Differences and similarities in I/O and provided functionalities between the Linux, Windows, and FreeBSD Operating Systems

Alex Hoffer

CS 444
May 2017
Spring 2017

Contents

1 Windows

1.1 I/O

1.1.1 Similarities to Linux

Both operating systems have a design goal to provide what [3] calls an "abstraction of devices", meaning a universal set of software tools by which the programmer can manage I/O functions. Additionally, both operating systems implement, in their own way, a *Hardware Abstraction Layer* (HAL), which is a generalized interface that allows programs to directly manipulate device hardware [4]. Note, with the HAL as a perfect example, both operating systems' inclusion of abstraction in their design decisions. Linux and Windows both desire to be able to manage the I/O requests of a large variety of devices and file systems, since they are intended to be commercially used operating systems. While they share this design principle, the way in which they approach this abstraction is the source of a lot of differences between the two systems.

1.1.2 Differences from Linux

A key difference between Windows and Linux that explains further differences is the fact that Windows and its kernel are separated and are only combined through the use of calls to the Application Binary Interface (ABI) [1], while Linux's device drivers are located within the kernel itself and not relying on calls to the ABI [2]. Programmatically speaking, in Windows, files and devices are managed as *objects* in the object-oriented paradigm, while in Linux files and devices are managed through the use of *file descriptors*, further exemplifying the classic UNIX saying that "Everything is a file". Additionally, Windows' I/O has three device driver layers that each I/O request can be subjected to (though a request can be handled by merely one of these layers). These three layers are *filter, function,* and *bus* [3]. Consider the following block of Microsoft C code courtesy of Microsoft's Github that initializes a function driver for a Bluetooth device:

```
NTSTATUS
DriverDeviceAdd(
    IN  WDFDRIVER        _Driver,
    IN  PWDFDEVICE_INIT  _DeviceInit
    )
```

On the other hand, Linux's drivers are defined as being *block*, *character*, or *network*, and an I/O request can't be processed by all three. Since Windows' kernel doesn't have any device driver development, it instead relies on the *Windows Driver Model* (WDM) to provide device drivers. When a piece of software in Windows requests an I/O operation, the Windows kernel dispatches the request to its *I/O Manager*, which translates the request into an *I/O Request Packet*, or IRP, which is then sent to the device driver layers. The function layer consists of the predominant drivers that map programming interfaces to specific devices, the bus layer helps out device hosting bus controllers, and the filter layer offers extra IRP processing. Meanwhile, in LinuxLand, these device drivers are wildly different: the character device driver type handles simple devices that can be read one byte at a time, the block device driver type handles complex devices whose requests come in the form of blocks of data, and networking options are there for moving data to and from a network.

1.2 Provided functionality

1.2.1 Similarities to Linux

In both operating systems, their respective *Application Programming Interfaces* (APIs) are written to respond to *events*, which is when a user wants something to happen or the device wants to pass a message to the I/O manager. Windows has three provided functionalities for programmers

to allow communication between Userland and I/O drivers: *buffered I/O*, *direct I/O*, and *memory mapping* [1]. Linux also provides these same three functionalities [2].

1.2.2 Differences from Linux

I

2 FreeBSD

2.1 I/O

2.1.1 Similarities to Linux

2.1.2 Differences from Linux

I

2.2 Provided functionality

2.2.1 Similarities to Linux

2.2.2 Differences from Linux

I

References

[1] Johnson M. Hart. *Windows System Programming (3rd Edition)*. Addison-Wesley Professional, 2004.

[2] Michael Kerrisk. *The Linux Programming Interface: A Linux and UNIX System Programming Handbook*. No Starch Press, San Francisco, CA, USA, 1st edition, 2010.

[3] Mark E. Russinovich, Alex Ionescu, and David A. Solomon. *Windows internals, Part 2*. Microsoft, 2012.

[4] Mark E. Russinovich, David A. Solomon, and Alex Ionescu. *Windows internals*. Microsoft Press, 2012.