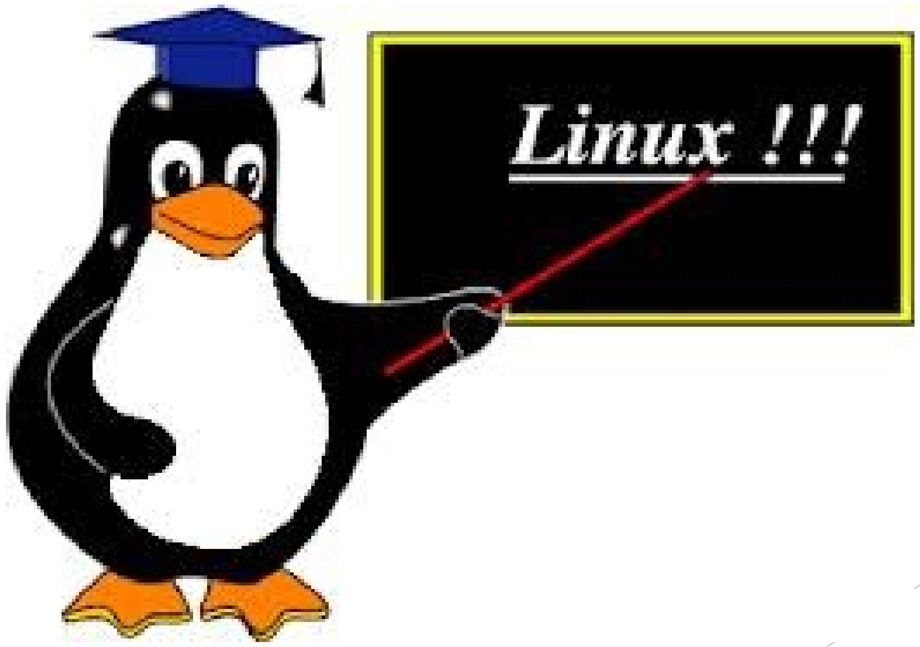
How to develop Device Drivers for Linux OS?

Presented from:

Rashid Siddiqui

For: Bin Tang

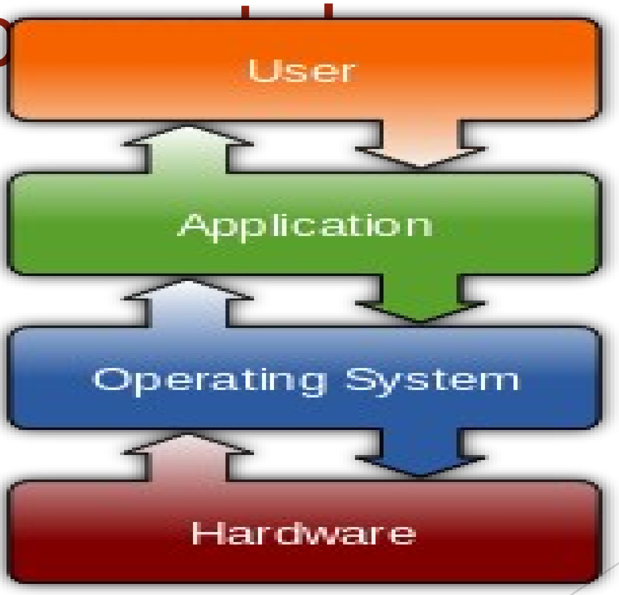


What is a device driver?

Software that handles or manages a hardware controller for a particular device!

What is a Controller?

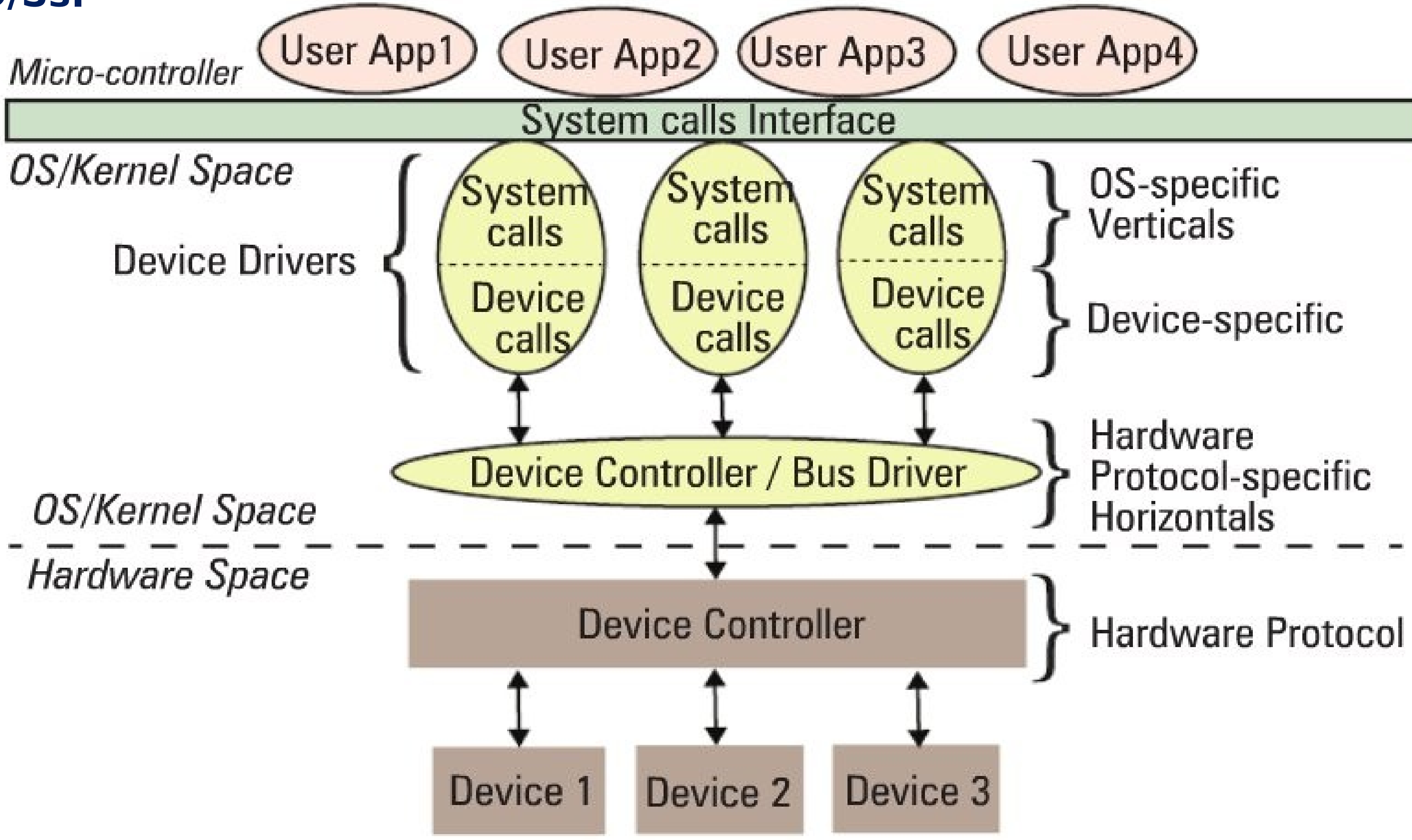
Is a piece of hardware that acts as the interface between the motherboard and the other comp



Device Data Sheet:

