Bootowanie systemu

(aktualizacja: 2019-04-17)

GRUB

Po zakończeniu procesu inicjalizacji sprzętu i testów rozruchowych BIOS ładuje do pamięci kod znajdujący się w pierwszym sektorze dysku twardego (sektorze rozruchowym rozpoczynającym się od adresu zerowego) i uruchamia go (przekazuje do niego kontrolę). W przypadku sektorów rozruchowych typu MBR (i kompatybilnych z nim), kod ten może liczyć maksymalnie 446 bajtów (gdyż na kolejnych pozycjach znajduje się tablica partycji) i jego zadaniem jest załadowanie i uruchomienie pozostałej częsci programu rozruchowego (np. może użyć programu w partycji oznaczonej jako bootowalna).

GRUB jest jednym z najpopularniejszych w środowisku linuxowym boot menagerów / boot loaderów. Jego wykonywalna przy starcie komputera wersja składa się z 3 części:

- stage1 (boot.img) - ma za zadanie uruchomić stage1.5 (zawiera jego położenie na dysku)
- stage1.5 (core.img) - zawiera moduły pozwalające na dostęp do systemu plików zawierającego stage2 oraz informację o jego położeniu (dysk, partycja, ścieżka)
- stage2 (/boot/grub) - zawiera wszystkie dodatkowe moduły oraz konfigurację

Może on być zainstalowany zarówno w MBR, jak i w bootowalnej partycji. W tym pierwszym przypadku w MBR wgrywany jest stage1, natomiast stage1.5 w zależności od używanego typu tablicy partycji znajduje się albo tuż za MBR - w przerwie pomiędzy MBR a pierwszą partycją (dla partycji MBR/msdos), albo w dedykowanej partycji BIOS boot partition (w przypadku GPT).

W przypadku komputerów opartych na UEFI firmware odpowiedzialny jest za zinterpretowanie tablicy partycji (GPT) i załadowanie programu rozruchowego z pliku znajdującego się na specjalnej partycji EFI (EFI System partition) z systemem plików FAT32. GRUB może pełnic funkcję tak uruchamianego boot loadera (w tym przypadku całość potrzebnego kodu stage1 i stage1.5 znajduje się w pliku na tej partycji).

Zadaniem GRUBa jest umożliwienie wyboru, załadowanie i uruchomienie jądra systemu operacyjnego z zadanymi opcjami. Jego działanie kończy się w momencie gdy jądro przejmuje sterowanie.

Konfiguracja

Współcześnie konfiguracja GRUB'a jest generowana automatycznie w oparciu o `/etc/grub.d/` i `/etc/default/grub` przez `grub-mkconfig` w oparciu o wykrywanie typów dysków na których jest zainstalowany system (raid, lvm, typ systemu plików, ...) oraz unikalne identyfikatory dysków, partycji, itd. Mimo to warto znać przynajmniej podstawy działania grub'a, jego komend, aby być wstanie coś zrobić gdy zamiast menu startowego ujrzymy tylko konsolę awaryjną. Poniżej zamieszczam przykładowy plik konfiguracyjny `/boot/grub/grub.cfg` z komentarzami opisującymi zanczenie różnych poleceń i opcji.

```
# oficjalna dokumentacja: http://www.gnu.org/software/grub/manual/grub.html

# korzystanie z konsoli na porcie szeregowym
#serial --speed=115200 --unit=1 --word=8 --parity=no --stop=1
#terminal_input serial console
#terminal_output serial console
# za to na którym RS jest grub odpowiada --unit w serial
#
# korespondujące opcje uruchamiania dla Linuxa:
#   module    /boot/vmlinuz [...] console=tty0 console=ttyS1,115200n8
# korespondujące opcje uruchamiania dla XENa:
#   multiboot /boot/xen.gz  [...] com2=115200,8n1 vga=mode-0x0319 console=com2,vga
#   module    /boot/vmlinuz [...] console=tty0 console=hvc0
#
# w opcjach kernela szczególną rolę odgrywa ostatni podany terminal (ostatnia opcja console)
# jest on związany z /dev/console i na nim pokazywany jest init, ponadto należy uruchamiać getty
# na odpowiednim urządzeniu poprzez wpis w inittab np.:
#   T0:23:respawn:/sbin/getty -L /dev/ttyS1 115200 vt100
# lub (dla wariantu XENowego, UWAGA: w przykładzie tylko runlevel 4 i 5):
#   T1:45:respawn:/sbin/getty 38400 hvc0

# timeout w sekundach dla domyślnej pozycji menu
```

```
set timeout=60
set default="0"

# ładowanie modułów - typ tablicy partycji
insmod part_gpt
#insmod part_msdos

# ładowanie modułów - RAID
insmod mdraid1x # mdraid metadata 1.x
#insmod mdraid09 # mdraid metadata 0.9
#insmod raid5rec # faulty RAID4/5
#insmod raid6rec # faulty RAID6

# ładowanie modułów - LVM
insmod lvm

# ładowanie modułów - File System
insmod xfs

# ustawienie root'a
set root='lvm0-root'

menuentry 'Debian GNU/Linux' {
        # obraz jądra Linuxa z przekazywanymi do systemu opcjami
        linux   /boot/vmlinuz-3.12-trunk-amd64 root=/dev/mapper/lvm0-root ro rootdelay=3 quiet
        # obraz initrd
        initrd  /boot/initrd.img-3.12-trunk-amd64
}

menuentry "SBM Boot Manager" {
        linux16  /boot/extra/memdisk floppy
        # memdisk z pakietu syslinux-common (/usr/lib/syslinux/memdisk)
        initrd16 /boot/extra/sbootmgr.dsk
        # SBM Boot Manager (http://btmgr.sourceforge.net/about.html)
        # z http://ftp.lanet.lv/ftp/mirror/Slackware/isolinux/sbootmgr/
}
```

