Linux

Obsah

1 Úvod

2 Konfigurace jádra

Než je možné jádro Linxu přeložit je třeba provést konfiguraci, která umožní přizpůsobit chování systému pro daný hardware, zvýšit výkon změnšit velikost výsledného binárního souboru (zakázáním nepotřebných modulů).

Konfiguraci jádra lze rozdělit do několika bodů:

1. **Architektura a vlastnosti procesoru (Processor type and features)** - absolutní základ, který je třeba nastavit pro správné fungovanání na dané platformě:
   - **Typ procesoru** - například intel Core, AMD ryzen, ARM, ...
   - **Podpora vícejádrových procesorů** - pokud má daný procesor více jader, umožní plánovači systému efektivně rozdělovat jádra běžícím procesům
   - **Bezpečnostní funkce procesoru** - tyto volby zvyšují bezpečnost výsledného jádra, jedná se o volby jako randomizace RAM adres nebo ochrana zásobníku proti přetečení
2. **Obecné nastavení jádra** -

2.1 Obecné nastavení (General configuration)

2.1.1 Komprimace jádra (Kernel compression mode)

Kernel compression mode je nastavení v konfiguraci Linuxového jádra, které určuje, jakým kompresním algoritmem bude výsledný soubor jádra (tzv. kernel image, obvykle vmlinuz nebo bzImage) po kompilaci komprimován. To že je jádro na disku v komprimované podobě umožňuje úsporu paměti disku, rychlejší bootkování, menší využití RAM nebo rychlejší síťové bootování. Když vyberete kompresní algoritmus a zkompilujete jádro, kód dekompresoru pro vybraný algoritmus je zabudován přímo do počáteční části jádra. Když bootloader (např. GRUB) načte komprimované jádro do paměti a předá mu řízení, první věcí, kterou jádro udělá, je dekomprese sebe sama do paměti. Teprve potom začne spouštět zbytek systému.

Jádro Linuxu podporuje několik kompresních algoritmů, které se liší kompresním poměrem a rychlostí dekomprese. Základní volby jsou

- **GZip** - dobrý kompresní poměr, ale pomalejší dekomprese
- **BZip2** - lepší kompresní poměr nez GZip ale výrazně pomalejší
- **LZMA** - velice dobrý kompresní poměr ale zároveň velice pomalá dekomprese, vhodný pro embedded systémy s omezenou pamětí
- **XZ** - podobné vlastnosti jako LZMA
- **LZO** - nižší kompresní poměr, ale velice rychlá dekomprese, vhodné pro systémy s potřebou rychlého bootování, kde velikost jádra je méně důležitá
- **LZ4** - ještě nižší kompresní poměr ale ještě rychlejší dekomprese než LZO, vhodné pro desktopové systémy

2.1.2 Preemtion Model

Model preemptibility jádra (anglicky Kernel Preemption Model nebo též Preemption Model) je klíčové nastavení Linuxového jádra, které ovlivňuje jeho odezvu (responsivitu) a propustnost (throughput). Určuje, kdy může jádro přerušit (preemptovat) běžící kód jiného procesu nebo samotného jádra, aby se mohla spustit úloha s vyšší prioritou.

V kontextu operačních systémů znamená preempce možnost operačního systému odejmout procesoru kontrolu nad běžící úlohou (procesem nebo kódem jádra) a přidělit ji jiné úloze, obvykle té s vyšší prioritou, nebo jednoduše té, která čeká na vykonání.

V jádře Linuxu se preempce týká kódu běžícího v jádrovém režimu (kernel mode). Tradičně, když se kód jádra spustí, běží, dokud sám neskončí, nebo dokud nenarazí na

blokující operaci (např. čekání na I/O). Během této doby nemůže být přerušen jinou úlohou, dokonce ani tou s vyšší prioritou. To zajišťuje jednoduchost a minimalizuje riziko složitých stavů, ale na úkor odezvy.

