Writing Assignment 2

Alec Merdler
CS 444: Operating Systems II
Spring 2017

1 Introduction

Computers are fundamentally useless without the ability to receive different inputs and emit unique output. Whether the data comes from a hard drive, solid state drive, mouse, keyboard, or virtual reality headset, the operating system needs to be able to translate and deliver the data to the various programs that need it. Conversely, programs running on the operating system need a way to provide feedback back out to devices. The design and implementation of handling I/O differs between three major operating systems: Windows, FreeBSD, and Linux. However, there are also many similarities amongst them as well.

2 Windows

2.1 I/O Components

The Windows I/O System consists of many executive components that together manage devices. The Windows operating system uses a number of executive components that work together to manage input and output devices connected to the computer [1].

Security Uniform security and naming across devices in order to protect shareable resources.

Asynchronous High performance asynchronous packet based I/O to allow for the creation and maintenance of scalable applications.

Multiple Platforms Services that allow drivers to be written in a high level language and easily ported between different machine architectures.

Extensibility Layering and extensibility to allow for the addition of drivers that tranparently modify the behavior of other drivers or devices, without requiring any changes to the driver whole behavior or device is modified.

Dynamic Loading Dynamic unloading and loading of device drivers so that they can be loaded on demand and will not consume system resources when not in use.

Plug and Play 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 discover and activate device interfaces.

Power Management Support for power management so that the system or individual devices can enter low power states.

File Systems 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).

Driver Tools Windows Management Instrumentation (WMI) support and diagnostics tools so that drivers can be managed and monitored through WMI scripts and programs.

2.2 I/O Processing

Windows provides a hardware abstraction layer (HAL) at the lowest level in order to encapsulate the different processor and interrupt signals and instead expose a uniform set of application programming interfaces (APIs) to device drivers. The next layer is the actual device drivers. This layer is followed by the I/O system layer, which includes the various executive components listed in the previous section. Most I/O operations in Windows are synchronous, which means the application thread will block while the device performs its data operations. Asynchronous I/O operations are enabled by specifying the FILE-FLAG-OVERLAPPED flag when calling the CreateFile function. This will allow the application to continue executing while the device performs the I/O operation [1].

```
HANDLE WINAPI CreateFile(
  _In_     LPCTSTR               lpFileName,
  _In_     DWORD                 dwDesiredAccess,
  _In_     DWORD                 dwShareMode,
  _In_opt_ LPSECURITY_ATTRIBUTES lpSecurityAttributes,
  _In_     DWORD                 dwCreationDisposition,
  _In_     DWORD                 dwFlagsAndAttributes,
```

```
  _In_opt_ HANDLE                hTemplateFile
);
```

Above is the implementation of the CreateFile function. This function is used to create or open files or other I/O devices. The Windows scheduler processes the file descriptors using a C-LOOK algorithm. Once the list contains requests, the drivers will iterate through the list from the lowest to highest sector [2].

3 FreeBSD

3.1 I/O Components

There are multiple different types of file descriptors that are used for difference processes. These file descriptors types and uses are as follows:

- PIPE - pipe
- FIFO - named pipe
- SEM - POSIX semaphore
- PTS - pseudo-teletype master device
- DEV - device not reference by a vnode
- MQUEUE - POSIX message queue
- KQUEUE - event queue
- CRYPTO - cryptographic hardware
- SHM - POSIX shared memory
- PROCDESC - process
- VNODE - file or device
- SOCKET - communication endpoint

These file descriptors are important because each data structure that is used to implement the I/O scheduler has one of these types. Also, there are multiple different flags that go along with these file descriptors that will tell the operating system whether to block or not or if the process will be asynchronous or synchronous [3].

3.2 I/O Processing

Below is the implementation of the standard disk scheduler in FreeBSD. A significant difference between this scheduler and the Linux scheduler is the ability to delete, load, and unload. Delete is used to free resources when a scheduler is not in use. Load/unload are used can hook into the respective scheduler events to perform specific operations. Every time init is called, it will allocate a struct containing device information, which is referenced using a file pointer [3].

```
struct _disk_sched_interface {
       struct _disk_sched_interface *next;
       char *name;     /* symbolic name */
       int refcount;   /* users of this scheduler */

       void (*disksort)(struct buf_queue_head *head, struct buf *bp);
       void (*remove)(struct buf_queue_head *head, struct buf *bp);
       struct buf *(*get_first)(struct buf_queue_head *head);
       void (*init)(struct buf_queue_head *head);
```

```
        void (*delete)(struct buf_queue_head *head);
        void (*load)(void);     /* load scheduler */
        void (*unload)(void);   /* unload scheduler */
};
```

[4]

4 Conclusion

The design and implementation of I/O in FreeBSD is very similar to Linux, in that they both use C-LOOK algorithm. All three operating systems provide file descriptors to specify asynchronous data operations, but only FreeBSD requires the type of descriptor. Windows has several layers of abstraction, including the hardware abstraction layer (HAL) to provide a uniform interface for all device drivers across different hardware architectures. This is similar to the Linux kernel itself, which provides a uniform set of APIs across countless devices. FreeBSD also has a form of HAL.

References

[1] M. R. D. Solomon and A. Ionescu. (2012) Windows internals, part 2 6/e.

[2] Microsoft. Createfile function. [Online]. Available: https://msdn.microsoft.com/en-us/library/windows/desktop/aa363858(v=vs.85).aspx

[3] M. K. McKusick and G. V. Neville-Neil, *The Design and Implementation of the FreeBSD Operating System*, 2nd ed. Boston: Addison-Wesley Print, 2015.

[4] M. K. McCusick and G. V. Neville-Neil. (2015) Design and implementation of the freebsd operating system 2/e.