CS444 Written Assignment 2

Nicholas Skinner

May 9, 2018

Input and Output

General input and output is an essential component of all operating systems, At its core, an operating system will require an ability to manipulate data in one way or another in order to conduct a process, and subsequently report the information back to the user. Data given in this manner can be stored through the use of files, we're going to be looking more specifically at how FreeBSD, Linux, and Windows all manage their data, as well as whether or not the data is abstracted for the users sake or not. An example of this could be folders in linux, they can be opened like files, they can be viewed and edited in an editor such as vim. The files can br created, edited, stored or transferred on block or similar character devices. But before we get ahead of ourselves in talking about points of cryptography, scheduling, as well as algorithms that are used to manage devices, we will need to establish a basic definition of what a block is, as well as define what a character device is. Block devices exist as randomly accessible data that is stored in a fixed size chunk, an example of this could be a hard drive, where data is randomly accessible. Whereas character devices are in-order accessed individual bytes of data, which are also more colloquially referred to as data streams, an example of this would be a keyboard[1].

FreeBSD and Linux share a common ancestor, that ancestor being unix, as such, FreeBSD contains many structures that are similar to if not identical to linux Input/Output management systems[2]. An example of this is that a core component of the two operating systems is the use of files for any and all processes as well as system storage. This is achieved through usage of file descriptors, where these descriptors are accessed and connected via pipes and sockets, a process known as redirection. This system of using files for everything is abstracted to the user, and kept to the black box that is the kernel. To access the descriptors and extract data from them can be done in the same function as files for non system-level applications. Outside of a a few differences of variables contained, Data Structures used by FreeBSD for Input/Output use the same Block Input/Output structure that linux uses[3].

In terms of I/O, Windows and Linux both work extremely differently, these differences range from their management systems all the way to their data structures that are used to manage devices. Windows takes an approach with a variety of ways for data structures to manage requests, with the standard being IRP[4]. IRP itself stands for Input/output Request Packet, and it is a representation of any given request as it is being processed. It also contains vital information in the header, information that provides the system how to handle this request, this would contain information like is it synchronous/asynchronous, etc. Data that is not contained in the IRPs header will be in two to four different stack locations. Stack locations are the Linux Block Input/Output for page data, which is more colloquially called a file pointer. For Windows, the default system that handles IRP on a standard plane windows machine is the Windows Input/Output Manager. The Windows I/O manager serves as a staging area that transfers IRP over to the correct drivers and destinations. Windows I/O accomplishes this task by using a layered stack approach, where each driver gets placed in a cataloged, then a given IRP is sent down the list to interact with as many drivers as it needs. When the request has finally been fulfilled, the Input/Output manager notifies the application that had initated the IRP. IRPs themselves are fulfilled when one or all of the following conditions have been met: When the IRP is canceled, When the IRP contains invalid parameters, or when the IRP no longer needs to be passed down the driver stack[5]. Once one of those conditions have been met, and the IRP was initiated by the Windows I/O manager, the IRP will be freed by the Windows I/O manager, if it was not initialized by the I/O manager, it will instead be sent back to the driver it was allocated from to be reused.

Those that develop for FreeBSD Linux and Windows have a lot of freedom over whatever encryption method they wish to use, whether it be AES, Blowfish, or RSA. Windows, cryptographic API provides AES, RSA, DSS as well as a few others[7]. While Linux and FreeBSD use similar encryption algorithms to what Windows provides, however their implementations are nearly identical in semantics, with some key features being underlying data as well as naming schemes being the main difference between the two.

The FreeBSD, Linux, and Windows operating systems all abstract their underlying file-system management. This abstraction may be done through the use of I/O management processes like the Windows I/O manager, or it may be done by file descriptors like in FreeBSD and Linux. Overall, it seems that Windows has a lot more safety nets and restrictions in place for their I/O requests. However, FreeBSD and Linux opt to remove the safety nets, instead allowing the user to have more freedoms. When it comes to the data structures that are used for keeping track of said I/O requests, all three operating systems seem to use a form of structure that is similar to BIO from Linux-like IRPs. For cryptography, all three platforms are capable of using the same techniques when they encrypt devices, the only difference being the underlying implementation of the algorithm. These operating system platforms have fairly similar, if not identical approaches to resolving the I/O problem, with differences stemming from what control they allow the users to have.

References

[1] Robert Love. *Linux kernel development*. Addison-Wesley, 3 edition, 2015.

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

[3] Joseph Kong. *FreeBSD device drivers: a guide for the intrepid*. No Starch Press, 2012.

[4] Understanding the windows i/o system — microsoft press store. https://www.microsoftpressstore.com/articles/article.aspx?p=2201309&seqNum=3. (Accessed on 05/9/2018).

[5] When to complete an irp (windows drivers). https://msdn.microsoft.com/en-us/library/windows/hardware/ff565653(v=vs.85).aspx. (Accessed on 05/9/2018).

[6] Reusing irps (windows drivers). https://msdn.microsoft.com/en-us/library/windows/hardware/ff561107(v=vs.85).aspx. (Accessed on 05/9/2018).

[7] Cryptographic provider types (windows). https://msdn.microsoft.com/en-us/library/windows/desktop/aa380244(v=vs.85).aspx. (Accessed on 05/9/2018).