-

Final Paper:
Operating System Feature Comparison

Rhea Mae V. Edwards

CS 444
Spring 2017

I. Introduction

There are many components and characteristics to an operating system, and many of those features differ depending on the operating system that a device is using. Among all of those differences though, there are also many similarities between the different operating systems, such as the structure or simple implementations that is used to have a function do what it is suppose to do. Many operating systems purposes are meant to reach the same goal of meeting the for not only computer but for also its user, and how they go about to meet that goal varies.

This report will focus on comparing certain features of the Windows and FreeBSD operating systems to the Linux operating system. The three topics of processes, memory management, and file systems that an operating system implements within a computer, will be covered.

II. Processes

When it comes to the implementations of processes, threads, and CPU scheduling across a variety of operating systems, such as Windows, FreeBSD, and Linux, there are many similarities and differences that exist when they are compared to one another. This section with focus on comparing the implementation of Linux to each of the Windows operating system and the FreeBSD operating system.

Processes

A process can be defined as the flow at which a task is used during a program. Processes have states, at first initializing the process itself, then put into a ready queue, waiting to be ran. While being in the running state, is when the process ends naturally by exiting or somehow blocked. For example by being killed by an external process. If a process is blocked, the only way a process can move out of that state, is to be moved back into the ready queue, which it is then can only be moved forward back to the running state. Another characteristic of a process' state diagram is that a process can cycle through the running state and the ready queue.

Types of processes that can exist within an operating system are parent processes, child processes, orphaned processes, and zombie processes. Where child processes are created or forked from parent processes, and how orphaned processes are child processes that have been terminated but have not been removed completely and properly, and where the parent processes are still well running. Which then a zombie process is formed from an orphaned process that still has not been properly removed, and the parent process somehow ended without properly working with the orphaned process.

There are two ways a process can be processed, which are CPU bound, usually being faster, or usually I/O bound, being slower. The way these processes are used differ between various types of operating systems where some characteristics can still remain the same.

A. Windows

The Windows operating system runs by multiprocessing, being considers a CPU bounded process used throughout the operating system itself [1]. The Linux operating system supports real time processes, which work fairly quickly being considered as a CPU bound way of processing processes, causing similarities to the Windows operating system when compared [2].

FreeBSD

FreeBSD is a multi-tasking operating system, which is considered as a CPU bounded process for processing processes [3]. Linux implements real time processes, similar to FreeBSD, which work fairly quickly being considered as a CPU bound way of processing processes [2].

Threads

Threads are the pieces of executable program instructions that consist of a process. The ways threads are purposely used can differ and also hold similarities throughout various operating systems.

Windows

The Windows operating system implements threads through the method of multithreading [1]. Based on Linux's way of processing processes, relating to the implementation of threads throughout the operating system, Linux is similar to the way Windows implements multithreading with its threads within a process [2].

FreeBSD

FreeBSD being a multi-tasking operating system, characterizes FreeBSD implementation of threads in a multithreading fashion [3]. The FreeBSD is similar to Linux's way of processing threads, when analyzing the comparison with FreeBSD's implementation of multithreading and Linux's implementation of multiprocessing [2].

CPU Scheduling

CPU scheduling is the way a process is conducted by the planning and execution of the threads within a process. There are two different types of schedulers, low level schedulers and high level schedulers. Low level schedulers run the queue per CPU, select threads from the highest priority queue, and runs a round robin system for a quantum per thread. High level schedulers are primarily for SMP, where there is processor affinity, equidistribution, and multi-core and multi-socket implementation, being a O(1), constant time scheduler.

Various operating system implement their CPU schedulers differently which can also hold similar characteristics.

Windows

The Windows is a priority-driven preemptive scheduler, which implements CPU Scheduling as a low level type of scheduler [1]. The Linux operating system uses pre-emptive scheduling, which is considered as a high level scheduler, differing from the Windows operating system that implements a high level scheduler [2].

FreeBSD

FreeBSD implements its CPU scheduler in a multitasking environment, which would be considered as a high level type of scheduler [4]. The Linux operating system uses pre-emptive scheduling, which is considered as a high level scheduler, similar to the FreeBSD operating system [2].

III. **Memory Management**

The Basis of Memory

The memory subsystem for an operating system is very important and essential for the extensional use and capabilities that truly makes an operating system in what it is.

