OS 2

SPRING 2018

FINAL WRITING ASSIGNMENT

PREPARED BY R. HAYDEN ANDERSON

CONTENTS

1 Introduction 3

2 Processes, Threads, and CPU Schedulers 3

2.1 Introduction . . . 3

2.2 Windows Implementation . . . 3

2.2.1 Processes & Threads . . . 3

2.2.2 CPU Scheduling . . . 3

2.2.3 Why Do They Exist . . . 4

2.3 FreeBSD Implementation . . . 4

2.3.1 Processes & Threads . . . 4

2.3.2 CPU Scheduling . . . 4

2.3.3 Why Do They Exist . . . 4

2.4 Conclusion . . . 4

3 I/O 5

3.1 Introduction . . . 5

3.2 Windows I/O . . . 5

3.2.1 Data Structures . . . 5

3.2.2 Crypto . . . 5

3.3 FreeBSD I/O . . . 5

3.3.1 Data Structures . . . 5

3.3.2 buffers . . . 6

3.3.3 Multi Filesystem Support . . . 6

3.4 Conclusion . . . 6

4 Memory Management 6

4.1 Introduction . . . 6

4.1.1 Windows Memory Management 6

4.1.2 FreeBSD Memory Management 7

4.2 Conclusion 8

5 Conclusion 8

1 INTRODUCTION

The Windows, FreeBSD, and Linux operating systems have a lot of differences among them. This paper details the main differences between both Windows and Linux and FreeBSD and Linux. Primarily the differences in processes, threads, and CPU scheduling. The following section compares their differences with I/O and the differences in data structures used, cryptography, buffers, and even multi file system support. The last section discusses memory management; specifically pages, hugepages and their sizes as well as the general algorithms and techniques each operating system uses in order manage its memory effectively.

2 PROCESSES, THREADS, AND CPU SCHEDULERS

2.1 Introduction

Running a program on different instances can lead to the same result but the processes have many similarities and differences. Windows, FreeBSD, and Linux all have processes that are similar in the fact that they are all instances of a running program on the computer. Each of these operating systems use, design, and manage these processes in very different ways. All three systems use CPU schedulers that all work very differently from each other. Each process can have multiple threads which split up the overall goal of the process and pseudo-simultaneously run in parallel. Processes, threads, and the CPU scheduler among Windows, FreeBSD, and Linux all have many similarities and differences that will be discussed.

2.2 Windows Implementation

2.2.1 Processes & Threads

In windows, processes work very similarly to Linux in every way except for structure. Linux processes all stem from the initial process that spawns new processes that in turn, spawn new processes. Each process in Linux can exists in three states; a parent, a child, or a zombie. Linux has a very structured hierarchy of parent-child processes, where Windows does not have the parent-child hierarchy. Linux and Windows both benefit from the parent-child relationship because it can be quicker and less overhead is needed. One main difference between windows and Linux in terms of processes are that Windows can end parent processes without creating zombie children processes. Windows still keeps track of these processes to force the zombie process to end as well. The ability to end the zombie process causes a lot of overhead for windows as it has to know which processes spawn other processes to prevent zombie processes from occurring. Even though the zombie process requires a significant amount of overhead, it allows for quicker clean up of zombie processes to ensure there is no wasted space. While Linux and Windows have similar processes, there are a variety of differences that can have significant impacts on processing time.

2.2.2 CPU Scheduling

The CPU scheduler for both Windows and Linux use similar types of management and priorities for their processes. In October 2007, the default scheduler for Linux became the Completely Fair Scheduler which uses a red-black tree to create a runqueue of processes with specific priorities in order to maximize overall CPU utilization[11]. Windows uses a somewhat different file priority system. Each process receives a priority level between 0 (lowest priority) to 31 (highest priority) where Linux only has 19 different priority levels. Each thread with the same priority is considered equal and receive different time slices from the system in a round-robin fashion[6]. Both operating systems run their CPU scheduling queues from high to low priority. By allowing different levels of priority, you can differentiate your processes more easily because there is a greater amount of range of levels. While this benefit can be great for having multiple processes, it can also require more overhead than a system with fewer levels. Linux and Windows use very similar CPU schedulers resulting in similar results with few differences.

2.2.3 Why Do They Exist

I think the differences in CPU structure and management make sense between Windows and Linux Operating system. For Linux you have more power users that know what is going on between processes and which process is parent of another process. Windows is based for a graphical user interface. When you close a parent process, there's a lot more disconnect between what window is running another process sometimes, so allowing the child process to continue to run while the parent closes makes sense. Linux has a lot more emphasis on the command line. When you start a program from the command line, that instance of the command line is the parent process that calls the child process program. It makes sense that closing that parent process that is running the child process closes.