S preempčním jádrem se situace mění. Jádro může být nakonfigurováno tak, aby se chovalo jako "klient" pro scheduler, což umožňuje přerušení (preemptování) kódu jádra jiným, naléhavějším úkolem.

Volba modelu preemptibility je kompromis mezi:

- **Odezvou (Responsiveness)**: Jak rychle systém reaguje na externí události (např. stisk klávesy, síťový paket, multimediální data). Dobrá odezva je klíčová pro desktopové systémy, audio/video aplikace a real-time systémy.
- **Propustností (Throughput)**: Celkové množství práce, které systém zvládne za daný čas. Vyšší propustnost je často prioritou pro servery, databázové systémy nebo HPC (High Performance Computing), kde je cílem dokončit co nejvíce úloh, i za cenu mírně delších latencí.

Obecně platí **Více preempce = lepší odezva**, ale potenciálně mírně nižší propustnost kvůli vyšší režii (overhead) z neustálého přepínání kontextu. **Méně preempce = vyšší** propustnost, ale potenciálně horší odezva.

Dostupné nastavení v jádru:

- **No Forced Preemption (Server)** - nejméně preemtivní model, kód jádra běží nepřerušovaně, dokud sám nedokončí, nebo dokud nenarazí na definovaný bod přerušení (čekání na IO, explicitní volání funkce shedule()). Ideální pro servery, databáze, HPC, kde je priorita na maximální propustnost a stabilní, předvídatelný běh s minimálním rušením.
- **Voluntary Preemption (Desktop)** - kompromisní model. Kód jádra má do sebe vloženy dodatečné "body přerušení". Jádro si kontroluje, zda je proces s vyšší prioritou připraven ke spuštění, a pokud ano, dobrovolně se vzdá procesoru. Dříve běžná volba pro desktopové systémy, než se rozšířila plná preempce.
- **Preemptible Kernel (Low-Latency Desktop)** - toto je nejvíce preemptivní model pro běžné použití. Většina kódu jádra je plně preemptibilní, což znamená, že může být přerušena kdykoli jiným úkolem s vyšší prioritou (nebo jen jiným úkolem, který potřebuje CPU). Jádro se chová mnohem více jako normální uživatelské procesy. Standardní a doporučená volba pro desktopové systémy, multimediální stanice a interaktivní pracovní stanice.
- **Sheduler controlled preemtion model** - Místo, aby se rozhodovalo o preemptibilitě celého jádra při kompilaci staticky, tato volba přesouvá část rozhodování o preempci na běhový plánovač (scheduler). Jádro může dynamicky rozhodovat o tom, jak agresivně bude preemptovat, a to na základě zatížení systému, typu běžících úloh a dokonce i na základě konfiguračních parametrů, které lze nastavit za běhu.

3 Překlad jádra

3.1 Instalace jádra

Po dokončení kompilace je v adresáři se zdrojovými kódy jádra velké množství nově vytvořených souborů. Většina z nich jsou ale pouze dočasné soubory z procesu překladu, jako jsou objektové soubory, logy a podobně. Instalace je proces, vezme všechny výsledné přeložené soubory jako je soubor jádra a jeho moduly a přesune je do instalačního adresáře, kde je připraví k použití a pro další fázi nastavení nového systému.

Hlavním úkolem instalace je:

- **Instalace modulů jádra**: Umístění ovladačů a dalších částí jádra (které byly v menuconfig označeny jako [M]) na správné místo v adresářové struktuře systému.
- **Instalace samotného jádra**: Umístění hlavního spustitelného souboru jádra (vmlinuz nebo bzImage) do bootovacího oddílu.
- **Instalace pomocných souborů**: Zkopírování souborů jako System.map a config, které jsou užitečné pro ladění a referenci.
- **Aktualizace bootloaderu**: Informování bootloaderu (jako je GRUB), že existuje nové jádro, které může spustit.