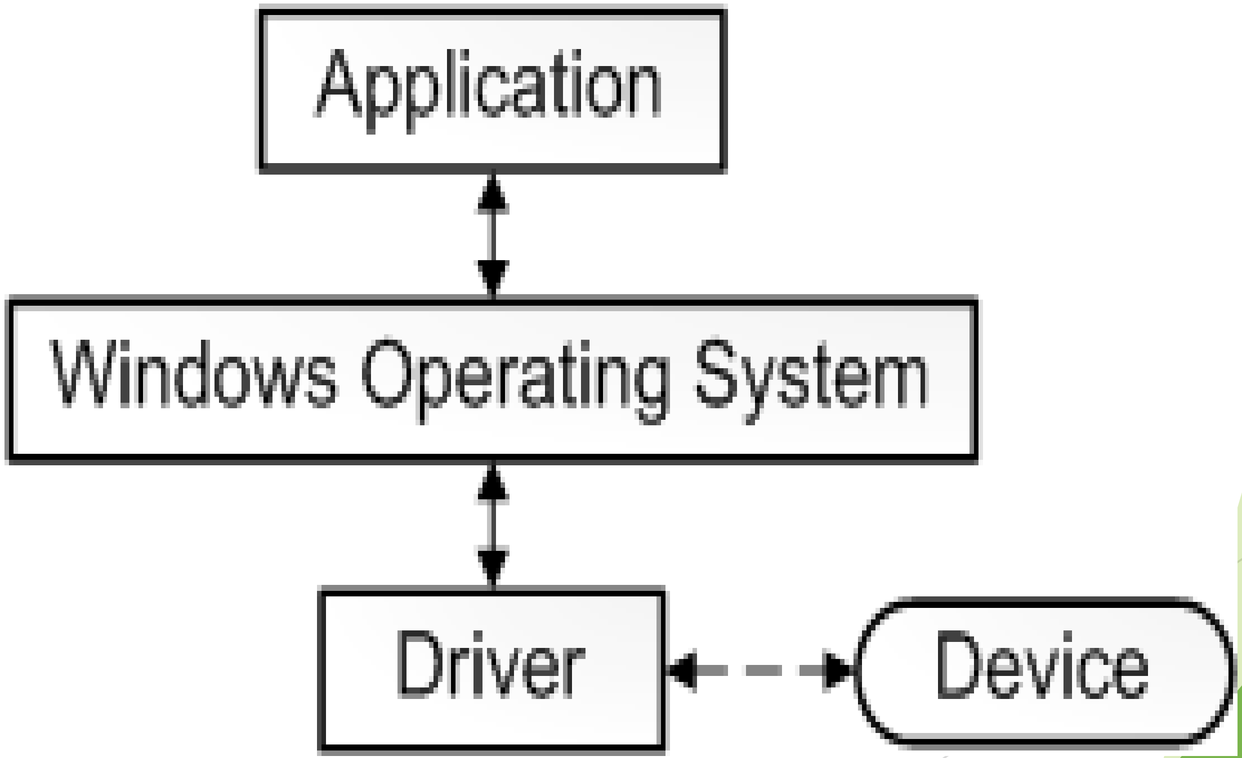
Gives technical details including its operation, performance, programming etc. Understanding and decoding required!

Device-specific portion of a device driver is the same across all O/Ss.



Functions of a device driver!

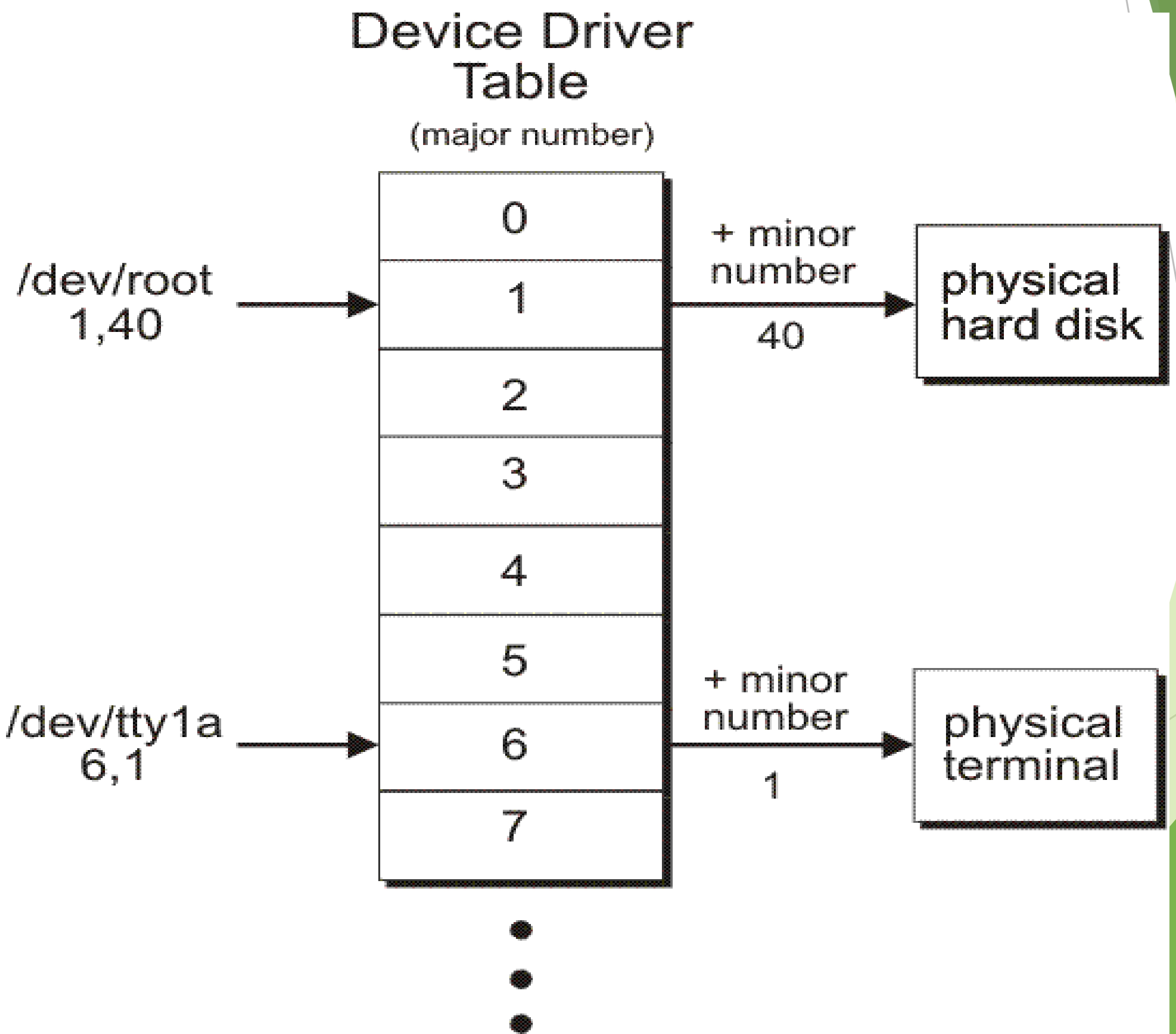
- implement device-specific functions for generic input/output operations.
- Communication interface O/S and hardware
- Defines and processes device commands.



