Linux Device Drivers

An Introduction to Device Drivers

Role of Device Drivers

The role of device driver is providing mechanism, not policy

What is the difference between mechanism and policy?

The distinction between mechanism and policy is one of the best ideas behind the Unix design. Most programming problems can indeed be split into two parts: "what capabilities are to be provided" (the mechanism) and "how those capabilities can be used" (the policy). If the two issues are addressed by different parts of the program, or even by different programs altogether, the software package is much easier to develop and to adapt to particular needs

Example →

Unix divides the graphics display into
- → X-Server :- which knows about hardware and provides a unified interface
- → session & window managers :- uses the interface provided by the X-Server knows nothing about the underlying hardware

So, the device drivers are policy-free
- floppy disk drivers are policy-free - the role being showing the diskette as continuous array of data blocks

Higher level system provide the abstraction on whether users can access or who can access, whether via filesystem or without, and other features

The programmer is also free to decide the underlying architecture of the device driver.

for example:-

if a driver will be used by multiple users, the programmer is free to choose what kind of concurrency to use.

Splitting the kernel

In Linux kernel, several processes work together to attend to different task these kernel task can be split into

* Process management
* Memory management
* Filesystems
* Device control
* Networking

Loadable Modules

One of the good features of Linux kernel is to extend the runtime with set of kernel features.
This mean you can extend the capabilities of the kernel, as well as remove them, while the system is runing

This pieces of code are known as module
* insmod :- dynamically linking to the runtime program
* rmmod :- removing / unlinking the module

Class of devices and Modules

The way Linux treats a device distinguishes into main 3types :-

* char module
* block module
* network module

Character device

A character (char) device is one that can be accessed as a stream of bytes (like a file); a char driver is in charge of implementing this behavior. Such a driver usually implements at least the open, close, read, and write system calls.

The text console (/dev/console) and the serial ports (/dev/ttyS0 and friends) are examples of char devices, as they are well represented by the stream abstraction. Char devices are accessed by means of filesystem nodes, such as /dev/tty1 and /dev/lp0.

The only relevant difference between a char device and a regular file is that you can always move back and forth in the regular file, whereas most char devices are just data channels, which you can only access sequentially.

Block devices

Like char devices, block devices are accessed by filesystem nodes in the /dev directory. A block device is a device (e.g., a disk) that can host a filesystem.

In most Unix systems, a block device can only handle I/O operations that transfer one or more whole blocks, which are usually 512 bytes (or a larger power of two) bytes in length. Linux, instead, allows the application to read and write a block device like a char device—it permits the transfer of any number of bytes at a time. As a result, block and char devices differ only in the way data is managed internally by the kernel, and thus in the kernel/driver software interface.

Like a char device, each block device is accessed through a filesystem node, and the difference between them is transparent to the user. Block drivers have a completely different interface to the kernel than char drivers.