System bazowy

dr inż. Krzysztof Konopko

e-mail: k.konopko@pb.edu.pl

System Bazowy

Program wykładu:

- Główny system plików
- Minimalistyczny system z zastosowaniem Busyboksa.
- Uruchamianie systemu.

Główny systemy plików

- Podczas instalacji Linuksa tworzona jest struktura folderów na dysku lub dyskach zainstalowanych w komputerze.
- Podstawowa struktura katalogów jest dość ściśle określona wg standardu FHS (z ang. Filesystem Hierarchy System).
- Wszystkie pliki i katalogi znajdują się w katalogu głównym "/", nawet jeśli są przechowywane na innych dyskach fizycznych.
- Systemy plików są montowane w określonym miejscu w hierarchii katalogów
  - kiedy system plików jest zamontowany w katalogu (tzw. punkt montowania), jego zawartość odzwierciedla zawartość urządzenia,
  - kiedy system plików jest odmontowany, punkt montowania jest zwalniany.
- Takie podejście pozwala aplikacjom na dostęp do plików i katalogów, bez względu na ich prawdziwe położenie.

Główny systemy plików

- Szczególny system plików jest zamontowany w katalogu głównym hierarchii, zidentyfikowanym jako /.
- Ten system plików jest nazywany główny system plików (z ang. *root filesystem*).
- Główny system plików jest pierwszym montowanym systemem plików i nie może być zamontowany w normalnym programem `mount`
- Jest on montowany bezpośrednio przez jądro, zgodnie z następującą opcją `root = .`

Główny systemy plików

Główny system plików może być montowany z wielu różnych urządzeń w tym:

- z partycji dysku twardego, czy z dysku USB,
  `root=/dev/sdXY`
- z karty SD,
  `root=/dev/mmcblkXpY`
- z pamięci flash (np. NAND),
  `root=/dev/mtdblockX`

Montowanie głównego systemy plików przez sieć:

Główny system plików może być montowany także z sieci przy użyciu protokołu NFS. W systemach wbudowanych umożliwia to:

- szybszą aktualizację w systemie plików, bez konieczności restartowania systemu (znacznie szybszą niż przez port szeregowy),
- przechowywanie dużego głównego systemu plików, nawet jeśli nie ma wystarczającej ilości pamięci wewnętrznej albo zewnętrznej w systemie docelowym (np.: można posiadać natywne narzędzia deweloperskie, zamiast posługiwać się cross-kompilacją).

Montowanie głównego systemy bezpośrednio w pamięci:

Możliwe jest również, aby główny system plików był zintegrowany z jądrem Linuksa (jest wtedy ładowany łącznie z jądrem do pamięci).

Mechanizm ten nazywa się `initramfs` i umożliwia:

- integrację skompresowanego archiwum (w formacie `.cpio`) zawierającego system plików z jądrem Linuksa,
- załadowanie takiego archiwum przez bootloader.

Realizacja takiego systemu jest użyteczna w przypadku:

- szybkiego uruchamiania bardzo małych systemach plików (ponieważ system plików jest całkowicie załadowany do pamięci w czasie startu systemu, uruchamianie aplikacji jest bardzo szybkie).

Struktura katalogów systemu Linux

Katalog	Opis
/bin/	Podstawowe pliki wykonywalne dostępne dla wszystkich użytkowników.
/boot/	Pliki programu rozruchowego.
/dev/	Pliki urządzeń.
/etc/	Pliki konfiguracyjne.
/etc/opt/	Pliki konfiguracyjne dla katalogu /opt/.
/etc/X11/	Pliki konfiguracyjne X w wersji 11.
/home/	Katalogi domowe użytkowników.
/lib/	Biblioteki dla programów z katalogów /bin/ i /sbin/.
/mnt/	Punkt montowania innych niż natywny systemów plików.
/media/	Punkty montowań dla nośników wymienialnych (pojawiło się w FHS-2.3).
/opt/	Opcjonalne aplikacje.
/proc/	Wirtualny system plików informujący o stanie systemu i poszczególnych procesów.
/root/	Katalog domowy użytkownika root.
/sbin/	Pliki wykonywalne do zarządzania systemem.
/tmp/	Pliki tymczasowe, których stan nie jest gwarantowany po zamknięciu systemu.