Major and Minor Device Numbers

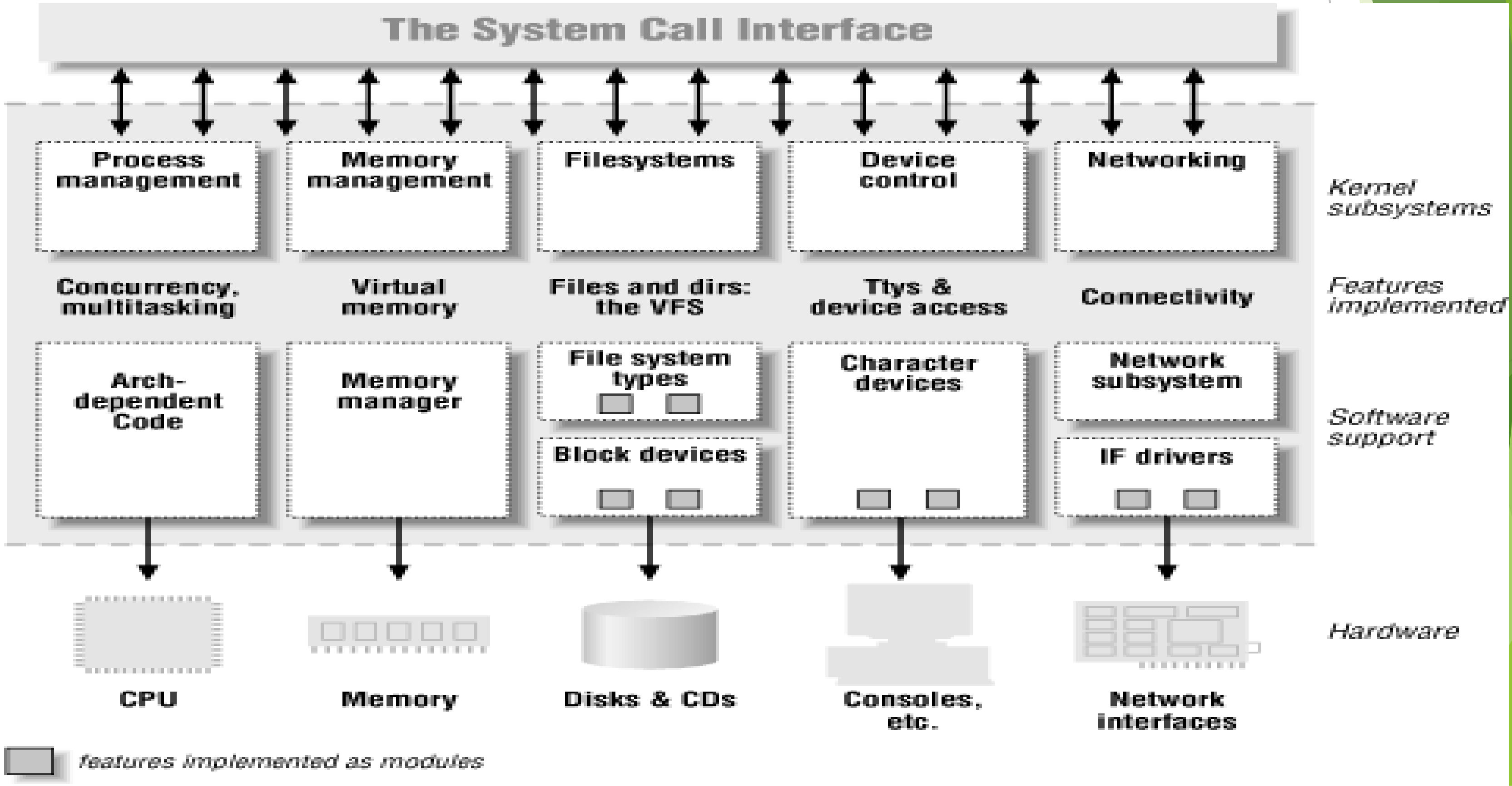
Major number is same for devices controlled by the same Device Driver.

Minor Device Number distinguish between different devices and their controllers.



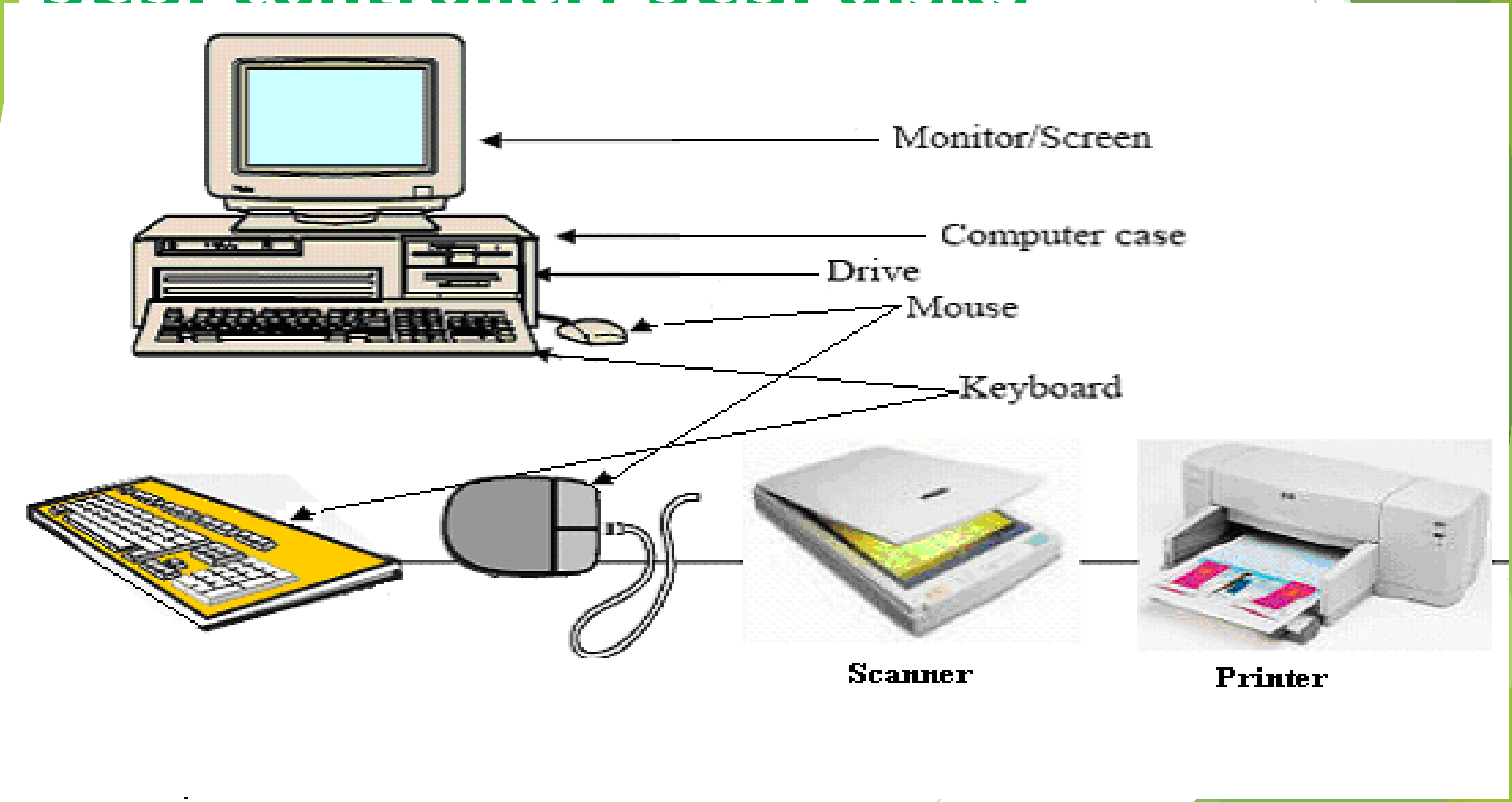
Each physical device has its own hardware controller.

1. **Character devices simplest, are accessed as files, applications use standard system calls to open them, read from them, write to them and close them exactly as if the device were a file.**
2. **This is true even if the device is a modem being used by the PPP daemon to connect a Linux system onto a network.**
3. **As a character device is initialized its device driver registers itself with the Linux kernel by adding an entry into the chrdevs vector of device_struct data structures.**
4. **The device's major device identifier (for example 4 for the tty device) is used as an index into this vector. The major device identifier for a device is fixed.**