Different types of memory can be store in two different location within an operating system. One location is the heap, which tends to store dynamically allocated memory or variables, and the other location being the stack, which holds local variables used within a file. Usually, anything that is related to memory that is not the stack, bss (block started by symbol), text or data, is located within the heap. In further detail, information that is persistent and non-local, and that will have to be cleaned up at some point within a program will be on the heap, and on the other hand, any pieces of information that is non-persistent and local to a program, and that is usually meant to be cleaned up automatically by the system, is located within the stack. With the inclusion of address binding, helps the operating system actually know where every variable lives within memory, which is set to each variable when the program is executed.

There are two different types of memory, logical or virtual memory and physical memory. In simple terms, on a modern system, a logical process is able to be done because of the physical connections of a device. For example, logical address space is backed up with physical address space. Logical addresses are different from physical addresses within a system. There is memory isolation with virtual addresses, because it is unwanted to share memory to the same physical addresses.

The structure that is used to organize these memory addresses are page tables. From a virtual page to a physical page, there is some sort of mapping that is done in order to have such memory addresses to be transferred. Each physical address within a table contains a number of virtual addresses. These virtual addresses are split between the reserved space to either hold the page number or to be the offset of the address. Page tables are organized in a tree structure.

The typical organization of these page tables is a tree structure consisting of three levels, even though theoretically there can be numerous levels to such a tree structure. The first level being the page global directory, then moving o the second level being the page middle directory, and then the third level actually being the level that contains the the page table, which each page table contains physical addresses of pages that each lead to a physical page. There are also different sizing names depending on the size of a tree structure and its pages overall.

So knowing that page tables process address space for memory, the actual location for page tables in a system goes the every back-end to the control register 3 of a device. This register holds the pointer of the current process, being the base address for memory.

Memory Management

Managing memory within an operating system can be done in a variety of ways. Within the x86 paging modes, there are mainly four different modes that can be practiced:

1) No Paging:
   By not applying a paging system with memory, is a management mode, which does have a limit in how much memory an operating system can store overall.
2) 32-bit Mode:
   Such a mode has a 32-bit law (logical address width) and at the most a 40-bit paw (physical address width).
3) PAE Mode:
   This mode is similar to the 32-bit mode of page management, where it has a 32-bit law (logical address width), but can go up to having a 52-bit paw (physical address width).
4) IA-32e:
   For this mode of page management, it has a 48-bit law (logical address width) and up to a 52-bit paw (physical address width). IA-32e is also considered the official standard, the mode that is mostly used throughout the industry.

And with each paging modes, there are different paging structures that manage and organize each practice in its own way.

An operating system can hold up to so much memory, but it does has its limits. When an operating system runs out of space to hold its memory, a method is used called page swapping, which uses other pages (that are already in use or have been used) to remove and write the newest memory to. This process temporary removes and stores usually the least used pages within memory to use. There is also the idea of thrashing with deciding which pages to remove and use within the system again, which takes the least-wanted page behavior, but such method usually slows down the hard drive at about 90% when in use.

Windows' Memory Management

The Windows operating system's memory management organization has its similarities and differences when its compared to Linux's memory management system.

Overall, Windows' memory management, by default, contains 32-bit processes 2GB in size. This size can be increased to 3GB or 4GB in size on a 32-bit or 64-bit Windows operating system, respectively. 64-bit processes of a Windows operating system have 8192 GB (8TB) of virtual memory space.

Windows' memory management also offers multiple services, such as:

- Address Mapping
- Paging
- Memory Mapped Files
- Copy-On-Write Memory
- Physical Memory Allocation and Use

Windows also has the idea of a working set, which is a set of pages that are physically present. There are also five (technically six, which has been found not to be all that useful) key parts, that can be used:

1) Working Set Manager
2) Process/Stack Swapper
3) Modified Page Writer
4) Mapped Page Writer
5) Zero Page Thread

Windows' memory management system is also fully re-entrant, with a finely grained locking structure. This provides:

- Allocation and Freeing Virtual Memory
- Shared Mappings
- Map Files
- Flush Page

There are two differentiating page sizes for withing Windows' pages, being large (2MB) and small (4kB) that it supports.

Also with Windows' memory management, there are a variety of page states that can exist, being:

- Free
- Reserved

    A state at which when a page is requested but not currently in use, which is also private.
- Committed

    A private state, that offers valid mapping, resident, and demand-zero.
- Shared

    A state that has valid mapping and resident.

When you create a process in Windows, it creates a container of threads. A user can have as many of processes as it wants, and when a thread is created, it is consisted in the heap, stack, or any other data structure it is held in.

