Writing Assignment 1

Bradley Imai

CS 444
Spring 2017
1 May 2017

Abstract

Windows, FreeBSD and Linux each take different methods as how they are generalized as a low level operating system (OS). Within these operating systems there lies similarities and differences. In the following, processes, threading and CPU scheduling will be broken down and explained among these operating systems.

1 Processes

An application consists of one or more processes. A process, in the simplest terms, is an executing program. One or more threads run in the context of the process.

1.1 Similarities

The concepts of processes as a whole, Windows, FreeBSD and Linux have many similarities within each other. In general, throughout these operating systems, processes contain program code, signals, files and threads. According to [11], once a program is loaded into the memory and it becomes a process, it can be divided into four sections, stack, heap, text and data. In the stack, the process contains the temporary data such as method/function parameters, return address and local variables. In the heap, it is dynamically allocated memory to a process during its run time. In the text which includes the current activity represented by the value of Program Counter and contents of the processors registers. Lastly data, this section contains the global and static variables.

1.2 Differences

Looking at theses operating systems there are a few differences within each other. Both FreeBSD and Linux call there processes a task where Windows, represents its processes as an Executive process. According to [9] each task or thread of execution is termed a process. Thecontext of a 4.4BSD process consists of user-level state, including the contents of its address space and the run-time environment, and kernel-level state, which includes scheduling parameters, resource controls, and identification information. Where in Windows, the executive process consists of the Process Environment Block (PEB). Within the PEB, it contains the process heap, thread, storage and executable image to be accessed from the user space. One of the main differences between Linux/FreeBSD and Windows is the forming of a new process. In Linux/FreeBSD, according to [1], a new process is created because an existing process makes an exact copy of itself. This child process has the same environment as its parent, only the process ID number is different. This procedure is called forking. After the forking process, the address space of the child process is overwritten with the new process data. This is done through an exec call to the system. Whereas Windows, a CreateProcess function will create a new process, which runs independently of the creating process. However, for simplicity, the relationship is referred to as a parent-child relationship [8].

1.3 Why do similarities and differences exist?

The very first difference between these three operating systems is the way they create their process. FreeBSD/Linux creates their processes by providing two separate calls in a procedure called forking which are known as tasks. Whereas Windows creates a CreateProcess function which runs independently of the creating process. In a nut shell, FreeBSD/Linux takes an extra step in creating their processes where Windows keeps its process creating function connected. A few similarities to these operating systems is their process address spaces and user mode and kernel mode. All three of these systems use a structure in running

2 Threads

A thread is the basic unit to which the operating system allocates processor time. A thread can execute any part of the process code, including parts currently being executed by another thread.

2.1 Similarities

One similarity that FreeBSD/Linux and windows share are their implementations of the system and kernel threads. The threads allow for jobs to be executed in kernel mode. According to [4] the executing code has complete and unrestricted access to the underlying hardware. It can execute any CPU instruction and reference any memory address. Kernel mode is generally reserved for the lowest-level, most trusted functions of the OS. Windows, FreeBSD and Linux all support threads that run on the kernel space. FreeBSD and Linux use kernel threads where Windows uses a system thread. All three of these operating systems do not have a task and also do not have address space [5] .

2.2 Differences

One of the main differences between Windows, FreeBSD and Linux is within the application of threads. Both FreeBSD and Linux processes and threads are basically the same besides that the threads share resources. In Windows, at least one thread has to be present in order for it to run [2].

2.3 Why do similarities and differences exist?

Reviewing the similarities and differences between these Operating systems are quite difference from each other. The main similarity is that they have threads that allow for jobs to be executed exclusively in kernel mode. However, FreeBSD and Linux share thread resources where in windows a thread is represented by an executive thread (ETHREAD) block [12]. ETHREADS do not have resources like FreeBSD and Linux.

3 CPU Scheduling

CPU scheduling is a process which allows one process to use the CPU while the execution of another process is on hold due to unavailability of any resource like I/O etc, thereby making full use of CPU. The aim of CPU scheduling is to make the system efficient, fast and fair [7].