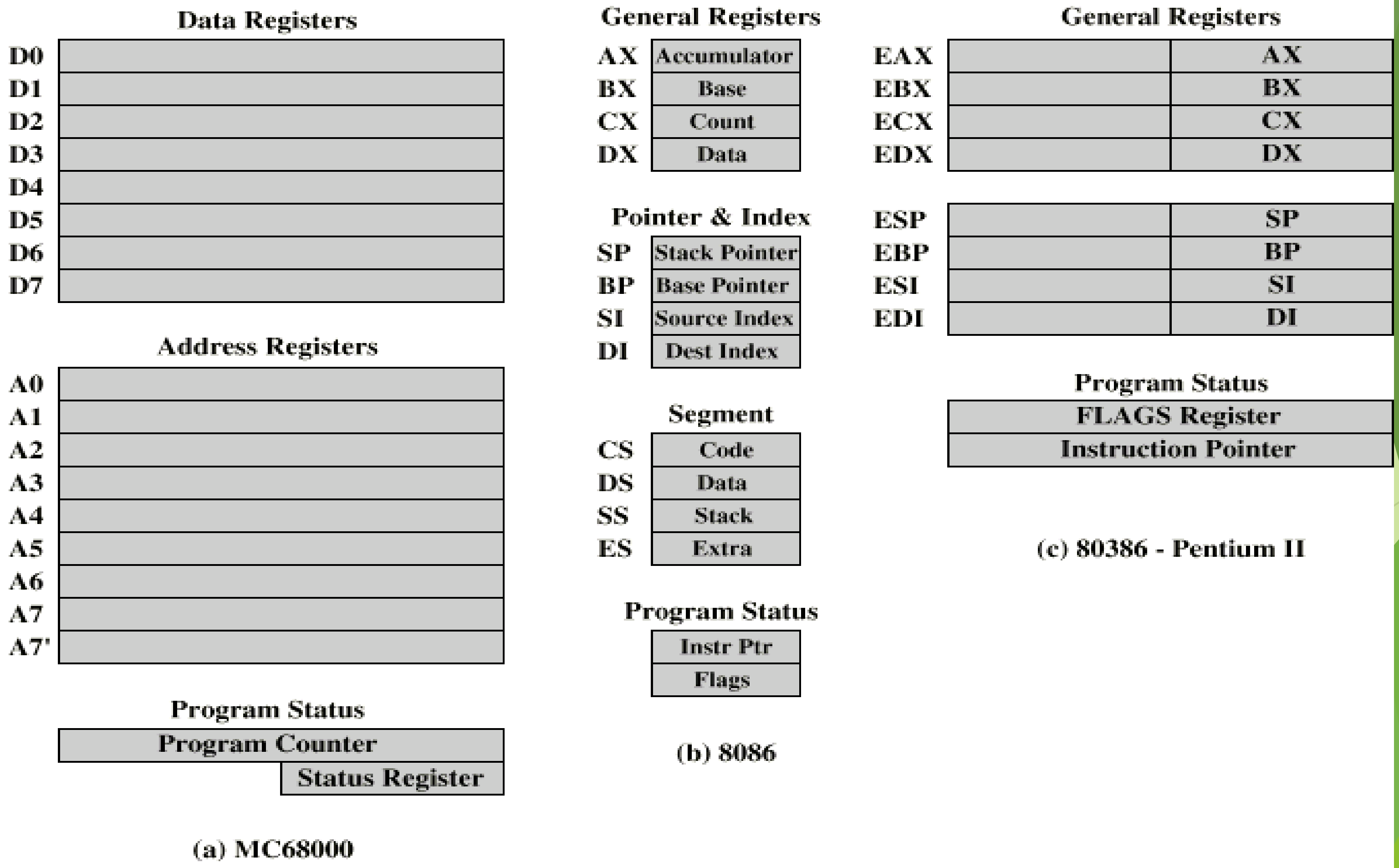
Different Types of Device Controllers

1. A superio chip: keyboard, mouse and serial ports
2. IDE controller: Controls IDE disks
3. SCSI controller: SCSI disks



Control and Status Registers
Functions of CSRS:

- Used to start and stop the device, initialize it and to diagnose problems with it.
- The code to manage is kept in the Linux kernel and not in every application.
- Control and Status Register (CSR) is a register in many central processing units that are used as storage devices for information about instructions received from machines.
- The device is generally placed in the register address 0 or 1 in CPUs and works on the concept of using a comparison of flags (carry, overflow and zero, usually) to decide on various If-then instructions related to electronic decision flows



Device Drivers: Properties

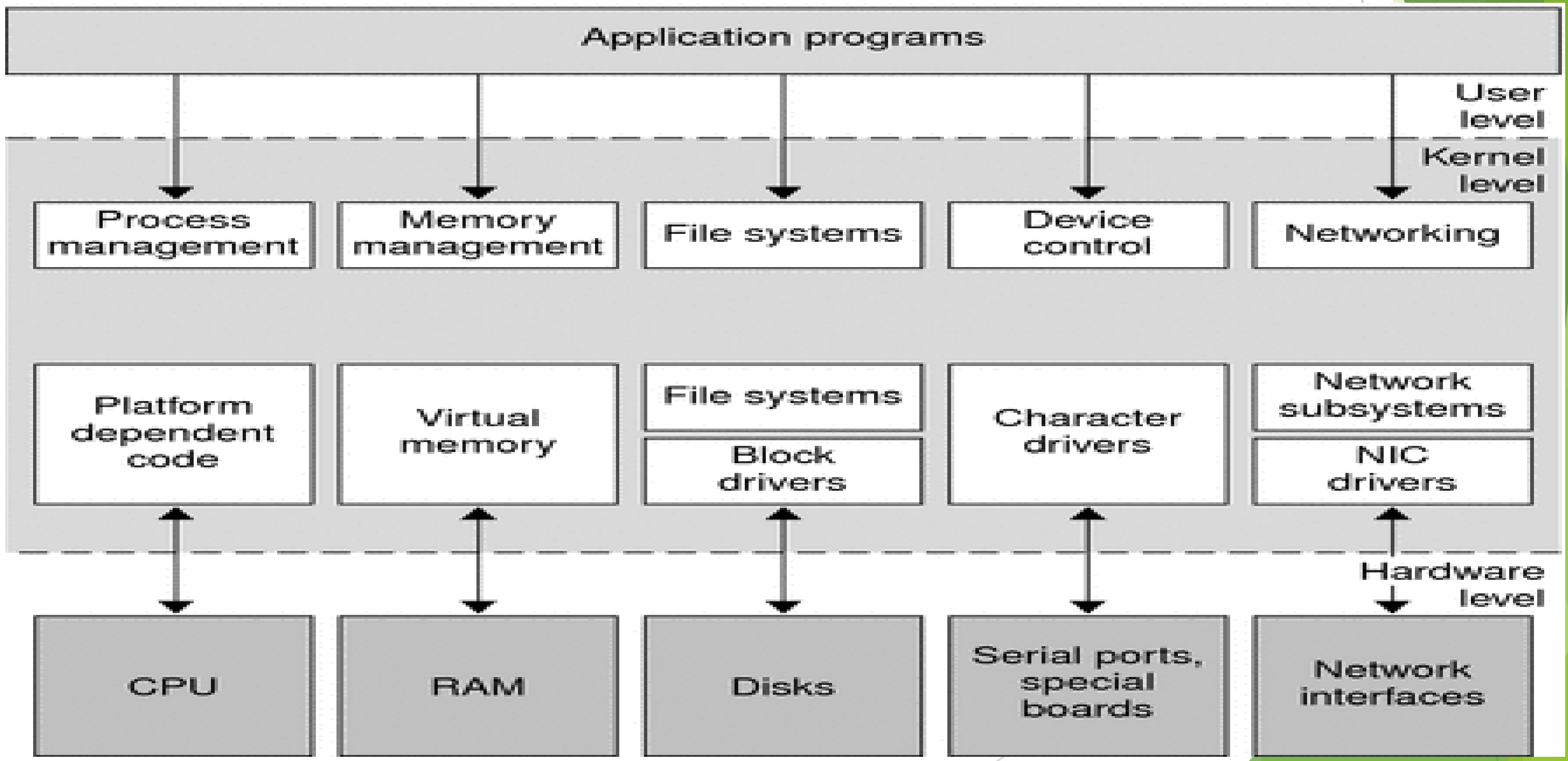
- A shared library of privileged, memory resident, low level hardware handling routines!
- Handle the peculiarities of the devices they are managing.
- Abstract the handling of devices.
- Devices look like regular files: opened, closed, read and written
- Use same, standard, system calls that are used to manipulate files.
- A special file exists for every device, first IDE disk has /dev/hda.

Block (disk) and character devices created by (struct{mknod-command})

Major and minor numbers are used to describe the device.

Network devices are also represented by device special files but they are created by Linux as it finds and initializes the network controllers in the system.

1. **All devices controlled by the same device driver have a common major device number.**
2. **The minor device numbers are used to distinguish between different devices and their controllers, for example each partition on the primary IDE disk has a different minor device number. So, /dev/hda2, the second partition of the primary IDE disk has a major number of 3 and a minor number of 2.**
3. **Linux maps the device special file passed in system calls (say to mount a file system on a block device) to the device's device driver using the major device number and a number of system tables, for example the character device table, chrdevs .**



Linux supports three types of hardware devices: character, block and network.

Character devices:

a) Read and write without buffering
b) System's serial ports /dev/cua0 and/dev/cua1.

Block devices: written to and read from in multiples of the block size, typically 512 or 1024 bytes.

Accessed:

c) Randomly via the buffer cache
d) Via their device special file
e) Via file system.
f) Support a mounted file system.

Network devices are accessed:

g) BSD socket interface
h) Networking subsytems.

