Architektura jádra Linux

Jádro Linuxu (kernel) je monolitické jádro s modulární strukturou, které představuje srdce operačního systému. Poskytuje hardwarovou abstrakci, správu systémových prostředků a základní služby pro běžící programy.

Základní charakteristika

Linux kernel je monolitický, což znamená, že všechny základní služby (správa paměti, souborové systémy, ovladače) běží v privilegovaném režimu v jediném adresovém prostoru jádra. Na rozdíl od mikrojader (jako Minix nebo QNX) není funkcionalita rozdělena do samostatných serverových procesů. Přesto Linux podporuje dynamické načítání modulů, což mu dává flexibilitu podobnou mikrojádrům.

Vrstvy architektury

1. Hardwarová vrstva

Nejnižší úroveň představuje fyzický hardware - procesor, paměť RAM, disky, síťové karty, periférie. S touto vrstvou komunikuje jádro prostřednictvím specifických instrukcí a portů.

2. Rozhraní System Call Interface (SCI)

Toto rozhraní poskytuje systémová volání (syscalls), která umožňují uživatelským programům požádat jádro o služby. Systémová volání jsou jediným legálním způsobem, jak může uživatelský proces přistupovat k privilegovaným operacím. Příklady systémových volání:

`open()`, `read()`, `write()`, `close()` - práce se soubory `fork()`, `exec()`, `exit()` - správa procesů `socket()`, `connect()`, `send()` - síťová komunikace `mmap()`, `brk()` - správa paměti

Systémové volání probíhá přepnutím z user mode (neprivilegovaný) do kernel mode (privilegovaný) pomocí speciální instrukce procesoru (např. `int 0x80` na x86 nebo `syscall` na x86-64).

3. Vrstva správy procesů

Process Scheduler (plánovač procesů)

Linux používá Completely Fair Scheduler (CFS) zavedený v jádře 2.6.23. CFS se snaží spravedlivě rozdělit procesorový čas mezi všechny procesy podle jejich priority a „nice“ hodnoty. Klíčové koncepty:

Virtual runtime (vruntime) - sleduje, kolik procesorového času proces už obdržel Red-black tree - procesy jsou organizovány v červeno-černém stromu podle vruntime, proces s nejnižším vruntime běží jako první Time slices - CFS nedělí čas na pevné časové kvanta, ale dynamicky přiděluje čas

podle zatížení Prioritní třídy - real-time (SCHED_FIFO, SCHED_RR), normální (SCHED_NORMAL), dávkové (SCHED_BATCH), idle (SCHED_IDLE)

Process Management

Každý proces v Linuxu je reprezentován strukturou `task_struct`, která obsahuje:

PID (Process ID) a PPID (Parent Process ID) Stav procesu (running, sleeping, stopped, zombie) Registry procesoru a ukazatel zásobníku Informace o paměťovém prostoru (mm_struct) Seznam otevřených souborů Práva a oprávnění (UID, GID) Signály a jejich obsluhy

Linux používá copy-on-write mechanismus při `fork()` - nový proces sdílí paměťové stránky s rodičem, dokud některý z nich nezačne do paměti zapisovat.

4. Vrstva správy paměti (Memory Management)

Virtuální paměť

Linux implementuje stránkování (paging) - virtuální adresový prostor je rozdělen na stránky (typicky 4 KB na x86). Každý proces má vlastní virtuální adresový prostor (typicky 0x00000000 až 0xFFFFFFFF na 32-bit systémech), který je mapován na fyzickou RAM. Page tables (tabulky stránek) překládají virtuální adresy na fyzické. Linux používá víceúrovňový systém:

PGD (Page Global Directory) - nejvyšší úroveň PUD (Page Upper Directory) PMD (Page Middle Directory) PTE (Page Table Entry) - nejnižší úroveň, obsahuje fyzickou adresu

Zóny paměti

Fyzická paměť je rozdělena do zón:

ZONE_DMA - paměť pro přímý přístup starších zařízení (0-16 MB na x86) ZONE_NORMAL - běžně použitelná paměť (16 MB - 896 MB na 32-bit x86) ZONE_HIGHMEM - vysoká paměť (nad 896 MB na 32-bit systémech) ZONE_MOVABLE - paměť pro přemístitelné stránky

Alokátory paměti

Buddy allocator - alokuje bloky fyzické paměti v mocninách 2 (4K, 8K, 16K...), efektivně předchází fragmentaci Slab allocator (SLUB v novějších jádrech) - cachuje často používané objekty (např. task_struct, inode), minimalizuje overhead alokace Vmalloc - alokuje virtuálně souvislou paměť (fyzicky může být fragmentovaná) Kmalloc - rychlá alokace malých bloků, fyzicky i virtuálně souvislá paměť

Page cache a buffer cache

Page cache - cachuje obsah souborů v paměti, zrychluje čtení a zápis Buffer cache - cachuje bloková data z diskových zařízení Linux používá jednotný page cache od verze 2.4

Swapping a paging

Když dojde fyzická paměť, kswapd daemon začne přesouvat nejméně používané stránky na disk (swap partition/file). Algoritmus LRU (Least Recently Used) s modifikacemi určuje, které stránky vyměnit. OOM Killer (Out of Memory Killer) zabíjí procesy, když dojde paměť a swapping nestačí.

