

Kernel and Drivers

Kernel

- Monolithic
 - All modules run in kernel space
 - Interacts quickly with devices
 - Large and uses RAM
 - Less stable
 - Microkernels can be more stable
 - Runs minimum resources necessary
 - Small kernel space, large user space
 - Consumes less RAM, but worse performance overall
 - LT said “microkernels are stupid”.
- Open Source
- Modular
- Main component of the OS
- Provides Interface between hardware and processes
- Provides core services for user processes
- Manages resource sharing, memory, processed, device drivers
- Millions of lines of code of mostly C and some assembly
- Provides abstract high-level interface
- First component to be loaded at boot
- Occupies RAM memory space

Basic Features

- Management and abstraction of HW devices
- Processes and threads (and ways to communicate among them)
- Management of memory (Virtual memory and memory-space protection)
- I/O facilities (filesystems, NW interfaces, serial interfaces, etc.)
- Housekeeping functions (startup, shutdown, timers, multitasking, etc.)

What sysadmins do:

- Change parameters that are accessible from user space.
- Install/uninstall/update device drivers.
- Occasionally you need to recompile the kernel after modifications.

Versioning

- Kernels are under constant development
- Some kernels have extended support, others do not.
- Some older kernels are more stable, but some newer ones have brand new features.
- Kernel and distribution versions develop separately!

uname -r

- Versions are names using semantic versioning: major version, minor version, patch level.

Given a version number MAJOR.MINOR.PATCH, increment the:

1. MAJOR version when you make incompatible API changes,
 2. MINOR version when you add functionality in a backwards compatible manner,
 3. PATCH version when you make backwards compatible bug fixes.
- How do you install a new kernel - from kernel source tree.
 - Should always have two kernels – one is a backup.
 - The safest way to upgrade – to install newer distro.

Device Driver

- Part of the kernel, not user processes.
- Usually loadable modules loaded into the kernel.
- Software that handles or manages controllers for attached hardware.
 - Interface between OS and devices
- Provides an abstraction layer between hardware and kernel's programming interface.
- Shared libraries of privileged, memory resident, low level HW handling routines.

O'Reilly: They are distinct "black boxes" that make a particular piece of hardware respond to a well-defined internal programming interface; they hide completely the details of how the device works. User activities are performed by means of a set of standardized calls that are independent of the specific driver; mapping those calls to device-specific operations that act on real hardware is then the role of the device driver.

- DD are developed rapidly to match development of new hardware.
 - Still compatibility issues
- System specific.
- Are not always compatible with other Unix/Linux systems.
- Vendors are trying to keep up.
- Before installing device – make sure drivers are available and you have them installed.
- Some drivers are needed during boot time, others (USB) are not detected till later on.

USB Devices

- Incorporates plug-and-play features.
- De-facto standard interface technology for peripheral devices.
 - (In the past there were many cable types and buses)
 - HDDs, SSDs, cameras, smartphone/tablets, printers/scanners, kb/mice, microphones/webcams, game controllers.
- Not reliable for streaming audio/video.

Wireless Devices

- Require antennas and agreement on frequencies
- Wifi, bluetooth, NFC

SATA

- Bus interface standard for attaching storage devices to traditional computers
- Replaced IDE in PCs.
- High capacity and low price
- Serial interface (unlike IDE which is parallel)
- Used for consumer mostly, not enterprise.

SCSI

- Mostly for peripheral devices, common for storage
- Parallel interfaces
- Used mostly in enterprise, longer cable.
- Can daisy chain
- Much faster (up to 24Gb/s)
- More reliable.

Where are devices located?

- /proc – lists info reported by kernel at runtime
 - /proc/devices - lists all character/block devices
- /sys – creates hierarchical view of all devices
 - /sys/devices - has files that expose device details
- /dev – device driver files
 - /dev/mapper - includes Lvs, encrypted devices, etc.
- /etc – config files for many components

2 Groups of Devices

- Hotpluggable – can be physically added or removed without a reboot
 - Detected by system when plugged-in
 - Most modern distros support hotpluggable devices for many bus types
 - USB, FireWire, PCIe
- Coldpluggable – require a reboot.
 - RAM, CPU, internal storage.
 - Some require complete power-off.

udev

- Device manager that automatically detects and configured HW devices
- Created every time machine boots and dynamically generates names for devices.
 - Names not always predictable

- This is why we refer to devices by UUID in /etc/fstab instead of sda, etc.
- Can encourage predictability by establishing rules in /etc/udev/rules.d
- Function of systemd
- Initializes during boot
- Handles both hotpluggable and coldpluggable devices
 - Hotpluggable modules loaded dynamically at runtime
 - Coldpluggable modules loaded when system is rebooted
- Can create rules in /etc/udev/rules.d
 - You can get info about device by running command
sudo udevadm info /dev/sda