Network Devices (represented by a device data structure)

Hardware: like Ethernet card sends and receives packets of data

Software only: Loopback device which is used for sending data to yourself.

Initialized registered at kernel boot time.

Device data Structure contains:

Information about the device and the addresses of functions that allow the various supported network protocols to use the device's services.

These functions are mostly concerned with transmitting data using the network device.

The device uses standard networking support mechanisms to pass received data up to the appropriate protocol layer.

All network data (packets) transmitted and received are represented by sk_buff data structures,

Flexible data structures that allow network protocol headers to be easily added and removed.

How the network protocol layers use the network devices, how they pass data back and forth using sk_buff data structures.

Under BSD sockets,
# include <Header Files>

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
```

For Winsock you need winsock.h.
(Or, if you need Winsock 2-specific functionality, winsock2.h.)

Interface Flags

These describe the characteristics and abilities of the network device:

IFF_UP	Interface is up and running,
IFF_BROADCAST	Broadcast address in `device` is valid
IFF_DEBUG	Device debugging turned on
IFF_LOOPBACK	This is a loopback device
IFF_POINTTOPOINT	This is point to point link (SLIP and PPP)
IFF_NOTRAILERS	No network trailers
IFF_RUNNING	Resources allocated
IFF_NOARP	Does not support ARP protocol
IFF_PROMISC	Device in promiscuous receive mode, it will receive all packets no matter who they are addressed to
IFF_ALLMULTI	Receive all IP multicast frames
IFF_MULTICAST	Can receive IP multicast frames

Common Interfaces for Character, Block and Network Devices

- Kernel can treat often very different devices and their device drivers absolutely the same.
- Linux maintains tables of registered device drivers as part of its interfaces with them
- These tables include pointers to routines and information that support the interface with that class of devices.
- Linux kernel Uses the same interface for SCSI and IDE
- Linux is dynamic, needs different device drivers for different devices
- Its configuration scripts enable device drivers to be included at kernel build
- When initialized they may not discover any hardware
- Drivers can be loaded as kernel modules when needed
- When initialized register themselves with the kernel

Common Attributes of Linux Drivers

- Kernel Code: Without errors
- Standard Interfaces : A terminal driver and SCSI device provide file I/O and buffer cache interfaces to the kernel