3.1 Similarities

One of the main similarities between Windows, FreeBSD and Linux is their priority based scheduling. According to Microsofts website, windows scheduler controls multitasking by determining which of the competing threads receives the next processor time slice. The scheduler determines which thread runs next using scheduling priorities [14]. The default scheduler for FreeBSD is called the ULE. The default ULE assigns a scheduling priority and a CPU by the high-level and low-level scheduler. The high level scheduler determines in which run queue they are placed. In selecting a new thread to run, the low-level scheduler scans the run queues of the CPU needing a new thread from highest to lowest priority and chooses the firs thread on the first nonempty queue [10]. In Linux, the Completely Fair Scheduler (CFS) is a process scheduler which handles CPU resource allocation for executing processes, and aims to maximize overall CPU utilization while also maximizing interactive performance [6].

3.2 Differences

The main difference between these operating systems is found in various places. Windows uses the Muti-Level Feedback Queue (MLFQ) as their base scheduler [3]. It observes the execution of a job and prioritizes it accordingly. This allows it to deliver excellent overall performance and makes progress for long-running CPU-intensive workloads. Whereas FreeBSD uses the ULE as its default scheduler and Linux uses the Completely Fair Scheduler.

3.3 Why do similarities and differences exist?

All three of these systems have priority based scheduling and has been proven to be one of the best ways in scheduling. With a priority scheduler, the scheduler simply picks the highest priority process to run [13].

References

[1] ”4.1.5.1. Process creation.” Linux - Process creation. N.p., n.d. Web. 28 Apr. 2017.

[2] ”About Processes and Threads.” About Processes and Threads (Windows). N.p., n.d. Web. 29 Apr. 2017. ¡https://msdn.microsoft.com/en-us/library/windows/desktop/ms681917(v=vs.85).aspx¿.

[3] Arpaci-Dusseau, Remzi, and Andrea C. Arpaci-Dusseau. Operating systems: three easy pieces. S. l.: Arpaci-Dusseau , 2015. Print.

[4] Atwood, Jeff . ”Coding Horror.” Understanding User and Kernel Mode. N.p., n.d. Web. 28 Apr. 2017. ¡https://blog.codinghorror.com/understanding-user-and-kernel-mode/¿.

[5] ”Chapter 15. The Process Address Space.” Chapter 15. The Process Address Space - Shichao’s Notes. N.p., n.d. Web. 29 Apr. 2017. ¡https://notes.shichao.io/lkd/ch15/¿.

[6] ”Completely Fair Scheduler.” Wikipedia. Wikimedia Foundation, 24 Apr. 2017. Web. 27 Apr. 2017.

[7] ”CPU Scheduling.” CPU Scheduling in OS — Operating System Tutorial — Studytonight. N.p., n.d. Web. 27 Apr. 2017. ¡http://www.studytonight.com/operating-system/cpu-scheduling¿.

[8] ”Creating Processes.” Creating Processes (Windows). N.p., n.d. Web. 29 Apr. 2017. ¡https://msdn.microsoft.com/en-us/library/windows/desktop/ms682512(v=vs.85).aspx¿.

[9] McKusick, Marshall K., Keith Bostic, Michael J. Karels, and John S. Quarterman. ”The Design and Implementation of the 4.4BSD Operating System.” 2.4. Process Management. N.p., n.d. Web. 28 Apr. 2017.

[10] McKusick, Marshall Kirk., George V. Neville-Neil, and Robert N. M. Watson. The design and implementation of the FreeBSD operating system. Upper Saddle River: Addison-Wesley/Pearson, 2015. Print.

[11] Tutorialspoint.com. ”Operating System - Processes.” Www.tutorialspoint.com. Tutorials point, n.d. Web. 28 Apr. 2017.

[12] ”Processes, Threads, and Jobs in the Windows Operating System.” Processes, Threads, and Jobs in the Windows Operating System — Microsoft Press Store. N.p., n.d. Web. 28 Apr. 2017

[13] Process Scheduling. N.p., n.d. Web. 27 Apr. 2017. ¡https://www.cs.rutgers.edu/ pxk/416/notes/07-scheduling.html¿.

[14] ”Scheduling.” Scheduling (Windows). N.p., n.d. Web. 28 Apr. 2017. ¡https://msdn.microsoft.com/en-us/library/windows/desktop/ms685096(v=vs.85).aspx¿.