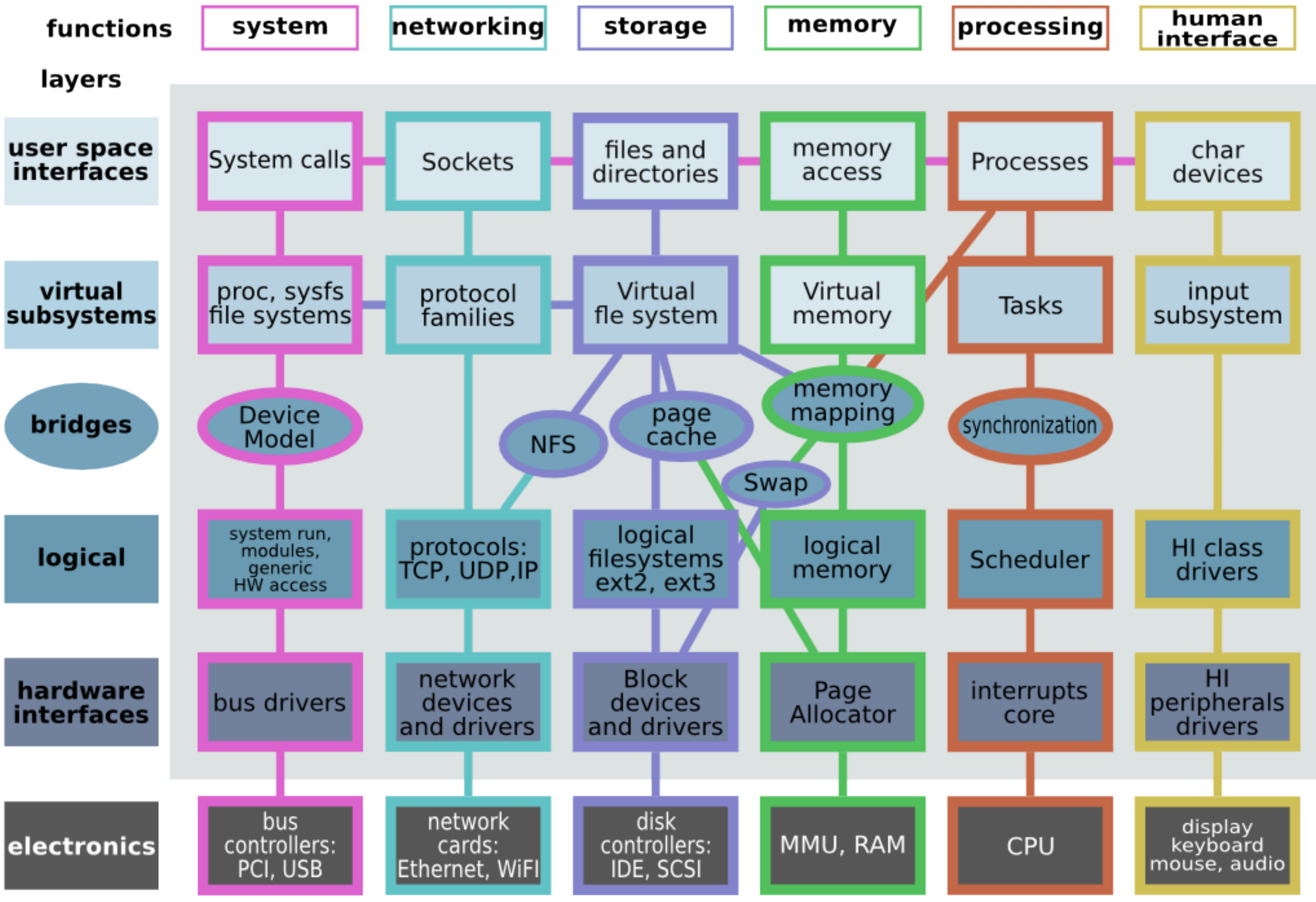
Advanced Linux System Administration

Topic 10. The Linux Kernel

Pablo Abad Fidalgo
José Ángel Herrero Velasco

Departamento de Ingeniería Informática y Electrónica

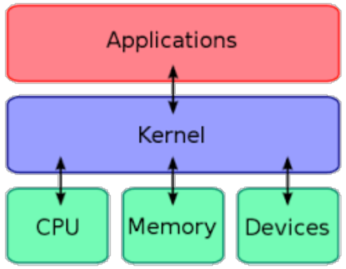
The Linux Kernel



Index

- Introduction:
  - Kernel types.
- Static Reconfiguration:
  - Configuration.
  - Compilation.
  - Install.
- Dynamic Reconfiguration:
  - /proc.
  - LKM: Loadable Kernel modules.
- Device Driver Modules.