Więcej informacji

- GRUB @ wiki.archlinux.org
- UEFI @ wiki.archlinux.org

Start systemu - initrd, init i pliki startowe

Start systemu rozpoczyna się od załadowania do pamięci obrazu jądra wraz z parametrami oraz (opcjonalnie) initrd i przekazania kontroli do jądra przez program rozruchowy (np. GRUB, LILO). W przypadku korzystania z initrd obraz ten przekształcany jest na RAM-dysk w trybie zapisu-odczytu i montowany jako rootfs z którego uruchamiany jest `/init` (nie `/sbin/init`). Po zakończeniu jego działania (lub od razu gdy nie używamy initrd) uruchamiany jest program wskazany w opcji `init=` jądra (typowo domyślnie jest to `/sbin/init`) z rootfs wskazanego w opcji `root=` jądra.

initrd

Zawartość obrazu rozruchowego można podejrzeć wypakowując jego zawartość przy np. pomocy następującej funkcji:

```
extractInitRd() {
        [ $# -ne 2 ] && echo "USAGE extractInitRd path/to/initrd extratDir" && return
        initrd=`realpath "$1"`
        mkdir -p "$2"; cd "$2"
        if file "$initrd" | grep "ASCII cpio archive" >& /dev/null; then
                # extract microcode
                blocks=`cpio -i < "$initrd" |& awk '/^[0-9]* blocks$/ {print $1}'`
                echo "unpack $blocks blocks of microdode from initrd"

                # extract ramdisk
                mkdir initrd; cd initrd
                dd if="$initrd" of=/dev/stdout bs=512 skip=$blocks | zcat | cpio -i
        else
                # no microcode ... extract ramdisk
                zcat "$initrd" | cpio -i
```

```
        fi
}
```

Jednak zwykłe ponowne spakowanie go nie przynosi pożądanych rezultatów, ponadto współczesnych obrazów nie da się już montować przez mount -o loop. Celem dostosowywania initrd do własnych potrzeb polecam przyjrzenie się strukturze katalogów `/usr/share/initramfs-tools/` i `/etc/initramfs-tools/`. W szczególności wart przeanalizowania jest skrypt `/usr/share/initramfs-tools/init`, który jest uruchamiany zaraz po odpaleniu obrazu startowego (jest kopiowany przy tworzeniu initrd.img na `/init` w obrazie ram dysku). Po wykonaniu stosownych zmian w tych plikach obraz można zbudować poprzez wywołanie `mkinitramfs` lub wygodniejszego `update-initramfs -u`.

