LPIC-1 101-500 – Lesson 9

101.1 Determine and configure hardware settings

The *sysfs* (*/sys*) virtual filesystem

- **/sys** is a virtual filesystem and just like **/proc** and **/dev** exists only in memory
- Contains information about system devices, device drivers as well as communication buses like **pci**, **usb**, **scsi** etc.
- Can be used for some basic configuration
- `$ mount -t sysfs sysfs /sys #`
  `mount sysfs on /sys`
- `$ df -hTa | grep sysfs # display`
  `# sysfs`
  `sysfs  sysfs  0  0  0  - /sys`

The *proc* (*/proc*) filesystem

- **/proc** is another virtual filesystem and as such exists only in memory
- The files and directories under it are created on startup and during operation of the system
- Provides information on running services
- Provides information on system devices
- Provides many other useful information about the system
- `$ mount -t proc proc /proc # mount the /proc filesystem`
- `$ df -hTa | grep \/proc$ # display proc`
  `proc   proc   0     0      0    - /proc`

The *udev* device manager and the */dev* directory

- The /dev directory hosts all the system devices:
- **`$ ls -laR /dev`**
- The files and directories inside /dev are dynamic but in some legacy systems they are static
- In modern systems devices are dynamically created using the **udev** device manager
- Under **udev** the **/dev** directory exists as a **devtmpfs** virtual filesystem
- **`$ df -hTa | grep \/dev$`** `# display /dev`

```
udev  devtmpfs 3.9G  0  3.9G   0% /dev
```

The D-Bus service

- **D-Bus** (Desktop Bus) is an Inter-Process communication and RPC mechanism for UNIX and Linux systems
- It allows the communication between various programs in the same system
- Originally used by graphical environments it is now an essential part of most Linux systems (either with GUI or not)
- In modern systems it is tightly coupled with **systemd**

USB Controllers

- Open Host Controller Interface (**OHCI**)
  - USB 1.1 (12 Mbit/s)
- Universal Host Controller Interface (**UHCI**)
  - USB 1.1 (12 Mbit/s)
- Enhanced Host Controller Interface (**EHCI**)
  - USB 2.0 (480 Mbit/s)
- Extensible Host Controller Interface (**xHCI**)
  - USB 3.0, 3.1, 3.2 (5 Gbit/s, 10 Gbit/s, 20Gbit/s)

USB devices categories

- Human Interface Device (HID):
  input devices (keyboard, mice, touchscreens, etc)
- Communication Devices: Modems, Network cards
- Mass Storage Devices: USB Hard Drives, USB Flash Drives, Memory cards
- Audio: Audio devices
- IrDA: Infrared devices
- USB Printers

Display USB devices with `lsusb`

- **`$ lsusb`** `# show USB buses and devices`

Options:

- **`-v`** `# verbose information for USB devices`
- **`-t`** `# show hierarchical topology of Usb buses and devices`

Display USB devices with `lsusb`

- **$ lsusb** # show USB buses and devices

```
Bus 008 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 007 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 006 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 005 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 004 Device 002: ID 08ff:2810 AuthenTec, Inc. AES2810
Bus 004 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 003 Device 002: ID 093a:2510 Pixart Imaging, Inc. Optical Mouse
Bus 003 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 002 Device 002: ID 0951:1603 Kingston Technology DataTraveler
1GB/2GB Pen Drive
Bus 002 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 001 Device 004: ID 17ef:1004 Lenovo
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

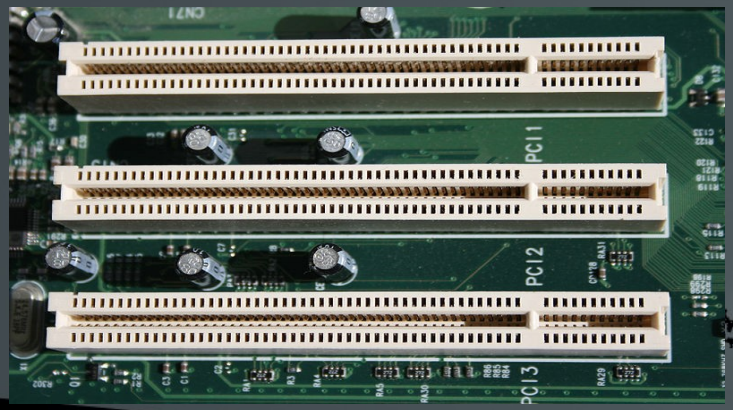
Display USB devices with `lsusb`

- **$ lsusb -t** # show USB device topology

```
/:  Bus 08.Port 1: Dev 1, Class=root_hub, Driver=uhci_hcd/2p, 12M

/:  Bus 07.Port 1: Dev 1, Class=root_hub, Driver=uhci_hcd/2p, 12M

/:  Bus 06.Port 1: Dev 1, Class=root_hub, Driver=uhci_hcd/2p, 12M

/:  Bus 05.Port 1: Dev 1, Class=root_hub, Driver=uhci_hcd/2p, 12M

/:  Bus 04.Port 1: Dev 1, Class=root_hub, Driver=uhci_hcd/2p, 12M

/:  Bus 03.Port 1: Dev 1, Class=root_hub, Driver=uhci_hcd/2p, 12M
    |__ Port 1: Dev 2, If 0, Class=HID, Driver=usbhid, 1.5M

