V tomto článku si ukážeme čo to sú ramdisky, jednotlivé druhy a aj to ako fungujú, spôsob ich využitia a ako to súvisí s tvorbou distribúcie alebo úpravou distribúcie. Pripútajte sa, výlet do krajiny zázrakov začína

Nadviažem týmto teraz na predchádzajúci článok kde som splietal také veci o vygenerovaní nového initrd ak ste nejakým spôsobom zmenili kernel distribúcie (opäť predpokladám pokročilú znalosť Linuxu). Prečo by nás to ale malo zaujímať a čo to vlastne je?

Ako iste viete, máme rôzne druhy blokových zariadení a typov súborových systémov (FS). Ramdisk je vlastne tiež takéto úložné zariadenie. Alokuje časť pamäti RAM a tvári sa to ako normálne úložne zariadenie (pevný disk, flash, CD....) takže ho môžete mu spraviť mount a zapisovať/čítať z neho. Všetko je (prakticky povedané) rovnaké ako keby sa údaje zapisovali napríklad na pevný disk s tým rozdielom že sa to deje len na úrovni RAM. Skrátene povedané používate RAM ako disk pre ukladanie súborov a ich čítanie. Toto má obrovské výhody. Ak si porovnáme prístupovú dobu a rýchlosť zápisu/čítania disku oproti RAM tak tam je niekoľkonásobný rozdiel a disk sa v rýchlosti s RAM ani veľmi rovnať nemôže, je dosť ďaleko vzadu.

Takýchto typov ramdiskov máme niekoľko druhov. V dávnejších linuxových časoch sa systémovému zavádzaču dával parameter „root=“, špecifikujúci root partíciu odkiaľ sa má zaviesť systém, to bolo v pohode keďže vtedy sa využívali prakticky len diskety a pevné disky a nie nejaké krolomné zavádzacie voodoo techniky. V procese zavedenia systému proste len kernel spustil proces init s PID 1, ktorý sa už staral o zvyšok na userspace úrovni. Technológia ale napreduje a objavili sa také veci ako zdieľané disky, kryptované FS atď... a tu nastal problém so zavedením systému. Predstavme si že náš root FS sa nachádza na sieťovom disku, ktorý vzdialene mountujeme. Jadro systému by sa muselo napojiť na server, mountnuť vzdialený disk a odtiaľ spustiť init, to ale je pre kernel problém a jednak takéto prasačinky by fakt nemalo riešiť práve jadro ale nejaký program a práve hlavne preto vznikli ramdisky, ktoré priniesol Linux 2.6 .

Ako prvý bol na svete „ramdisk“. Bol to súbor používajúci sa ako fake blokové zariadenie,

pričom mal pevne danú veľkosť a bol naformátovaný na špecifický FS (napr. ext2). Svojím spôsobom sa to dá prirovnať k loopback zariadeniu. Tento súbor bol na normálnom lokálnom disku, a kernel ho najprv načítal do pamäte a mountol ako root FS. Tento ramdisk musel obsahovať súbor linuxrc, ktorý následne kernel spustil. Úlohou tohto súboru bolo vykonať, nazvime to „pre-boot“ úlohy ako napríklad kontaktovanie servera a pripojenie vzdialeného disku, alebo napríklad rozšifrovanie lokálneho kryptovaného disku alebo rôzne iné úlohy, ktoré je potrebné vykonať ešte predtým než je pripojený systémový disk. Samozrejme, ramdisk mohol obsahovať aj ľubovoľné iné programy potrebné k úlohám, ktoré sa spúšťali zvnútra linuxrc. Po tom čo spravil čo trebalo zavolal už z normálneho FS proces init a boot systému pokračoval tak ako ho klasicky poznáme.

Tu nastávali ale časom problémy. Spustený linuxrc mal náhodné PID (nepredvídateľné) a ak zavolal nejaké iné programy, ktoré neboli riadne ukončené tak dochádzalo k vytvoreniu Zombie procesov a podobného bordeľu, ktorý mohol viesť k nestabilite systému. Riešenie priniesla nová generácia ramdisku zvaná (init)ramfs (popri ňom vznikol aj tmpfs).