2.3 FreeBSD Implementation

2.3.1 Processes & Threads

When comparing FreeBSD and Linux, both stem from the UNIX architecture and have very similar process structures. Both FreeBSD and Linux have one initial process "Init" with a Process ID of 1, and each processes thereafter is a fork from that process or its offspring. Each process has a Process Identifier (PID) and the POSIX functions used with each operating system are more or less interchangeable. There are not any significant differences in the processing structure of FreeBSD and Linux.

2.3.2 CPU Scheduling

The comparisons of FreeBSD and Linux have some similarities with process structure but there are some major differences in how each system manages the process and threads. FreeBSD uses different priority structures for both processes and threads. Their process priorities range from -20 to 20 while Linux priority is from 0-19. Thread priorities are a big difference between FreeBSD and Linux. FreeBSD priority thread ranges are from 0 to 255 and Linux priorities are just for processes. Another difference with priorities within FreeBSD is that the number given for a priority is followed, they only behave differently when in different set ranges. There are five set ranges of priority; interrupt threads are the priority range of 0-63, top half kernel threads are for threads 64-127, and real time user threads consist of the priority range 128-159. There are also time sharing user threads for the priorities 160-223 and idle user threads for the priority range of 224-255[5]. The algorithm used with FreeBSD process and thread management takes into account both the priority of the process as well as the recent amount of time used for the thread. While having the large range of priorities can be beneficial while using the algorithm with FreeBSD, there is still significant overhead needed to maintain the range. Linux continues to require less overhead because of the smaller range of priority.

2.3.3 Why Do They Exist

Keeping track of all of these threads and processes required a lot of extra CPU usage overhead that Linux just doesn't need. These differences can be helpful in some situations and I think can be very beneficial if you want to use a lot of hosting with your machine. It is also consistent and stable where some forms of Linux can be fairly unstable.

2.4 Conclusion

There are a lot of similarities between Windows and Linux and FreeBSD and Linux. However, there are some really big differences between them as well. For windows the main difference is the process and thread structure. There is a large cost for Windows to keep up with all of the information needed for the children of killed parents processes. While keeping track of zombie processes can be nice, Windows has a much larger time and CPU cost compared to Linux. FreeBSD and Linux are very similar and even share the same UNIX kernel, but vary differently when it comes to process and thread priority and CPU scheduling. The algorithm used to assign processes within the CPU is very unique and works better for a specific use.

3 I/O

3.1 Introduction

The Inputs and Outputs of operating systems such as Windows, FreeBSD, and Linux are used as the operation or process to interact with other devices or humans. The methods used to change how the system interacts with us changes significantly between the operating systems. The I/O ranges from the file structure and how it is displayed to the user to the cryptography to the scheduling used to manage the devices. By understanding the differences in how the operating systems impact the communication among devices or humans, it is easier to select the operating system to best suit the needs of the process. Windows, Linux and FreeBSD all have some similarities and differences that will be analyzed through the data structure, cryptography, buffers, and multi-file system support.

3.2 Windows I/O

3.2.1 Data Structures

For the Windows operating system to talk amongst itself, the Windows Driver Model (WDM) and Windows NT device drivers use I/O Request Packets (IRP's). The WDM is a framework for layers of drivers to interact and communicate with each other via messages. The messages they use are the IRP's. The windows device drivers are small programs that control and operate particular pieces of hardware devices on the computer. These drivers all run together make the WDM framework which, in turn, makes the whole system run together. Drivers are all hardware specific and essentially an API for operating systems to use to control the piece of hardware. The IRP messages are sent via bus drivers. Windows uses a very Object Oriented driver stack. Conversely, Linux uses request structures to process and share requests throughout the system. Essentially, the system and its communication is made up of buffers with file pointers. When writing or reading the buffers, the Linux kernel calls requests in the request structure. The file pointers within Linux call routines for requests within a request structure. While both operating systems function differently, they both accomplish the communication in an efficient manner. [7]

3.2.2 Crypto

Cryptography in Windows is primarily independent from the system and are up to the user to implement. In Windows, users are able to create their own cryptography or stick with the default Cryptographic Service Provider(CSP). The default CSP uses RSA, AES, DSS, Diffie-Hellman, and more [6]. To keep the CSP from behaving incorrectly, Windows periodically checks and scans the CSP in order to prevent malicious viruses and to make sure the user has not tampered with things they should not be tampering [9]. Similarly, Linux uses an independent system for cryptography functions as well. While Cryptography is important in prevention of malicious viruses, it is up to the user to choose which system to implement, regardless of the operating system they are using.