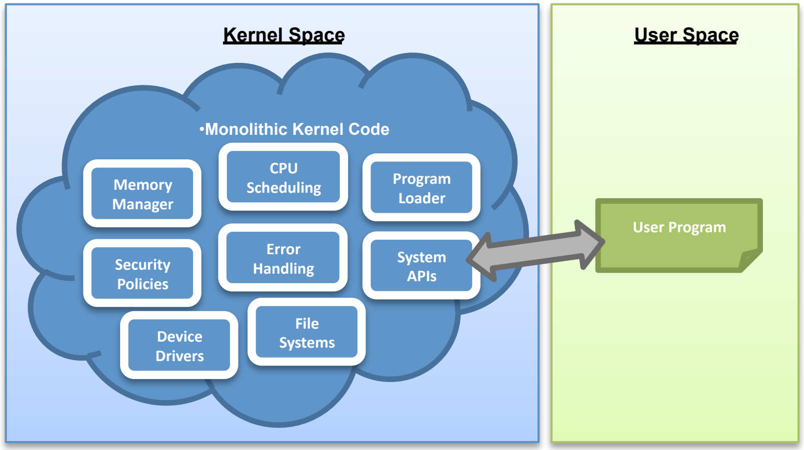
Introduction (Kernel)



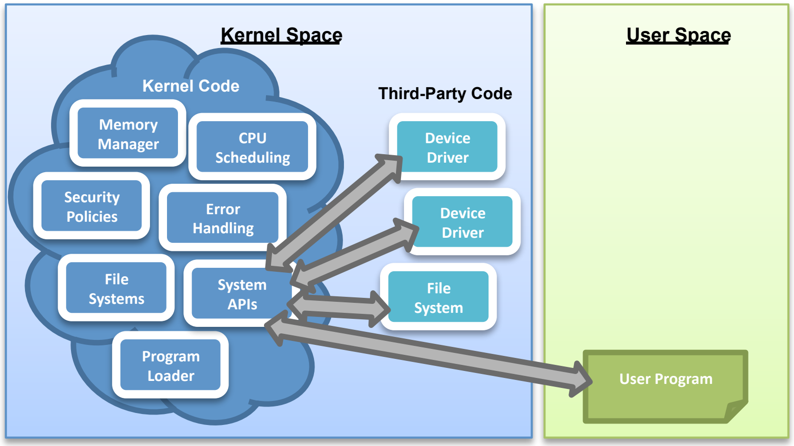
- Hides HW under an abstract, high level programming interface.
- Creates these concepts from low-level HW features:
  - Processes (time-sharing, protected address spaces).
  - Signals and semaphores.
  - Virtual memory (swapping, paging, mapping).
  - Filesystem (files, directories namespace).
  - General input/output (specialty hw, keyboard, mouse, USB).
  - Communication (between processes / network connections).
- Linux kernel mostly written in C (+ a few assembly (/linux/arch)).
- Source code available (git repository):
  - https://**git**hub.com/torvalds/**linux**.

Introduction (Kernel)

- Two basic approaches:
  - Monolithic kernels:
    - All functionality is compiled together.
    - All code runs in privileged kernel-space.



    - **Modular kernel** (also monolithic):
      - Most functionality compiled into the kernel, some functions loaded dynamically.
      - All functionality runs in kernel-space.



  - Microkernels:
    - Only essential functionality is compiled.
    - All other functionality runs in user space.

Introduction (Kernel)

- Usually, distributions include a kernel generic enough to avoid further reconfiguration.
- However, **reconfiguration** is sometimes unavoidable:
  - Adding a new hardware device.
  - Performance optimizations:
    - Pre-compiled kernels provide many unnecessary components (compatibility).
  - Routine updates (security patches).
- How can we "adjust" the kernel?:
  - **Statically**, re-compiling the whole kernel:
    - (Source code + compiler + a few more things).
  - **Dynamically**, through /proc params or modules.

