Writing 2:
I/O and provided functionality among Linux, Windows, and FreeBSD

Webster Cheng

May 8th, 2018

CS544
Operating Systems II
(Spring 2018)

Abstract

This article describe how Windows and FreeBSD implements I/O and provided functionality, and compare each to Linux. The article contain some codes to support the article.

1 INTODUCTION

In computer, I/O is critical communication between users to hardware. In addition, managing I/O of data stream is a very important issue that need to handle variety devise and maximize the computer resource utilization. In the article, there will describe Windows and FreeBSD I/O design and compare to Linux I/O design.

2 WINDOWS

According the Windows on-line document, they separate their I/O drivers into user-mode drivers and kernel-mode drivers for security concern that is same as process design [1]. For kernel I/O, they use several I/O scheduling to manage I/O process such as Synchronous, Asynchronous, Fast I/O, Mapped File I/O and Scatter/Gather I/O [2]. All the windows I/O request are using I/O request packet (IRPs) processing including tradition I/O request [1]. IRP is the critical design in Windows I/O system and there syntax of structure can be seen as figure 1. Furthermore, drivers are leveled such as Lowest-Level driver and Intermediate-Level driver. Also, they set a standard procedure for all drivers to follow [1]. In addition, they present their devices and hardware as objects that I/O manager can find appropriate objects to complete the work easily [1].

```
typedef struct _IRP {
  PMDL              MdlAddress;
  ULONG             Flags;
  union {
    struct _IRP  *MasterIrp;
    PVOID        SystemBuffer;
  } AssociatedIrp;
  IO_STATUS_BLOCK IoStatus;
  KPROCESSOR_MODE RequestorMode;
  BOOLEAN           PendingReturned;
  BOOLEAN           Cancel;
  KIRQL             CancelIrql;
  PDRIVER_CANCEL  CancelRoutine;
  PVOID             UserBuffer;
  union {
    struct {
    union {
      KDEVICE_QUEUE_ENTRY DeviceQueueEntry;
        struct {
        PVOID DriverContext[4];
        };
      };
      PETHREAD    Thread;
      LIST_ENTRY ListEntry;
    } Overlay;
  } Tail;
} IRP, *PIRP;
```

Code. 1 Windows IRP structure [3]

As above mention, Windows have four different types of I/O. First of all is Synchronous and Asynchronous I/O. Synchronous is the program waits until the I/O request completed and return a result code. Asynchronous design is for the multiple I/O requests and the I/O perform won't interrupt executing program [2]. The default of program is synchronous I/O perform that is logical for some applications which need complete data to work. Fast I/O is an I/O perform that generates an IRP and interact to the driver directly [2]. Mapped File I/O is an important design to Windows that the user can manage a storage of device by referring a segment of virtual memory [2]. This design support a huge file I/O performance efficient but it waste slack space for a small size of file [4]. `CreateFileMapping` and `MapViewOfFile` funcitons let the Mapped Filed I/O feature can be done in the user mode. Scatter/Gather I/O is special feature in Windows. Using these function, a program can send one I/O request to separate buffer in virtual memory [2].

Windows driver usually follow the Windows Driver Model (WDM) which allows driver can be used in all Windows operating system [5]. According to Windows on-line document, there are three type of drivers which are the bus driver, filter drivers and individual devices. First is bus driver that is used for most common buses such as USB, PCI and SCSI. Individual devices are run by function drivers which is similar as character driver. Those character devices mean un-buffer disk. Filter drivers are used to filter I/O requests for devices or buses. In Windows, filter drivers have three different type of drivers which are bus filter drivers, lower-level filter drivers, and upper-level filter drivers. Those design that driver can fit into Windows user and kernel mode design.

