

# Windows Device Drivers and Plug and Play

# Why device drivers?

- Operating systems must have an effective way to allow a user to work with computer devices/peripherals without worrying about the specifications of the hardware.
  - External hard drives
  - Game controllers
  - Printers
  - Video cards
  - Sound cards
- Operating systems cannot be expected to know how to control every device that the user wants.
- Analogous to operating systems abstracting system hardware.

# Ok, so what is a Driver?

- A driver is piece of software that allows the operating system to handle and interact with a computer device. Usually supplied by device manufacturer in the form of a CD or floppy disk.
- Design goal of device drivers is abstraction. Essentially needed to tell the operating system how to control a piece of hardware or a device by converting OS function calls into device specific calls.
- Performance of devices is dependant on the driver's code and how efficient it is in conjunction with the OS.

# Sounds simple!

- Yes. For the user.
- Majority of device drivers execute in kernel mode which means that driver writers must have an in-depth understanding of the platform they are coding for, both on the hardware and software level.
- Code for a driver must be as complete and efficient as possible. Any problems will be difficult and dangerous to examine.
- They must also be coded to the specific computer architecture
  - 16-bit (Windows 3.11)
  - 32-bit (Windows 2000/XP)
  - 64-bit (Linux and Windows Vista)
- Drivers are usually specific to the manufacturer hardware since they have the most knowledge of how the hardware works. Because of this, specific drivers are not useable by hardware other than what it is coded for.

# So how did Windows handle drivers?

- Windows 3.0 introduced virtual device drivers, or VxD. Previously, DOS applications had direct access to hardware and devices by reading and handling interrupts. This in turn led to conflicting resource usage since each DOS application ran as its own “virtual” machine and had to take more resources to handle them.
- These “drivers” did not actually drive devices. Starting with Windows 95, they allowed an application or hardware to run at “ring 0”, or an unprotected level, with full access to memory and system resources.

# What about current iterations of Windows?

- VxD's are no longer supported by windows. Rather, Windows 2000 and XP use what is known as the Windows Driver Model, or WDM.
- WDM exists in the intermediary layer of Windows 2000 kernel-mode drivers and was introduced to increase the functionality and ease of writing drivers for Windows. WDM standardized developer drivers for any device using Windows as well as base lining the amount of code needed.
- Communication occurs via I/O Request Packets, or IRP's. Any driver categorized as a WDM is layered in a hierarchy:
  - **Class drivers:** Provides an interface between the levels of the WDM architecture. Miniport and other class drivers are built upon this foundation. Can be loaded and unloaded whenever the user decides.
  - **Miniport drivers:** The more common devices: USB devices, Audio, SCSI, network adapters, printers. These drives are source and binary compatible, they allow functionality between Windows 2000 and XP without the need for device specific drivers.
  - **Software bus drivers:** Windows provides bus drivers for common buses: PCI, SCSI, USB, FireWire. Functionality is built into the actual bus. OS handles it.
  - **OS Services:** This is the layer that applies the OS functionality.
  - **Virtual device drivers:** Windows 98 and Windows Me only. Legacy hardware only.
- In the layered architecture of Windows kernel-mode drivers, class/mini port drivers are functional drivers.

# WDM Issues

- WDM is a great improvement over the simple resource access that VxD provided to applications and hardware, but it still suffers from problems for driver developers:
  - WDM has a very steep learning curve.
  - I/O cancellation is almost impossible to get right.
  - Thousands of lines of support code are required for every driver.
  - No support for writing pure user-mode drivers.

# And for the future?

- Rather than update the WDM, Microsoft has begun to implement the Windows Driver Framework, or WDF, for Windows Vista and beyond.
- Based off of WDM, but will look to address the issues with WDM. WDF will come in two types:
  - Kernel Mode Driver Framework (KMDF) for kernel device drivers.
  - User Mode Driver Framework (UMDF) for user devices.

# Plug-and-Play

- WDM can be seen as an almost pseudo Plug-and-Play (PnP) system: ability for the OS to support many different types of devices and/or peripherals without the need for handling hardware specifications.
- Difference lies in that WDM strictly works with having the OS handle drivers. True PnP devices require both software and hardware support, including the drivers the device uses, the Basic Input/Output System (BIOS), the physical bus line, and the device itself.

# Plug-and-Play

- PnP introduced with Windows 95.
- Tells the software (device drivers) where to find various pieces of hardware (devices) by establishing channels of communication between each physical device and its driver by allocating "bus-resources" in hardware:
  - I/O addresses
  - Memory regions
  - IRQs
  - DMA channels (LPC and ISA buses only).
- Note: difference between a USB PnP card and USB ports. Card is the actual PnP device, wired with a bus to your motherboard. USB ports cannot work without the card being configured.

# Plug-and-Play

- A true PnP device needs four requirements:
  - The OS must be PnP compatible
  - The device itself must be PnP compatible
  - The BIOS must support PnP
  - The device driver must be 32-bit or better
- PnP is actually a process that is loaded and executed when a computer boots up its BIOS. Thus, a machine must be turned off before the installation of a PnP device.
- Computer buses that the PnP device is being put into must also recognize interrupts. This is now built into the bus itself. (PCI, USB, FireWire)
- The OS must also be capable of handling the bus interrupts. It must watch the configuration and apply any changes by finding the correct driver.

# PnP Process

- Most of the work in loading a PnP device occurs in the BIOS when a computer boots up.
- The BIOS has a list of devices that it will boot in a specific order. When it reaches any PnP bus, it will follow this procedure:
  - Create a resource table of the available IRQs (Interrupt Requests), DMA channels and I/O addresses.
  - Identify all devices using a PCI or ISA bus.
  - Load the last known system configuration from the ESCD (Extended System Configuration Data). This configuration is stored in BIOS's nonvolatile memory from previous boots.
  - Compares a devices current configuration from the loaded one. If the configuration is the same, continue the booting process. If the configuration is new, the BIOS will reconfigure the system, by eliminating any additional system resources taken by non-PnP devices and reassigning them to reconfigure the PnP device in question.
  - Update the ESCD area by saving to it the new system configuration and continue the normal booting.

# A few examples of common PnP devices

- USB
- FireWire
- Modem
- CD-ROM Drives
- DVD Drives
- Computer printer
- Network Card
- Computer Keyboard
- Graphics Card
- Computer display
- Game controller
- Computer Mouse

# Conclusion (in my opinion)

- There is still a long way to go to achieve perfect or near-perfect device handling in computers today, but we have made leaps and bounds from DOS and VxD's direct application memory access and security threats that it posed.
- Windows XP has shown to be the most reliable device manager system yet for its users, but it still needs to improve its resource usage and the tools for driver developers.

# Bibliography

- <http://www.wikipedia.com>
- <http://www.pcguides.com/ref/mbsys/res/pnp.htm>