3.3 FreeBSD I/O

3.3.1 Data Structures

FreeBSD and Linux share more than just the same kernel, they also share a very similar I/O structure. The kernel reads and sends everything as a stream of bytes [2]. This commonizes every program and then entire I/O stream into something that each receiving program or device driver can read. The contents of the I/O stream could have a structure, like the typical text document which consists of lines of ASCII characters separated by ASCII line-feed character. The FreeBSD uses the C-Look elevator for its disk scheduling algorithm. The C-Look elevator puts write requests into an order from highest on disk to lowest on disk physically from the base of the write-head. The write-head writes requests from the top to bottom of the disk and when it reaches the lowest request it immediately jumps to the highest request and then goes back down through the list. The linux operating system uses the Deadline elevator as its disk scheduling algorithm. The deadline algorithm keeps for queues of requests; one for read operations, one for write operations, and

two for deadlines or expirations of requests. Using a queue for expiration requests forces the elevator to not put off any processes for too long. The read queue has highest priority because reading is what holds up most operations. The write-head then writes in batches of requests from each queue [10]. By default read operations are timed out after 500 milliseconds and write operations expire at a full five seconds. By having a consistent scheduling algorithm, the efficiency of high priority process increases.

3.3.2 buffers

One big difference between FreeBSD and Linux is the use of scatter/gather in the I/O. Using scatter input, the ability to use multiple buffers to house a single read system call is introduced. Scatter gather also allows writes to be written to multiple buffers in a single write by using a pointer to an array of buffers and lengths. Linux by default only uses traditional read and write calls. Adding the abstraction of using pointers to write instead of writing in one contiguous space of memory is very time consuming and can often double the run time of the application in use [2].

3.3.3 Multi Filesystem Support

Another difference between FreeBSD and Linux is the inherent use of a dynamically loaded operating systems. The ability to use vnodes and load multiple filesystems when they are first referenced by the mount system call was initially introduced to just the FreeBSD operating system [2]. On the other hand, Linux introduced this ability much later.

3.4 Conclusion

The Windows and Linux operating system differ greatly when it comes to I/O. Linux is essentially pure file pointers and data streams while Windows is more object oriented and structures. The FreeBSD system and Linux are near identical in structure but with many subtle differences in configurations and how they work at a very low level. One of the bigger differences being the elevator used and the use of scatter/gather I/O and multiple file system support.

4 MEMORY MANAGEMENT

4.1 Introduction

Memory management is different among Windows, FreeBSD, and Linux. Some of the operating systems have very little differences while others have very large differences. Memory management is how the operating system directly creates, reads, references, allocates, and frees physical space on Random Access Memory (RAM). Physically a system has RAM sticks that are volatile, they reset when there is no current within the system. Memory management is the process of using the space on RAM for quick access and abstracting them to be kept virtually and just mapping virtual memory space to the physical space. RAM is for quick access, what items on your machine the user or other processes need to access quickly instead of having to go through slow access like hard disc drives.

All of the operating systems use pages. A page of memory is a fixed size contiguous block. Each of these pages are set as an entry into a page table. Using page tables to map locations of pages allows for quick access.

4.1.1 Windows Memory Management

Both Windows and the Linux operating system use pages. Both operating systems have the same smallest page size of 4 Kibibytes[7]. Windows and Linux also have larger page sizes it can use called hugepages. Windows has a 2 Mebibyte and a Gibibyte hugepage size while Linux typically uses the 4 Kibibyte it also has an 8 Kibibyte option][4]. These page sizes are fixed when they are in memory so when a program needs to allocate memory to perform its task it takes up the whole page even if it only uses a small fraction of it. The set sizes allow the page table to easily access each page

by different offsets. Windows uses a tree like structure to determine for the page tables where the pages are located and how to access them. Linux on the other hand uses a linked list type of data structure for the offsets. In both, memory is heavily using pointers to first go from program to virtual memory to physical mappings. [4]

In terms of paging, Windows brings in eight pages of data into memory simultaneously in order to save time and it assumes all; these pages are needed. Conversely, Linux only swaps pages into memory that are directly required for the current subprocess instead of the entire process at once[4]. By only loading in necessary pages, Linux saves time on swapping and the amount of physical memory required at one time. This is just one reason why Traditional Linux uses less memory for its system than Windows.