2.1 FreeBSD

FreeBSD has similar design as Unix in many points including I/O system. FreeBSD have three main kinds of I/O filesystem, character-device interface and socket interface [6]. Filesystem is for actual files I/O request that data stream represent a simple linear array of bytes. In character-device interface, it will access to devices without structured. Under the character-device interface have two types of device. First is exactly character device such as terminals and second is block devices such as disks and tapes. The character device files are created by `mknod` function that allow the special device to be accessed or modified. FreeBSD elevator algorithm is `disksort` function as seen as code 2. There also have some common function for user to manage the I/O streams such as open, read, write and close that are similar to most operating system design. In FreeBSD, they thought their I/O subsystem as a stack that contain user requests on the top of stack and put the devices at bottom [7]. FreeBSD I/O stack are system call interface, active file entries, OBJECT/VNODE, filesystems, page cache, GEMO, NVMe, and drivers from top to the bottom [7]. They have two kind of scheduler Common Access Method and GEOM [7]. The GEOM modular is for disk-I/O request. They support enter to the operating system by supporting various software RAID configurations. In the GEMO layer, the I/O requests are changed from `struct buf` or `struct bio` that are from system calls [8]. The default of GEMO scheduling is FIFO. The scheduler of common access method CAM can control data stream accurately by identified various type of device to approach the efficient performant. Except of disks I/O request, CAM handle all of I/O requests. In CAM, they separate their devise into SCSI-based disks and ATA-based disks that use same scheduling policy.

```
1 DiskSort(drive queue *dq, buffer *bp)
2 {
3   if(active list is empty)
```

```
4   {
5     place the buffer at the fromt of the active list;
6     return;
7   }
8   if(request lies before the first active request)
9   {
10    locate the beginning of the next-pass list;
11    sort bp into the next-pass list;
12  }
13  else
14  {
15    sort bp into the active list;
16  }
17 }
```

FreeBSD DiskSort pseudo code [6].

3 Compare to Linux

Linux have its own way to handle I/O requests such as block device request queue [9]. Linux I/O scheduler provide different type of I/O scheduling – anticipatory, deadline, completely fair queuing, and no-op schedulers. Some of scheduler are similar as Windows and FreeBSD schedulers such as no-op scheduler which is FIFO. The default of Linux scheduler is CFQ which support synchronous and asynchronous requests same as Windows I/O. Linux and FreeBSD both have block and character devices classification. However, Windows have one different type of device which is filter device classification. In I/O request structure, Linux and FreeBSD have similar design because of base on Unix that are different than Windows I/O structure design. In Windows, they have priority to manage their I/O request that are different than Linux and FreeBSD. The most different between Windows and Linux is Windows spate their I/O process into two user and kernel layer. Also, Windows uses I/O priority that Linux didn't have it in there system. The most similar part of three operating system is they all provide the abstract interface for user interactive with various type of devices.

REFERENCES

[1] microsoft.com, *Overview of The Windows I/O Model*. [online] https://docs.microsoft.com/en-us/windows-hardware/drivers/kernel/overview-of-the-windows-i-o-model [Accessed: May 8, 2018].

[2] microsoft.com, *Understanding the Windows I/O System*. [online] https://www.microsoftpressstore.com/articles/article.aspx?p=2201309&seqNum=3 [Accessed: May 8, 2018].

[3] microsoft.com, *IRP Structure*. [online] https://msdn.microsoft.com/en-us/library/windows/hardware/ff550694(v=vs.85).aspx [Accessed: May 8, 2018].

[4] wikipedia.org, *Memory mapped file*. [online] https://en.wikipedia.org/wiki/Memory-mapped_file [Accessed: May 8, 2018].

[5] microsoft.com, *Introduction to WDM*. [online] https://docs.microsoft.com/en-us/windows-hardware/drivers/kernel/introduction-to-wdm [Accessed: May 8, 2018].

[6] chapin, *I/O in FreeBSD*. [online] http://www.sai.syr.edu/čhapin/cis657/IO.pdf [Accessed: May 8, 2018].

[7] M. W. Losh, *I/O Scheduling in FreeBSD's CAM Subsystem*. [online] https://people.freebsd.org/ imp/bsdcan2015/iosched-v3.pdf [Accessed: May 8, 2018].

[8] M. K. McCusick,& V. George,*Design and Implementation of the FreeBSD Operating System*. Addison-Wesley, Inc. 2015.

[9] L. Robert,*Linux Kernel Development*. Pearson Education, Inc. 2010.