5. Vrstva správy souborových systémů

Virtual File System (VFS)

VFS je abstraktní vrstva, která poskytuje jednotné rozhraní pro různé souborové systémy (ext4, XFS, Btrfs, NFS, procfs, sysfs…). Uživatelské programy používají standardní funkce (open, read, write) bez znalosti konkrétního souborového systému. Klíčové struktury VFS:

superblock - reprezentuje připojený souborový systém inode - reprezentuje soubor nebo adresář (metadata: oprávnění, velikost, časové razítka) dentry (directory entry) - reprezentuje položku v adresáři, spojuje název souboru s inodem file - reprezentuje otevřený soubor (obsahuje ukazatel na aktuální pozici)

Dentry cache (dcache)

Cachuje nedávno používané dentry objekty v paměti, výrazně zrychluje operace s cestami (path lookup).

Inode cache (icache)

Cachuje inode struktury, aby se nemusel při každém přístupu číst disk.

Konkrétní souborové systémy

ext4 - výchozí FS na mnoha Linuxových distribucích, journaling, extenty, velké soubory (až 16 TB) XFS - vysoce výkonný, škáluje na petabyty, dobré pro velké soubory Btrfs - moderní copy-on-write FS, snapshoty, RAID, komprese F2FS - optimalizováno pro flash paměti NFS, CIFS/SMB - síťové souborové systémy procfs, sysfs - virtuální souborové systémy pro informace o jádře

6. Block I/O vrstva

Tato vrstva zprostředkovává komunikaci mezi souborovými systémy a blokovými zařízeními (disky).

Block layer

Bio (block I/O) - základní struktura reprezentující I/O operaci Request queue - fronta čekajících I/O operací I/O Scheduler - optimalizuje pořadí operací pro lepší výkon (CFQ, Deadline, NOOP, BFQ, mq-deadline)

Schedulery se snaží:

Minimalizovat pohyb hlavičky disku (elevator algoritmy) Předcházet starvation (některé požadavky čekají příliš dlouho) Rozlišit mezi sekvenčním a náhodným přístupem

Generic Block Layer

Poskytuje jednotné rozhraní pro všechna bloková zařízení, podporuje:

Merging - spojování sousedících požadavků Plugging - odložení odeslání požadavků za účelem optimalizace Device mapper - virtualizace blokových zařízení (LVM, RAID, šifrování)

7. Vrstva zařízení a ovladačů (Device Drivers)

Typy zařízení

Character devices - proud znaků bez náhodného přístupu (terminály, sériové porty) Block devices - náhodný přístup k datovým blokům (disky, SSD) Network devices - síťová zařízení, nemají speciální soubory v /dev

Ovladače

Ovladače mohou být:

Zabudované do jádra (built-in) Moduly - dynamicky načítané soubory `.ko` (kernel object)

Moduly umožňují:

Načítání ovladačů za běhu (`insmod`, `modprobe`) Odebrání nepoužívaných ovladačů (`rmmod`) Menší velikost jádra Aktualizace ovladačů bez restartu

Subsystémy

USB subsystem - správa USB zařízení, USB core a USB host controller drivers PCI subsystem - správa PCI a PCIe zařízení Input subsystem - klávesnice, myši, touchpady, joysticky Graphics subsystem - DRM (Direct Rendering Manager), framebuffer

8. Síťová vrstva (Network Stack)

Linux implementuje kompletní TCP/IP stack s podporou mnoha protokolů.

Vrstvový model

Network Device Drivers - nejnižší vrstva, komunikace s hardware (Ethernet, Wi-Fi) Link layer - LLC, Ethernet framing Network layer - IP (IPv4, IPv6), ICMP, routing Transport layer - TCP, UDP, SCTP Application layer - socket API pro uživatelské aplikace

Socket API

Procesy komunikují přes síť pomocí socketů:

SOCK_STREAM - TCP sockety (spojované, spolehlivé) SOCK_DGRAM - UDP sockety (nespojované, nespolehlivé) SOCK_RAW - přímý přístup k IP vrstvě

Netfilter framework

Umožňuje filtrování a manipulaci síťových paketů:

iptables/nftables - konfigurace firewallu Connection tracking - sledování stavu spojení NAT - překlad adres (SNAT, DNAT, masquerading)

Networking features

Traffic Control (tc) - QoS, bandwidth shaping Network namespaces - izolace síťových stacků (používá Docker) VLAN, bridging, bonding - virtuální sítě Wireless subsystem (mac80211, cfg80211)

9. Meziprocessová komunikace (IPC)

Linux poskytuje několik mechanismů IPC:

Pipes a named pipes (FIFO) - jednosměrná komunikace mezi procesy Signals - asynchronní notifikace (SIGKILL, SIGTERM, SIGUSR1…) Message queues - fronty zpráv Shared memory - sdílené paměťové oblasti mezi procesy, nejrychlejší metoda Semaphores - synchronizace přístupu ke sdíleným zdrojům Sockets - síťová i lokální komunikace (Unix domain sockets)

10. Bezpečnost a přístupová práva

Tradiční Unix přístupová práva

Každý soubor má:

Owner (vlastník) a Group (skupina) Permissions - read ®, write (w), execute (x) pro owner, group, others Special bits - SUID, SGID, sticky bit

Linux Security Modules (LSM)

Framework umožňující implementaci pokročilých bezpečnostních modelů:

SELinux (Security-Enhanced Linux) - mandatory access control, používá Red Hat/Fedora AppArmor - profily omezující přístup aplikací, používá Ubuntu/SUSE Smack - zjednodušený MAC systém TOMOYO - pathname-based MAC

Capabilities

Jemné dělení root oprávnění - místo „vše nebo nic“ může proces mít jen některá privilegia:

CAP_NET_ADMIN - správa sítě CAP_SYS_TIME - změna systémového času CAP_KILL - posílání signálů jiným procesům CAP_CHOWN - změna vlastnictví souborů

Namespaces

Izolace různých aspektů systému (základ kontejnerů):

PID namespace - izolace process IDs Network namespace - vlastní síťový stack Mount namespace - vlastní souborový strom UTS namespace - hostname a domain name IPC namespace - izolace IPC objektů User namespace - mapování UID/GID

Cgroups (Control Groups)

Omezení a měření využití zdrojů skupinou procesů:

CPU omezení Paměťové limity I/O bandwidth limity Freezer - pozastavení skupiny procesů

11. Interrupt handling

Hardware interrupty

Když hardware potřebuje pozornost CPU (disk dokončil operaci, přišel síťový paket), vygeneruje interrupt. CPU okamžitě přeruší běžící kód a skočí na interrupt handler (ISR - Interrupt Service Routine). Proces:

Hardware vyvolá IRQ (Interrupt Request) CPU uloží kontext a skočí na příslušný handler Handler provede top half - rychlé kritické zpracování Bottom half (softirq, tasklet, workqueue) - odložené

nekritické zpracování CPU obnoví kontext a pokračuje

Softirqs a tasklets

Softirq - „měkké“ přerušení, zpracování odložené práce s nižší prioritou než hardware interrupt Tasklet - jednodušší forma softirq pro ovladače Workqueue - práce provedená v context procesu (může spát)

12. Synchronizační primitiva

Jádro musí chránit sdílená data před souběžným přístupem:

Spinlocks - aktivní čekání, drží CPU, používá se pro krátké kritické sekce Semaphores - proces může uspat, používá se pro delší kritické sekce Mutexes - binární semafor, jednodušší API RCU (Read-Copy-Update) - lock-free mechanismus pro čtení dat, writer vytvoří kopii Atomic operations - atomické operace garantované hardware (test-and-set, compare-and-swap) Memory barriers - zajištění pořadí paměťových operací v SMP systémech

13. Architektury CPU

Linux kernel podporuje mnoho architektur:

x86/x86-64 - Intel a AMD procesory ARM/ARM64 - mobilní zařízení, embedded systémy, servery (Apple M1, AWS Graviton) PowerPC - IBM servery RISC-V - otevřená architektura MIPS, SPARC, s390 - méně běžné

Arch-specific kód je v adresáři `arch/` ve zdrojácích jádra.

Zdroje jádra

Celé jádro je napsáno primárně v C (asi 98 %) s částmi v assembleru pro kritické low-level operace. Zdrojový kód obsahuje přes 30 milionů řádků a je organizován do:

`kernel/` - core funkce (scheduler, signals, timers) `mm/` - správa paměti `fs/` - souborové systémy a VFS `net/` - síťový stack `drivers/` - ovladače zařízení (největší část) `arch/` - architektura-specifický kód `include/` - hlavičkové soubory

Uživatelský prostor vs. Kernel space

Linux jasně odděluje:

User space - neprivilegovaný režim, běžné aplikace, nemohou přímo přistupovat k hardware Kernel space - privilegovaný režim, plný přístup k hardware a paměti

Tento model protection rings (na x86 Ring 3 pro user, Ring 0 pro kernel) zajišťuje stabilitu - chybný uživatelský program nemůže shodit celý systém.

Závěr

Linux kernel je komplexní, vysoce optimalizovaný kus software, který kombinuje monolitickou architekturu s modulárním designem. Jeho úspěch spočívá v open-source modelu vývoje, excelentní podpoře hardware, škálovatelnosti od embedded zařízení po superpočítače a důrazu na výkon a stabilitu. Díky aktivní komunitě tisíců vývojářů po celém světě se jádro neustále vyvíjí a přizpůsobuje novým technologiím.