Researching of the VMCS

Fall 2017 CS544 Operating System

Weijie Mo

November 10th 2017

I. Introduction

As we all know, Windows, Linux, FreeBSD are three common operating systems, and I/O, I/O scheduling is fundamentally what a computer was designed to do. Therefore, in this paper, I want to talk about I/O, its basic concepts like data structure, algorithms, types of devices, scheduling, and how its processed and scheduled between Windows and FreeBSD, and how Windows and FreeBSD implements devices and drivers. Then I would be comparing and contrasting them to Linux, distinguish their differences and similarities, and provide some code to identify them better.

II. Comparison

In a modern operating system, I/O(Input/Output) is an essential aspect of computer to complete various functions. Processor is responsible for executing a variety of computing tasks, and through the memory to control the whole memory space, but as a practical computer system, these functions are not enough, it also needs to be connected to other external equipments so that could make computer really ”useful”. Although the functions and purposes of these devices vary, the processor takes standard interface technology to deal with them through the device’s controller. Therefore, from the hardwares aspect, the interface technology supported by the processor is the basis for computer system to work together.

Windows, as a general operating system, provides an extensible I/O processing framework that allows third-party hardware companies to produce specialized system modules to control their hardware devices. This I/O processing framework is known as the Windows I/O model. I/O manager, PnP manager, Power manager, WMI service and device drivers formed the I/O system in Windows. The I/O manager is the heart of the I/O system, it connects applications and system components to virtual, logical, and it defines the infrastructure that supports device drivers[1]. There are mainly four types of data structures related to I/O system, File object, driver object, device object and I/O request packet.[1] The I/O requests are issued by user processes in the form of I/O request packets (IRP) to the I/O manager. IRPs data structure is similar with the File Entry structure in the FreeBSD, however, the structure itself is not the same. Synchronous and Asynchronous operations both exist in Windows and FreeBSD.

Windows Device object represents the device in the logical or physical space in the system, also contains information such as location, request queue. The I/O manager creates a driver object when a driver is loaded into the system, and it then calls the drivers initialization routine (DriverEntry), which fillss in the object attributes with the drivers entry points[1]. Similar to the FreeBSD operating system, Device drivers consist of a set of routines that are called to process the various stages of an I/O request[1]. Windows supports a wide range of device driver types and programming environments. Even within atype of device driver, programming environments can differ, depending on the specific type of device for which a driver is intended. The broadest classification of a driver is whether it is a user-mode or kernel-mode driver. Windows supports a couple of types of user-mode drivers:Windows subsystem printer drivers;User-Mode Driver Framework (UMDF) drivers[1]. There are also many types of kernel-mode drivers, which can be divided into the following basic categories:File system drivers;Plug and Play drivers;NonPlug and Play drivers[1].The File system drivers is similar to FreeBSD, however, plug and

play drivers and non-plug and play drivers that is unique to Windows. The Windows subsystem printer devices is similar to character devices implemented in Linux and FreeBSD and also User mode driver Framework.

As for FreeBSD, it is unlike windows uses descriptors to do all user process I/O. Descriptors just are small unsigned integers obtained from the system calls. For user processes, all I/O are done through descriptors[2]. System calls that refer to open files take a file descriptor as an argument to specify the file. The file descriptor is used by the kernel to index into the descriptor table for the current process (kept in the filedesc structure, a substructure of the process structure for the process) to locate a file entry, or file structure[2]. The file entry provides a file type and a pointer to an underlying object for the descriptor. For data files, the file entry points to a vnode structure that references a substructure containing the filesystem-specific information[2]. Device drivers in windows are Kernel-Mode Driver Framework and User-Mode driver framework. The is contains a similar queue I/O structure that is implemented in both FreeBSD and Linux.In FreeBSD, all devices are managed by the DEVFS filesystem[2]. When a program accesses a device directly by calling the open() system call with a path that terminates within the DEVFS filesystem, such as /dev/cu, the DEVFS filesystem searches for a matching entry in its internal list of devices and, if it finds a match, calls the the open() routine that is present in the devices cdevsw[2]. The kernel provides a generic disksort() routine that can be used by all the disk device drivers to sort I/O requests into a drives request queue using an elevator sorting algorithm[2]. Here is a part of the algorithm of disksort():

```
void disksort(
drive queue *dq,
buffer *bp);
{
if (active list is empty){
place the buffer at the front of the active list;
return;
}[2]
```

There are two types of I/O devices in Linux, character and block. Character devices only have one position the current one whereas block devices must be able to navigate back and forth between any location on the media. As for block devices, a sector is the smallest addressable unit on a block device, The sector size is a physical property of the device, and the sector is the fundamental unit of all block devices the device cannot address or operate on a unit smaller than the sector, although many block devices can operate on multiple sectors at one time[3]. When a block is in memory, it is called a buffer. Buffer contains a buffer head which is also called descriptor. As for I/O schedulers, Linux uses elevator algorithms. It sorts and merges nearby sector values so that all requests are optimally serviced. There are various schedulers implemented on the elevator algorithm. First is the deadline scheduler that has 3 queues(read, write, merged) and ensures that no request starves. Following this we have the Anticipatory scheduler that tries to reduce the seek storm that accompanies read requests. Besides, the simplest I/O scheduler in Linux kernel, Noop. As for data structure in Linux, the simplest and common one is obviously linked list. Here is a part of code to introduce linked list:

```
Struct node *{
void * data;
struct node*next;
```

struct node*prev;

};

The other one is Queue data structure, it also is implemented in FreeBSD system, because file entry could point to a pipe that implemnts a kfifio structure.

REFERENCES

[1] Windows internals. *M. Russinovich, D. Solomon, A. Ionescu and M. Pietrek. Microsoft, 2012*.

[2] The Design and Implementation of the FreeBSD Operating System. *Marshall Kirk McKusick, George.V.Neville-Neil, Robert N.M. Watson. Addison-Wesley Professional; 2 edition, September 15, 2014*.

[3] Linux Kernel Development. *Robert Love, 2010*.