Writing Topic 1: Processes

CS 444 Fall 2016

Kevin Stine

PROCESSES

Most modern operating systems are built utilizing processes and threads. In order to maximize the efficiency of handling these processes and threads, operating systems rely on CPU scheduling to monitor their behavior. Three of the major modern operating systems: Windows, FreeBSD, and Linux all utilize processes, threads and scheduling. A process provides the resources which are needed to execute a program. This would include items such as the virtual address space, executable code, unique processes identifiers, environment variables, and at least one thread of execution. A thread is an entity within a process that can be scheduled for execution. These threads maintain exception handlers, scheduling priorities and a unique thread identifier. Multiple threads are able to exist within one process, each executing concurrently and sharing resources. These threads are managed independently by the scheduler of the operating system. CPU scheduling is utilized to keep all the computer resources busy or to allow multiple users to share system resources effectively. Each of the three respective operating systems implement some way of managing processes, threads and scheduling.

Microsoft Windows supports preemptive multitasking, a concept which creates the effect of simultaneous execution of multiple threads from multiple processes. While multiple threads from multiple processes are not actually capable of running simultaneously, the OS does a great job at hiding that fact. To the user, they see multiple windows open, multiple programs running and can assume that everything is just working simultaneously when in fact, the OS is working hard to provide each processes with a fair amount of time on the CPU. Within Windows, threads are scheduled to run based on their scheduling priority, which ranges from zero (lowest priority) to 31 (highest priority). [3] Despite having a zero priority, only the zero-page thread is allowed to have a priority of zero. Windows also utilizes the round-robin method for process scheduling. The round-robin scheduler utilizes time-sharing, giving each job a time slice which is the amount of time it is allowed on the CPU. If the job is not completed when the time slice is up, the job is interrupted and resumed once another time slice is assigned to that process. Windows treats all threads with the same priority as equal, and begins by assigning time slices in a round-robin fashion to all threads with the highest priority. If those processes are not ready to run, the system moves to the next highest priority processes and assigns them a time slice. Windows creates a base priority, which is a combination of the process priority class and the thread priority level. The process priority classes can be idle, below normal, normal, above normal, high priority, and real time. The thread priorities can be idle, lowest, below normal, normal, above normal, highest and time critical.

On the other hand, FreeBSD implements a few things in a different fashion. In FreeBSD, all threads that are runnable are assigned a scheduling priority which determines the run queue they are placed in. The system scans the run queues from highest to lowest priority and chooses the first thread on the first non empty queue. If there are multiple threads on the queue, the system will run the threads in a round robin fashion. If a thread blocks, it is not put back onto any of the run queues and if a thread uses up the time quantum, it is placed as the end of the queue. [1] FreeBSD has found that the longest quantum that could be used without loss is 100ms. It also uses a time-share thread scheduling algorithm which is based on multilevel feedback queues. The system will adjust the priority of a thread dynamically to reflect resource requirements and the amount of resources consumed by the thread. If a thread that is not currently running is assigned a higher priority than the currently running thread, the system switches to that thread immediately if in user mode, otherwise it switches to the higher-priority thread once the current thread exits the kernel. This system favors interactive jobs by increasing the scheduling priority of threads that are blocked waiting for I/O for 1 or more seconds, and lowers the priority of threads that utilize a lot of CPU time. Threads are given a certain level of priority, between the range of -20 and 20 with the normal

value being 0. Negative values increase the threads priority while positive values decrease them.

Linux also uses preemptive multitasking where the scheduler decides when a process is to cease running and a new process is to resume running. The scheduler used by Linux is the O(1) scheduler since the scheduler can run in constant time. Similarly to FreeBSD, Linux sets the priority on a scale from -20 to 19 with the default priority being zero. The more negative the values are, the less nice they are, which makes them the highest priority. The Linux scheduler dynamically determines the timeslice of a process based solely on its priority. This allows for higher priority processes to run longer and more often than lower priority processes. [2] A process with a nice value of -20 receives the maximum timeslice and a process with a nice value of 19 receives the minimum timeslice. In Linux, the minimum timeslice is 10ms, the default time slice is 150ms, and the maximum timeslice is 300ms.

Windows, FreeBSD, and Linux all have their own implementations for handling processes, threads and scheduling. Both Windows and Linux utilize a type of preemptive multitasking and while FreeBSD can preempt threads executing in kernel mode, it is generally not done. Each of the three operating systems use a round-robin algorithm for handling all processes without a priority. Round robin seems to be a relatively simple and easy to implement algorithm which prevents the issue of starvation. Since all three operating systems use time slices, it makes sense that they would all use round robin. Setting the priority of threads and processes differs between Windows and Linux. On Windows, the priority is determined by a combination priority class of the process and the priority level of the thread, giving a unique number on the scale from zero to thirty-one. On FreeBSD and Linux, the priority is measured on the nice scale, from -20 to 19 (Linux) or 20 (FreeBSD). The amount of time used for the timeslice is very different between FreeBSD and Linux. FreeBSD uses a constant 100ms for their time slice, while Linux determines the time slice dynamically, based on the process priority and ranges from 10ms to 300ms. It seems like across the board, round-robin is the best choice for scheduling threads without a priority. From there however, the three operating systems take different routes on how to determine priority, and how to determine the time slice that each thread will be given.

REFERENCES

[1] Marshall Kirk McCusick and George V. Neville-Neil, Design and Implementation of the FreeBSD Operating System 2/e, Addison-Wesley, 2015, ISBN: 978-0-32196897-5

[2] Robert Love, Linux Kernel Development 3/e, Addison-Wesley, 2010, ISBN: 978-1-672-32946-3

[3] Windows Dev Center, About Processes and Threads, Microsoft Corporation