Część z opcji przekazywanych jako parametry jądra jest obsługiwana przez initrd - wybrane z tych opcji:

- `break=top|modules|premount|mount|mountroot|bottom|init` - przerywa initrd (i uruchamia powłokę busybox) w wskazanym momencie (w initramfs-tools 0.92o jest błąd powodujący iż break-point init nie działa - przed wywołaniem maybe_break init jest wstawione unset break), dalsze bootowanie można wznowić Ctrl+D
- `rootdelay=xxx` - wstawia opóźnienie xxx sekund przed próbą montowania głównego systemu plików (pomiędzy inicjalizacją urządzeń fizycznych a startem skryptów local - tworzących urządzenia md, lvm itp), jej uzycie może być konieczne w przypadku korzystania z rootfs na lvm, soft-raid itp
- `rootflags=` - podaje flagi montowania głównego systemu plików (przekazane do opcji -o komendy mount)
- `boot=` - określa jakie skrypty celem zamontowania root fs ma zainkludować initrd (domyślnie local), boot=xxx nakazuje wykonanie skryptu /scripts/xxx (np. /scripts/nfs) przed montowaniem głównego systemu plików
- `debug` - zwiększa głośność initrd
- `blacklist=` - rozdzielana przecinkami lista modułów których ładowania chcemy zabronić na etapie initrd
- `init=` - określa jaki program ma być wykonany zaraz po zakończeniu initrd

Więcej opcji linii poleceń jądra znaleźć można w `man bootparam` oraz `man init`. Zaznaczyć tu należy iż zgodnie z `man bootparam` parametry postaci nazwa=wartosc nie rozpoznane jako parametry jądra są przekazywane jako zmienne środowiskowe, obecnie trafiają one w tej postaci do skryptów wywoływanych przez init (jednak zdarzało się że gdzieś ginęły). Natomiast same procesy związane z logowaniem użytkownika nie przekazują tych zmiennych do środowiska obecnego po zalogowaniu, zatem aby z nich skorzystać konieczne jest przetwarzanie `/proc/cmdline` (ze względów na nie pewność co do ustawienia środowiska przez init warto tak postępować także w skryptach init). Można to realizować w następujący sposób `ZMIENNA=`awk 'BEGIN {RS="[\n]"; FS="=";} $1=="ZMIENNA" {print $2}' /proc/cmdline`` lub:

```
for tmp in $(cat /proc/cmdline); do
        case $tmp in
                ZMIENNA=*)
                        ZMIENNA=${tmp#ZMIENNA=}
                        ;;
        esac
done
```

Działanie skryptu `/init` z initrd.img kończy się wywołaniem (poprzez exec) programu `run-init`, którego zadaniem jest chroot-owanie się do wcześniej zamontowanego katalogu z rootfs'em oraz execowanie się na wskazany program rozruchowy (typowo `/sbin/init` bądź inny wskazany w opcji jądra) na nowym root filesystemie.

klasyczny init

Klasyczny init konfigurowany jest z uzyciem pliku `/etc/initab` i oferuje 7 poziomów działania (0 - wyłączenie, 1 - pojedynczy użytkownik (single), 2...5 - standardowe (najczęściej używany 2), 6 - restart). Podczas zmiany poziomu rozruchu przetwarza skrypty startowe (będące na ogół zwykłymi skryptami powłoki) z odpowiedniego katalogu /etc/rcX.d (X- poziom rozruchu na który przechodzi init). Pliki w katalogach /etc/rcX.d mają nazwy postaci YZZxxxx, gdzie Y to S gdy dany skrypt wykonywany ma być z parametrem "start" albo K gdy z parametrem "stop", ZZ to dwucyfrowy numer decydujący o kolejności wykonania, xxxx - nazwa skryptu. Typowo są one dowiązaniami symbolicznymi do skryptów umieszczonych w /etc/init.d. Do zarządzania nimi można użyć polecenia

`update-rc.d`. Przy starcie systemu najpierw wykonywane są skrypty z /etc/rcS.d a następnie zadanego poziomu (zazwyczaj /etc/rc2.d, rzadziej z /etc/rc1.d - tu na ogół i tak są same "stop", a jeszcze rzadziej z innych).

systemd

Współcześnie `/sbin/init` wskazuje najczęściej na systemd. W odróżnieniu od klasycznego podejścia nie opiera się on na skryptach umieszczanych w `/etc/init.d` i linkowanych do odpowiedniego `/etc/rc*.d` a na opisach usług w postaci plików konfiguracyjnych typu `.desktop` / `.ini`, jednak często obsługuje też skrypty klasycznego inita.

