

Technical Case Study:  
**Hardware Interface - Device Drivers  
For  
Infrastructure Management Systems**

This is subsequent to Summary of Project –

VMware - ESX Server to Facilitate:  
IMS, Server Consolidation, Storage & Testing with Production Server



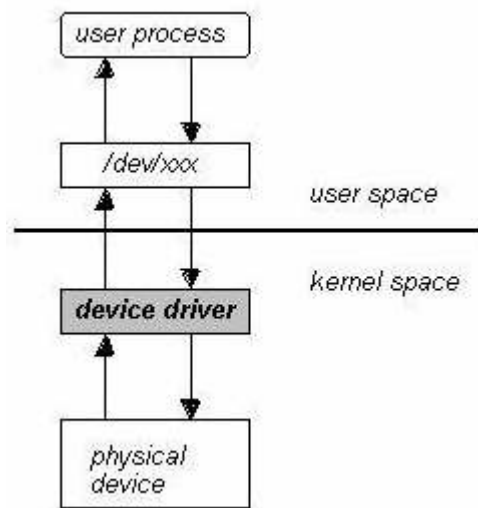
**VAssure** | Virtualization Labs | trRIMS | Offshore-QA | BI | Portals

<http://www.vassure.com>

## Introduction:

The purpose of device driver is to handle requests made by the kernel with regard to a particular type of device. There is a well-defined and consistent interface for the kernel to make these requests. By isolating device-specific code in device drivers and by having a consistent interface to the kernel, adding a new device is easier.

A driver acts like a translator between the device and programs that use the device. Each device has its own set of specialized commands that only its driver knows. In contrast, most programs access devices by using generic commands. The driver, therefore, accepts generic commands from a program and then translates them into specialized commands for the device.



**Figure 1: Driver acts like a translator**

This project mainly deals with how to write a device driver program for new device to access to the ESX Server and to make the driver compatible with the ESX Server.

VMware ESX Server delivers high performance I/O for PCI-based SCSI, RAID, Fibre Channel, and Ethernet controllers. To achieve high performance, these devices are accessed directly through device drivers in the ESX Server machine.

VMware - ESX Server device drivers that support specific devices within the following families of I/O controllers are:

- SCSI Controllers (Attached to Non-Removable SCSI Devices Only)
- RAID Controllers
- Fibre Channel Adapters (SCSI Protocol Support Only)
- Ethernet NICs

If any new device is to be attached to the ESX Server, driver should be developed for that device and then attached to the ESX Server

Sometimes modification needs to be done on the drivers for better performance

Reasons for modifies the drivers:

- To make the driver compatible with the ESX Server
- To tune the driver for performance
- To fix generic bugs in the driver

A device driver is a software module that resides within the kernel and is the software interface to a hardware device or devices. In general, there is one device driver for each type of hardware device.

## Device Drivers can be classified as:

- Block device drivers
- Character device drivers (including terminal drivers)
- Network device drivers
- Pseudo device drivers

### Block Device Driver

A block device driver is a driver that performs I/O by using file system block-sized buffers from a buffer cache supplied by the kernel. The kernel also provides for the device driver support interfaces that copy data between the buffer cache and the address space of a process.

Block device drivers are particularly well-suited for disk drives, the most common block devices. For block devices, all I/O occurs through the buffer cache.

## Character Device Driver

A character device driver does not handle I/O through the buffer cache, so it is not tied to a single approach for handling I/O. You can use a character device driver for a device such as a line printer that handles one character at a time. However, character drivers are not limited to performing I/O one character at a time (despite the name "character" driver). For example, tape drivers frequently perform I/O in 10K chunks. You can also use a character device driver when it is necessary to copy data directly to or from a user process.

Because of their flexibility in handling I/O, many drivers are character drivers. Line printers, interactive terminals, and graphics displays are examples of devices that require character device drivers.

A terminal device driver is actually a character device driver that handles I/O character processing for a variety of terminal devices. Like any character device, a terminal device can accept or supply a stream of data based on a request from a user process. It cannot be mounted as a file system and, therefore, does not use data caching.

