Processes between Windows, FreeBSD, and Linux share the common concept of being the core instance of any singular running program. However, there are stark differences between the various systems for how they are managed and designed. Each system contains different default managing systems (also known as schedulers), as well as varying design patterns. Overall Processes are the backbone of any given operating system as they can be swapped in and out of the CPU to run multiple applications in a concurrent or a preconceived concurrent manner; giving the user freedom to multi-task. As comparisons and differences will be made in context to Linux, Windows, and FreeBSD's vanilla configurations.

Processes in windows have unique differences; however, some overall concepts are extremely similar to their Linux counterparts. The structures of processes for Windows and Linux differ greatly. In Linux, a process contains relationships with others that represents a hierarchy stemming from a root node that holds three differing states: parent, child, and zombie\cite{wwwtldpo42:online}. Windows differs greatly in that there is no hierarchy of parent-to-child relationships. As well parents processes of a child can be killed without creating a zombie\cite{CS3013Op9:online}. These differences cause the creation of new process (in Windows) to be extremely costly compared to Linux, because of the system needing to track process information to prevent zombies from occurring. In addition to overhead caused by process bookkeeping, Windows also has another service for checking the application against various compatibility databases allowing programs to run on varying systems

On an abstracted view, each system uses some form of priority management system for processes; however, under the hood they have small differences that set them apart. The Scheduling system between Windows and Linux's CFS (Completely Fair Scheduler) both use a priority queuing system to maintain order. The levels in the windows priority system range from zero to thirty-one which are divided into two categories: Windows API and Windows Kernel\cite{windowsbookpt1}. These categories help distinguish the difference between system level and application level processes (assigning them to their respective Queues.) This differs greatly in Linux, which uses a priority system of negative-twenty to nineteen \cite{love\_2015} where the negative priorities represent higher priority (typically for system processes). These priority management systems are essentially bookkeepers for the schedulers.

The two priority management systems (priority level) for these systems are a bookkeeper for the scheduler itself.These schedulers work is extremely distinct between the Completely fair Scheduler and Windows Scheduler. The Windows Scheduler parses its collections of processes and uses an algorithm called a multi-leveled queuing algorithm\cite{Scheduli20:online}. This algorithm is similar to a round robin algorithm where it allocates CPU time depending on the group of processes. First, it runs processes in order of highest to lowest priority in groups consisting of the same priority level. When all processes of the lowest priority are done being serviced, it begins servicing the highest priority group and repeats the algorithm. This is also similar to the CFS scheduler that implements a Red-Black tree "to manage the list of runnable processes and efficiently find the process with the smallest runtime"\cite{love\_2015}.

The two priority management systems (priority level) for these systems are a bookkeeper for the scheduler itself.These schedulers work is extremely distinct between the Completely fair Scheduler and Windows Scheduler. The Windows Scheduler parses its collections of processes and uses an algorithm called a multi-leveled queuing algorithm\cite{Scheduli20:online}. This algorithm is similar to a round robin algorithm where it allocates CPU time depending on the group of processes. First, it runs processes in order of highest to lowest priority in groups consisting of the same priority level. When all processes of the lowest priority are done being serviced, it begins servicing the highest priority group and repeats the algorithm. This is also similar to the CFS scheduler that implements a Red-Black tree "to manage the list of runnable processes and efficiently find the process with the smallest runtime"\cite{love\_2015}.

%FreeBSD to Linux

FreeBSD/Linux’s process structure and management are more aligned with each other's than Windows (though not entirely.) By that, each system uses a monolithic kernel, where all processes stem from one initial process during its boot procedure \cite{freeBsdBook}. That leads to processes in FreeBSD having the same parent, child, and zombie hierarchy of Linux-even down to having the same properties like PID (Process Identifiers) and POSIX functions being relatively interchangeable between the two systems. The differences lie in them having a slightly different priority range for processes; ranging from negative twenty to twenty, instead of nineteen like Linux\cite{setprior86:online}. Additionally, when child processes are forked off a parent, they inherit the parent's priority but can degrade priority as CPU time is assigned. This will be expanded on later.

Where FreeBSD and Linux different greatly is from a scheduler implementation standpoint. In fact, FreeBSD's scheduler uses a multilevel queue\cite{ThreadSc37:online} similar to Window’s scheduler. The reason a multilevel queue is used is because this particular implementation is better suited for severing environments than user desktops, like Linux. A thread within a process in FreeBSD is broken into two different categories: user mode, where typically application code is run in a protective manner for the system; and kernel mode, which is the mode for kernel processes like managing I/O or drivers \cite{freeBsdBook}. Management for a process itself is done by FreeBSD’s default ULE scheduler (which is meant to be a play on words for schedule.) The ULE scheduler, unlike the CFS, tends to favor either shorter running, or blocking processes like small scripts or text editors. However, if a short running/blocking process requires longer quantums, its priority will drop to prevent the potential of starving long-running background processes of CPU time. When a background process is run, they typically have fixed quantums but receive more access to a CPU. For example, the "compilation of a large application, may be done in many small steps, and No individual step runs long enough to have its priority degraded" \cite{freeBsdBook}.

%conclusion

Overall processes in Windows and FreeBSD have strong similarities to Linux, but differ in the some important ways. FreeBSD contains a unique scheduling algorithm that allows it to assign and reassign priorities as a process receives CPU time. In terms of similarities because Linux is a distant derivative of FreeBSDs underlying UNIX kernel they share almost identical process structures and even down to priority scales, with the only difference being algorithmic implementations. For windows processes receive different priority ranges, as well as process structure at the cost of creation time (because of bookkeeping of process information). The tradeoff allowing for orphaned child processes to be run unaffected of a missing or killed parent. Where Windows and Linux compare are the overall concepts of their schedulers, requiring fixed priorities but allowing variable or dynamic time slices to the CPU itself. Some of the qualities stated for FreeBSD and Windows can be altered to resemble Linux by changing schedulers or altering other other parts of their kernel.