Linux device drivers

Jernej Vičič

February 28, 2018

Overview

What is a driver?

- It is a special kind of computer program.
- The main purpose is to control external devices..
- It executes privileged instructions.
- It should be inserted into the kernel.
- Two interfaces - kernel, hardware.
- The basic shape of the program is OS-specific.

GNU/Linux device drivers

- It is basically a set of service functions.
- This set is conceptually an object.
- In C the same concept is depicted by `struct`
- Specifically: `struct file_operations { ...; };`
- D+The definition is in header file: `/usr/src/linux/include/linux/fs.h`
- or: `/usr/src/linux-headers-(uname-r)-generic-pae/include/linux`

Types of drivers

- Character diver
  - The device processes a few bytes at a time.
    (examples: keyboard, printer, modem)
- Block driver:
  - The device processes groups of bytes (blocks) at a time.
    (examples: hard disk, CD-ROM)

More GNU/Linux drivers:

- network driver,
- mouse driver,
- SCSI driver,
- USB driver,
- video driver,
- ”hot-swap” driver,
- other drivers.

The devices are represented as files

- Well documented Application Programming Interfaces (API)
  `open()`, `seek()`, `read()`, `write()`, `close()`, ...
- normally needs a special file in a special directory
  (for devices it is usually `/dev`)

A driver

- Char and block devices:
  - Use of major numbers (major-number) for driver identification;
  - Use of minor numbers (minor-number) to identify devices controlled by the same driver.
- Kernel needs a unique driver name.
- Users need a device (file) as an interface to the driver functions.

How to make a driver

- Present your goals;
- Make a plan to reach your goals;
- Prototype you ideas;
- Debug your prototype if necessary ALMOST ALWAYS:(;
- Build your final version of the driver;
- Document your work for future uses.

Duality of Linux

- User space
- Kernel space

Duality of Linux 2

Various layers within Linux, also showing separation between the userland and kernel space

User mode	**User applications**	For example, bash, LibreOffice, GIMP, Blender, 0 A.D., Mozilla Firefox, etc.			
	Low-level system components:	**System daemons**: *systemd*, *runit*, *logind*, *networkd*, *PulseAudio*, ...	**Windowing system**: *X11*, *Wayland*, *SurfaceFlinger* (*Android*)	**Other libraries:** *GTK+*, *Qt*, *EFL*, *SDL*, *SFML*, *FLTK*, *GNUstep*, etc.	**Graphics**: *Mesa*, *AMD Catalyst*, ...
	C standard library	`open()`, `exec()`, `sbrk()`, `socket()`, `fopen()`, `calloc()`, ... (up to 2000 subroutines) *glibc* aims to be POSIX/SUS-compatible, *uClibc* targets embedded systems, *bionic* written for Android, etc.			
Kernel mode	**Linux kernel**	`stat`, `splice`, `dup`, `read`, `open`, `ioctl`, `write`, `mmap`, `close`, `exit`, etc. (about 380 system calls) The Linux kernel System Call Interface (SCI, aims to be POSIX/SUS-compatible)			
		Process scheduling subsystem	IPC subsystem	Memory management subsystem	Virtual files subsystem	Network subsystem
		Other components: ALSA, DRI, evdev, LVM, device mapper, Linux Network Scheduler, Netfilter Linux Security Modules: *SELinux*, *TOMOYO*, *AppArmor*, *Smack*			
Hardware (CPU, main memory, data storage devices, etc.)					

Figure: Linux duality.

User space

- Virtually unlimited (OK limited by the size od the (virtual) memory);
- Used by most applications;
- Slower;
- Safer.

User space

- Limited size;
- Seldom used by applications (only on special requests);
- Special modules that need unobstructed attention (cannot be stopped);
- Faster;
- Less safe.

Kernel modules

- Modules are parts of code that are loaded into system kernel.
- Enable extension of kernel functionality.
- We do not need a restart of the whole system to load a module into kernel,
- Example: a driver enables the system to access a selected device.
- Modules enable creation of smaller kernels (otherwise all functionality would be always available eating space).
- Modules enable support for future devices.

What is a driver?

- A driver represents the middle layer between the operating system and a device - mostly hardware.
- Drivers enable a standardised access to any device.

GNU/Linux kernels

Change every 60 - 80 days (new version) from 2.6 onwards.

- 1999 – Linux 2.2.0,
- 1999 – 2.2.13 IBM mainframe patch,
- 2001 – Linux 2.4.0,
- 2003 – Linux 2.6.0.
- 2004 – Linux 2.6.? new kernel every 2-3 months,
- 2011 – Linux 3.0 the number is not a big leap, just 3rd decade of linuxe kernel
- 2015 – Linux 4.0
- 2016 – Linux 4.4

- 2016 – Linux 4.5 item 2016 – Linux$_4$.15*latest*

Kernel 2.4

- Older kernel version GNU/Linux.
- Still in some OLD systems.
- The compilation was different, see folder kernel2.4!

Kernel 2.6

- Prinaša popolnoma nov način prevajanja jedrnih modulov.
- Oglejmo si primer v mapi kernel2.6!

Kernel 4.?

- Newest GNU/Linux kernel.
- Just evolution, no revolution 2.6.

Kernel modules usage

- Linking with devices;
- Inserting modules into the kernel;
- Deleting modules from kernel;
- Using modules.

Linking with the devices

- On UNIX systems user applications access the devices through device files.
- All devices are "exposed" on /dev.
- Two numbers define each device (major, minor) numbers.
- Major is used by the kernel to represent the link between the device and the file (basically the driver).
- Minor is used by the driver to identify a device.

Linking with the devices 2

- Prepare a file on /dev.
- `mknod /dev/znakec c 60 0`
- "c" means this will be a character device
- 60 is the major number
- 0 minor number

Inserting modules into kernel

- insmod
- rmmod
- modprobe [-l]
- modprobe -l lists all modules
- modules are listed in: /lib/modules/$(uname-r)
- lsmod lists all modules that are installed in the kernel at the precise moment.