Similarities to Linux

The overall and main structural format and layout of the Windows memory management and Linux memory management are closely similar. Certain characteristics and functionality that have been explained and described in further detail.

Differences from Linux

A difference that Windows operating system has from the Linux operating system is that Linux calls there larger sized pages huge pages, versus where Windows calls them large page, which has been explained previously. When a process is created in Linux, it also creates a container of groups, which varies from Windows memory management that creates a container of threads or processes.

FreeBSD's Memory Management

The FreeBSD operating system's memory management organization also has its similarities and differences when its compared to Linux's memory management system (and possibly with its compared to the Window's memory management system as well).

Similarities to Linux

A similarity that FreeBSD has to Linux, is that it handles its BM structure differently that Linux. FreeBSD's memory management system also does its licensing differently when compared to Linux.

Differences from Linux

Similarly to Linux, the FreeBSD's memory management system also has page tables and more along those lines, and open source implementation.

IV. **File Systems**

In the olden days, what was considered a file system, was simply just the sector numbers of a file. That is all that an operating system really needed to keep track of files throughout the system. But the idea of just having numbers as file names, was not very tolerable by an average human mind, so the invention of extensions, naming files, and having a more detailed and user-friendly file system came into light.

Characteristics of a File System

File systems not only consist of file and directory structure, but they also have certain capabilities, abstractions, and implementation details that are also very important to consider. Three requirements that make a file system a functional file system are that it is able to hold more than one file, it can survive a reboot, and one is able to navigate through the system itself. Most commonly, a file system's structure stores its files as a sequence of bytes, which undergoes a record based system and a hierarchical structure (which is mainly for its user, not for the operating system actually). A file type consists of a magic number and a file extension, being that the magic number is for the operating system and the file extension for the minds of the eyes using the creation.

A file system is characterized by the structure of a B-tree, a B-tree file system (Btrfs), and that it also consists of a file allocation table (FAT). The idea of a file system is mostly completely abstract, so on top of all of the file system within an operating system, there is a virtual file system (VFS). It acts like a universal I/O system where it mainly reads and writes.

The structure of a virtual file system, generally follows the following flow of control:

1) First off, a system call is made in user space.
2) Which then writes to the a virtual file system write, moving into kernel space at the virtual file system later.
3) While still remaining in kernel space, moves into a file system-specific write at the file system later.
4) Then gets access to any relevant media, in regards to what has been requested to do.

File abstraction contains everything that is needed for a file. There are four abstractions within a virtual file system:

1) Files - Representing an open file
2) dentries - Path components

   Which also consists of the idea of "walking a file system", in other words, traversing the tree structure of the file system. For example, with the path of */bin/vi*, the *root*, *bin*, and *vi* are dentries, with less error. Namespaces is where a file system is rooted.
3) Mount Points/Superblocks - Structure representation of mounted file systems
4) inodes - Logical representatives in a file system

Behind a File System

Storing files in the system inherits the idea of memorization, where the data number is stored in cache, which is used at the algorithm level versus the system level. In memory cache and block cache, there are not always files being dealt with. There are two types of locality within a operating system:

1) Temporally Local - Focuses on the idea that a file was used once before, it would most likely be want to be used again
2) Spatially Local

Now focusing more on the concept of spatially local, such locality has a faster I/O on average. The amortization, "the cost of algorithms over a sequence of operations" [5] can be calculated with the following equation:

$$hA_C + (1-h)A_D$$

where... $h =$ Hit Rate (in 10's to 100's microseconds), $A_C =$ Cache Access Time, $A_D =$ Disk Access Access Time

The higher the hit-rate of a file system, the better. And some ways to have a higher hit-rate, include the following

- Bigger Cache

  Which is not typically possible though
- Predictive Caching

  When there is caching done while pre-loading
- Invention Model

  When the system knows what to get rid of (but not very realistic though)
- Intelligent Caching

  Knowing what to cache, machine learning, and knowing about files' access amount

Usually files are stored on the hard-drive or on the disk called backing store. And along with storing files, there is the reading and writing of files, where a file is read from the disk, through cache where a copy of the file is made, which then is given to the program. Writing works a little differently, the source is different and it can be out of sync, which can be a problem at times. But some writing strategies that may be better to be practiced in the long-run include the following:

- Write-Through
- Write-Back

  Where some files may have to be marked as "dirty"
- No-Write
  - Where files are completely ignore the cache, not a common strategy, but a strategy
- Ideal Eviction Strategy
  - Usually done when the cache is full.

