%Introduction

An interrupt is a more performance efficient alternative to polling for operating system level processes, or programs within the user realm. Interrupts allow the program the ability to notify a CPU at any given time that a program or kernel critical charge requires attention, as well as a time slice, or quantum time from the CPU. Synchronization within the Kernel exists as an abstraction of interrupts on the kernel and user realm levels. Synchronizing permits multiple processes that are running concurrently to share data between each other without the risk of corrupting the data string by altering it mid-read/write. These abstractions of interrupts are implemented through a variety of methods including mutexes, semaphores, and spinlocks. These implementations do vary between operating systems though, seemingly by mixing up the underlying syntax of the implementation while retaining the same general idea. FreeBSD, Linux, and Windows all possess a few specific traits that distinguish their interrupts from each other, while all also retaining the standardized concept of an interrupt.

%FreeBSD

Within the realm of FreeBSD, interrupts are handled by two different methods: Trap Handlers, and System Calls. Trap handlers appear to normally occur because of some kind of unintentional error, some normal examples of these are an invalid pointer, or division by zero\cite(freeBsdBook}. On the occurrence that a trap handler catches an interrupt, it will begin its process by saving the current state of the erroring process. Then it will determine the interrupt information, then it checks the higher-priority processes to see if they will be affected by this call, then exits safely. When interrupts are born from a system call instead of a kernel level error within the system, they are capable of blocking data, causing mutual exclusion (Also known as a mutex) within the kernel. In the freeBSD kernel, software interrupts are given the highest priority, and appear to typically be used for networking devices due to their random retrieval of packets\cite{freeBSDBook}, whereas hardware interrupts are a tier below network, and will be serviced after networking interrupts. When an interrupt is to be serviced, it will require a full service switch by the processor in order for it to begin assigning Time slices, or quantums.

Synchronization within FreeBSD contains many of the same naming schemes as its Linux counterpart, but the mechanics it abstracts are strikingly different. An example of this is in user realm programs, programmers are given an option to use assets like mutexes to ensure that only on thread can access data at a time. Within FreeBSD there are rules to prevent the deadlocking of processes, and to achieve this, threads with access to multiple areas of data are only permitted one mutex lock at a time \cite{43Contex5:online}. This exists as a fail safe, and is not implemented by the kernel in linux. There exist alternative options for interrupts within FreeBSD as well, programmers are allowed a lighter interrupt known as the INTR\\_FAST flag, this flag appears to have been used more frequently within earlier versions of the FreeBSD kernel, but now goes underused as it required additional used memory and process time\cite{83Genera16:online}.

%Wandows

Inside the domain of Windows, interrupts are handled through a trap dispatching mechanism. This method allows for a driver to force a processor to suspend what it is currently doing in order to execute an action on a freshly executing thread. Once an interrupt is caught by the trap handler, it is sent to get dispatched by interrupt service routines, not unlike Linux. Interrupts are also indexed the Interrupt Dispatch table, similar to the Linux system call table it allows programs within the user realm to create calls to a function by its interrupt number in order run some form of subroutine, whether that be a function call, or just retrieval of data. Trap handlers are also a vital asset for Windows, as they can assist in informing the processor to differentiate if an interrupt is hardware or software related, as well as determine if it is able to run synchronously or asynchronously.

Software requests, while being less common than hardware interrupts, have their own special place on the Windows operating system. Some examples of tasks that are facilitated by software interrupts are: Asynchronous I/O, Thread Dispatch, and Timer Expiration, among many others. One specific classification of a software interrupt is named a ‘Dispatch or Deferred Procedure Call’ , which is otherwise known as a DPC. A DCP software interrupt is enacted when a thread is either sleeping or when it is terminated. The interrupt request is sent in order to facilitate a context switch\cite{win}. Windows also possesses an interesting feature applied to its interupt system, it has a large priority range from 0 to 31, and unlike Linux, the higher number will represent a higher priority. Generally, it appears that hardware interrupts occupy priorities that range from tier 3 to tier 31, whereas software interrupts have priorities of 2 or 1\cite{win}.

%Conclusion

The implementation of interrupts and synchronization between the platforms of FreeBSD, Linux, and Windows all appear to use generally similar constructs, with a few of the implementation differences existing as the methods of creating interrupts, and the implementation of varying locks. For FreeBSD, the biggest differences from Linux appear as its focus on the trap handler memory and CPU usage, whereas Windows differs through its use of the interrupt request, as well as how trap handlers will relay more detailed information about a specific interrupt.Overall though, all three platforms use some form of interrupts to assist in getting a newly executing thread started on its way all in similar ways.