Aké zmeny priniesol ramfs? Predovšetkým šlo o špecifikáciu pre init, už sa viac nespúšťal linuxrc ale disk musel obsahovať súbor init, ktorý sa spustil ako klasický a taktiež s PID 1. Čo už tento init robí je čisto len jeho vec. Ďalšou novinkou bolo použitie cpio archívu. Prvá generácia vyžadovala pre mountnutie ovládače pre FS v kerneli, nový RamFS je len obyčajný archív bez súborového systému, ktorý kernel len rozbalí do pamäte a spustí v ňom /init. Posledným takým významným vylepšením bola dynamická veľkosť. Predchádzajúci ramdisk mal pevne stanovenú veľkosť, ktorú zaberal v pamäti a bolo jedno že či súbory v vnútri majú len pár kB oproti 2MB ramdisku. Taktiež nemohol pochopiteľne presiahnuť veľkosť 2MB. Kdežto RamFS sa dynamicky zväčšuje a zmenšuje. Ak sa vnútri odstráni nejaký súbor tak sa automaticky uvoľní pamäť RAM, ktorá bola tým súborom zabratá, alebo analogicky zase ak sa tam nejaký súbor zapísal tak sa automaticky alokovala nová pamäť a ramdisk sa zväčšil. Samozrejme ale mohlo dôjsť k tomu že systém zamrzne pretože RamFS vyplní celú pamäť RAM a nezostane žiadna voľná, to je ale skôr také vedľajšie. Popritom vznikol aj tmpfs, ktorý je veľmi podobný ako ramfs ale má pevne danú veľkosť a taktiež swapuje na normálny disk.

Fajn, dosť bolo kecov, teóriu už ovládame a teraz ako to vôbec súvisí s tvorením a úpravou distribúcie? V predchádzajúcom článku sme upravovali súbor filesystem.squashfs, ktorý obsahoval kompletný súborový systém distra. Problém je v tom že ako z neho kernel zavedie systém? Na to je použitý ramdisk (initrd), ktorý tento súbor mountne ako root FS a spustí z neho proces init. Okrem toho sa ramdisk využíva na také fancy vecičky ako zobrazenie splashu

počas zavádzania systému, zostavenie RAID poľa atď...

Ukážeme si teraz ako si taký základný ramdisk zostavíme. Spravíme si ramdisk, ktorý nám kernel načíta a z neho sa nám spustí shell. Presuňte sa do vášho pracovného priečinka a vytvorte si základnú štruktúru ramdisku:

```
intense@Singularity ~/Desktop/blog $ mkdir -p bin lib etc
```

```
intense@Singularity ~/Desktop/blog $ ls
```

```
bin  etc  lib
```

Teraz si tam skopírujeme bash z nášho hosťovského systému:

```
intense@Singularity ~/Desktop/blog $ cp /bin/bash bin/
```

Komplikácie pri vytváraní systému v ramdisku môžu nastatať so závislosťami, musíme tam taktiež nakopírovať všetky súbory ako sú napr. *.so knižnice a konfiguráky pre danú binárku pretože ramdisk je určený na to aby bežal úplne nezávisle od normálneho systému takže musí obsahovať všetko potrebné k behu programov. Tu príde vhod utilitka ldd, prípadne rôzne nástroje ako sú equery v gentoo alebo pacman v arch linuxe pre vyhľadanie závislostí:

```
intense@Singularity ~/Desktop/blog $ ldd /bin/bash
```

```
linux-gate.so.1 =>  (0xb77de000)
```

libreadline.so.6 => /lib/libreadline.so.6 (0xb776c000)

libncurses.so.5 => /lib/libncurses.so.5 (0xb7728000)

libdl.so.2 => /lib/libdl.so.2 (0xb7724000)

libc.so.6 => /lib/libc.so.6 (0xb75cd000)

/lib/ld-linux.so.2 (0xb77c2000)

Vidíme zoznam knižníc na ktorých bash zavisí tak ich nakopírujeme do štruktúry nášho ramdisku. Nájsť konfiguračné súbory (alebo nejaké iné súbory), na ktorých závisí nejaký program môže byť dosť veľký problém, pomôcť vám môžu vyššie spomenuté utilitky alebo ešte lsof a strace. Takže nakopírujeme všetky závislosti pre bash (linux-gate.so.1 nepotrebujeme):

intense@Singularity ~/Desktop/blog $ cp /lib/libreadline.so.6 lib/

intense@Singularity ~/Desktop/blog $ cp /lib/libncurses.so.5 lib/

intense@Singularity ~/Desktop/blog $ cp /lib/libdl.so.2 lib/

intense@Singularity ~/Desktop/blog $ cp /lib/libc.so.6 lib/

intense@Singularity ~/Desktop/blog $ cp /lib/ld-linux.so.2 lib/

Našťastie bash sa zaobíde aj bez konfigurákov takže iné veci nepotrebujeme. To či máme všetko potrebné môžeme vyskúšať tým že sa chrootneme do štruktúry ramdisku a zároveň spustíme bash:

```
Singularity blog # chroot ./ /bin/bash

bash-4.2#
```

