# Processes, Threading and CPU Scheduling

George Crary

Operating Systems II

Spring 2017

**Abstract**

Between Windows, Linux and FreeBSD many similarities exist as in end as operating systems they fundamentally try to achieve the same things however differences do still exist. This analysis inspects these details and the origins that led to them. While both FreeBSD and Linux share lineage from Unix (Linux via Minix), Windows does not obviously. As a result of this the differences between these operating systems can be interpreted by the level of developer control that they come from. For example the Linux operating system development is purely concerned with the Linux kernel itself while the FreeBSD organization involves itself with both kernel development and core system utilities. BSD in comparison to Linux is subject to more structured design as a result of this. Windows on other hand has development interests from the kernel all the way out to userland. Alternatively the historical environment that these operating systems were developed in also have an influence worth noting. FreeBSDs Unix ancestors hail from the days of the 16-bit PDP11 while FreeBSD and Linux themselves began their lives on the x86 platform. Windows NT at the very least on the other hand also began life targeting the x86 platform as well.

## Contents

## I. PROCESSES

Processes throughout the operatings systems contain a large amount of similarities. Throughout the implementations of such concept all of them share such things as program code, open file descriptors, signaling and threading facilities. In addition other living aspects of the program are stored in a process such as run time, threads that are being executed, a data section for global variables and a private virtual address space for its given threads. Information for the CPU scheduler is also maintained within the process data structure as well. For FreeBSD and Linux these similarities are contained in their respective $task_struct$ data structure and are very similar as result of being Unix derivatives. In Windows these details are contained in the Windows Executive Process structure, also known as the EPROCESS structure. On top of that some process related data in Windows such as the Process Environment Block live in userland address space for accessibility reasons which is another difference [1].

Within the Unix realms FreeBSD and Linux both begin new processes with a fork system call. The fork calling process is turned into a parent process while the newly forked process is called a child process. In FreeBSD this can involve the rfork system call, which can handle controlling which resources get shared between the parent and child, while on Linux this is similarly handled with the clone system call [2]. Once the child process is up and running an exec system call can be made to run an entirely different program. This exec call overlays the program into the programs address space and begins executing it. The separation of these two system calls allows for any necessary preparation and cleanup to take place before the new program runs. Windows handles process creation in a similar fashion however its much more streamlined. Within Windows CreateProcess gets called and handles identifying the application type. As Windows supports executing Win16, Posix, MS-DOS executables these must be handled additionally with the standard exe executables. In those respective situations the appropriate Windows support image is executed and handled from there [1]. These differences between process creation really speak to the philosophical differences of the operating systems. The fork and exec system calls used in Linux and FreeBSD highlight the mindset of Do one thing right and do it well and showcase the composition of powerful and concise utilities. The Microsoft way of doing things showcases a commitment to backwards compatibility and total control by consolidating process creation into a single interface.

## II. THREADING

Threading is a common concept to all three operating systems. Significant differences begin to show however between Windows and the Unix related operating systems. Even within the Unix family threading is different. Threads are a facility for multiple paths of execution for a given process to occur at a single time while allowing for process resources to be shared. On a multicore machine this means threads can run in parallel potentially which enables a concurrent model of programming for a process. Within the Linux world threads are just seen as special processes and arent scheduled in a particularly different way. Within FreeBSD threads are looked upon as lightweight processes as context switching between threads is considerably faster than a context switch between processes [3]. Windows has a similar notion of threads as lightweight processes but also provides a userland scheduled threading facility called Fibers and another concept known as jobs. Jobs simply represent a collection of processes that are to be managed together.

Across the Unix systems the threading support is largely standardized thanks to POSIXs pthreads library.

One of the largest similarities between the three operating systems when it comes to threading is the User mode and Kernel mode distinction. Threads can operate in either User mode and Kernel mode and have their own respective access levels to address spaces. The user mode limitations to address space can help protect the operating system from exploitation in the event a server thread executing on the behalf of a client goes rogue. Separate stacks for these modes are given to threads as well in order for kernel specific operations to be cleanly supported from user mode operations.

## III. CPU Scheduling

CPU Scheduling is a common source of difference between the three operating systems. As the operating system must schedule the order of processes and thread operation across limited computing resources different approaches arise. Each operating system provides facilities for giving processes and threads different levels of priorities. In Linux for example this consists of a niceness value ranging from -20 to +19 with 0 as the default value. In FreeBSD this value ranges from 0 to 255 with different priority classes existing to serve the needs of user mode and kernel mode threads. Windows implements priority across the range 0 to 32 with the greater half representing real time priority levels and the lower end representing variable levels with 0 reserved for the zero page thread [1].

The schedulers behind each of these operating systems are drastically different. After Linux kernel version 2.5 was released a constant time scheduler, also known as the O(1) scheduler), was used. This scheduler could determine the next process to be ran in constant time. This behavior benefited high process count workloads seen on big iron workstations for example as the previous algorithm wasted a lot of time running in linear time. By version 2.6 this was changed to a new scheduler named CFS, the completely fair scheduler [2]. To handle scheduling fairness better CFS divides the processor time uniformly across N processes and then weighs given times appropriately with respect to the relative nice values.

FreeBSD uses the ULE scheduler which is named after a filename pun. This scheduler is divided into two schedulers. The low level scheduler and the high level scheduler. The former handles selecting a new thread to run from a non-empty run queue in a round robin fashion whenever a thread blocks [3]. The high level scheduler handles run queues for each CPU and maintaining the thread priorities. The Windows scheduler is similar to the ULE scheduler in terms of its high level CPU concerns and low level task picking. In Windows case however processor affinity is used to handle the high level case of where a thread may be executed.

## References

[1] Mark Russinovich, David Solomon, and Alex Ionescu, “Windows internals, part 1 6/e,” 2012.

[2] R. Love, “Linux kernel development 3/e,” 2010.

[3] M. K. McCusick and G. V. Neville-Neil, “Design and implementation of the freebsd operating system 2/e,” 2015.