- Kernel Mechanisms and Services: Device drivers use memory allocation, interrupt delivery and wait queues to operate
- Loadable: loaded on demand as kernel modules, makes the kernel very adaptable and efficient with the system's resources
- Configurable: built into the kernel and are configurable when the kernel is compiled
- Dynamic: As the system boots and each device driver is initialized it looks for the hardware devices that it is controlling.
- All the device drivers are initialized, even if the device does not exist

Block Devices

Block devices also support being accessed like files.

The mechanisms used to provide the correct set of file operations for the opened block special file are very much the same as for character devices.

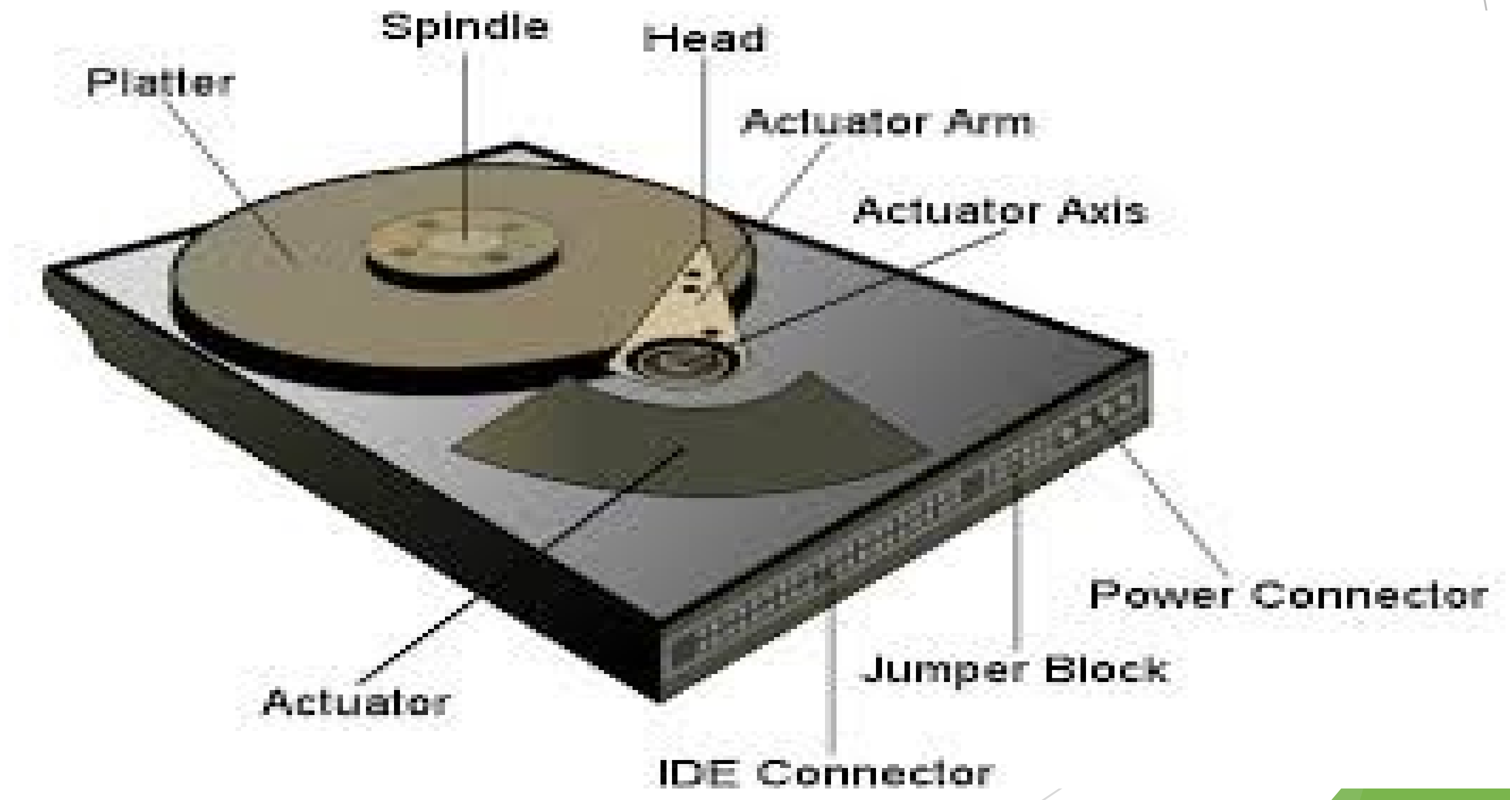
1. Linux maintains the set of registered block devices as the blkdevs vector.
2. It, like the chrdevs vector, is indexed using the device's major device number.
3. Its entries are also device_struct data structures.
4. Unlike character devices, there are classes of block devices. SCSI devices are one such class and IDE devices are another.
5. It is the class that registers itself with the Linux kernel and provides file operations to the kernel. The device drivers for a class of block device provide class specific interfaces to the class.
6. So, for example, a SCSI device driver has to provide interfaces to the SCSI subsystem which the SCSI subsystem uses to provide file operations for this device to the kernel.

Hard Disks

More permanent method for storing data, keeping it on spinning disk platters.

To write data, a tiny head magnetizes minute particles on the platter's surface.

The data is read by a head, which can detect whether a particular minute particle is magnetized.



IDE Disks (Integrated Disk Electronic) uses an interface

1. The master and slave functions by jumpers
2. Primary IDE controller, the next the Secondary IDE controller and so on.
3. IDE can manage about 3.3 Mbytes per second of data transfer to or from the disk and the maximum IDE disk size is 538Mbytes.
4. Extended IDE, or EIDE, has raised the disk size to a maximum of 8.6 Gbytes and the data transfer rate up to 16.6 Mbytes per second.
5. IDE and EIDE disks are cheaper than SCSI disks and most modern PCs contain one or more on board IDE controllers

C is reading data while A,B and D are writing data

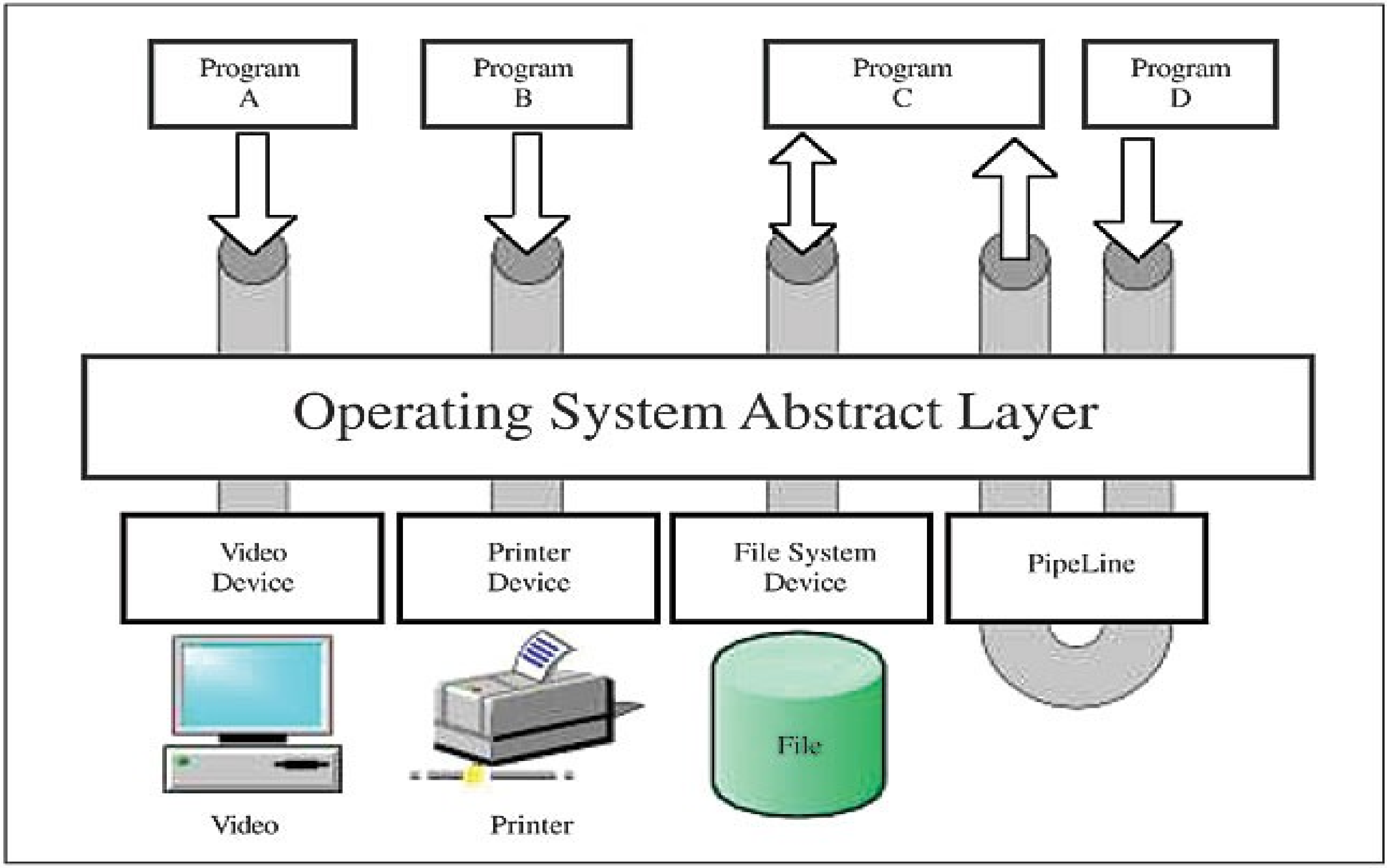


Figure 1. Device driver and file access. There are four programs (A, B, C, and D) reading data (C), writing data (A, B, D) or both (C). The arrows specify the data flow direction.