Struktura katalogów systemu Linux

Katalog	Opis
/usr/	Drugorzędowa hierarchia dla danych (ang. *Unix system resources*), dane tylko do odczytu.
/usr/bin/	Jak w hierarchii pierwszorzędowej, ale nie wymagane do uruchomienia, czy naprawy systemu.
/usr/include/	Standardowe pliki nagłówkowe (ang. *include files*).
/usr/lib/	Jak w hierarchii pierwszorzędowej.
/usr/sbin/	Jak w hierarchii pierwszorzędowej, ale nie wymagane do uruchomienia, czy naprawy systemu (np. demony różnych usług sieciowych).
/usr/share/	Dane niezależne od architektury, a więc między nimi współdzielone (ang. *shared*).
/usr/src/	Kod źródłowy (np. źródła kernela z jego nagłówkami)
/usr/local/	Trzeciorzędowa hierarchia danych lokalnych.
/var/	Pliki często ulegające zmianom takie jak: logi, bazy danych, tymczasowe pliki e-mail.
/var/lock/	Pliki blokady zasobów będących w użyciu.
/var/log/	Logi różnych aplikacji.
/var/run/	Informacje o działaniu systemu od ostatniego jego uruchomienia (np. aktualnie zalogowani użytkownicy, uruchomione demony).
/var/spool/	Miejsce przechowania oczekujących zadań (np. kolejki wydruku, nieprzeczytane e-maile).
/var/tmp/	Pliki tymczasowe, które w przeciwieństwie do /tmp powinny być zachowywane przy zamknięciu systemu.

Urządzenia plikowe:

Jedną z podstawowych roli jądra jest umożliwienie aplikacjom na dostęp do urządzeń sprzętowych. Programy działające w przestrzeni użytkownika odwołują się do dwóch typów urządzeń plikowych:

- urządzenia znakowe,
- urządzenia blokowe.

Urządzenia są identyfikowane przez jądro za pomocą trzech danych:

- typu (b - blokowe i c - znakowe),
- numer główny (major) - określa sterownik (lub klasę urządzenia - na przykład karta dźwiękowa),
- numer poboczny (minor) - określa numer funkcji sterownika (zazwyczaj numer kolejnego urządzenia, kolejnej partycji itp.) - zależy od implementacji konkretnego sterownika.

Tworzenie urządzeń plikowych:

Do ręcznego tworzenia urządzeń plikowych wykorzystuje się polecenie:

```
mknod /dev/<device> [c|b] major minor
```

Możliwy jest też mechanizm dynamicznej alokacji plików urządzeń z zastosowaniem :

- `udev` – narzędzie używane na stacjach roboczych i serwerach linuksowych
- `mdev` – dużo prostsze i mniejsze narzędzie chętnie stosowane w systemach wbudowanych.

Pseudopliki

Pseudo system plików /proc istnieje od początku Linux i pozwala na:

- dostarczanie statystyk na temat uruchomionych procesów w systemie,
- konfigurację różnych parametrów systemowych (/proc/sys) np.: zarządzanie procesami, zarządzanie pamięcią.

Ważne pliki które znajdziemy w /proc:

`/proc/cpuinfo`	Informacje o procesorze
`/proc/meminfo`	Użycie pamięci
`/proc/version`	Informacje o wersji i kompilatorze użytym do budowy jądra
`/proc/cmdline`	Linia poleceń jądra
`/proc/<PID>/environ`	Zmienne środowiskowe danego procesu
`/proc/<PID>/cmdline`	Polecenie uruchamiające dany proces
`/proc/<PID>/status`	Informacje na temat pracy procesu (jego stan, ilość pamięci, liczba wątków itp.). Ten plik jest głównym źródłem informacji dla narzędzia **ps**
`/proc/devices`	Informacje o zarejestrowanych urządzeniach blokowych i znakowych oraz numerach głównych im przydzielonych

Pseudopliki

Pseudo system plików /sys zawiera katalogi i pliki odwzorowujące strukturę wewnętrznego modelu urządzeń w jądrze. Znajdują się w nim informacje o magistralach, urządzeniach, sterownikach oraz ich wzajemnych połączeniach .

