INTRODUCTION

|  |
| --- |
|  |
| General input/output is essential to all operating systems whether it be Linux, Windows, or Freebsd. At their core all of these operating systems require the manipulation of data in one form or another to conduct processes and display information back to the user. Data is managed through the use of files for the three systems specified, whether abstracted for the users sake or not. An example of this, is folders in Linux can be opened like files viewed or edited in a text editor like Vim or Emacs. These files can typically be stored, transferred, created, or edited on block or character devices. Before expanding on the various functionality and components of these devices such as cryptography, scheduling, and algorithms used to manage these devices, a basic definition of both block and character devices needs to be established. Block devices are simply randomly accessible fixed size chunks of data that can be accessed in any order, an example is a hard drive. Character devices are in-order accessed individual bytes of data, also known as a data streams, an example is a keyboard\cite{Linux\_book}.\\ |
|  |

|  |
| --- |
| WINDOWS TO LINUX |
| Windows and Linux both work extremely differently in terms of I/O, ranging from their management systems to data structures used to manage requests to a given device. **Windows** uses a variety of different options for data structures to manage request, the standard being IRP's (input/output request packets)\cite{Understa77:online}. These IRP's are a representation of any given request as it is processed. **As** well as containing essential header data that impacts the way the request is handled, an example is whether the request was made to be asynchronous or synchronous. **Finally** data that is not contained in a IRP header is two to four different stack locations\cite{Understa77:online}. **Stack locations** are the Linux BIO (block Input/Output) equivalent or a page of data otherwise simply known as a file pointer. **The default** system that handles IRP's for any given vanilla Windows system is generically called the Windows I/O manager. **This manager** serves as a middle man transferring IRP's from driver to driver to its correct destination. **This is** done by a layered stack approach, each driver is cataloged a given IRP is sent down the stack to either one or multiple different drivers. **When the** request has been fulfilled the I/O manager simply notifies the application that initiated the IRP. IRP's are Completed or fulfilled when one or all of the following conditions are met: it is cancel led, it is contains invalid parameters, or it no longer needs to be passed down the driver stack\cite{WhentoCo44:online}. After these conditions are met the IRP is freed by the Windows I/O manager or sent back to the driver it was allocated from to be reused, if and only if the request was not issued from the Windows I/O manager \cite{ReusingI59:online}.\\ |
|  |

FREEBSD TO LINUX

|  |
| --- |
|  |
| Unlike Windows, FreeBsd contains many similar structures and I/O management as Linux, because of connected lineage of the two operating systems. For example a core component of FreeBsd as well as Linux is the use of Files for any and all processes and system storage. This is achieved through the use of file descriptors, these descriptors are not managed by a processes similar to the Windows I/O manager. Instead these file Descriptors are accessed and connected via pipes and sockets otherwise known as redirection, Linux uses an identical system.\cite{freeBsdBook}. However this system of using files for everything is typically abstracted to the user and Kept in a black box that is the kernel. To access descriptors and extract data from them can be done via the same functions as files for non system level applications. The data structures used by FreeBsd for I/O use the same BIO (Block input/out) structure as Linux with some differences with variables contained\cite{kong\_2012}.\\ |
|  |

|  |
| --- |
| CRYPTOGRAPHY |
| Overall Linux, Windows, and FreeBsd developers have the freedom of using any encryption method whether it be AES (Advanced Encryption Standard), Blowfish, or RSA. However others contain less encryption methods installed by default. Windows, cryptographic API have AES, RSA, DSS, and and a few others\cite{Cryptogr64:online}. while others like FreeBsd and Linux use similar encryption algorithms to Windows, but their implementations semantic-wise are nearly identical, with key features such as naming schemes and underlying data structures being used being the main difference between the two operating systems.\\ |
|  |

|  |
| --- |
| CONCLUSION |
| For the Linux, Windows, and FreeBsd operating systems, they all abstract their underlying file-system management. This has been done though the use of I/O management processes like the Windows I/O manager, and File descriptors for Linux and FreeBsd. Overall Windows has a more hands on approach to I/O request management that gives programmers more safety nets when developing applications. However Linux and FreeBsd drivers remove those safety nets and allow more freedom and less restrictions. When it comes to the data structures used for keeping track of I/O requests all of the named operating systems use a form of structure that resembles a BIO from Linux like IRP's. Finally Linux, Windows, and FreeBsd use or can use the exact same cryptography techniques when encrypting devices, the only different being the underlying implementation of the given algorithm. As an Overview of I/O between operating systems they are nearly identical in most aspects with the out lier being Windows because of its non open source nature. |
|  |