SCSI Disks use a bus up to 8 devices

1. Unique identifier set by jumpers on the disks.
2. Transferred synchronous or asynchronous between any two devices on the 32 bit wide bus, transfers up to 40 mbytes /s
3. Transfers both data and state information between devices
4. Single transaction between an initiator and a target can involve up to eight distinct phases.
5. You can tell the current phase of a SCSI bus from five signals from the bus.

SCSI Bus Information

Information needed by device driver

- The *irq* number is the interrupt
- The *base address* is the address of any of the device's control and status registers in I/O memory.
- DMA channel number
- Set at boot time as the device is initialized.

Network Protocol Information

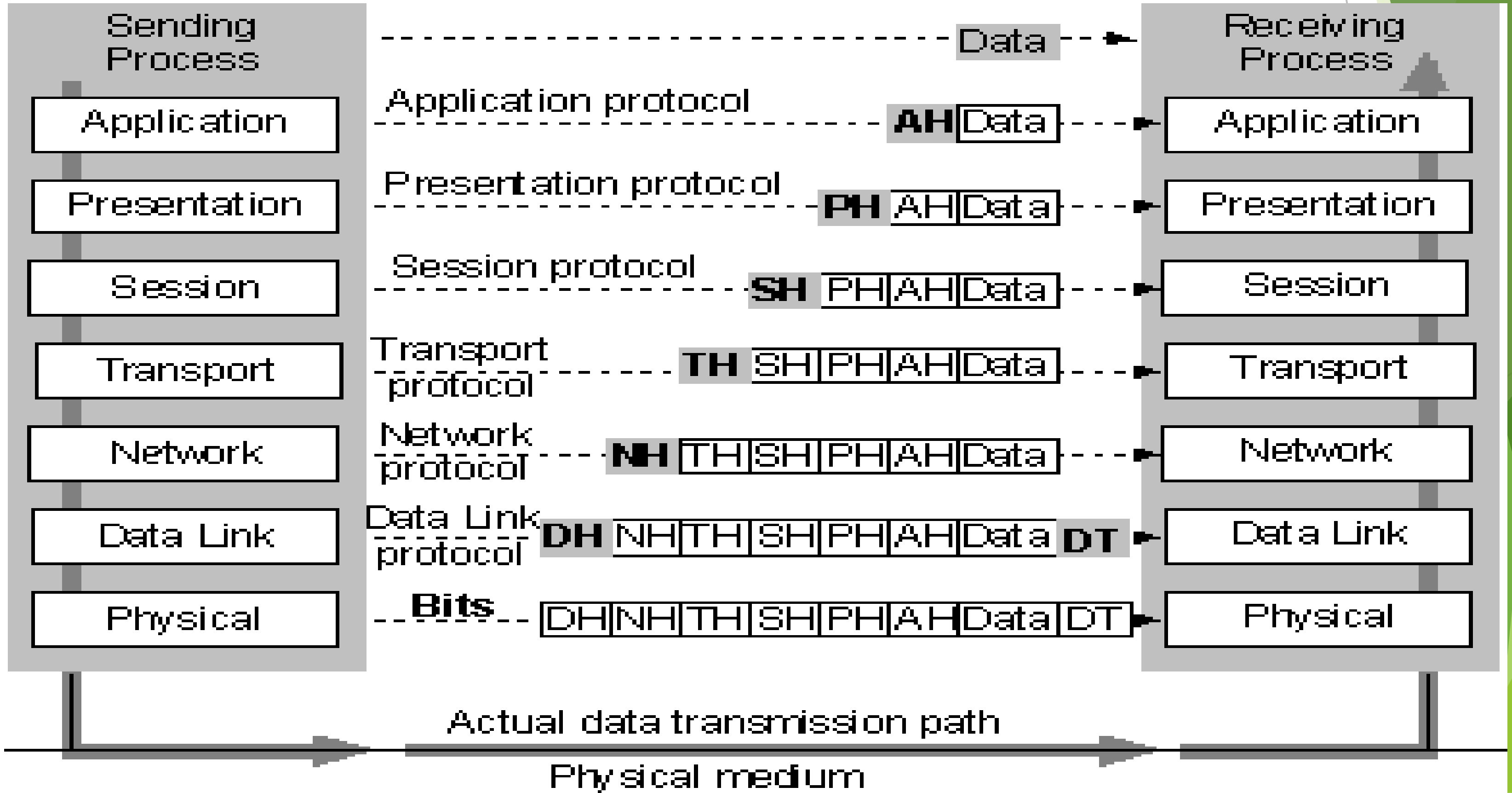
- Each device describes how it may be used by the network protocol layers:
- MAXIMUM TRANSMISSION UNIT MTU: The size of the largest packet that this network can transmit not including any link layer headers that it needs to add. This maximum is used by the protocol layers, for example IP, to select suitable packet sizes to send.
- Family: The family indicates the protocol family that the device can support. The family for all Linux network devices is AF_INET, the Internet address family.
- Type: The hardware interface type describes the media that this network device is attached to.
- Media: These include Ethernet, X.25, Token Ring, Slip, PPP and Apple Localtalk.
- Addresses: Device Data Structure holds a number of addresses that are relevant to this network device, including its IP addresses.

Network Devices Packet Queue

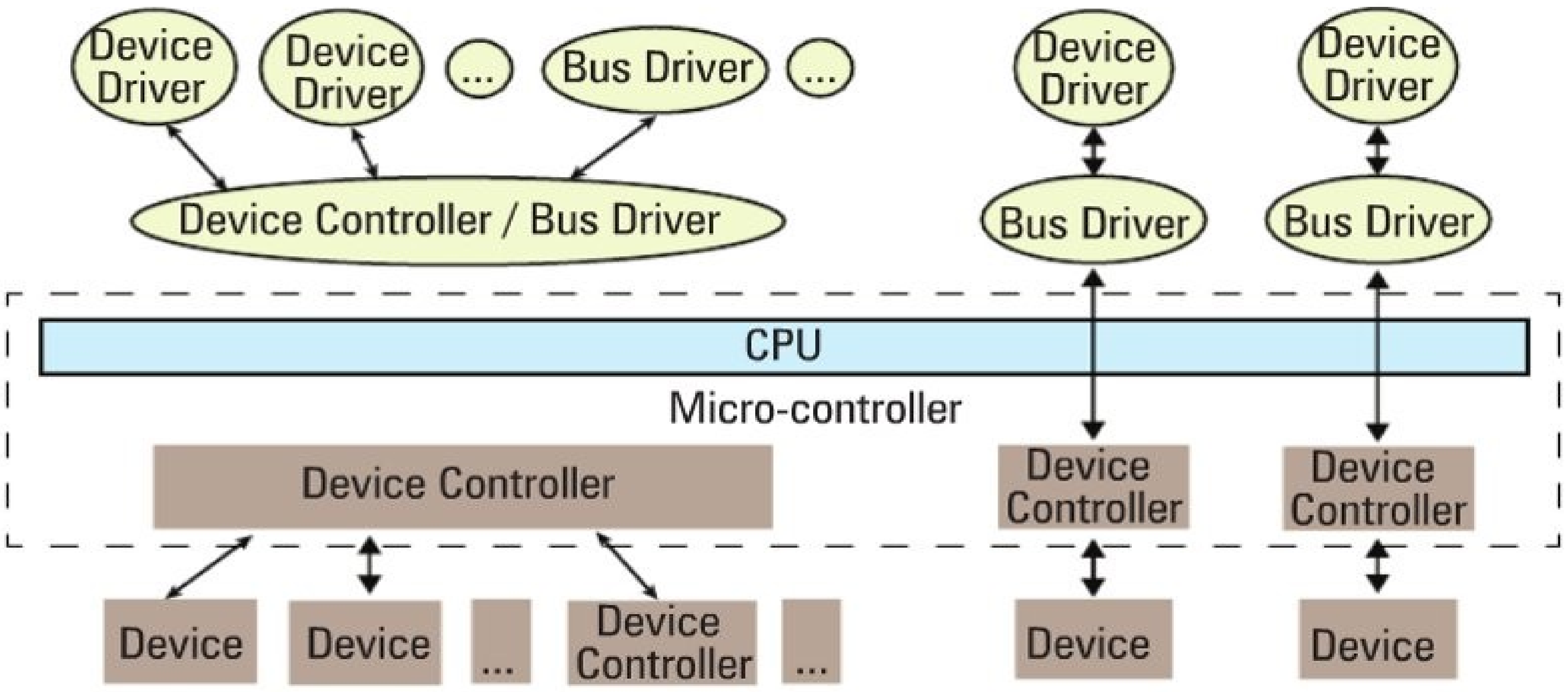
This is the queue of sk_buff packets queued waiting to be transmitted on this network device,

Support Functions

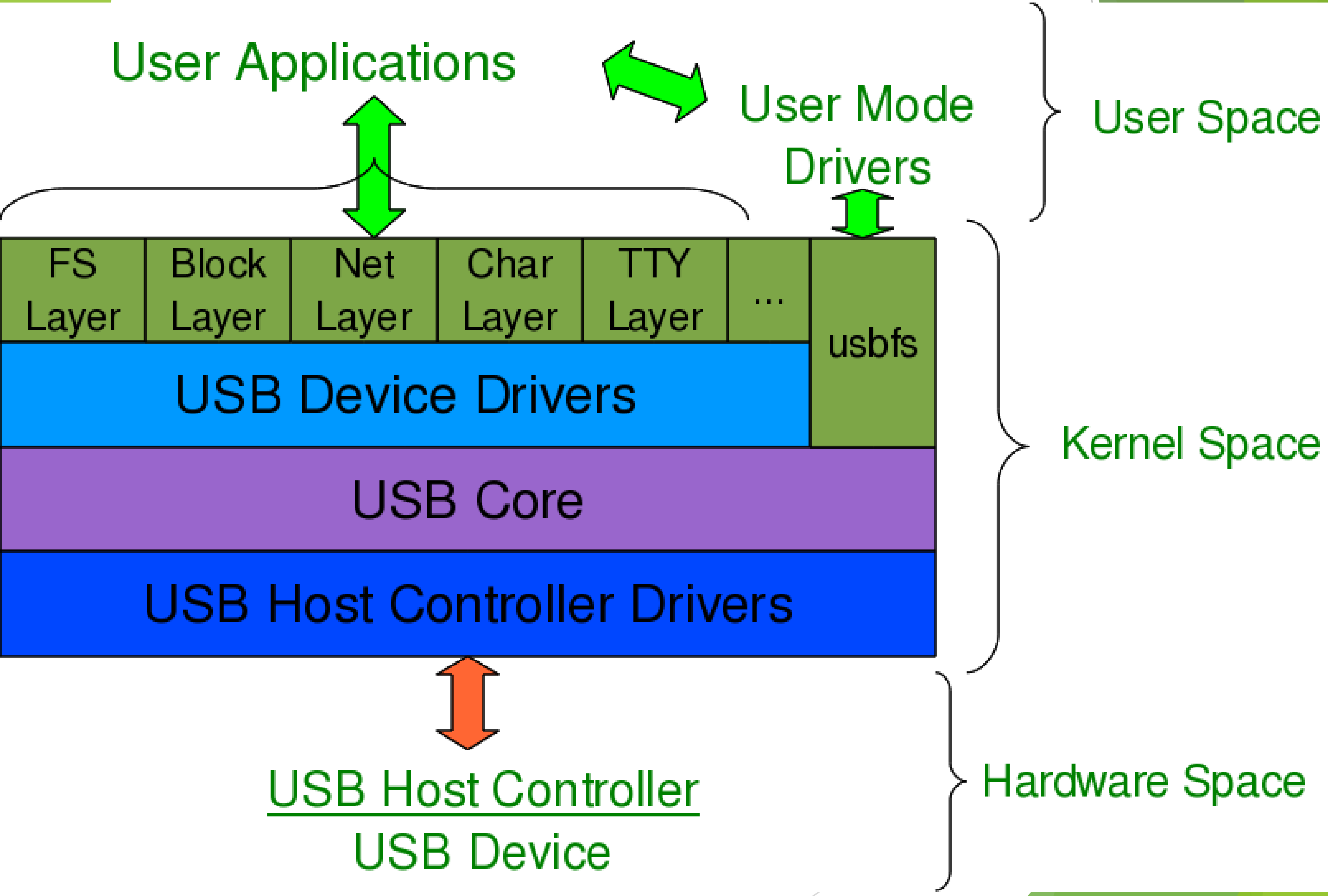
1. Each device provides a standard set of routines that protocol layers call as part of their interface to this device's link layer.
2. These include setup and frame transmit routines as well as routines to add standard frame headers and collect statistics.
3. These statistics can be seen using the ifconfig command.



A driver drives, manages, controls, directs and monitors the entity under its command. What a bus driver does with a bus, a device driver does with a computer device (any piece of hardware connected to a computer) like a mouse, keyboard, monitor, hard disk, Web-camera, clock, and more.

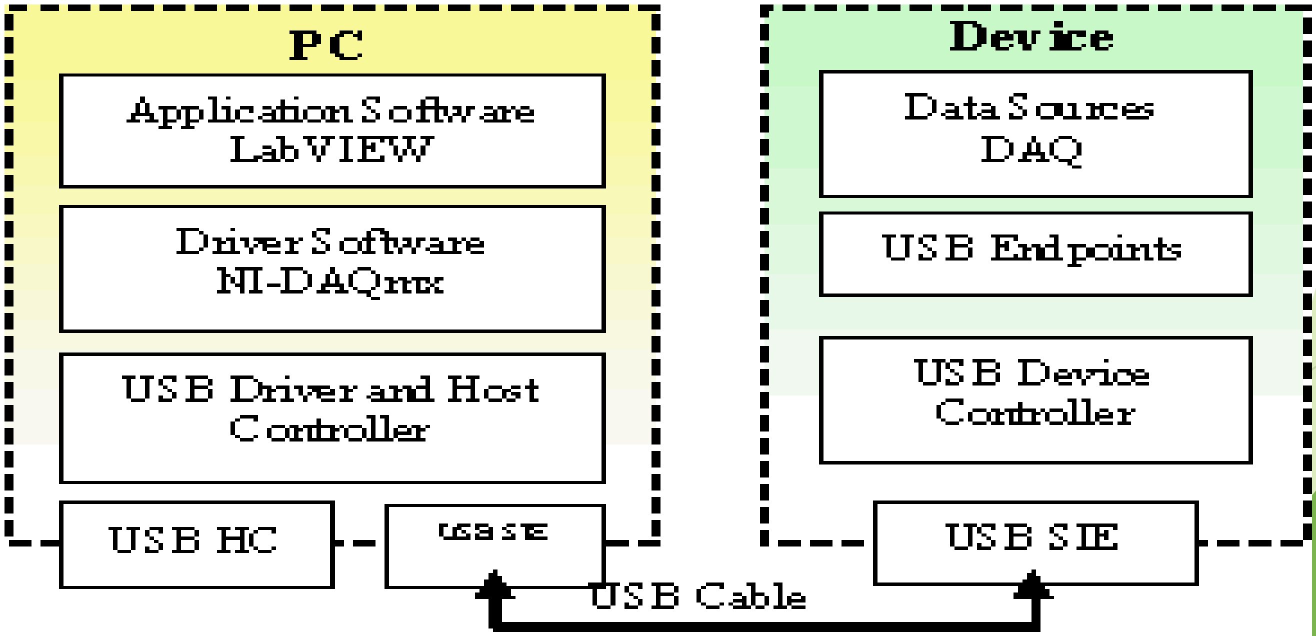


USB subsystem in Linux: It shows a top-to-bottom view.



USB protocol specifications used to detect USB devices