Super, všetko funguje. Ak ale skúsite použiť príkazy typu ls, mv a podobne tak dostanete hlášku typu „bash: mv: command not found“. To je preto lebo tento program v našom ramdisku neexistuje, samozrejme, niektoré príkazy (napríklad pwd), ktoré sú už v bashi vstavané môžete bez problémov používať. Ak chcete ale napríklad používať príkaz ifconfig, ls, mount a podobne, musíte si ich všetky nakopírovať do ramdisku ako aj ich závislosti rovnako ako sme to spravili s bashom. Takže teraz si tam napchajte všetko čo chcete mať v ramdisku, obmedzený ste len veľkosťou RAM vášho PC.

. Príklad nakopírovania utility ls do ramdisku:

```
intense@Singularity ~/Desktop/blog $ cp /bin/ls bin/

intense@Singularity ~/Desktop/blog $ ldd /bin/ls

linux-gate.so.1 =>  (0xb778e000)

librt.so.1 => /lib/librt.so.1 (0xb7749000)

libacl.so.1 => /lib/libacl.so.1 (0xb7740000)

libc.so.6 => /lib/libc.so.6 (0xb75e9000)
```

libpthread.so.0 => /lib/libpthread.so.0 (0xb75d0000)

/lib/ld-linux.so.2 (0xb7772000)

libattr.so.1 => /lib/libattr.so.1 (0xb75c9000)

intense@Singularity ~/Desktop/blog $ cp /lib/libattr.so.1 lib/

intense@Singularity ~/Desktop/blog $ cp /lib/libpthread.so.0 lib/

intense@Singularity ~/Desktop/blog $ cp /lib/libacl.so.1 lib/

intense@Singularity ~/Desktop/blog $ cp /lib/librt.so.1 lib/

Singularity blog # chroot ./ /bin/bash

bash-4.2# ls

bin etc lib

Keď ste si už do ramdisku napchali všetky potrebné veci, je čas ho zabaliť. Ako som už hore v teórií spomínal, potrebujeme mať súbor init, ktorý kernel spustí hneď po rozbalení obsahu ramdisku. Keďže bash je sám o sebe interpreterom pre shellovské scripty, môže byť aj init tým pádom normálny shell script:

```
#!/bin/bash

#spustime shell

/bin/bash;

#Ano, pri takomto ukonceni dostaneme kernel panic

#to nas ale teraz netrapi ;-)

exit 0;
```

Nezabudnite tomuto scriptu samozrejme nataviť príznak spustiteľnosti (chmod +x). Teraz už nám len stačí zabaliť náš ramdisk. RamFS je vlastne len gzipovaný cpio archív takže nie je nič jednoduchšie ;-) :

```
intense@Singularity ~/Desktop/blog $ find . -print0|cpio --null -ov --format=newc|gzip -9 >
```

../rambash.cpio.gz

.

./lib

./lib/ld-linux.so.2

./lib/libc.so.6

./lib/libncurses.so.5

./lib/libpthread.so.0

./lib/libacl.so.1

./lib/libreadline.so.6

./lib/libattr.so.1

./lib/libdl.so.2

./lib/librt.so.1

./init

./bin

./bin/ls

./bin/bash

./etc

5697 blocks

intense@Singularity ~/Desktop/blog $ ls ../

114.png 114.svg 79.png 79.svg blog rambash.cpio.gz

Super, teraz už len upravíme zavádzač aby sa náš ramdisk použil ako initrd (initrd je vlastne len RamFS použitý k zavedeniu systému). Otvorte konfigurák grubu, kde pomocou možnosti initrd zadáte cestu k vášmu ramdisku. Príklad:

title Exile Linux

root (hd0,1)

kernel /boot/Exile-kernel root=/dev/sdb2

initrd /boot/rambash.cpio.gz

Samozrejme váš kernel musí podporovať zavedenie systému z ramdisku. Skontrolujte si to v súbore /proc/filesystems či tam máte podporu pre ramfs. Ak si kernel kompilujete, možno využijete možnosť že môžete ramdisk zabudovať priamo do kernelu. Túto možnosť nájdete v konfigurácii kernelu pod General Setup -> Initramfs source file(s).