Device Files and Device Drivers

- /dev directory contains “device files” that can be accessed through both kernel and user space
- Most non-nw devices have 1 or more device files.
- All devices have major and minor numbers associated with them.
- Device file references are mapped to drivers using these numbers.
- Major number
 - Ids the driver with which file is associated – type of device.
- Minor number
 - Ids an instance of a given device type – unit number.
- Check documentation for the driver for major and minor.
- Two types of Device Drivers:
 - Block device drivers – read or written one block at a time (512B).
 - Character device drivers – read or written one byte at a time.
 - Some devices could act as both, Linux uses them as block devices (tapes and disks).
- Pseudo-devices – phantom device driver that controls no actual device
 - Pseudo-tty (PTY) simulates serial port.
 - /dev/zero – accepts and discards input, output is continuous zero bytes.
 - /dev/null – black hole that discards everything written to it, produces no output.
 - /dev/urandom – pseudo-random number generator.
- Network interfaces are devices, but have no device files in /dev
- Device files are created automatically.
- Occasionally you need to create device files by hand.
filename is the device to be created, type (c or b).

mknod *filename type major minor*

How to monitor devices

- lsblk, lsusb, lspci, ls /dev/sd*, dmesg (msg in kernel ring buffer from device drivers)

Device File Management

- Automated in Linux and FreeBSD
- Device file created automatically when new device is detected.
- Files are removed when device is unplugged or goes away.
- Removable storage media can be automounted as a filesystem.
- User-space daemon **udev** takes care of advanced procedures.
- **udev** gets its raw data from device information repository is **sysfs**

Common Hardware Issues

- Common Sources of problems:
- Missing or misconfigured drivers
- Incompatible user space software?

sysfs Repository

- Added to kernel at version 2.6
- Virtual, in-memory filesystem implemented by kernel
- Provides detailed and well-organized information about available devices, their configs, and state
- Info accessible both from kernel and user space.

Directory	What it contains
block	Information about block devices such as hard disks
bus	Buses known to the kernel: PCI-E, SCSI, USB, and others
class	A tree organized by functional types of devices ^a
dev	Device information split between character and block devices
devices	An ancestrally correct representation of all discovered devices
firmware	Interfaces to platform-specific subsystems such as ACPI
fs	A directory for some, but not all, filesystems known to the kernel
kernel	Kernel internals such as cache and virtual memory status
module	Dynamic modules loaded by the kernel
power	A few details about the system's power state; mostly unused

a. For example, sound and graphic cards, input devices, and network interfaces

- Command to query device information, trigger events, and control udev daemon, monitors udev and kernel events:
udevadm command to show all attributes for the device **sdb**:

udevadm info -a -n sdb

info – prints device-specific info

control – starts and stops udevd or forces to reload rules files

monitor – displays events as they occur

other commands: *trigger*, *settle*, *test*

- udevd uses rules: */etc/udev/rules.d* (local) and */lib/udev/rules.d* (default)
- Config file in */etc/udev/udev.conf*

Kernel Configuration

3 ways:

- Modify tunable (dynamic) kernel configuration parameters.
 - Use interface in */proc*
 - Can view and set options through files in */proc/sys* (kernel backdoor)
 - Some files are not writable.
 - *proc/sys/fs/file-max* - displays max number of open files (per-boot)
 - permanent changes are done with *sysctl* in */etc/sysctl.conf* that is read at boot time.
- Load new drivers and modules into existing kernel on the fly.

DD are distributed in 1 of 3 forms:

 - A patch against a specific kernel version.
`cd kernel_directory ; patch -p1 < patch file`
 - Loadable kernel module.
 - Script or package that installs driver. (most common)
- Build kernel by compiling source code.
 - Only when necessary, and ...
 - *When productivity gains you expect to get exceed the effort and lost time required for installation.*
- Kernel source files live in */usr/src/kernel_version*
- Do not edit the configuration file directly!
- Use Linux make targets:
 - Use graphical interface **make xconfig** (KDE)
 - Use graphical interface **make gconfig** (GNOME)
 - Ubuntu ships with both Unity and GNOME desktop environment.
 - Use terminal based **make menuconfig**, if not GNOME or KDE
 - Terminal based **make config** requires answering questions
- Build the binary
 - Have to be in kernel source directory
 - Run `make xconfig`, `gconfig`, `menuconfig`, based on what you have
 - Run `make clean` - not a must, but a good idea
 - Run `make`
 - Run `make modules_install`

- Run make install
- May have to deal with grub config file. Grub updater updates the list of available kernels and adds them to the menu.

Linux Loadable Modules (LKM)

- LM implementation is OS dependent.
- LKM support allows a device driver r any other kernel component) to be added or removed to kernel while it is running.
- Makes device driver installation easier.
- Drivers not loaded unless needed -> makes a smaller kernel.
- Always a chance to cause kernel panic.
- Stored in `/lib/modules/version`, where version is kernel version.
- Useful commands:
 - lsmod** -> lists modules
 - modprobe** -> installing modules - wrapper around **insmod** command
 - modprobe -r** -> removing modules

Kernel Errors

- Can happen on any properly configured system.
- Many causes, faulty hw being the most common: mem or hd failures.
- Can be caused by bugs in kernel implementation (rare)
- Device Drivers come from different sources.
- Kernel panic is a structured event with a lot of checks.
- Kernel failure types:
 - *Soft lockup* – system in kernel mode, preventing user-mode programs from running. Process is denied CPU cycles. Interrupts from nw and kb are still being serviced.
 - *Hard lockup* – same as above, but processor interrupts are unserved. Detected quickly.
 - *Panics* – computer will reboot.
 - Linux “oops” – anomalies that can often be repaired.