Do zarządzania plikami usług (listowania dostępnych, włączania/wyłączania autostartu, blokowania, itd) służą m.in. następujące polecenia programu `systemctl`:

```
systemctl  list-unit-files
systemctl  enable|disable  NAZWA_USLUGI
        ls /etc/systemd/system/*.target.wants/
systemctl  mask|unmask  NAZWA_USLUGI
systemctl  show|list-dependencies  NAZWA_USLUGI
systemctl  cat|edit  NAZWA_USLUGI
```

Poziomy startu, czyli to jakie usługi kiedy mają byćn uruchomione (z wyłączeniem rzeczy wynikłych z zależności - są one uruychamiane automatycznie na podstawie zależności zdefiniowanych w pliku usługi) opisywane są linkami w `/etc/systemd/system/*.target.wants/`

Do zarządzania aktualnym stanem usługi (jej startowania, zatrzymywania itd) a także wyświetlania timerów służy również program `systemctl` z następującymi poleceniami:

```
systemctl  list-units|list-timers
systemctl  status|start|stop|restart|reload  NAZWA_USLUGI
```

Jak wspomniano systemd jest kompatybilny z klasycznymi sktyptami startowymi (z `/etc/rc[0-6].d`, typowo nie dla `/etc/rcS.d`) - automatycznie generuje dla nich pliki konfiguracyjne usług i umieszcza je w `/run/systemd/generator.late/` a linki do usług związanych z danym poziomem uruchomienia w `/run/systemd/generator.late/*.target.wants/`.

systemd umożliwia także tworzenie katalogów, linków itp przy starcie systemu. Np. aby mieć `/tmp` i `/var/log` na tmpfs (przydatne dla systemów np. na kartach SD) możemy utworzyć plik `/etc/tmpfiles.d/on_tmpfs.conf` z następującą zawartością:

```
d  /run/tmp   1777 root root -
L+ /tmp       -    -    -    -  /run/tmp
L+ /var/tmp   -    -    -    -  /run/tmp

d  /run/log   0755 root root -
L+ /var/log   -    -    -    -  /run/log
```

Więcej informacji

- Systemd @ wiki.archlinux.org
- Getting Started With systemd on Debian Jessie
- How To Use Systemctl to Manage Systemd Services and Units
- Can systemd replace monit?

Moduły - ładowanie, konfiguracja, ...

Jądro Linuxa ma zmodularyzowana budowę i wiele sterowników (urządzeń, protokołów, ...) oraz innych funkcji zawartych jest w osobnych modułach. Moduły te mogą być ładowane podczas startu systemu bądź być ładowane dynamicznie lub ręcznie podczas działania systemu. Pliki określające jakie moduły i z jakimi parametrami chcemy załadować podczas startu umieszczane są w katalogu `/etc/modules-load.d`, gdzie w każdej linii nie zaczętej od # pierwszy wyraz to nazwa modułu a wszystko następne to parametry.

Za konfigurację modułów (oprócz wspomnianego pliku z listą ładowanych modułów) odpowiadają również pliki w katalogu `/etc/modprobe.d` (wszystkie pliki w tym katalogu są równoważne). W skrypcie startowym modułów znajduje się polecenie budujące drzewo zależności (depmod), jeżeli zależy nam na przyspieszeniu startu komputera możemy je wyłączyć i uruchamiać ręcznie po zmianie, dodaniu,

... jakiś modułów, bądź modyfikacji ich parametrów w wspomnianych wyżej ktalogach.

Trzeba także zaznaczyć że w nowszych jądrach gdzie wykorzystywany jest udev, większość modułów (domyślnie) ładowana będzie automatycznie - dlatego ważniejsza jest konfiguracja `/etc/modprobe*` niż `/etc/modules-load.d`. Trzeba także zwrócić uwagę na konfigurację `/etc/udev/*`.

Moduły są ładowane na etapie initrd (są to moduły określone w `/usr/share/initramfs-tools/modules*` oraz inne potrzebne do zamontowania rootfs, aby modyfikacje ich dotyczące konfiguracyjne mogły odnieść skutek należy przebudować initrd) lub na etapie normalnego init'a.

Tworzenie plików urządzeń - udev

udev jest mechcanizmem służącym do dynamicznego zarządzania zawartością katalogu `/dev`, jego konfiguracja znajduje się w katalogu `/etc/udev` (osobiście polecam swoje wpisy umieszczać na początku pliku udev.rules, kolejność wykonywania reguł określana jest przez porządek linków w `/etc/udev/rules.d` i jest ona istotna) i umożliwia m.in. (oczywiście to tylko parę przykładów zastosowań konfiguracji udev):

