**Operating System Feature Comperison: I/O**

by Sinan Topkaya

CS 444 - Spring 2016

Abstract: This paper will examine I/O and functionalities such as data structions and algorithms. I will be examining how Windows implements them and also how FreeBSD implements them then compare it with how Linux implements them. This paper will mainly focus on different types of devices, I/O scheduling and the like.

# WRITING ASSIGNMENT - I/O COMPARISON

## INTRO

The term I/O is used to describe any program, operation or device that transfers data to or from a computer and to or from a device. Every transfer is an output from once device and an input into another. Devices such as keyboards and mouses are input-only, while CD-ROM is both input and putput device. I/O is implemented differently in FreeBSD, Windows and Linux. This paper will have the information about their similarities and their differance on how they implement I/O.

## WINDOWS

The Windows I/O System consists of many executive components that together manage devices.

I/O System Components

1) Uniform security and naming across devices to protect shareable resources.
2) High-performance asynchronous packet-based I/O to allow for the implementation of scalable applications.
3) Services that allow drivers to be written in a high-level language and eaasily ported between different machine architectures.
4) Layering and extensibility to allow for the addition of drivers that tranparently modify the bhavior of other drivers or devices, withour requiring any changes to the driver whole behavior or device is modified.
5) Dynamic loading and unloading of device drivers so that drivers can be loaded on demand and not consume system resources when unneeded
6) Support for Plug and Play, where the system locates and installs drivers for newly detected hardware, assigns them hardware resources they require, and also allows applications to dis-cover and activate device interfaces
7) Support for power management so that the system or individual devices can enter low power states.
8) Support for multiple installable file systems, including FAT, the CD-ROM file sustem(CDFS), the Universal Disk Format (UDF) file system, and the Windows file system (NTFS)
9) Windows Management Instrumentation (WMI) support and diagnosability so that drivers can be managed and monitored through WMI applications and scripts.[1]

To implement the above features Windows I/O system consists executive components as well as device drivers.

The I/O system includes the hardware abstraction layer(HAL) at the bottom, this insulates drivers from the specifics of the processor and interrupt controller by providing APIs that hide differentces between platforms. We can think of HAL as the bus driver for all the devices. The next layer includes the different Drivers, these drivers are then accessed by the I/O system layer which include, WDM WMI routines, PnP manager, Power manager and I/O manager.

The I/O system layer work closesly with each other.

Windows Management Instrumentation support routines, called the Windows Driver Model(WDM) WMI provider, allow device drivers to indirectly act as providers. I/O manager is the heart of the I/O system. It connects applications and system components to virtual, logical, and physical devices, and it defines the infrastructure that support device drivers. The Pnp manager works closely with the I/O manager and a type of device driver called a bus driver to guide the allocation of hardware resources as well as to detect and respond to the arrival and removal of hardware devices. The power manager also works closely with the I/O manager and the PnP manager to guide the system, as well as individual device drivers, through power-state transitions.

Windows supports a wide range of device driver types and programming environment. These drivers could be user-mode or kernel-mode drivers.

*Windows Drivers*

User mode drivers include windows subsystem drivers, that translate device-indipendent graphics requests to printer-specific commands. These commands are then typically forwarded to a kernel-mode port driver such as the universal serial bus(USB) printer port driver. User-Mode Driver Framework(UMDF) is also another user-mode driver that windows supports. These drivers are hardware device drivers that run in user mode. They also communicate to the kernel-mode.

Kernel-mode drivers include WDM driver, Layered Driver.

WDM Drivers are device drivers. WDM includes support for Windows power management, Plug and Play, and WMI, and most Plug and Play drivers adhere to WDM. There are 3 types of WDM drivers:[1]

1) Bus drivers manage a logical or physical bus. Examples of buses include PCMCIA, PCI, USB, and IEEE 1394. A bus driver is responsible for detecting and informing the PnP manager of devices attached to the bus it controls as well as managing the power setting of the bus
2) Function drivers manage a particular type of device. Bus drivers present devices to function drivers via the PnP manager. The function driver is the driver that exports the operational interface of the device to the operating system. In general, it is the driver with the most knowledge about the operation of the device.
3) Filter drivers logically layer either above or below function driver, augmenting or changing the behavior of a device or another driver. For example, a keyboard capture utility could be implemented with a keyboard filter driver that layers above the keyboard function driver.

Support for an individual piece of hardware is often divided among several drivers, each providing a part of the functionality required to make the device work properly. In addition to WDM bus driver, function driver, and filter driver, hardware support might be split between the layered driver and they are as follows: [1]

1) Class drivers implement the I/O processing for a particular class of devices, such as disk, keyboard or CD-ROM, where the hardware interfaces have been standarlized, so one driver can serve devices from a wide variety of mufacturers
2) Miniclass drivers implement I/O processing that is vendor defined for a particular class of devices
3) Port devices implement the processing of an I/O request specific to a type of I/O port, such as SATA, and are implemented as kernel-mode libraries of functions rather than actual device drivers.
4) Miniport drivers map a generic I/O request to a type of port into an adapter type, such as a specific netwrok adapter. Miniport drivers are actual device drivers that import the functions supplied by a port driver.