Teraz už len reštartujte PC alebo zapnite virtuálnu mašinu a užite sa nábeh systému do vášho bash shellu. Ak máte vyriešené všetky závislosti a na nič ste nezabudli tak by malo všetko fachať bez problém. Týmto spôsobom sme si teraz vlastne vytvorili mini distribúciu, bežiacu čisto len v RAM, ktorá pozostáva len zo shellu bez všetkých tých fancy vecičiek okolo. Samozrejme že fantázií sa medze nekladú a tak si kľudne môžete zakomponovať do ramdisku aj Xka alebo rovno celú distribúciu. Môžete ale napríklad využiť aj to ako fungujú live distribúcie čo som už spomínal vyššie tak že si vytvoríte SquashFS, ktorý bude obsahovať vašu distribúciu a z ramdisku len tento FS mountnete a následne z neho zavediete váš systém. Veľké využitie majú ramdisky aj ako záchranné prostredia. Keďže ramdisk beží nezávisle od hosťovského systému, môžete bez problémov nabootovať do záchranného shellu alebo rovno aj celého prostredia aj keď váš lokálny systém nedokáže nabootovať. A tá najlepšia vec, systém bežiace v RamFS je fakt nenormálne rýchly keď si ho porovnáte s klasickým systémov zavádzajúcim z disku.

Pre tých fajnšmekrov sa nájde využitie pre ramdisk aj na ich lokálnom systéme. Príkladom môže byť premiestnenie /bin /lib a podobných súborových štruktúr do ramdisku, ktoré sa veľmi často používajú pri spúšťaní programov. Následne upravíte /etc/fstab a nastavíte mountpointy pre vaše ramdisky obsahujúce túto štruktúru a tak budú tieto súbory celý čas v RAM vďaka čomu sa vám budú spúšťať programy nenormálne rýchlo.

Týmto teraz ovládate vražednú sadu nástrojov, ktorú môžete využiť k vybudovaniu distribúcie

alebo rôznym úpravam distribúcie na vysokej úrovni keďže ovplyvňujete aj samotné zavádzanie systému.

Na záver vám dám ešte pár užitočných typov:

- Ak si buduje shell skúste sa pozrieť na BusyBox. Je to shell obsahujúci aj sadu zákaldných nástrojov ako je mount, ifconfig, cat, echo atď... takže keď si staviate nejaký shell tak nemusíte každý jeden nástroj kopírovať ku bashu (alebo inému shellu ak preferujete) čo môže byť dosť únavné.
- Dávajte si pozor ako sa váš systém ukončuje. Pri nesprávnom ukončení init dostanete kernel panic s hláškou že sa pokúsil zabiť init proces čo je nie je práve potešujúce vypnutie systému ;-)
- Po tom čo váš init vykoná nejaké základné veci podľa toho na čo ho máte, môžete execnutím spustiť init už normálneho systému a predať mu tak kontrolu nad systémom.
- Naraz môžete špecifikovať aj viac ako jeden ramdisk pomocou viacerých volieb initrd pre grub. Načítajú sa v poradí v akom sú zadané (najprv sa ale vždy načíta ten, ktorý je v kerneli zabudovaný ak ste tam nejaký dali) a všetky sú rozbalené do toho istého priestoru takže pozor aby vám jeden ramdisk neprepísal súbory toho druhého.
- Ak chcete pristupovať k zariadeniam tak si ich nakopírujte do /dev. V prípade že pristupujete k špecifickým funkciám, ktoré poskytuje jadro tak príde vhod dať do init nejaké sleep() aby ste ho nachvíľu uspali. Kernel totižto pri zavádzaní systému nemusí stihnuť inicializovať danú vec ešte pred tým než ju chcete použiť
- Môžete vypnúť cez /proc vo voľbe /printk aby sa vám vypisovali hlášky kernelu do shellu keď sa spustí. Príde to vhod ak nechcete aby sa vám tam zrazu začali hádzať nejaké hlášky keď pracujete so shellom.