Фундаментални неща в Линукс

08 октомври 2020 г. 13:41

Как стартира системата:

1. Първо BIOS проверява всички output и input устройства. Когато проверските свършат boot проецеса започва.
2. Boot sector на първият хард диск - буут лоудъра започва да търси секцията която съдържа данните необходими да зареди една операционна система
3. Boot loader зарежда линукс ядрото след това ядрото ще зареди пъвичният RAM диск които съдържа доста драйвъри и започва да зарежда драйвърите които зареждат файловата система на хард диска.
4. След като ядрото зареди започва Инициализация на системата. Исторически ядрото не се интересува от инциализиращият процес.
5. Идеята му е била да разбира inputa и output който му е подаден не зависимо от къде идва и да взаимодейства с компютърният hardware.
6. Като цяло ядрото извършва много функции, но някои от тях стартират вариация от програми познати като сервизи (services) които правят компютъра полезен
7. След като инициализиращият процес започне закачането на файловата система. Това е момента в който initial RAM не е необходима повече.
8. Продължават да зарежат сервизи.

Boot logs:

Логове които се генерират по време на зареждането. Веднъж компютъра е рестартиран логовете се изтриват и пресъздават.

dmesg

традиционен уред с който можем да видим ring buffer --> локацията където се запаметтяват всички буут логове.

Намира се рама. Зарежда много бързо защото нон стоп се пише информация по него.

Ще откриете детайли за хардуеъра които ядрото може да види и как ги активира и много съобщения от ниско ниво когато ядрото се "разправя с рама"

С тази команда ще проверявате дали ядрото си говори с хардуеъра който имате а не програмите да ли го виждат.

dmesg е завещаният метод за проверка на хардуеъра.

В новите системи се употребява systemd вместо старият init system оттам идва и командата journalctl -k

Уреда с който може да видите логове на ядрото в буфера използвайки systemd.

init:

- init - накратко обозначава 'initialization'
- Въз основа на System V използвана в unix подобни системи.
- sysvinit написана от miquelvan smoorenburg
- Сервизите започват един след друг, по сериен начин.

- Проблема се е зародил, когато някой от сервизите не зареди, следователно цялата система е запичала.

След като зареди първият рам диск започва търсенето на initialization system в /sbin/init след като ядрото намери init файла ядрото почва да търси конфигурацията на /etc/inittab (на инит).

Преди да го погледнем трябва да видим какво init точно търси и това е:

RUNLEVEL	ЦЕЛ
0	Halt(Спиране)
1	Single user Mode (режим на един потребител) ползван е root с цел поправяне на системата
2	Multi user mode (no networking) (Мулти потребителски режим (без мрежа)
3	Multi user mode (with networking) (Мулти потребителски режим (с мрежа)
4	unused (неизползван)
5	Multi-user mode, with networking and a graphical desktop (многопотребителски режим, с мрежа и графичен десктоп)
6	Reboot (Рестартиране)

/etc/inittab

<indentifier>:<runlevel>:<action>:<process>

id:3:initdefault:

si::sysinit:/etc/rc.d/rc.sysinit

10:0:wait:/etc/rc.d/rc 0

11:1:wait:/etc/rc.d/rc 1

12:2:wait:/etc/rc.d/rc 2

13:3:wait:/etc/rc.d/rc 3

14:4:wait:/etc/rc.d/rc 4

15:5:wait:/etc/rc.d/rc 5

16:6:wait:/etc/rc.d/rc 6

wait --> процеса които е посочен ще стартира само и само коагато runlevala е вкаран, и init ще чака терминирането си.

Традиционните папки които съдържат скриптовете за стартиране са в:

RedHat Based: /etc/rc.d

Debian: /etc/init.d

rc е акроним за run commands

В тези папки се съдържат скриптове символни връзки към оригиналните /etc/indit.d

В /etc/init.d/rc са скриптовете които оркестрират как runlevel скриптовете и какво се случва като runlevel се промени

Upstart

- Първо е разработен за ubuntu 2006 година от Скот Ремнант
- Първо е ползван в ubuntu6.10
- Последователно успеха му е включен в Redhat enterprise 6 debian и fedora 9
- За разлика от init предлага асинхроничен метод на стартиране на сервизите което е намалило времето за стартиране.
- Използва real-time events което init неможе да разбере.
- Което значи че не само спира и стартира сервизите но и знае дали са достъпни.
- Което води до факта че upstart може да разбере че нещо е спряло и да се опита да го включи.

