CS 444 Writing Assignment 1

Spring 2017

Hannah Solorzano

Abstract

There are considerable differences in how Windows and FreeBSD handle processes, threads, and scheduling in comparison to Linux. This paper explains how they differ, any similarities, and why these similarities exist.

I. Introduction

The differences in kernels is what makes each one specialized to excel at performing certain tasks. A kernel might consider video and audio to be higher priority than a time sensitive process whose quality suffers if the processes are not executed correctly. This essay will discuss the types of schedulers and context switching that are responsible for running processes and threads.

II. FreeBSD Kernel

A. CPU Scheduling

The open source operating system FreeBSD supports multiprogramming by using context switches to determine which thread is executed. The default scheduler, Timeshare Scheduler, recalculates the priority of the threads based on how much CPU is to be used and the amount of memory resources it possesses or calls for in order to be executed.

A secondary Real Time Scheduler uses an independent queue that is separated from the Time Share queue that is not apt to priority degradation, therefore, those processes are only surpassed by another Real Time whose priority is equal or higher.

The third queue contains processes that run at an idle priority. These threads are only executed if there are no other Real Time or Time Share threads available to run.

B. Priority Policy

The Time Share scheduler utilizes a priority base policy that prefers interactive programs, such as text editors, over long running threads. Interactive programs are preferred because of how they preform short bouts of computation quickly followed by inactivity or I/O.

On creation, threads are considered to be high priority at first which allows them to execute for a fixed time slice. If a thread uses its entire time slice, it's priority level is lowered, though those that are required to give up their time slice, like those that perform I/O, are able to keep their higher priority level. Threads can also be considered low priority if they require a significant portion of the CPU to execute [McK14].

III. Windows Kernel

The Windows kernel is similar to the FreeBSD kernel as it uses Real Time Scheduling as well as thread priority. The difference is that the priority of the threads are based on a 0 to 31 scale with 0 being the lowest and 31 being the highest. Windows context switches also play a part in determining which queue of threads are to be executed by checking if a particular queue is blocked or if there is a queue of higher priority that has become available to run.

A. Scheduling Priority

There are six classes of process priority levels:

- IDLE_PRIORITY_CLASS
- BELOW_NORMAL_PRIORITY_CLASS

- NORMAL_PRIORITY_CLASS
- ABOVE_NORMAL_PRIORITY_CLASS
- HIGH_PRIORITY_CLASS
- REALTIME_PRIORITY_CLASS

The default priority of a thread is set to NORMAL_PRIORITY_CLASS which is the third lowest priority class, though the levels can be specified on creation through the CreateProcess() function. Using this function, it is possible to specify a process to be IDLE_PRIORITY_CLASS for a thread that simply monitors the system in the background, or HIGH_PRIORITY_CLASS for processes that are time critical. Though making a HIGH_PRIORITY_CLASS process should be cautioned as these processes will take run time away from other processes and prevent them from running. Likewise, it is generally a bad idea to create a REALTIME_PRIORITY_CLASS as these processes will interrupt the mouse, keyboard, and any disk flushing [Rus09].

B. Context Switches

It is possible for a queue of threads to become blocked and are no longer able to execute. In this event, the scheduler will not allocate any run time to the threads regardless of how high their priority level is. The scheduler will then move on the next queue of highest priority in a process called context switching. The most common reasons for context switches is:

- A time slice has elapsed
- A thread with a higher priority is available to run
- A currently running thread has become blocked or is required to wait

IV. Linux Kernel

The Linux kernel utilizes several different queues for each processor, where the processor is only capable of running processes from its own queue while the others have the option of being idle. The queues are also able to balance themselves so that if one queue is too long, it will move processes to another queue to reduce its own length.

The queue that is given the ability to run is determined by an algorithm that repeatedly locates the queue of highest priority, calculates its time slice size, allows that process to run, then sets the process in a list of expired processes on completing its runtime. A time slice can also be specified when creating a process using the nice() function. See listing 1 where the default, max, and nice() time slices are declared in the Linux header file *include/linux/sched.h* [Uni06].

A. The Processor Scheduler

There are two types of process priorities in the Linux kernel: Static Priority and Dynamic Priority. Static priorities are used by RT schedulers and are given a number between 1 and 99 as their priority level. Dynamic priorities are used by non-RT schedulers whose priority level is calculated by adding the base time slice and the number of ticks of CPU time the process has remaining before its time slice expires.

B. Queues and Priorities

The higher the priority level, the more time that process has to run. For each priority level, there are two sets of queues allocated for their processes: Active and Expired. When a process completes its time slice, it is sent to the expired queue. Once all of the processes in the Active queue has been executed, the Active queue switches with the Expired queue.

Listing 1. the default, max, and nice() time slice

```
#define DEF_COUNTER (10*HZ/100) /* 100 ms time slice */
#define MAX_COUNTER (20*HZ/100)
#define DEF_NICE (0)
```

V. Kernel Comparison

The three kernels, FreeBSD, Windows, and Linux, all utilize a few similar methods for handling processes and threads. These common elements are the basic foundation of keeping a scheduler that can prioritize queues and manage time slices. The differences between the three kernels are what make the operating system unique and specialized.

A. The Similarities

Each kernel contains a scheduler that uses a priority system to identify which processes need to be executed first. This is to insure that processes that need to be ran immediately are not blocked by a process that is controlling something that is occurring in the background.

Another function they all provide is having at least two types of schedulers running the processes. One of the most common type of scheduler to have is the Real Time scheduler that runs the process queues based on priority in a first-in-first-out method.

B. The Differences

One of the biggest differences between the three kernels is how they assign the queue their priority levels. While FreeBSD gives higher priority depending upon the type of process it is, Windows uses a more in depth approach of using an algorithm to calculate the priority based on clock ticks and time slices. The Linux kernel also assigns priority based on process type, but it also allows a user to declare the priority level through the nice() function which creates the process.

Another difference is the number of queues the scheduler is working with. The Linux kernel has about 140 queues as each priority level receives a active and inactive queue. The Windows kernel possesses only a couple queues for those that are going to run, those that are blocked, and those that have finished their time slice. Similarly, the FreeBSD kernel also uses only a few queues to handle processes.

C. Reason for Differences

Each of the kernels handles processes a little bit differently which is what makes the kernel unique and able to specialize in handling certain types of programs better than other kernels. An example of

this is how Windows is better at running audio and video which allows the user to watch videos and play video games. FreeBSD is also adept at running audio and video, but is noticeably slower than that of Windows. Linux, while capable of running a few video games, specializes in being a light weight operating system that can host web apps, troubleshoot other computers, and work closely with the hard drive.

VI. Conclusion

Overall, while Linux, FreeBSD, and Windows operate their kernels with similar basic concepts and ideas, the differences that lie deeper in their scheduler organization is what sets them apart. These details allow for the scheduler to balance user input, I/O, and audio/ video, and seamlessly provide run time for all the processes waiting in the queue.

REFERENCES

[McK14] Marshall Kirk McKusick. Thread Scheduling in the FreeBSD Operating System. http://www.informit.com/articles/article.aspx?p=2249436&seqNum=4/, 2014. [Online; accessed 01-May-2017].

[Rus09] Mark E. Russiovich. Processes, Threads, and Jobs in the Windows Operating System. https://www.microsoftpressstore.com/articles/article.aspx?p=2233328&seqNum=7/, 2009. [Online; accessed 01-May-2017].

[Uni06] Columbia University. Linux Scheduler. https://www.cs.columbia.edu/~smb/classes/s06-4118/l13.pdf/, 2006. [Online; accessed 01-May-2017].