/sys zawiera między innymi następujące podkatalogi:

/sys/fs - załadowane systemy plików - używane do kontrolowania i konfiguracji zachowania sterowników

/sys/devices - urządzenia rozpoznane przez jądro podzielone na magistrale, do których są podłączone

/sys/dev - zarejestrowane w jądrze pliki urządzeń (podzielone na znakowe i blokowe)

/sys/bus - magistrale wykryte przez jądro; każda magistrala ma dwa podkatalogi: **devices** - lista urządzeń wykrytych na magistrali, **drivers** - lista zarejestrowanych sterowników; w katalogu danego sterownika są dowiązania do urządzeń, które aktualnie obsługuje i vice versa - każde urządzenie ma dowiązanie do sterownika,

/sys/class - zawiera podkatalogi dla danego typu urządzeń **(input, mmc, printer** itp.)

/sys/firmware - znajdujące się tu pliki obsługują firmware niektórych urządzeń

/sys/kernel – informacje z jądra oraz plików konfigurujących ich działanie,

/sys/power – kontrola stanu zasilania niektórych urządzeń,

/sys/module – informacje o wszystkich załadowanych modułach i ich parametrach.

Minimalistyczny system

W celu podjęcia pracy, system Linux potrzebuje przynajmniej kilku aplikacji:

- aplikacji `init`, która jest pierwszą aplikacją uruchamianą przez jądro po zamontowaniu głównego systemu plików,
  - jądro próbuje uruchomić: `/sbin/init`, `/bin/init`, `/etc/init` i `/bin/sh`,
  - w przypadku initramfs odszukiwana i uruchamiane jest tylko `/init`

  Aplikacja init jest odpowiedzialny za uruchomienie wszystkich aplikacji i usług w przestrzeni użytkownika
- powłoki umożliwiającej użytkownikom interakcję z systemem,
- podstawowych aplikacji np. do pracy z plikami,

Busybox

To podstawowe uniksowych narzędzia uruchamiane z wiersza poleceń zintegrowane w jeden projekt. Zaprojektowany z myślą o systemach wbudowanych: bardzo konfigurowalny, bez zbędnych funkcji.

Wszystkie narzędzia są zebrane w jednym pliku wykonywalnym:

/bin/busybox

Aplikacji zintegrowanej Busyboksie tworzone są poprzez dowiązania symboliczne do pliku /bin/busybox

Cały dobrze wyposażony system stosujący Busyboksa ma mnie niż 500 KB (statycznie skompilowany z uClibc) lub mniej niż 1 MB (statycznie skompilowany z glibc).

Komendy dostępne w Busyboksie

[, [[, addgroup, adduser, adjtimex, ar, arp, arping, ash, awk, basename, bbconfig, bbsh, brctl, bunzip2, busybox, bzcat, bzip2, cal, cat, catv, chat, chattr, chcon, chgrp, chmod, chown, chpasswd, chpst, chroot, chrt, chvt, cksum, clear, cmp, comm, cp, cpio, crond, crontab, cryptpw, cttyhack, cut, date, dc, dd, deallocvt, delgroup, deluser, depmod, devfsd, df, dhcprelay, diff, dirname, dmesg, dnsd, dos2unix, dpkg, dpkg_deb, du, dumpkmap, dumpleases, e2fsck, echo, ed, egrep, eject, env, envdir, envuidgid, ether_wake, expand, expr, fakeidentd, false, fbset, fbsplash, fdflush, fdformat, fdisk, fetchmail, fgrep, find, findfs, fold, free, freeramdisk, fsck, fsck_minix, ftpget, ftpput, fuser, getenforce, getopt, getsebool, getty, grep, gunzip, gzip, halt, hd, hdparm, head, hexdump, hostid, hostname, httpd, hush, hwclock, id, ifconfig, ifdown, ifenslave, ifup, inetd, init, inotifyd, insmod, install, ip, ipaddr, ipcalc, ipcrm, ipcs, iplink, iproute, iprule, iptunnel, kbd_mode, kill, killall, killall5, klogd, lash, last, length, less, linux32, linux64, linuxrc, ln, load_policy, loadfont, loadkmap, logger, login, logname, logread, losetup, lpd, lpq, lpr, ls, lsattr, lsmod, lzmacat, makedevs, man, matchpathcon, md5sum, mdev, mesg, microcom, mkdir, mke2fs, mkfifo, mkfs_minix, mknod, mkswap, mktemp, modprobe, more, mount, mountpoint, msh, mt, mv, nameif, nc, netstat, nice, nmeter, nohup, nslookup, od, openvt, parse, passwd, patch, pgrep, pidof, ping, ping6, pipe_progress, pivot_root, pkill, poweroff, printenv, printf, ps, pscan, pwd, raidautorun, rdate, rdev, readahead, readlink, readprofile, realpath, reboot, renice, reset, resize, restorecon, rm, rmdir, rmmod, route, rpm, rpm2cpio, rtcwake, run_parts, runcon, runlevel, runsv, runsvdir, rx, script, sed, selinuxenabled, sendmail, seq, sestatus, setarch, setconsole, setenforce, setfiles, setfont, setkeycodes, setlogcons, setsebool, setsid, setuidgid, sh, sha1sum, showkey, slattach, sleep, softlimit, sort, split, start_stop_daemon, stat, strings, stty, su, sulogin, sum, sv, svlogd, swapoff, swapon, switch_root, sync, sysctl, syslogd, tac, tail, tar, taskset, tcpsvd, tee, telnet, telnetd, test, tftp, tftpd, time, top, touch, tr, traceroute, true, tty, ttysize, tune2fs, udhcpc, udhcpd, udpsvd, umount, uname, uncompress, unexpand, uniq, unix2dos, unlzma, unzip, uptime, usleep, uudecode, uuencode, vconfig, vi, vlock, watch, watchdog, wc, wget, which, who, whoami, xargs, yes, zcat, zcip