Item	Lines	%
./usr	845	0.0042
./init	5,739	0.0283
./samples	8,758	0.0432
./ipc	8,926	0.0440
./virt	10,701	0.0527
	…	
./tools	232,123	1.1438
./kernel	246,369	1.2140
./Docume	569,944	2.8085
./include	715,349	3.5250
./sound	886,892	4.3703
./net	899,167	4.4307
./fs	1,179,220	5.8107
./arch	3,398,176	16.7449
./drivers	11,488,536	56.6110

Index

- Introduction:
  - Kernel types.
- **Static Reconfiguration:**
  - **Configuration.**
  - **Compilation.**
  - **Install.**
- Dynamic Reconfiguration:
  - /proc.
  - LKM: Loadable Kernel modules.
- Device Driver Modules.

Static Reconfiguration

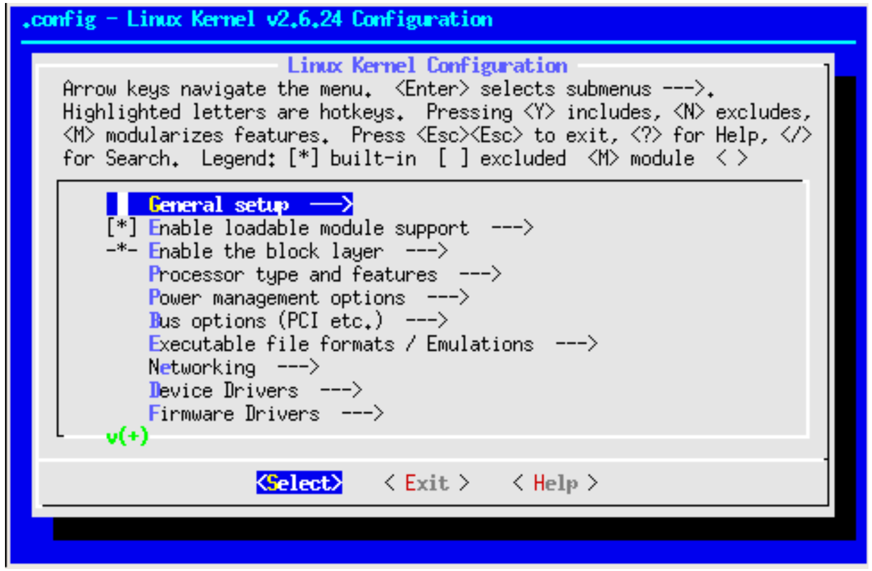
Working Directory:

/usr/src

- Step 1. Obtaining kernel **source code:**
  - From www.kernel.org (recommended stable versions):
    - Complete version: linux-4.X.X.tar.xz (~80MB).
    - Patches: patch-4.X.X.xz (~50-100k). (Applied to current kernel, with patch command).
  - From repositories:
    - apt-get install linux-source-4.X.X.
- Step 2. **Configuration:**
  - Kernel configuration in file /usr/src/linux-4.X.X/**.config:**
    - Each line contains a keyword (device/subsystem) and an argument: CONFIG_SCSI=y.
    - Driver can be not selected (#), built into the kernel (=y) or built as a module (=m).
  - Extremely complex process, requires deep hw and system understanding.
  - Two ways to create .config: from scratch or adjusting a well known config.

Static Reconfiguration

- Step 2. **Configuration** (cont.):
  - From **scratch:** make <config/**menuconfig**/xconfig>:
    - config: starts a character based question and answer session.
    - menuconfig: starts a terminal-oriented configuration tool (requires ncurses package).
    - xconfig: X based configuration tool.
  - From scratch (2): make **defconfig:**
    - creates a config file that uses default settings based on the current system's architecture.



Static Reconfiguration

- Step 2. **Configuration** (cont.):
  - Adapting a pre-built .config: make <oldconfig/silentoldconfig>:
    - oldconfig: update a config file (copied from another system or from previous kernel) to be compatible with the newer kernel source code (questions).
    - silentoldconfig: do not show questions answered by the config process.
  - More building options: make help.
- Step 3. **Compilation:**
  - Build the kernel + System.map :
    - [root si ~] make bzImage (after correct compilation kernel appears in arch/i386/boot).
  - Build modules (see next section):
    - [root si~] make modules.
  - Build ramdisk if modules are required to access booting device:
    - [root si~] mkinitrd –o /boot/initrd-4.X.X.img 4.X.X. (Example: our FS uses LVM/RAID).