The Linux address structure is broken up into four parts; Global Directory, Middle Directory, Page Table, and Offset. Windows address structure is only broken up into two parts; the Page Number and the Page Offset. To replace pages, Windows uses a first in first out (FIFO) replacement algorithm. So the ”oldest” page is used to be overwritten which means there are many page faults that can happen since each spot of memory does not need to be there for the same amount of time. Windows has about 155 page faults. Linux on the other hand uses the Least Recently Used (LRU) algorithm, while this algorithm can sometimes line up with Windows' FIFO, the LRU actually takes into account that each piece of memory does not use it for the same amount of time[4]. When memory needs to overwrite blocks of memory it looks for the page that has not been used for the longest period of time. This allows the memory that could have been more recently used to be used quicker while the least used block can be overwritten with something that is more used. Using the LRU is much faster in both time and amount of space. Using the LRU only has 12 page faults.[1]

To find the pages to replace, both Windows and Linux dived the virtual pages into four lists. Windows uses the Modified Page List, the Stand-by Page List, the Free Page list, and the Zeroed Page List to keep track of the blocks of memory in use. Linux uses the Active List, the Inactive-dirty List, the Inactive Clean List, and the Free List to determine the pages in use and which can be replaced.

4.1.2 FreeBSD Memory Management

The FreeBSD and Linux architectures use very similar applications when doing memory management both at the page and kernel level. This stems from the fact that they both are derived from the same UNIX parent OS and have only a couple differences from growing. Two major differences between the FreeBSD and the Linux operating system are their page layer and kernel memory allocation as well as their paging system. Both Linux and FreeBSD originally used the buddy-system as their page allocator[1]. The buddy system creates a tree-like structure to allocate pages in sizes of sizes that are powers of 2. Unfortunately, using powers of 2 causes a lo of internal fragmentation and wastes a lot of valuable memory space.

The FreeBSD system adopted the Zone Allocator which runs alongside the buddy-system. With the zone allocator, kernel modules register the sizes of memory they are most likely going to request at run time. The zone allocator then carves chunks of memory for each zone. It then utilizes a background garbage collector to free chunks of previously allocated memory and show them as ”free”. Kernel modules that predate the zone allocator continue to just use the buddy system, while all other new ones use the zone allocator.[1]

The Linux operating system adopted the slab allocator system. The slab allocator is a modified version of the zone allocator. It has three options of use when using a slab; the SLAB slab, the SLOB slab, and the SLUB slab. Each of these types of slabs are very similar but differ in the order of which they allocated and store pages.

When FreeBSD runs out of physical memory, it swaps out entire processes and frequently swaps into any file system for temporary storage of the idle process until it has more physical space to store[1]. On the other hand, when Linux runs out of physical memory space it just freezes some processes and relies on its page replacement algorithms to get rid of dirty pages and use swap partitions. This change allows the FreeBSD operatinmg system to be more memory efficient and in some cases much more usable. The FreeBSD system also has a slightly different approach to page replacement[1]. When a page needs replacing, FreeBSD uses the Global Least Regularly Used (GLRU) and page-coloring and picks from a pool of replacement pages in order to evenly distribute cache colliding pages. Linux also uses GLRU but uses a biased page replacement. Linux's page replacement is biased towards pages with a lower page fault rate[4].

4.2 Conclusion

The Windows and FreeBSD operating systems both have different memory management techniques that they use in comparison with Linux. Both Linux and Windows use normal pages and hugepages of different size, and have a big difference in how they use swap memory. They also differ in using the LRU and the FIFO algorithms for replacing pages. FreeBSD and Linux have a lot of small differences, mainly the differences in allocators and the paging system.

5 Conclusion

In conclusion, the Windows operating system has a lot of differences in comparison to Linux; some very small but most are very large differences. FreeBSD on the other hand is very similar to Linux in a lot of aspects, which is expected considering they contain the same kernel and are branches off from UNIX. Th bulk of differences between FreeBSD and Linux is their techniques in memory management and a couple algorithms used in its I/O management. All of these operating systems have some major differences but are able to get the same things accomplished and are best suited to different kinds of users.

REFERENCES

[1] Rohit Dube. A comparison of the memory management sub-systems in freebsd and linux, Sep 1998.
[2] freebsd.org. *I/O System*. 2017.
[3] GeeksforGeeks. *Disk Scheduling Algorithms*. 2017.
[4] All Answers ltd. Compare the memory management of windows with linux, Mar 2015.
[5] Marshall Kirk McKusick, George V. Neville-Neil, and Robert N.M. Watson. *The Design and Implementation of the FreeBSD Operating System*. Addison-Wesley Professional, 2014.
[6] Microsoft. *Processes and Threads*. 2018.
[7] Mark E. Russinovich, David A. Solomon, and Alex Ionescu. *Windows Internals, Part 1 (Developer Reference)*. Microsoft Press, 2012.
[8] Mark E. Russinovich, David A. Solomon, and Alex Ionescu. *Windows Internals, Part 1 (Developer Reference)*. Microsoft Press, 2012.
[9] Wikipedia. *Cryptographic Service Provider*. 2016.
[10] Wikipedia. *Deadline scheduler*. 2017.
[11] Wikipedia. *Completely Fair Scheduler*. 2018.