Minimalistyczny system - jądro

- Instalacja i konfiguracja toolchaina (np. dostępnego w Ubuntu)

```
sudo apt-get install gcc-arm-linux-gnueabihf
```

- Pobranie plików źródłowych jądra:

```
git clone --depth=1 https://github.com/raspberrypi/linux
```

- Skopiowanie pliku z konfiguracją jądra dla raspberry pi 2 (skorzystamy z toolchaina dostarczonego przez Ubuntu)

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- \
   bcm2709_defconfig
```

- kompilacja jądra modułów i systemu Device Tree

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- zImage \
   modules dtbs
```

Minimalistyczny system podstawowe narzędzia Busybox

- Pobranie plików źródłowych Busyboksa:

```
wget –c http://www.busybox.net/downloads/busybox-
   1.27.2.tar.bz2
```

- Wypakowujemy archiwum

```
tar –xf busybox- 1.27.2.tar.bz2
```

- Konfigurujemy

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- defconfig
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- menuconfig
```

- Kompilujemy

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- \
```

- Instalujemy

```
make install
```

- Przegrywamy utworzony system plików do naszego systemu docelowego

```
cp _install/{bin,sbin,usr} /home/user/target
```

Tworzymy urządzenia plikowe i skrypty startowe

- Tworzymy podstawowe drzewo katalogów:

```
mkdir dev /dev/pts etc etc/init.d proc sys root
```

- Tworzymy podstawowe urządzenia plikowe niezbędne przy starcie systemu

```
sudo mknod dev/console c 5 1
sudo mknod ded/null c 1 3
```

- Tworzymy plik `/init`, który jako pierwszy będzie uruchomiony przez system

```
ln –s bin/busybox init
```

- Tworzymy plik konfiguracyjny dla programu `init`, który znajduje się w pliku /etc/inittab

```
null::sysinit:/etc/init.d/rcS
ttyAMA0::askfirst:/sbin/getty 115200 ttyAMA0
tty1::askfirst:/sbin/getty 38400 tty1
```

Tworzymy urządzenia plikowe i skrypty startowe

- Tworzymy skrypt startowy `/etc/init.d/rcS`:

```
#!/bin/ash
/bin/mount –t proc proc /proc
/bin/mount –t sysfs sys /sys
/bin/mount –t devpts devpts /dev/pts
echo „sswb” > /proc/sys/kernel/hostname
echo /sbin/mdev > /proc/sys/kernel/hotplug
/sbin/ip link set lo up
/sbin/mdev -s
```

Tworzymy urządzenia plikowe i skrypty startowe

- Tworzymy użytkownika (tylko konto administratora – root), w tym celu do pliku /etc/passwd wpisujemy:

```
root:x:0:0:root:/:/bin/ash
```

- jeszcze potrzebne hasło, generujemy je poleceniem:

```
echo „haslo” | openssl passwd -1 –stdin
```

- i wstawiamy je do pliku `/etc/shadow`

```
root:_tu_nasze_haslo:16000:0:999999:7:::
```

I to koniec mamy gotowy system teraz trzeba go jeszcze uruchomić. :)

Bootloader

Bootloader jest fragment kodu odpowiedzialnym za:

- podstawową inicjalizację sprzętu,
- załadowanie aplikacji, zazwyczaj jądra systemu operacyjnego, z pamięci flash, z sieci, lub z innego typu pamięci nieulotnej,
- często też rozpakowania skompresowanego jądra,
- uruchomienie jądra.