Static Reconfiguration

- Step 2. **Configuration** (cont.):
  - Adapting a pre-built .config: make <oldconfig/silentoldconfig>:
    - oldconfig: update a config file (copied from another system or from previous kernel) to be compatible with the newer kernel source code (questions).
    - silentoldconfig: do not show questions answered by the config process.
  - More building options: make help.
- Step 3. **Compilation:**
  - Build the kernel + System.map:
    - [root si ~] make bzImage (after correct compilation kernel appears in arch/i386/boot).
  - Build modules (see next section):
    - [root si~] make modules.
  - Build ramdisk if modules are required to access booting device:
    - [root si~] mkinitrd –o /boot/initrd-4.X.X.img 4.X.X. (Example: our FS uses LVM/RAID).

Symbol (variable/function) to address table
Example: ffffffff8104d148 t swap_pages
Employed for debugging kernel crashes

Static Reconfiguration

- Step 4. **Installation:**
  - Copy kernel image, System.map and ramdisk to /boot:
    - [root si~] cp arch/i386/boot/bzImage /boot/bzImage_KERNEL-VERSION.
    - [root si~] cp System.map /boot/System.map-KERNEL_VERSION.
    - [root si~] ln –s /boot/System.map-KERNEL_VERSION /boot/System.map.
  - Install kernel modules (already built):
    - [root si~] make modules_install (installed in /lib/modules/KERNEL_VERSION).
  - Configure bootloader (grub2):
    - [root si~] update-grub.
    - Do not remove old kernels (new might not boot). Put them in /boot/grub/menu.lst.

```
…
title Test Kernel (4.X.X)
    root (hd0,1)
    kernel /boot/bzImage-4.X.X ro root=/dev/sda1 ro quiet
    initrd /boot/initrd-4.X.X.img
…
```

Static Reconfiguration (DEBIAN)

- Debian provides tools to compile + build a package for the kernel:
  - Append compiled kernel information to the software database.
  - Ease the management of multiple kernels (clean).
  - All the tools included in **kernel-package** (apt-get install kernel-package).
- Alternative Steps with debian (**make-kpkg**):
  - Step 2. Configuration: make-kpkg --config:
    - Equivalent to make oldconfig.
  - Step 3. Compilation: make-kpkg --initrd kernel_image modules_image:
    - Generates a .deb file with name: linux-image-[version]_[arch].deb.
    - Recommended to do a make-kpkg clean previously.
  - Step 4. Installation: as easy as dpkg -i linux-image-XXX.deb.

Index

- Introduction:
  - Kernel types.
- Static Reconfiguration:
  - Configuration.
  - Compilation.
  - Install.
- **Dynamic Reconfiguration:**
  - **/proc.**
  - **LKM: Loadable Kernel modules.**
- Device Driver Modules.

Dynamic Reconfiguration

- Kernel recompilation is not a usual task (very complex and delicate).
- Usually, kernel is "fine-tuned" dynamically:
  - Through /proc directory and/or Loadable Kernel Modules (LKM).
- **/proc:** pseudo File System representing current kernel status:
  - Details about system hardware (/proc/cpuinfo or /proc/devices).
  - Information about any process currently running:
    - cmdline: command line arguments.
    - cwd: link to current working directory.
    - environ: env variables.
    - exe: executable.
    - maps: memory maps to executable & library files.
    - mem: memory held by process.
    - … (see man proc).

```
[ root si /tmp ] ls /proc/2719
attr              environ   mem         root
auxv              exe       mountinfo   sched
cgroup            fd        mounts      sessionid
clear_refs        fdinfo    mountstats  smaps
cmdline           io        net         stat
coredump_filter   limits    oom_adj     statm
cpuset            loginuid  oom_score   status
cwd               maps      pagemap     task
```

Dynamic Reconfiguration (/proc)

- /proc employed for:
  - Input (configuration): echo 32768 > /proc/sys/fs/file-max.
  - Output (monitoring): /proc/stat.
- Command **sysctl:** configure kernel parameters at runtime:
  - Syntax: $ sysctl [option] <arguments>:
    - Option –a: display all values currently available.
    - Option –w: change a variable value (sysctl –w proc.sys.fs.file-max=32768).
    - Option –p: load settings from a file.
    - https://www.kernel.org/doc/Documentation/sysctl/kernel.txt.
- Permanent modifications: /etc/sysctl.conf.

Dynamic Reconfiguration (LKM)

- **Loadable Kernel Modules** (LKM):
  - Add code to the kernel while it is running (avoiding recompilation).
- Advantages:
  - No need to rebuild the kernel (keep using the untouched kernel).
  - Easier system problem diagnosis:
    - Kernel -> running; Kernel + LKM -> died; problem located at Module.
  - Faster development/maintenance (no rebuild/reboot).
- But…:
  - Some pieces MUST be built into the base kernel:
    - Anything required to boot far enough to load LKMs, for example, the driver of the disk drive that contains root filesystem.