I/O Processing depends of the type of I/O in Windows. For example asynchronous I/O allows an application to issue multiple I/O requests and continue executing while the device performs the I/O operation. Synchronous I/O is when the application thread waits while the device performs the data operation and returns a status code when the I/O is complete. When used in their simplest form, Windows ReadFile and WriteFile functions are executed synchronously. Regardless of the type of I/O request, internally I/O operations issued to a driver on behalf of the application are performed asynchronously, that is, once an I/O request has ben initiated, the device driver returns to the I/O system.

Another usefull I/O is called the fast I/O. This is a special mechanism that allows I/O system to bypass

generating an IRP and isntead go directly to the driver stack to complete an I/O request. Mapped File I/O is an important feature of the I/O system, one that the I/O system and the memory manager produce jointly. Mapped file I/O refers to ability to view a file residing on disk as part of a process vitual memoery. A program therefore can access the file as a large array without buffering data. Scattered/Gather I/O is also supported by windows. This is a high performance I/O and avalaible through ReadFileScatter and Write FileGather functions.

```
// This is the ReadFileScatter function that is used for Scatter/Gather I/O
BOOL WINAPI ReadFileScatter(
  _In_       HANDLE                  hFile,
  _In_       FILE_SEGMENT_ELEMENT    aSegmentArray[],
  _In_       DWORD                   nNumberOfBytesToRead,
  _Reserved_ LPDWORD                 lpReserved,
  _Inout_    LPOVERLAPPED            lpOverlapped
);

//This is the WriteFileGather function that is used for Scatter/Gather I/O
BOOL WINAPI WriteFileGather(
  _In_       HANDLE                  hFile,
  _In_       FILE_SEGMENT_ELEMENT    aSegmentArray[],
  _In_       DWORD                   nNumberOfBytesToWrite,
  _Reserved_ LPDWORD                 lpReserved,
  _Inout_    LPOVERLAPPED            lpOverlapped
);
```

## FREEBSD

Properties of the I/O System in FreeBSD includes anonymozation of hardware devices, both to user and to the other parts of the kernel.

There are 3 main kinds of I/O and they are filesystem, socket interface and character device interface. Filesystem are actual files, they have hierarchical name space, locking, quotas, attribute management and protection. Socket interface deals with netwrok requests and the character device interface, allow user to access to devices in an unstructured way.

In FreeBSD device drivers are modules of code dedicated to gandling I/O for a particular piece or kind of hardware, they define a set of operation via well-know entry points.

### *A. Device Drivers*

Device Driver in FreeBSD are processes first by autoconfiguration and initilization, in this step the devices are probed and software state is initialized, this step is called only once. One this is done the Top Half is where system calls or VM systems are taken care of. The buttom hald(interrupt handlers) is where handlers are taken care of, and last but optional Error Handlig is where crash-dump routine is defined and only called when an unrecoverable error accurs. I/O queuing is means the data movement between Top and Buttom halves of the Kernel, typically the top hald, receives a request from user through system call, then records this request in data structure. After that it places data structure in queue and start the device f necessary. For many devices, they are defined as a byte stream fro example /dev/lp0 is the printer driver, /dev/tty00 terminal driver, /dev/mem memmory driver, there are also some non byte strea devices for example a high speed graphic device. Most of the drivers in FreeBSD has character interfaces except for networks.

FreeBSD offers 3 I/O solution when processing I/O operations, which are Polling I/O, which repeatly checks a set of descriptors for avaliable I/O, Nonbloking I/O, which completes immediately with partial results and finally signal-driven I/O, which notifies process when I/O becomes possible.[2]

## CONCLUSION

Without I/O, computers are useless. And when choosing the OS you want to use, knowing how the I/O functions in that OS is really important. The I/O should be able to make devices reliable, witohut any failures, and it should make them work in an efficient way. Overall and I/O should allow byte or block devices, these devices should be accessed either sequently or randomly. Some devices require contonual monitoring while other generate interrupts when they need service.

Overall all of the operating systems(Linux, Windows, FreeBSD) implement I/O in a very similar way, therefore I think the most important thing to think of is the performance reasons. In Windows kernel has a ”graphical subsystem” dedicated for graphics processing while linux does not have it. The FreeBSD bootloader can load binary drivers at boot-time. This allows third party manufacturers to distribute binary-only driver modules that can be loaded into any FreeBSD. This is bit different to the Linux because open-

source has to be used for Linux. While Microsoft allows third party devices without a problem. Although Windows and Linux are really different in implementation of I/O, they also have alot of similarities. For example in Linux tasklets are non preemtible procedures called with iterrupts enabled, however Windows offers the same feature which is called Windows Deferred Procedure Calls (DPCs).

## REFERENCES

[1] D. S. Mark Russinovich and A. Ionescu, *Windows Internals, Part 1 6/e*. Microsoft Press, 2012.

[2] M. K. McCusick and G. V. Neville-Neil, *Design and Implementation of the FreeBSD Operating System 2/e*. Addison-Wesley, 2015.