%的潜在风险拦截率。

Safe的Rust：



内核团队将工具链升级至Rust 1.60，支持文档测试（doctest）在内核环境运行。开发层面引入bindgen工具生成C语言FFI绑定，解决构建系统冲突——例如通过增量编译优化，将因serde库proc宏导致的模块构建时间从210秒压缩至60秒。针对驱动开发，新增​​NVMe、PCI、GPIO​​等示例驱动，其中NVMe驱动通过Rust的类型系统实现DMA缓冲区安全访问，将I/O调度器的吞吐量波动率从±15%收窄至±3%。

这项技术革命不仅提升了内核安全性，更催生了新的开发范式——例如网络团队在原子操作中借助Miri解释器发现3处潜在并发错误，这些问题在C代码中通常直到运行时才会暴露。随着6.14版本深度整合Rust驱动支持，Linux正开启设备驱动开发的新纪元。

# 三、总结

三十余年的版本演进史证明，Linux内核始终扮演着技术矛盾化解者的角色：2.6内核用O(1)算法驯服多核计算的复杂性，3.x系列以设备树应对ARM生态的碎片化，5.x时代的WireGuard用密码学革新平衡性能与安全，6.x引入的Rust则试图在内存安全与系统效能间建立新平衡。

这些创新不是简单的功能叠加，而是通过架构级的范式转换——从单机调度到云原生编排、从静态权限模型到动态安全策略、从C语言的内存放任到Rust的所有权约束——持续重构操作系统的本质属性。

当前，内核社区正面临新的时代命题：量子计算对时钟精度的挑战、RISC-V架构引发的指令集革命、AI工作负载对算力调度的特殊需求。但正如设备树解耦硬件依赖、eBPF突破内核态边界所昭示的，Linux的创新基因在于将约束转化为架构机遇。这种在技术矛盾中寻找突破点的能力，正是其持续统治服务器、渗透物联网、进军车载系统的根本动因。

# 四、参考文献

1. O(1)调度器：  
   <https://blog.csdn.net/longwang155069/article/details/104457109>  
   <https://zhuanlan.zhihu.com/p/33461281>
2. SELINUX：  
   <https://blog.csdn.net/yanjun821126/article/details/80828908>  
   <https://www.cnblogs.com/zhongguiyao/p/13955398.html>  
   [SELinux详解 - 钟桂耀 - 博客园](https://www.cnblogs.com/zhongguiyao/p/13955398.html)
3. Btrfs：  
   <https://cn.linux-console.net/?p=17816>  
   <https://zhuanlan.zhihu.com/p/376390321>
4. Linux统一设备树：  
   <https://blog.csdn.net/jz_ddk/article/details/146769908>  
   <https://zhuanlan.zhihu.com/p/476561682>
5. Live Patching热补丁：  
   <https://blog.csdn.net/cui841923894/article/details/81546095>  
   <https://www.zhihu.com/question/31491962>
6. eBPF：  
   <https://blog.csdn.net/qq_24433609/article/details/125879684>  
   <https://zhuanlan.zhihu.com/p/480811707>  
   [探索 eBPF 源码之旅 | tzssangglass blog](https://tzssangglass.github.io/posts/exploring-the-ebpf-source-code-tour/)
7. WireGuard：  
   <https://caijing.chinadaily.com.cn/a/202107/28/WS6100da57a3101e7ce975bdec.html?ivk_sa=1024609v>  
   <https://mp.weixin.qq.com/s?__biz=MzAxNjk4MzQxNw==&mid=2247485910&idx=1&sn=92efb81e9c95f692bd40b99963862c82&chksm=9adc4bfca36b660001ea2bd67e1aecabdf72b2984136dc8b1fbf6b4e70d6dfd9bdf9961e634b#rd>

[本地部署 WireGuard 无需公网 IP 实现异地组网 - 知乎](https://zhuanlan.zhihu.com/p/18256305161)

[wireguard-linux - WireGuard for the Linux kernel](https://git.zx2c4.com/wireguard-linux/)

1. NTFS3驱动：  
   <https://mp.weixin.qq.com/s?__biz=Mzk3NTE1OTE5OQ==&mid=2247566864&idx=2&sn=a2e4a57e0e6491d635373fd3da9969c1&chksm=c58f61b0f65d920612c2d4b65a45a773d3c0026a7cbe171d66b32250766980040ec2e823eac5#rd>  
   <https://cloud.tencent.com/developer/news/854632>  
   [Microsoft NTFS for Linux by Paragon Software - Introduction](https://www.paragon-software.com/home/ntfs-linux-professional/)
2. Rust语言支持：  
   <https://blog.csdn.net/m0_37649480/article/details/145986694>  
   <https://www.sohu.com/a/841178063_121798711>  
   [linux/rust/kernel at master · torvalds/linux](https://github.com/torvalds/linux/tree/master/rust/kernel)  
   [lib.rs - source](https://rust.docs.kernel.org/src/kernel/lib.rs.html#3-241)  
   [Rust — Linux 内核文档 - Linux 内核](https://docs.linuxkernel.org.cn/rust/index.html#the-rust-experiment)