- Określenie numeracji kart sieciowych na podstawie adresów sprzętowych: `KERNEL=="eth*", SYSFS{address}=="mac:adres:wybranej:karty:sieciowej:", NAME="eth0"` (jeżeli komputer posiada dwie karty sieciowe a w konfiguracji udev nie określimy ich kolejności numeracja ich może się zmienić po restarcie - nawet jeżeli używają różnych modułów jako sterowników).
- Określanie własnych nazw urządzeń: np. `BUS=="scsi", SYSFS{model}=="USB Storage-CFC ", NAME{all_partitions}="flash/cf"` określa że wszystkie partycje (tak na zapas) karty CF która może być umieszczona w czytniku USB Mass Device Storage (widzianym jako urządzenie magistrali scsi) o nazwie modelu "USB Storage-CFC " mają mieć nazwy `/dev/flash/cfX`, gdzie X jest numerem partycji.
- Konfigurację dowiązań symbolicznych do urządzeń znajdujących się w katalogu `/dev`: np. `KERNEL=="radio0", SYMLINK+="radio"` nakazuje utworzenie dowiązania symbolicznego `/dev/radio` do `/dev/radio0`.
- Oba powyższe w jednym - np. `BUS=="serio", KERNEL=="mouse*", DRIVER=="psmouse", SYMLINK="input/mouse0", NAME="input/psmouse"` nakazuje myszkę PS/2 podłączyć do `/dev/input/psmouse` oraz utworzyć do niej dowiązanie `/dev/input/mouse0`
- Konfigurować uprawnienia dostępu do urządzeń - poprzez dodanie do wpisu tworzącego urządzenie `OWNER="`*`wlasciciel`*`", GROUP="`*`grupa`*`", MODE="`*`prawa_numerycznie`*`"` lub interesującego nas fragmentu tego wyrażenia.

Przy tworzeniu i testowaniu reguł udev'a pomocna może być komenda udevadm, która umożliwia testowanie reguł dla danego urządzenia (określonego ścieżką względem `/sys`) - np. `udevadm test /devices/platform/ipmi_si.0/ipmi/ipmi0`, uzyskanie informacji o istniejącym urzadzeniu w dev - np. `udevadm info --query=all --name=`*`ipmi0`*, czy też przeładowanie i wykonanie reguł na działającym systemie (w celu np. zmiany nazw interfejsów sieciowych):

```
udevadm control --reload-rules
udevadm trigger --action=add --verbose --subsystem-match=net
```

Udev do identyfikacji urządzeń oprócz czytania informacji z `/sys/` korzysta z różnych pomocniczych programów są to m.in.:

- `/sbin/blkid /dev/sda` - informacje dotyczące filesystemu, (tablicy partycji) - UUID, LABEL, etc
- `/lib/udev/ata_id --export /dev/sda1` - informacje dotyczące urządzenia ATA - MODEL, SERIAL, WWN
- `/lib/udev/scsi_id --export --whitelisted /dev/sda1` - informacje dotyczące urządzenia SCSI - MODEL, SERIAL, WWN
- `/lib/udev/v4l_id /dev/video0` - informacje dotyczące urządzenia V4L

Od pewnego czasu stosowane są dziwne nazwy interfejsów (np. z całym numerem MAC w nazwie), jeżeli chcemy uniknąć takiego zachowania i powrócić do klasycznych nazw interfejsów typu ethX (przydatne np. w bootowalnych nośniach USB) możemy zablokować nowy tryb nazewnictwa poprzez: `ln -s /dev/null /etc/udev/rules.d/80-net-setup-link.rules` i `ln -s /dev/null /etc/udev/rules.d/73-special-net-names.rules`. Można także utworzyć plik `/etc/udev/rules.d/80-net-names` przypisujący nasze nazwy do interfejsów wpisami typu: `SUBSYSTEM=="net", ACTION=="add", ATTR{address}=="aa:bb:cc:dd:ee:ff", NAME="lan1"`

Zazwyczj całkiem dobrym miejscem na umieszczanie swoich wpisów jest `/etc/udev/udev.rules`. Na koniec wspomnę że udev'owi nie zawsze udaje się ładować wszystko autmagicznie i np. aby mieć

`/dev/lp0` należy załadować (np. przez `/etc/modules`) moduł `lp`. Podobnie warto tam dopisać `ipv6`. Podobną rolę jak wpisy udev mogą pełnić w niektórych przypadkach parametry modułów określające z jakim numerem ma występować urządzenie (tak jest np. w modułach v4l) - możemy je podać w konfiguracji modprobe. Z kolei nieskuteczną drogą jest określanie numerów urządzeń poprzez wpisy alias w konfiguracji modprobe.

Licencja