Dynamic Reconfiguration (LKM)

- What LKMs are used for:
  - **Device drivers:** allow communication between kernel and a piece of HW.
  - **Filesystem drivers:** interpret the contents of a File System as files and directories.
  - **System calls:** make your own syscall or modify an existing one.
  - **Network driver:** interprets a network protocol (IPX link -> IPX driver).
  - TTY line disciplines, executable interpreters.
- Where are modules:
  - Files with extension .o and **.ko** (since 2.6 version).
  - /lib/modules/4.X.X.

```
alu@si:/lib/modules/3.16.0-4-amd64/kernel/drivers$ ls
arch   crypto   drivers   fs   lib   mm net   sound
```

Dynamic Reconfiguration (LKM)

- **LKM Administration:**
  - Command **insmod:** insert a module into the Linux kernel:
    - Syntax: $ insmod <module_files> [params].
  - Command **ismod:** show the status of modules in the Linux kernel:
    - Reads the content of /proc/modules.
  - Command **rmmod:** remove a module from Linux kernel.
  - Command **modinfo:** show information about a kernel module:
    - Syntax: $ modinfo [modulename/filename].
  - Similar to software packages, many modules are not self-contained, and rely on other modules to load and operate successfully.
  - Command **depmod:** generate the file modules.dep and map files.
  - Command **modprobe:** insert a module into the kernel, <u>solving previously dependencies</u>.

Dynamic Reconfiguration (LKM)

- **Automatic** LKM **Loading** and Unloading:
  - A LKM can be loaded automatically when the kernel first needs it (through the **kernel module loader**).
  - Kmod service performs background monitoring, making sure modules are loaded by modprobe (a user process that executes modprobe is created) as soon as they are needed by the kernel.
  - Optional part of the Linux kernel (select CONFIG_MODULES in .config).
  - Example:
    - [root si ~] rmmod vfat fat.
    - [root si ~] mkfs.vfat /dev/fd0.
    - [root si ~] mount /dev/fd0.
  - File /etc/modules lists the modules that must be loaded at boot time.

Dynamic Reconfiguration (LKM)

- **Automatic** LKM **Loading** and Unloading:
  - A LKM can be loaded automatically when the kernel first needs it (through the **kernel module loader**).
  - Kmod service performs background monitoring, making sure modules are loaded by modprobe (a user process that executes modprobe is created) as soon as they are needed by the kernel.
  - Optional part of the Linux kernel (select CONFIG_MODULES in .config).
  - Example:
    - [root si ~] rmmod vfat fat.
    - [root si ~] mkfs.vfat /dev/fd0.
    - [root si ~] mount /dev/fd0.

It works!!!

  - File /etc/modules lists the modules that must be loaded at boot time.

Dynamic Reconfiguration (LKM)

- **Installing** new Modules from their source code:
  - Example: add support for a new network device named “snarf”.
  - in /usr/src/linux-XXX/drivers/net create the directory snarf and copy inside the .c and .h files provided by the developer.
  - Modify the following files:
    - drivers/net/Makefile: add “obj-$(CONFIG_SNARF_DEV)+= snarf/.
    - drivers/net/Kconfig: add 2 lines: 1. “config SNARF_DEV” 2. “Tristate ‘Snarf device support’:
      - Tristate means it can be built into the kernel (Y), built as a module (M) or not built at all (N).
    - First line allows selecting the device in configure, second line says it can be loaded as a module.
  - Compile the module and copy the .ko to /lib/modules.
  - Better option: follow the procedure.

```
# make modules SUBDIR=…
# make modules_install SUBDIR=…
# depmod
# modprobe <module_name>
```

Dynamic Reconfiguration (LKM)

- **Installing** new Modules from their source code:
  - Example: add support for a new network device named “snarf”.
  - in /usr/src/linux-XXX/drivers/net create the directory snarf and copy inside the .c and .h files provided by the developer.
  - Modi
    - d
    - d
      s

All these steps are not strictly necessary for loading your module into the kernel.

They are required if you want to include this module into the monolithic part.

They are required if you want to manage your module through .config file.

Compiling and using insmod is enough.

