Writing Assignment 2

Isaac Stallcup

CS 444

Spring 2017

1 Introduction

2 Windows

Please note: all Windows material is from [1]. This statement is designed to replace a citation [1] after every single sentence for reading pleasure, but feel free to mentally insert one before each period in the following sections on Windows. They have been inserted for you in some cases.

2.1 I/O

At the heart of the I/O system in Windows, the I/O Manager connects components of I/O system together to ensure their proper functioning, communicating via a packet-based system of data structures called I/O Request Packets. The I/O Manager defines the model by which I/O requests are delivered to device drivers [1].

Device drivers help smooth communication to and from connected devices. They translate high-level commands from devices into low-level commands to the operating system. They do this by relaying instructions from the I/O Manager to devices and communicating the results of these commands back to the I/O Manager [1].

2.2 Data Structures

One data structure that the I/O system uses is the I/O Request Packet. An IRP stores information required to process an I/O request. Constructed when an I/O request is made, the packet allocates and initializes the IRP before storing a pointer to the *file object* of the call responsible for generating the I/O request [1].

File objects are represented by data structures implemented in the kernel that represent handles to devices. They contain a ”name” (the actual file that the file object refers to) as well as location data, permissions, information for I/O priority and scheduling, and a pointer to the device associated with it [1].

When file objects are called, Windows must determine which driver to use to handle the file object based on the its name. Driver objects are data structures that handle these operations, and in turn create device objects. These device objects are also data structures representing physical or logical devices on the system. Driver objects can create and be connected to multiple device objects in order to fulfill whatever requests the file object makes. When a driver object's last device object has been deleted, the driver object is unloaded [1].

Essentially:

IRPs are created then given a file object, which is a handle to the file associated with the IRP. The file object, when called, causes a driver object to come into being, which spawns the required device object for the needs of the file object [1].

2.3 Types of Devices

There are several types of device drivers, divided into two broad categories: user- and kernel-mode drivers. User-mode drivers are divided into printer drivers (which unsurprisingly deal with functionalities associated with printing) and User-Mode Driver Frameworks. User-Mode Driver Frameworks are a large group of hardware devices that interact with the user [1].

The kernel-mode drivers are file system drivers, Plug and Play (PnP) drivers, and non-Plug and Play drivers. File system drivers deal with I/O to files and communicate with storage and networking devices. PnP drivers deal with hardware including mass storage devices, video and audio devices and networking adapters. Non-PnP drivers deal with kernel functionality (essentially everything else) [1].

2.4 I/O Scheduling

In Windows, I/O Scheduling falls under the umbrella of I/O Prioritization. I/O Prioritization not only includes scheduling but determines which processes get preferential treatment throughout the scheduling process (usually those that deal with user interface) and those that do not (background processes like antivirus or search indexing) [1]. The levels of priority available in Windows are (in order of descending urgency) critical, high, normal, low and very low.

Windows has two methods to schedule priority-ranked processes: hierarchy prioritization and idle prioritization. The former handles processes of all priority except very low, and sorts each incoming process into a queue based on its priority, grouping like to like (all processes of a similar priority are grouped together). All critical priority processes must be completed before the high priority queue is dealt with, and all high priority processes must be completed before the normal queue comes up, and so on. Each queue is processed in a FIFO manner [1].

Idle prioritization uses a separate queue for the lowest priority I/O requests. Starvation is prevented via a mechanism that requires at least one idle priority I/O request be processed per unit time. Additionally, a delay occurs after after the last idle priority I/O before another is sought in order to prevent the low priority I/O requests from arising amidst high priority requests [1].

3 FreeBSD

3.1 I/O

3.2 Data Structures

3.3 Algorithms

3.4 Cryptography

4 Comparisons to Linux

Fig. 1. ”Stuff”

```
printf("Hello, World!");
```

References

[1] A. I. Mark Russinovich, David A. Solomon, *Windows Internals, 6th edition, Part 2.*