Często też oprócz tych podstawowych funkcji, bootloadery udostępniają też powłokę umożliwiającą wprowadzanie komend wykorzystywanych np. do ładowania danych z pamięci masowej lub sieci, kontroli pamięci, diagnostyki sprzętu i testowania, itp..

Uruchamianie systemu Raspberry Pi

W przypadku Raspberry Pi 2 system składa się z:

- układu SoC Broadcom BCM2836 (CPU + GPU + DSP + SDRAM)
  - 900 MHz quad-core ARM Cortex-A7,
  - 1 GB SDRAM (współdzielona z GPU),
  - Broadcom VideoCore IV @ 250 MHz.

Start tych układów przebiega w następujących etapach:

- Zaraz po zasileniu RPi uruchamiany jest GPU; na ten moment CPU i RAM pozostają jeszcze nieaktywne (w stanie „reset”).
- Wyspecjalizowany, dodatkowy procesor wczytuje z ROM i wykonuje bootloadera pierwszego etapu (ang. 1st stage bootloader). Kod ten jest wpisywany do ROMu w czasie produkcji układu.
- Bootloader pierwszego etapu montuje partycję startową FAT z karty SD, wczytuje zawartość pliku „bootcode.bin” do pamięci podręcznej L2 procesora GPU (ang. L2 cache) i wykonuje go (CPU i RAM nadal w stanie resetu).

Uruchamianie systemu Raspberry Pi

Dalsze kroki uruchamiania systemu:

- Kod „bootcode.bin” to tzw. bootloader drugiego etapu (ang. 2nd stage bootloader). Uruchamia pamięć RAM i wczytuje do GPU z karty SD bootloader trzeciego etapu (ang. 3rd stage bootloader), zapisany w pliku „start.elf”. Kod „start.elf” jest jednocześnie systemem operacyjnym GPU (ang. firmware), który pozostanie w jego pamięci i pomaga Linuksowi dostawać się do jego zasobów. GPU jest uruchamiane.
- GPU wczytuje plik konfiguracji RPi „config.txt”, zawierający m.in. ustawienia częstotliwości poszczególnych elementów SoC (np. CPU, GPU) oraz podział pamięci RAM między CPU i GPU (plik „fixup.dat”).
- GPU wczytuje plik konfiguracyjny „cmdline.txt”, zawierający parametry do startu jądra Linuksowego.
- GPU wczytuje do RAM „kernel.img” obraz jądra Linuksa.
- GPU zmienia stan CPU na aktywny.
- CPU rozpoczyna wykonywanie kodu jądra linuksowego.

Uruchamianie systemu Raspberry Pi

- Przygotowujemy naszą kartę tworząc na niej dwie partycje, jedną mniejszą około 50 MB, w formacie FAT (z tej partycji uruchomi się nasze jądro) oraz większą w systemie ext4 (tutaj znajdzie się nasz główny system plików).
- przegrywamy na pierwszą partycję pliki: bootcode.bin, fixup.dat, start.elf (można je pobrać komendą

  `git clone git://github.com/raspberrypi/firmware.git`)
- tworzymy pliki config.txt (http://elinux.org/RPiconfig) i command.txt
- przegrywamy jądro oraz urządzenia Device Tree

```
sudo scripts/mkknlimg arch/arm/boot/zImage \
mnt/fat/kernel.img
sudo cp arch/arm/boot/dts/*.dtb mnt/fat/
sudo cp arch/arm/boot/dts/overlays/*.dtb* \
mnt/fat/overlays/
```

- przegrywamy nasz główny system plików na partycję `/mnt/ext4/`
- odmontowujemy kartę pamięci i uruchamiamy nasz system. :)

Dziękuję za uwagę

Zapraszam
za tydzień :)