    - First line allows selecting the device in configure, second line says it can be loaded as a module.
  - Compile the module and copy the .ko to /lib/modules.
  - Better option: follow the procedure.

```
# make modules SUBDIR=…
# make modules_install SUBDIR=…
# depmod
# modprobe <module_name>
```

Dynamic Reconfiguration (LKM)

- **Installing** new Modules:
  - Fortunately developers usually provide modules with some level of automation for installation.
  - **Kernel Patch** (compiled and installed as a module):
    - [root si ~] cd /usr/src/linux; patch -p1 < patch_file.
    - The patch leaves its code in /usr/src/linux/drivers.
    - Build the kernel and install.

```
# cd /usr/src/linux
# make modules_prepare
# make modules SUBDIR=…
# make modules_install SUBDIR=…
# modprobe <module_name>
```

  - **Script** (the common case):
    - The developer provides a .tgz including an installation script that performs the whole task.
    - LKMs can be EXTREMELY complex.

Index

- Introduction:
  - Kernel types.
- Static Reconfiguration:
  - Configuration.
  - Compilation.
  - Install.
- Dynamic Reconfiguration:
  - /proc.
  - LKM: Loadable Kernel modules.
- **Device Driver Modules.**

Device Driver Modules

- **Device:** name of a physical or logical device:
  - Physical: disk, tape, sound card…
  - Logical: terminal, net port…
- **Device Driver:** kernel modules that define the communication between the kernel and a device:
  - Interrupts, DMA, data transfer…
- **Device File:** special files that allow apps to interact with devices through the kernel:
  - Do not contain data, just a “frontend” to access device management function inside the kernel.

Device Driver Modules (Device File)

```
brw-r-----    1 root     root        8,    0 Mar 10  2006 /dev/sda
```

- **Main features:**
  - Physical: character (serial/parallel ports, sound card) or block (Hard disk) device.
  - Major and Minor device numbers:
    - Major indicates the driver being used with that file (from the list in /proc/devices).
    - Minor is employed by the driver to identify multiple devices using the same driver (Partitions).
- All device files are found in **/dev** directory:
  - Standard devices (stdin, stdout, stderr), memory (mem) virtual mem (kmem).
  - Specials (null, zero, random).
  - IDE devices (hdXX), USB/SCSI/SATA (sdXX), RAID devices (mdXX).
  - Virtual terminals (ttyX), parallel and serial ports (lpX), optical devices (CDRom).

Device Driver Modules (Device File)

```
brw-r-----    1 root      root         8,    0 Mar 10   2006 /dev/sda
```

- **Main features:**
  - Physical: character (serial/parallel ports, sound card) or block (Hard disk) device.
  - Major and Minor device numbers:
    - Major indicates the driver being used with that file (from the list in /proc/devices).
    - Minor is employed by the driver to identify multiple devices using the same driver (Partitions).
- All device files are found in **/dev** directory:
  - Standard devices (stdin, stdout, stderr), memory (mem) virtual mem (kmem).
  - Specials (null, zero, random).
  - IDE devices (hdXX), USB/SCSI/SATA (sdXX), RAID devices (mdXX).
  - Virtual terminals (ttyX), parallel and serial ports (lpX), optical devices (CDRom).

Device Driver Modules (Device File)

- Pseudo Devices (logical):
  - Appear in /dev, but do not correspond to real hardware devices:
    - Example: console connections are assigned a TTY(serial pseudo-terminal).
    - More: remote connections (/dev/pts/X), specials (/dev/null).

- **Using** dev files (same as the rest of files):
  - Example: reproducing a sound: [root si~] cat sound.au > /dev/audio.
  - Useful tool: [root si~] ln –s /dev/null .history.
- Manual **creation** of dev files:
  - Script MAKEDEV.
  - Command **mknod:** create a block/character file (dev file):
    - Syntax: $ mknod <file_name> <type> <major> <minor>.

Device Driver Modules (<u>Device File</u>)

- From 2.6, /dev is automatically controlled by **udev:**
  - Udevd service: when a device is added or removed from the system, the kernel informs udev (- hotplug).
  - According to the content in /etc/udev/ (rules for device creation), udevd will create a device file in /dev.
- The **/sys** directory (sysfs):
  - Introduced in kernel 2.6. This is a pseudo File System (similar to /proc):
    - It has detailed information about status and configuration of present devices.
    - View device topology as a simple file system.
    - Previously, most of this information could be found in /proc.
  - Relatively new, many features still not used:
    - In the future might replace /dev and udev.