/:  Bus 02.Port 1: Dev 1, Class=root_hub, Driver=ehci_hcd/6p, 480M

/:  Bus 01.Port 1: Dev 1, Class=root_hub, Driver=ehci_hcd/6p, 480M
    |__ Port 6: Dev 4, If 0, Class='bInterfaceClass 0x0e not yet
handled', Driver=uvcvideo, 480M
    |__ Port 6: Dev 4, If 1, Class='bInterfaceClass 0x0e not yet
handled', Driver=uvcvideo, 480M
```

The PCI Bus

- The PCI Bus is used for connecting system extension cards
- These cards are Coldplug (The system needs to be off before they are plugged or unplugged)
- PCI-X: PCI Extended extension of PCI which is still compatible with traditional PCI
- PCI Express (PCIe) replaces PCI and PCI-X



Display USB devices with `lspci`

- **$ lspci** # show PCI controllers and devices

  00:00.0 Host bridge: Intel Corporation Mobile 4 Series Chipset Memory Controller Hub (rev 07)

  00:01.0 PCI bridge: Intel Corporation Mobile 4 Series Chipset PCI Express Graphics Port (rev 07)

  00:03.0 Communication controller: Intel Corporation Mobile 4 Series Chipset MEI Controller (rev 07)

  00:03.2 IDE interface: Intel Corporation Mobile 4 Series Chipset PT IDER Controller (rev 07)

  00:03.3 Serial controller: Intel Corporation Mobile 4 Series Chipset AMT SOL Redirection (rev 07)

  00:19.0 Ethernet controller: Intel Corporation 82567LM Gigabit Network Connection (rev 03)

  00:1a.0 USB Controller: Intel Corporation 82801I (ICH9 Family) USB UHCI Controller #4 (rev 03)

  00:1a.1 USB Controller: Intel Corporation 82801I (ICH9 Family) USB UHCI Controller #5 (rev 03)

Display USB devices with `lspci`

Options:

- `-t` # show hierarchical topology of controllers and devices
- `-v` # verbose information
- `-vv` # very verbose information
- `-vvv` # very very verbose information

Kernel Modules

- Modules are object files that extend the functions of the Linux kernel
- They can be automatically or manually loaded or unloaded on demand
- They are used for device drivers, filesystems, protocols, etc
- They can be found under the directory:
  ```
  $ ls /lib/modules/$(uname -r)
  ```
- For legacy systems (kernel versions 2.4 and earlier), they have a **.o** extension
- For modern systems (kernel versions 2.6 and later), they have a **.ko** extension
  - In some systems (like CentOS) the modules are compressed with **xz** so they have a **ko.xz** extension
- **$ find /lib/modules/$(uname -r) -name "*.ko"** # find all modules for version **$(uname -r)**

Module Configuration Files

- /etc/modules.conf # for kernel version 2.4
- /etc/modprobe.conf # for kernel 2.6 and above
- /etc/modprobe.d/*.conf
- Modern systems use the **/etc/modprobe.d/*.conf** format
- These files set different rules for the proper resolution of conflicts that arise because of dependencies
- For setting up modules to load on startup we can use these:
- /etc/modules
- /etc/modules-load.d/*conf

Show active modules with `lsmod`

- **$ lsmod** # show loaded modules

```
Module                  Size  Used by

nls_iso8859_1          12713  1
nls_cp437              16991  1
vfat                   21708  1
fat                    61374  1 vfat
usb_storage            53538  1
uas                    17996  0
xt_multiport           12597  1
iptable_filter         12810  1
ip_tables              27456  1 iptable_filter # ←
                                               # iptables_filter
                                               # depends on
                                               # iptables

x_tables               29581  3
xt_multiport,iptable_filter,ip_tables
parport_pc             36959  0
ppdev                  17113  0
binfmt_misc            17565  1
joydev                 17606  0
```

Insert modules on a running kernel with `insmod`

- In legacy systems we simply use the module name:

```
# insmod msdos
```

- In modern systems we have to define the exact path:

```
# insmod /lib/modules/2.6.38-11-generic/kernel/fs/fat/msdos.ko
```

- **insmod** does not load dependent modules. They have to be manually loaded in the correct order:

```
# insmod fat ; insmod msdos
```

Remove modules from a running kernel with `rmmod`

- `$ rmmod msdos # remove module msdos.ko from the running kernel`

Options:

- `-a # remove all unused modules`

Add/remove modules from a running kernel with `modprobe`

- The **modprobe** command is more powerful that insmod or **rmmod** because it can load the called module as long as the modules it depends upon
- `$ modprobe msdos # this will add the msdos.ko modules as well as the fat.ko module, which is a dependency for msdos.ko`
- `$ modprobe -r msdos # this will unload msdos.ko, along with fat.ko`

Add/remove modules from a running kernel with `modprobe`

Options:

- **-a** # load all modules! Combined with **-t** it will load all modules of a certain category
- **-t <module category>** # successively load modules from a certain category until some one loads correctly, e.g. **-t net**.
- **-c** # show module configuration
- **-r** # remove module
- **-v** # verbose output

Information about modules with `modinfo`

- **$ modinfo msdos** # show information about msdos.ko

```
filename:       /lib/modules/4.9.0-6-amd64/kernel/fs/fat/msdos.ko
description:    MS-DOS filesystem support
author:         Werner Almesberger
license:        GPL
alias:          fs-msdos
depends:        fat
retpoline:      Y
intree:         Y
vermagic:       4.9.0-6-amd64 SMP mod_unload modversions
```

License