Describing the ideal eviction strategy in greater detail, this writing strategy is the most ideal case. The ideal eviction strategy blocks what files are least useful and throws away those files when needed. This writing strategy is a little more efficient than the least recently used method, which throws away the oldest untouched blocks. The least recently used (LRU) strategy is usually defined as LRU-n, where n represents the number of levels used, where this strategy involves a multi-level list to complete is task.

A summary of how the least recently used method works is that there is usually a hot list and a cold list, a two level list. The block with the higher hit rates, ones usually with the greater priority are kept in the hot list, and the all of the other blocks are kept in the cold list. Files are cycled through the cold list, which is circular, and files found there with the lowest hit rate, in other words the oldest blocks, are thrown out and used for other files when the cache is full.

The more levels this list has, the more ideal it becomes, but each level becomes more insignificant to the system. In reality, there are not usually methods with more than three levels. All of this structural information is stored as a radix tree, which is purely logical and read from the disk. And on another note, each block within this system has its own process id, in order to differentiate between each one.

Windows' File System: NTFS

The new technology file system (NTFS) is the main file system for Windows. This file system scales to about 16 exabytes, 256 terabyte system limit as of Windows 7. Most likely have scaled up to be greater as times goes on, and operating systems have improved and updated throughout the years. There are file junctions where you can then recombine files, with the existence of alternate data streams. Such a file system also supports software arrays.

Similarities to Linux

Similarly to Linux, it characterizes B-tree code, along with the aspects of soft and hard links used throughout the system.

Differences from Linux

At the file level, the new technology file system has native compression support. Either it completely finishes or does not start at all. With this file system, it also natively encrypts files. The new technology file system supports the attribute of storage pools.

FreeBSD's File System: ZFS

The zettabyte file system (ZFS) is the main file system for FreeBSD. This file system scales to the zeta-byte range at 10^{21}. In addition, it has a time machine attribute of this system that adds time of a file as a part of the file itself, which is a unique feature that make such a file system really special. Such an attribute provides a more organized and well-ran system of file overall, where it is used to read and write caches, besides for superblocks. At the on-disk state, being a transactional file system, it tends to be pretty consist. Checksums of this file system also stores everything of a file that it reads and writes.

Also the zettabyte file system contains many types of characteristics that play a part of its archival quality, such as...

- Checking bugs
- Getting parity errors
- Ability to reconstruct aspects related to the file system

The primary goal of this file system is the use for large arrays, where in most cases uses a spinner drive to do so, which are more cost effective compared to solid state drives. This file system is also open sourced, able to talk to Window natively, and can have million of snapshots throughout its system, which is pretty impressive.

Similarities to Linux

Zettabyte file system uses a hierarchical file system (HFS+), which is similar to Linux. This file system also implements copy-on-write being a large file system, which can have files point to the same block. This action is done at the block level of the system. Then in regards to the underlying hardware layout, the zettabyte file system is completely independent of its existence.

Differences from Linux

Zettabyte file system never overwrites old data, it keeps it around, unlike Linux, due to its inheritable file structure. This file system characterizes grade-z layouts, which are more effective for its great data storage needs. And another unique feature of the zettabyte file system is that it also characterizes a pool storage file system.

V. Conclusion

All in all, there are many more components to an operating system that can be done in a variety of ways, such as the implementations of interrupts, synchronization, and I/O and additional provided functionality. Processes, memory management, and file systems are just a couple one can compare and talk about, and how these comparisons differ and share similar characteristics across operating systems such as Windows, FreeBSD, and Linux. There is so much more one can learn! Just need to get out there and have an open mind and experience it.

References

[1] W. Stallings. (2005). *The Windows Operating System* [Online]. Available: http://avellano.fis.usal.es/~lalonsoamp_inf/windows.pdf

[2] D. Rusling. (1999). *Chapter 4 Processes* [Online]. Available: http://www.tldp.org/LDP/tlk/kernel/processes.html

[3] *3.8. Processes and Daemons* [Online]. Available: https://www.freebsd.org/doc/handbook/basics-processes.html

[4] *2.4. Process Management* [Online]. Available: https://www.freebsd.org/doc/en/books/design-44bsd/overview-process-management.html

[5] T. H. Cormen., C. E. Leiserson, R. L. Rivest, C. Stein. (2009). *Chapter 17: Introduction to Algorithms, third edition* [Online]. Available: https://mitpress.mit.edu/books/introduction-algorithms