Final Paper

Final Paper CS444 Spring2018

Brian Huang

1 Introduction

This paper will be comparing three operating systems: Linux, Windows and FreeBSD. Many of the kernel implementations of things like I/O and processes will be similar between FreeBSD and Linux as they have a similar origin. Windows on the other hand will have the most differences between the three operating systems as it is not a branch of Linux. In this paper the implementation of processes, Input and output and memory managements will be compared and contracted between Windows and FreeBSD to Linux.

2 Process, Threads and, Scheduling

In operating systems there are three important pieces that the whole operating system is dependant on. These are processes, threads and a scheduler. A process is simply a program or a set of instructions executing on the operating system. This is true for all operating systems, but the way they are implemented can be vastly different. Inside of these processes are things called threads. Threads handle the process' tasks and are used for executing the various instructions that the process needs to execute. In an operating system there can be many threads and processes, so a scheduler is needed to schedule which process or thread will run and at what time and for how long. Schedulers vary from operating system to operating system, but they all do the same job, assigning what will happen and when.

2.1 Windows to Linux

A process in windows is a container for a set of resources for the process to use when executing their instructions. In this container there is a private virtual address space, a executable program instruction, a list of open handles to system resources such as semaphores, ports, and files, and a process ID. Each of these processes will also point to the process that created it or its parent. If a parent process is terminated, the child process will still point to the nonexistent parent. This does not pose a problem to Windows, as nothing relies on this information, so having a child with a nonexistent parent does not cause any issues to the system. Windows and linux differ in the fact that windows does not really keep track of parent and child. A child process in

windows can still exist even after it's parent is terminated, but in linux terminating a parent with and still having the child exist can cause a zombie or orphan process, which could cause problems in the long run.

A thread in windows is an entity within the process that is scheduled for execution. Without a thread in a process, nothing will happen. In Windows a thread has the contents of a set of CPU registers to represent the state of the processor, two stacks one for kernel mode and the other for user mode, a private storage area, and a unique identifier. These are some essential components of a windows thread. All threads within the same process shares the same virtual address space, which means all the threads in the same process will all share the same resources given to that process. What this also means is that all threads in that process have full read-write access to the process' virtual address space. Threads in linux are unique in the fact that there are no threads, only processes. In linux there is nothing special about a thread and no special data structures. Threads are just standard processes that share resources with other processes. Each "thread" is unique to the kernel, but they processes that share their resources and address space with other processes. The main difference between Windows and Linux is this uniqueness. The difference of these two is that one processor would be assigned to the windows process with threads, but in linux one processor would be assigned to each of the threads, as they are not actually threads but they are processes.

Scheduling in windows is priority driven, meaning that the highest priority process runs first. There are 32 priorities in windows ranging from 0 to 31. Real time levels have the priority level of 16 - 31, variable levels have the priority of 15 - 0, and 0 is reserved for the zero page thread. Processes also have processor affinity where the processor can limit a process' ability to to run. These affinities can be altered by altering the API or by setting an affinity mask. A process is given a quantum to run, which is just a length of time that a process can run, then another process of the same priority level will the run after the quantum is over. Scheduling is implemented at the kernel level, so there is no specific module that handles scheduling. The code that handles scheduling is spread all throughout the kernel. The scheduling system in linux is fairly similar to that of windows. The scheduler is priority driven, however the scale in linux runs from -20 to +19 with a default priority level of zero. In Linux these are called "nice" values, where -20 is the highest priority while +19 is the lowest. Linux also has a second range for each process for the real-time priority. This ranges from 0 to 99 and acts opposite from the nice values, where 99 is the highest priority and 0 is the lowest. Both of these operating systems use a round-robin style to execute their processes. They first execute all their high priority processes, then move to a lower level of priority and execute all of those.

2.2 FreeBSD to Linux

A process in FreeBSD has an address space containing a mapping of its program's object code and it's global variables. Processes in FreeBSD also have a set of kernel resources. These resources include credentials, signal state and any descriptor array that gives it access to files, pipes sockets and other devices. A process in FreeBSD must have system resources such as memory and the underlying CPU. Processes in FreeBSD are pretty similar to those in Linux. Both of them stem from a central process using the fork() and exec() commands. This means that both Linux and FreeBSD have the same structure of parent, child, orphaned, and zombie processes. They

even have similar ways of assigning PID's for each process.