1. Hardware-space detection is done by the USB host controller.
2. HCI is a register-level interface that enables a host controller for USB
3. HC driver would translate the low-level into higher-level USB protocol-specific information.
4. The USB protocol formatted information about the USB device is then populated into the generic USB core layer (the usb core driver) in kernel-space, thus enabling the detection of a USB device in kernel-space, even without having its specific driver.
5. After this, it is up to various drivers, interfaces, and applications (which are dependent on the various Linux distributions), to have the user-space view of the detected devices..



Decoding a USB device section

- one or more configurations.
- A configuration is a profile, where the default one is the commonly used one
- Linux supports only one configuration per device — the default one.
- Every configuration may have one or more interfaces.
- An interface corresponds to a function provided by the device.
- There would be as many interfaces as the number of functions provided by the device.
- MFD (multi-function device) USB printer can do printing, scanning and faxing uses at least three interfaces, one for each of the functions.
- So, unlike other device drivers, a USB device driver associated/written per interface,
- rather than the device as a whole — meaning that one USB device may have multiple device drivers, and different device interfaces may have the same driver — though, of course, one interface can have a maximum of one driver only.
- It is okay and fairly common to have a single USB device driver for all the interfaces of a USB device.

USB end-points

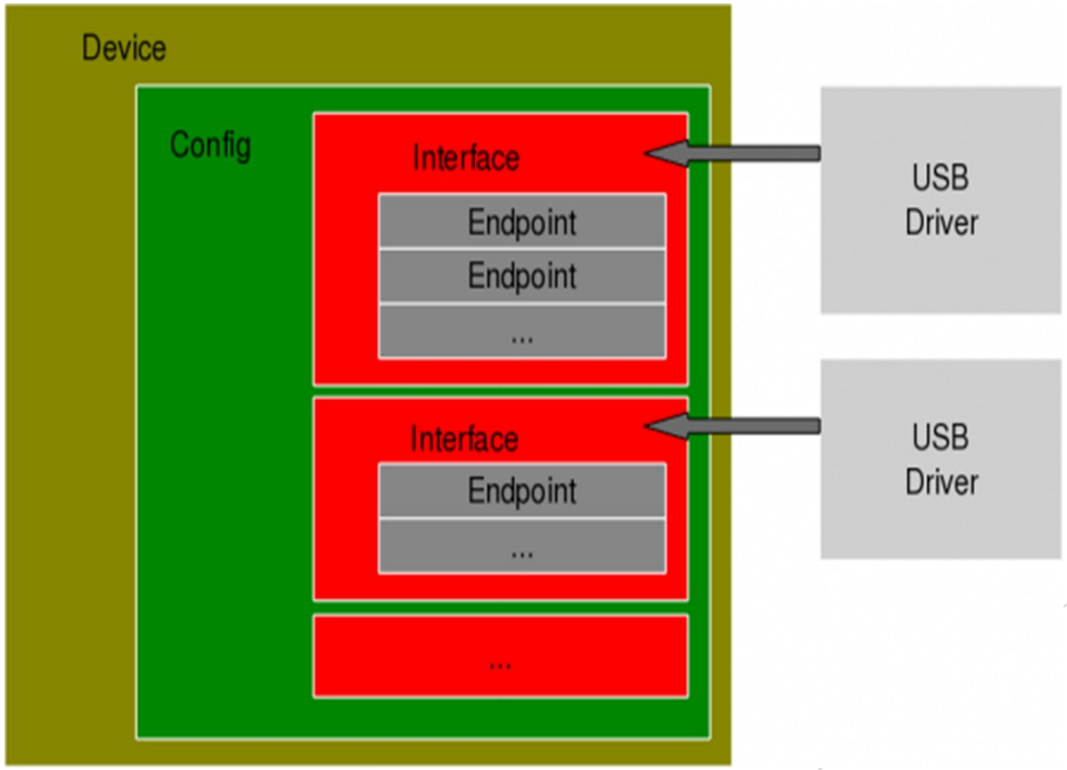
A pipe for transferring information either into or from the interface of the device, depending on the functionality.

An interface has one or more endpoints

Types: control, interrupt, bulk and isochronous.

Usb protocol specification: usb devices have an implicit special control end-point zero, the only bi-directional end-point.

Pictorial representation of a valid usb device, based on the above explanation.



References:

1. Device Drivers, Part 1: Linux Device Drivers for Your Girl Friend

By AKumar Pugalia on November 1, 2010
in Concepts, Developers, Overview · 2 Comments

2. David A Rusling 3 Foxglove Close, Wokingham, Berkshire RG41 3NF, United Kingdom

3. Device Drivers, Part 11: USB Drivers in Linux

By Anil Kumar Pugalia on October 1, 2011 in Coding, Developers · 5 Comments

4. NI CompactDAQ under the Hood – Technologies That Drive USB Performance

Publish Date: Apr 24, 2014 | 18 Ratings | **3.67** out of 5 |

Thanks!