3.2 Postup instalace

Proces instalace se odehrává stále v kořenovém adresáři, kde jsou zdrojové kódy zkompilovaného jádra.

3.2.1 Instalace modulů jádra

Pro instalaci modulů jádra se používá příkaz:

```
$ sudo make modules_install [INSTALL_MOD_PATH=<cesta>]
```

Tento příkaz zkopíruje všechny zkompilované moduly (.ko soubory), do správného adresáře v systému nebo na adresu definovanou v proměnnou **INSTALL_MOD_PATH**. Standardně se instalují do /lib/modules/¡verze_jádra¿/.

4 Initramfs

Initramfs (zkratka pro initial RAM filesystem) je malý, dočasný souborový systém, který jádro Linuxu načítá do operační paměti (RAM) během rané fáze bootování. Jeho hlavním účelem je připravit prostředí, ve kterém jádro dokáže najít a připojit skutečný kořenový souborový systém (root filesystem), na kterém je nainstalovaný celý operační systém.

Hlavní důvod existence initramfs spočívá v modularitě Linuxového jádra a rozmanitosti hardwaru:

1. **Chybějící ovladače v jádře** - Jádro Linuxu může být zkompilováno s mnoha ovladači jako moduly (soubory .ko), nikoli zabudované přímo v jádře. To šetří paměť a disk, protože se načítají jen potřebné ovladače. Problém nastává, když jádro potřebuje ovladač pro přístup k disku, na kterém je kořenový souborový systém (např. ovladač pro SATA řadič, NVMe disk, RAID pole, LVM svazek nebo šifrovaný disk). Pokud tento kritický ovladač není zabudován přímo v jádře, jádro ho nemůže načíst, protože k němu nemá přístup (nemá disk připojený!). Initramfs tento problém řeší protože obsahuje tyto kritické ovladače jako součást svého malého souborového systému.
2. **Složitá Úložiště** - Kořenový souborový systém nemusí být vždy na jednoduchém, přímo přístupném oddílu. Může být na šifrovaném disku (LUKS), na LVM (Logical Volume Management) svazku, RAID poli nebo dokonce po síti (NFS). Initramfs obsahuje nástroje (jako cryptsetup, lvm, síťové nástroje), které jádru umožní tyto složité konfigurace dešifrovat, aktivovat nebo připojit.
3. **Flexibilita při Bootování** - Initramfs umožňuje spouštět bootovací skripty (často init nebo linuxrc uvnitř initramfs), které provádějí předběžné operace, jako je dotazování se uživatele na heslo pro dešifrování disku, kontrola integrity souborového systému nebo příprava síťového připojení pro NFS root.

4.1 Fungování initramfs

1. BIOS/UEFI se spustí a inicializuje základní hardware.
2. Bootloader (např. GRUB) se spustí.
3. GRUB načte do paměti soubor jádra (vmlinuz) a soubor initramfs (initramfs-¡verze_jádra¿.img)
4. Jádro se spustí, to znamená že jádro dekomprimuje samo sebe a přenese obsah initramfs do RAM a připojí ho jako svůj dočasný kořenový souborový systém.
5. Initramfs se spustí. Tím jádro spustí první proces uvnitř sebe (obvykle /init nebo /linuxrc). Tento skript (běžící v initramfs) načte potřebné moduly jádra (pro disk, šifrování, LVM atd.). Provede se dešifrování disku (pokud je třeba), Aktivují se LVM svazky nebo RAID pole. Kořenový souborový systém je v této fázi nalezen a připraven.
6. "Hand-off" (předání kontroly). Jakmile je skutečný kořenový souborový systém připraven, proces uvnitř initramfs přepne kořenový souborový systém (pivot_root) z initramfs na ten skutečný disk. Systém pak spustí finální proces init (nebo systemd) z hlavního souborového systému, a zbytek operačního systému se načte normálně.
7. Initramfs je poté uvolněn z paměti a již se nepoužívá.

5 Adresářová struktura