Threads in FreeBSD have a structure that contains a scheduling list which in turn contains the thread priority, scheduling priority, the amount of time spent sleeping and status flags. It also contains TSB which is the user and kernel mode execution states, the kernel stack and the machine state. Threads in Linux and FreeBSD are implemented similarly where they are both implemented in a heavyweight fashion. What it means to be heavyweight is that instead of creating threads of a process another process is created instead. These processes essentially just share the same resources, but each one has its own processor to complete its task. This is different from windows which uses a lightweight system.

A programs priority in FreeBSD is characterized according to the amount of computation and the amount of I/O the process will be doing. When a child process is created is shares the same priority level as the parent. The priority for a process is periodically recalculated based on the amount of CPU time that it has used and the amount of memory resources it requires for execution times. Real time processes are processes that have a deadline, and in FreeBSD there is a completely separate queue for these. This queue is completely unaffected by the regular time shared processes, and priorities for real time processes do not change or degrade. FreeBSD also has an idle priority, which is simply the lowest priority and only runs when it is the only thing left, or there are no more runnable processes available. Time shared processes also use a priority for scheduling, but there is a bias for interactive programs such as a word processor. The scheduler in Linux and FreeBSD are fairly similar, where they both use a priority queue and another queue for real time processes. The main difference is that priorities in Linux are not biased or ever recalculated. They usually remain static, where as in FreeBSD there is a bias towards interactive programs and priorities are periodically recalculated.

2.3 Summary

Of these three operating systems Windows differs the most from FreeBSD and Linux, as FreeBSD and Linux are somewhat like relatives. They share similar process creation and similar thread implementation. One large difference of these three operating systems is the fact that windows can have a child without a parent. In FreeBSD and Linux this could eventually cause major problems, but in Windows it is completely fine as nothing relies on that information.

3 Input/Output and Crypto

General input and output is necessary for all operating systems to function. Whether the operating system is Windows, Freebsd or linux, they all need some sort of input and output to manipulate their data in some form. They manipulate this data to execute process and display information to the user. Data is usually managed through some kind of file, an example of this is in linux, where directories can be edited by any sort of text editor like vi or nano. The files can be moved, created and edited through these editors. To start talking about the more complicated topics in more detail we need to understand the definition of a block and character device. Block devices are devices that are able to randomly access fixed size chunks of data. They need to be able to be accessed in any order, an example of this is any sort of storage device such as an USB, hard drive and

solid state drive. A character device are devices that are accessed in order, this is also known as data streams. These devices include things like microphones, keyboards and mice. Now that we have established these two definitions it will be easier to discuss the details of things like functionality, cryptography and scheduling.

3.1 Windows to Linux

In terms of input and output Windows and Linux are drastically different. Their differences range from their style of management systems to the data structures they use to manage requests made by devices. Windows uses Input/Output request packets, also known as IRP's to manage requests. These packets are representations of any request and contain essential header data that will affect the way the request the device made is handled. The system that handles these packets is called the WIndows I/O manager. The manager is responsible for transferring the packets to the various drivers, then to its correct location. The manager does this by using a layered stack, where each driver is cataloged a packet, then it is sent to the bottom of the stack to either one or many different drivers. When the request is completed the manager notifies the application that it has initiated the packet. The request is completed when either one or all of the following conditions are met: If the request is cancelled, if it contains invalid parameters or it no longer needs to be passed down the driver stack. If one of these conditions are met the request is freed by the I/O manager or it is sent to the drive it was allocated to. The main difference here is that instead of a manager, Linux uses a stick to manage input and outputs which is also known as block input/output. The request made with linux are similar uses a stack and file pointers to manage its requests.

3.2 FreeBSD to Linux

The input output structure and management techniques the Freebsd uses are somewhat similar to that of linux. This is mainly because of how the two operating systems are connected in their lineage. Freebsd's core component is similar to Linux's where it uses files for all process and system storage. This is done by using file descriptors which are not managed by any sort of manager like windows uses. The file descriptors are instead accessed and connected by pipes and sockets. The system of using files for everything is usually hidden from the user in Linux and Freebsd and kept in the kernel. Accessing the descriptors and their data is done by using the same functions as files. The similarities of Freebsd and Linux continue as they also use the same data structure of block input and output. The only difference here is a slight difference in some of the variable the data structure contains.