Следователно:

init е статичен и не реагира натурално на промени в системата.

upstart е динамичен и може да реагира натурално

промяна в системата на линукс се нарича event

ивентите действат като спусъчни механизми на jobs

jobs са две категории tasks и сервизи (services)

Systemd:

- Премахва нуждата от bash скриптове и зависимоста от тях както в upstart, защото трябва да минат през баш интерператора което води до образуването на много процеси, всеки един от тези скриптове ще трябва да отвори библиотека.
- Следователно това не е ефектиевен начин за ползването на време с цел зареждане на сервизи.
- Systemd заменя баш скриптовете с изкомпилиран C код който върши същата работа.
- Продължава да поддържа скриптове от init --> авторите гарантират 99% съвместимост със старта init система която е била доста прозрачна.
- systemd работи с unit files.

Могат да бъдат открити в /usr/lib/systemd/system --> не променяйте поради ъпдейтите на системите:

- /etc/systemd/system тук може да променяте тези файлове.
- /run/systemd/system runtime файлове.

systemctl list-unit-files

- Компоненти на unit файловете:
- следват ini формата на MS-DOS
- Description=Multi-User System
- Documentation=man:systemd.special (7)
- Нуждае се от basic target

- man systemd.unit

sed - 'stream editor'

Използва се за вариация от различни задачи.

За LPIC ще трябва да знаете как да правите търсене и заместване с тази команда.

sed 's/desktop/workstation/' list.csv

- Буквичката 's' посочва ще правим substitution (замяна).Наклонените черти се ползват с цел да сложим думата която искаме да махнем.
- Работи и с променливи.
- Което автоматично посочва, че командата ще бъде екстремно полезна в баш скриптове, чиято цел имат да променят дадени конфигурационни файлов. Просто казано командата, може да бъде богинята на автоматизацията ви. Когато учим за елементарни баш скриптове ще разберете.

За да могат да работят променливите използвайте двойни кавички.

За да направите промяната перманентно използвайте -i:

-i[SUFFIX], --in-place[=SUFFIX]

edit files in place (makes backup if SUFFIX supplied)

sed 's/desktop/workstation/g' list.csv > new_list.csv

Този пример показва, че можем да насочим стандартният изход към нов файл.

Съдържанието на новият файл ще има съдържание чието е променено.

split

Използва се да "хване" файл и да го пръсне на индивидуални парчета.

Обикновенно всяко парче файл съдържа до 1000 линии, но това може да бъде променено като стойност или различен файлов размер.

- split random_text.txt
- Ще видите, че командата създава 16 различни файла започващи с x*.
- rm x* ще ги премахне.

- split -d -n2 random_text.txt
- Където -d ще каже на командата направи файловете по цифрова логика.
- -n2 направи два файла. (колко на брой файлове)
- split -b 100 -d random_text.txt
- find . -size -100 ще ни покаже, че всички файлове са 100 байта големи.
- Можем да сложим 100k за килобайти 100m за мегабайти и т.н.
- split -b 100 (k,M,G) в зависимост дали файла е достатъчно голям.

nl - принтира боря на редовете от един файл.

nl –a

od - октален дъмп

od random_text.txt

Показва съдържанието на файла в октален формат първата колона е байт офсета другите са думите.

od -c random_text.txt

Message Digests:

Дайджест на съобщенията е криптографска функция, съдържаща низ от цифри, създадени по формула за еднопосочно хеширане.

Целта им е да защитят целоста на част от данни или медия. За да открият промени по която и да е, част от съобщението използват хеш-стойности.Стойностите могат да предупредят собственика за авторските права за всякакви промени, приложени към тяхната работа.

Хеш-номерата представляват конкретни файлове, съдържащи защитени произведения. Едно съобщение дайджест е присвоено на определено съдържание на данни. То може да се позовава на промяна направена умишлено или случайно. Поддтиква собственика да индефицира модификацията както и индивида(ите) които са направили промяната.

Тези съобщения са още алгоритмични числа.

- md5sum test.sh
- md5sum test.sh > test.md5
- nick@fly:~/md5sum$ vim test.sh
- nick@fly:~/md5sum$ md5sum test.sh
- 82ff7d7c60ac72be4caf65d43dc89a5d test.sh
- nick@fly:~/md5sum$ cat test.md5
- fcf3ca8f8d95f56998bfc0ddf134a5c4 test.sh

- Промяната веднага е забележима.
- Когато си сваляте някое .iso от интернет, често разработчика дава и файл съдържащ md5 стойноста.
- По време на проверката вие ще може да сверите дали .iso съдържанието е кърпнато (развалено,променено).
- md5sum test.sh
- md5sum -c test.md5

- nick@fly:~/md5sum$ sha256sum test.sh
- ce0f78d2a09831ac430a1d799ae618bfd8facf71a6988539a42073713354f0c8 test.sh
- nick@fly:~/md5sum$ sha256sum test.sh > test.sha256
- nick@fly:~/md5sum$ sha256sum -c test.sh
- test.sh   test.sha256
- nick@fly:~/md5sum$ sha256sum -c test.sha256
- test.sh: OK
- nick@fly:~/md5sum$
- sha256sum Много по силен алгоритъм с цел криптиране.
- sha512sum Още по силен алгоритъм.

cat още значи concatenate (събиране)

Пример:

Примера показва успешно събиране на два файла в един.