## Network Device Driver

A network device driver attaches a network subsystem to a network interface, prepares the network interface for operation, and governs the transmission and reception of network frames over the network interface. This book does not discuss network device drivers.

## Pseudo device Driver

Not all device drivers control physical hardware. Such device drivers are called "pseudo device" drivers. Like block and character device drivers, pseudo device drivers make use of the device driver interfaces. Unlike block and character device drivers, pseudo device drivers do not operate on a bus. One example of a pseudo device driver is the pseudo terminal or pty terminal driver, which simulates a terminal device. The pty terminal driver is a character device driver typically used for remote logins.

## When a Device Driver is called:

The kernel calls a device driver during:

- Auto configuration

The kernel calls a device driver (specifically, the driver's probe interface) at auto configuration time to determine what devices are available and to initialize them.

- I/O operations

The kernel calls a device driver to perform I/O operations on the device. These operations include opening the device to perform reads and writes and closing the device.

- Interrupt handling

The kernel calls a device driver to handle interrupts from devices capable of generating them.

- Special requests

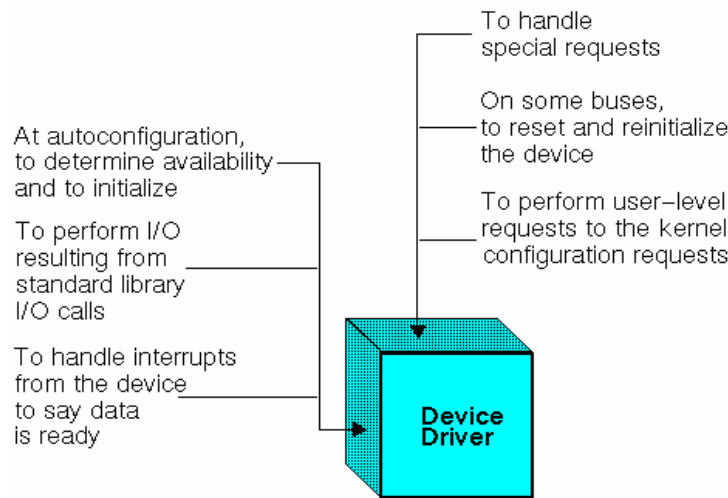
The kernel calls a device driver to handle special requests through ioctl calls.

- Reinitialization

The kernel calls a device driver to reinitialize the driver, the device, or both when the bus (the path from the CPU to the device) is reset.

- User-level requests to the sysconfig utility

The kernel calls a device driver (specifically, the driver's configure interface) to handle requests that result from use of the sysconfig utility. The sysconfig utility allows a system manager to dynamically configure, unconfigure, query, and reconfigure a device. These requests cause the kernel to call the device driver's configure interface. In addition, the driver's configure interface performs one-time initializations when called by the boot software or by the sysconfig utility.



**Figure 2: When the Kernel Calls a Device Driver**

## Device Driver Configuration:

Device driver configuration consists of the tasks necessary to incorporate device drivers into the kernel to make them available to system management and other utilities. After write device driver it needs to create a single binary module (a file with a .mod extension) from the driver source file (a file with a .c extension). After create the single binary module, it need to configure into the kernel so that this can test it on a running system.

There are two methods of device driver configuration:

- Static configuration
- Dynamic configuration.

**Static Configuration** consists of the tasks and tools necessary to link a device driver (single binary module) directly into the kernel at kernel build time. **Dynamic Configuration** consists of the tasks and tools necessary to link a device driver (single binary module) directly into the kernel at any point in time.

Auto configuration is a process that determines what hardware actually exists during the current instance of the running kernel at static configuration time. The auto configuration software calls the driver's probe, attach, and slave interfaces. Thus, the driver's probe, attach, and slave interfaces cooperate with the bus's configuration interface to determine if devices exist and are functional on a given system.

**Compiled by: Sridhar Gattu**  
[sridhar.gattu@vassure.com](mailto:sridhar.gattu@vassure.com)