3.3 Overall Crypto

The developers of Windows, Freebsd and Linux are able to use any encryption techniques they want. WIndows uses Advanced Encryption Standard(AES), Rivest–Shamir–Adleman(RSA), and Digital Signature Standard(DSS) as well as a couple others. Freebsd and Linux use some of the same encryption algorithms as windows, and their implementations are mostly identical. The main difference in the implementations are naming, and data structures that are used.

3.4 Summary

Windows, Freebsd and Linux all abstract their file management system under a layer of I/O management. For Windows this is done through the Windows I/O manager and for Linux and Freebsd this is done through file descriptors. These operating systems all also uses the same cryptography techniques when encrypting their devices, where the main difference between the three is the implementation of whatever algorithm it is using.

4 Memory Management

Managing memory is the process of creating addresses that references areas of the available physical memory. This includes RAM, memory caches, and flash memory. Abstracting physical memory is normally done by using virtual memory by using addresses. One example of this is the usage of pages and page tables in Linux. In all operating systems managing memory involves allocating, reallocation and, deallocating memory and organizing it in a way programs can access the memory quickly and efficiently. This is important as kernel level programmers an easier time when developing kernel applications.

4.1 Windows to Linux

The way windows manages memory is slightly different than the linux kernel. One of the first major difference between the two is the size of the pages. Windows pages can only be 2MB or 4KB, while linux uses 4KB and 8KB. The pages however are handled in a similar manner by using paging directories and tables. Addresses are mapped for memory are structured like a tree, where branches come up from the root to different tables. Using a tree data structure minimizes the search times and helps organize the page addresses in a uniform way which helps potential kernel developers. Pages in Linux are essential when it comes to data management for drivers, while in Windows some additional services such as managers, process and stack swappers, page writers and a thread whose sole purpose is to write zeros to the pages. Linux uses something like a FIFO queue depending on the process allocating memory on page uses, while Windows uses states to manage their page uses. These states are free, reserved, committed, and shareable. Free specifies an allocated but unused page, reserved will be eventually allocated and then used, committed means it is private to a specific process, and shareable means it is not private, mapped and any process or thread is able to access it. These page states that windows uses helps it set priorities for process allocation.

4.2 FreeBSD to Linux

The way that FreeBSD and Linux manages memory are primarily the same as many of the core components are exactly the same. There are however some unique attributes that FreeBSD has that Linux does not. One of these is if memory is limited for a system FreeBSD will swap a process into swap memory on a hard drive. This is called context swapping, however it is not completely unique to FreeBSD. Linux and FreeBSD also have similar naming schemes for memory allocation and freeing functions. Where FreeBSD starts to be unique is its memory management technique. When memory is somewhat limited for all processes the FreeBSD kernel will then start to slowly distribute shared amounts of physical memory to each of the processes on the system until it is all

evenly distributed. Exceptions can sometimes be made during the initialization of a process or if the process just tells the kernel how much memory it needs. The kernel will simply just give that process that amount of memory, which will overall aid performance. To prevent completely running out of memory for a process, FreeBSD allows the sharing of virtual memory space in random access memory for processes and threads. The result of this is any change a process makes in the kernel will be visible to other process that share its data.

4.3 Summary

Windows, freeBSD and Linux all roughly implements memory management is a similar way. All three operating systems all implement a form of memory addressing and indexing into a virtual memory space where it connects to some physical memory. All three operating systems also have similar functionality when it comes to allocation, allocation and deallocation of pages. The main difference between the three operating systems is FreeBSD handles limited physical memory differently as it limits resources to all processes and shares the remaining memory to all processes. Windows the size of the pages is what really sets it apart from the other two operating systems. However many differences these three operating systems in memory management, they all have the same ability to use mapping libraries to easily index files and data into address spaces in the kernel.

5 Conclusion

Overall the differences between the three systems mainly lie in windows and Linux, as FreeBSD could be considered an offshoot of the Linux operating system. These three operating systems may implement things differently, but in the end they are all able to accomplish the same thing even with different implementations.

References

[1] A. L. Mark Russinovich, David A. Solomon, “Windows internals part 1,” Microsoft Press, 2012, (Accessed on 5/9/2018).

[2] R. Love, “Linux kernel development,” Pearson Education, 2010, (Accessed on 5/9/2018).

[3] R. N. W. Marshall Kirk Mckusick, George V. Neville-Neil, “The design and implementation of the freebsd operating system,” pearson Education, 2015, (Accessed on 5/9/2018).