Примери които ползват Regular Expressions в някой команди:

- cat passwd | sed -n '/nologin$/p'
- Отваряки файла passwd ще принтираме всяка линия която съдържа nologin в своя край.
- cat passwd | sed '/nologin$/d' > filtered.txt
- Отваряйки файла passwd ще изтрия всички users които са но логин директно местейки ги във файл който се казва filtered.txt

Egrep:

Команда която търси в посочен файл линия по линия.

- Връщаки ни линиите които съдържат патерната която търсим.
- Еквивалента и е grep -E
- egrep 'bash$' passwd
- Ще ни покаже само и само линиите съдържащи bash в края си.
- egrep -c 'bash$' passwd
- Ще преброи линиите които съдържат bash в своя край.
- egrep '^man|nologin$' passwd
- Ще извади линиите които съдържат man отпред и ще ни покаже линиите с nologin.

Fgrep:

Търси базирано на низ за разлика от патерна. Също ползва file globbing вместо RegEx.

- Еквивалент на grep -F
- fgrep -f strings passwd
- Тук съм създал файл който съдържа определени низове по които ще търси в passwd
- fgrep -f strings passwd*
- Пример с File Globbing.

Менижиране на споделени библиотеки:

Файлове които съдържат функции които другите апликации ползват.

- Имат .so екстеншън които посочва shared object (споделен обект)
- Намират се в /lib, /usr/lib(за 32 битови системи) и /usr/lib64 (за 64 битови системи)
- Можете да ги откриете още в:

1. /usr/local/lib

2. /usr/share

Менижиране на споделени библиотеки:

Два типа:

Динамични .so

Статични .a компилира се за апликации които трябва да си потвърдят, че когато извикат определена функция то тя е същата версия на функцията която са повикали.
Често ще откривате апликации които когато си използват функциите те трябва да точно определената версия. Като игри и графични софтуеъри.

Команди:

ldd :

Принтира споделеният обект (shared object dependencies)

Пример:

Това което ще видите са споделените обекти (файлове) и с кои файлове са свързани. Ще забележите, че различни команди ползват различен подбор от библиотеки. Също и че някои от тях използват едни и същи като ср и ls.

Реално идеята да ползват едни и същи библиотеки е като в ср и ls: -с цел да се намали времето за програмиране от разработчиците.

Нека видим папката:

1. ls cat /etc/ld.so.conf.d/
2. Нека видим конфигурацията:
3. cat /etc/ld.so.conf.d/libc.conf
4. Нека видим новата:
5. cat x86_64-linux-gnu.conf
6. Нека видим какво има вътре:
7. ls /lib/x86_64-linux-gnu/

LD_LIBRARY_PATH

Стара стойност на средата която посочва пътя към бибилиотечните файлове от които може да се чете.

Например ако сте инсталиралира нова джава в папката /opt

export LD_LIBRARY_PATH=/opt/java/jre/lib

Tar:

Управлява .tar файлове , още наречени tarball, които се използват за архивиране и трансфер на файлове.

- tar архивите запазват структурата на файловете и директориите, както и метаданните на файловете (собственост, права, времена на създаване, достъп и модификация и др.) .
- tar поддържа компресиране на файловете по време на архивиране.

Опции:

-c (create) – архивиране

-x (extract) – разархивиране

-t (test) – показва съдържанието на архива без да го разархивира

-f (file) – указва името на архива

-z – използва gzip компресия

-j – изпозлва bzip2 компресия

-J – използва xz компресия.

- tar -czvf name-of-archive.tar.gz /path/to/files .
- tar -czvf name-of-archive.tar.gz .

Cpio:

cpio – управлява .cpio архивни файлове , които са основата на RPM пакетите.

- cpio не обхожда рекурсивно поддиректориите (трябва да бъде указан списък с директории)
- cpio се справя по-добре от tar при грешки в носителите.
- Опции: -i (input mode) - разархивиране -o (output mode) - архивиране -F (file) - указва името на файла в/от който ще архивира/разархивира
- RPM пакетите са изградени от cpio архиви и RPM метаданни (header).
- rpm2cpio - премахва RPM header-а с цел достъп до файл от пакета без неговото инсталира

Проверка на използвано дисково пространство:

du
Df

Awk:

Мощен език за обработка на текстови потоци, поддържащ регулярни изрази, математически оператори, променливи, входно-изходни оператори и др.

- awk '{print "Welcome to awk command"}'
- awk -F: '{print $1}' /etc/passwd

$0 for the whole line.

$1 for the first field.

$2 for the second field.

$n for the nth field.

- awk -F: '{print $1, $3}' /etc/passwd
- awk -F: '/bash$/ {print $1}' /etc/passwd