WRITING TOPIC ONE

AN EXAMINATION OF PROCESSES, THREADS AND CPU SCHEDULING

December 4, 2016

Steven Silvers
Oregon State University
CS 444 Operating Systems II

Process implementation in Windows, FreeBSD and Linux differ in some ways but are similar in many others. An Example of this is that Windows, FreeBSD and Linux all store processes in memory as structs, such as the task_struct in Linux[1] or the Process Environment Block(PEB) structure in Windows[2]. Another similarity between Windows, FreeBSD and Linux is that new processes inherit security permissions of their parent process[3]. What this means is that if a parent process doesn't have permission to view a particular file, any process created by that parent also cannot view that particular file.

A major way that FreeBSD and Linux differ from Windows is how they create new processes. In Linux and FreeBSD new processes can only be created when an existing process makes a call to the system call fork() which creates a duplicate process called a child process, which can then be overwritten with the new process to be ran using the system call exec() in Linux or execve() in FreeBSD[4]. In the Windows operating system the fork() and exec() system calls are combined into a single system call named CreateProcess()[3]. While this simplifies creating new processes in Windows, should the situation arise where you wish to only call fork() without the exec() you can do this in Linux and FreeBSD but not in Windows.

Threads are used fairly similarly in Windows, FreeBSD and Linux. A thread is what the operating system assigns processing time to, and any one process could have one or multiple threads related to it. Windows, FreeBSD and Linux all have the capability of running threads in two different modes, user mode and kernel mode[4]. When a thread is executing application code it will operate in user mode, which is a protection mode with fewer privileges than kernel mode. When a thread makes a request for services from the operating system it will operate in kernel mode. The purpose of having the two separate modes is to protect the system from possible damage when running an application. There is no point in having your system exposed to outside applications when it doesn't need to be, so it is best to keep it protected.

Scheduling is a key operation of all operating systems. Scheduling is basically the operating system's way of deciding which thread should get processing time and most operating systems are fairly similar in how they do this but still vary slightly. The action of changing between what thread is currently being ran is called context switching. Context switching allows the operating system to give an illusion of multiprogramming, it can appear to run multiple applications at the same time even though only one or two threads might have processing time at any given moment[4]. Context switching does require some overhead, which creates a lower bound on just how small of time slices the operating system can give to threads. If the time slices were smaller than the context switching overhead, threads wouldn't actually have any time to run process code and the system would not work.

The Windows operating system uses a priority based scheduling system. In this type of scheduling system each thread is given a priority level from zero to thirty-one where zero is least priority and thirty-one is highest priority. The system treats all threads on the same priory level as equals, assigning time slices using the round robin method to the highest priority level with threads available to run[5]. If a thread at a higher priority becomes available to run, the system will stop giving time slices to whatever level it is currently on and then start assigning time slices to that higher level thread. For example, if the system is currently doing round robin assignment on threads with priority twenty-two and then a thread with priority twenty-eight becomes unblocked the system will stop giving time slices to the twenty-two level threads and move up to the twenty-eighth level. Round Robin is a scheduling algorithm that assigns time slices equally in a circular fashion with no priority. While the Round Robin algorithm doesn't consider priority, the Windows system only uses the algorithm against the threads on the highest available priority level.

FreeBSD schedules time slices in a slightly different way than Windows. While FreeBSD's default scheduler also uses a priority system like Windows, FreeBSD is biased towards interactive programs like text editors and other similar applications[4]. If a thread uses its entire time slice without exiting have their priority level lowered. This system allows new processes and interactive processes maintain high priority while long running process get pushed down the priority line.

Scheduling in modern versions of Linux typically will use the Completely Fair Scheduler(CFS) which has been in use since Linux 2.6.23[6]. The idea behind the Completely Fair Scheduler is that all processes should get equal time using the processor. It maintains this balance between the processes by detecting if any processes are not getting fair time on the processor, and then corrects this to bring the system back into balance. The CFS manages processes using a red-black tree, which is a self balancing tree that can operate on any node in the tree in O(log n) time giving us efficient process insertion and deletion[6].

Scheduling practices vary less from operating system to operating system and more from the different goals of the operating system. For example, a Linux system and a Windows system both configured to be general use operating systems will schedule tasks more similarly than say two Linux systems where one is general use and the other is a mainframe. Two different operating systems both focused on general use would need to be much more focused on multiprogramming because users typically want to use more than one application at a time, whereas the mainframe system would be more focused on completing tasks quickly and wouldn't really care to appear to be running more than one application at a time.

References

[1] K. McGrath, *Operating Systems*. Top Hat, 2016.

[2] “Microsoft developer resources: Peb structure.” https://msdn.microsoft.com/en-us/library/windows/desktop/aa813706(v=vs.85).aspx. Accessed: 10-17-2016.

[3] “Microsoft developer resources: Createprocess function.” https://msdn.microsoft.com/en-us/library/windows/desktop/ms682425(v=vs.85).aspx. Accessed: 10-17-2016.

[4] G. V. N.-N. Marshall Kirk McMusick, *Design and Implementation of the FreeBSD Operating System*. Addison-Wesley, 2nd ed., 2015.

[5] “Microsoft developer resources: Createprocess scheduler.” https://msdn.microsoft.com/en-us/library/windows/desktop/ms685100(v=vs.85).aspx. Accessed: 10-17-2016.

[6] “Inside the linux 2.6 completely fair scheduler.” http://www.ibm.com/developerworks/linux/library/l-completely-fair-scheduler/. Accessed: 10-17-2016.