Florida A&M University

Operating Systems Research Paper

Christopher Griffin

COP 3610

Dr. Harmon

April 9, 2018

**Table of Contents**

Introduction…………………………………………………………………………….3

Process Scheduling……………………………………………………………………..3

Threads………………………………………………………………………………….5

Inter-Process Communication…………………………………………………………..6

Memory Management…………………………………………………………………..8

I/O………………………………………………………………………………………11

Security…………………………………………………………………………………13

Power Management……………………………………………………………………..14

File System……………………………………………………………………………...15

User Interface……………………………………………………………………………18

Conclusion……………………………………………………………………………….18

References……………………………………………………………………………….20

During the 1940s and 1950s, man was forced to use vacuum tubes to operate and program a functional digital computer. Now, here in 21st century, it is phenomenal how far computers have advanced. Man has even discovered how to turn a machine that previously would have had millions of vacuum tubes into a computer that can fit into a person’s pockets. However, to discover how this is possible, one must look at the modern day operating systems that control these computers. An operating system is a piece of software that manages a computer’s hardware resources, as well as acting as an intermediary between the computer’s hardware and software applications. Several components and attributes contribute to the workings of the modern operating system. In addition, many different operating systems have been created using different variations and combinations of these components. How do the most popular operating systems function? What about their predecessors? To answer these prominent questions, nine different features of several different operating systems will be explored.

One of, if not, the most important functionalities of any operating system is the ability to decide which process should undergo execution at a given moment. This function is completed using a process scheduling algorithm. Some of the more common examples of process scheduling algorithms are First In First Out (first process that is ready can begin execution), Highest Priority (process with the highest priority beings execution), and Round Robin (process executes until a time limit, also known as time quantum, has been reached, causing it to exit execution). Round Robin is an example of preemptive scheduling, meaning that it can interrupt a process while it is in execution to and send said process out of execution. Unix, Windows, Mac OS X, and iOS operating systems all use a combination of highest priority scheduling and round robin scheduling. They use highest priority to select the process to execute and use round robin (pre-emptive) scheduling to periodically pull processes from execution. In the Mac OS X system, if a higher priority process becomes ready while a lower priority process is running, that process will be preempted in favor for the higher priority (Singh). In Unix, if more than one processes have the highest priority, the process that has been waiting the longest will execute first (Bach 247). IBM OS 360/370 allows first in first out scheduling or non-preemptive highest priority scheduling (Lunny 7). The Android and Ubuntu operating system uses a more unique form of scheduling called Completely Fair Scheduling. This is another example of preemptive scheduling where a process can be scheduled within a constant amount of time, regardless of what is running (Kayande 18).

Also, deadlocks can occur during process scheduling. This is when multiple processes are awaiting resources that will never arrive because said resources are forever held by the other processes. There are three ways of treating deadlock. One way is to just ignore it, which is what Windows and Unix does (“Operating System | Process Management | Deadlock Introduction”). Other operating systems choose to try to anticipate and recover from deadlocks. In some way, OS 360/370 takes this approach by canceling the jobs responsible for said deadlock after some time (Lunny 10). The final way is deadlock prevention, which operating systems like Mac OS X performs.

Multics (being an older operating system) used some very unique methods for scheduling processes. In 1967, a scheduler known as the “Foreground – Background” scheduler was created, acting as Multics’s first scheduling algorithm. In essence, this scheduler is very similar to the Round Robin with Multilevel feedback scheduling algorithm discusses already. However, in this algorithm, lower priority queues are given a longer time quantum than higher priority queues. In addition, when a process’s time limit is finished, it is sent to the next lower quantum from the one it was in before. In 1975, Bob Mullen created another scheduling algorithm known as the “Percentage” or “Workclass” scheduler. This algorithm assigns different percentages of the machine to different groups of users. This algorithm actually acts on top of the Foreground- Background scheduler as that algorithm still hands the scheduling within each given group. This scheduler would later be improved by Mullen throughout the next year and would later be renamed the “Deadline” scheduler (“The Multics Scheduler”).

A fundamental part of any operating system is how it utilizes processes and threads. A process is simply an instance of a program. A thread is path of execution within a process. It is also referred to as a “lightweight process” due to having many similar properties as a process. Unlike a process, threads can be executed concurrently and can share information with each other. Also, a thread contains its own program counter, register set and stack space. Though referred to as “tasks” back in 1967, the traditional “thread” was not developed until the late 1970s for Unix. Thus, older operating systems such as Multics, OS 360/370, CDC NOS VE/BE, and Burroughs MCP did not have access to threads (O’Sullivan).

Windows and Unix take on threads is quite interesting. A “single” thread is actually two threads: a user and kernel thread. The user thread only runs when the kernel thread switches to it. These two operating system’s scheduling algorithms revolve around threads, not processes, including having their own queues.

Windows has a facility known as Win32 thread pool that handles the allocation of a limited number of threads and the maintenance of thread queues. Threads are assigned their own ID number and can be dynamically allocated via a Win32 system call (Tanenbaum 911-912).

In Android, all threads have an assigned priority, with its initial priority being equal to the priority of the thread that created it. A thread can also be marked as a “daemon” which is used to call the “main” method. Ultimately, the Java Virtual Machine will execute threads until an exit is called or all non-daemon threads are destroyed (“Thread”).

With Mac OS X, threads can change their priority levels during scheduling. They can also be marked as real-time priority which will allow them to inform the scheduler how long it will need to run during the next several clock cycles. It will also be able to mention whether or not the execution will need to be contiguous (“Kernel Programming Guide”).

While processes are being scheduled, it may be important for processes to communicate with each other and even synchronize their actions. This is a methodology known as “Inter-process Communication”. Two notable ways the processes can communicate are shared memory and message passing. Shared memory utilizes some variable that is shared between multiple processes. Message passing involves a communication link being established between multiple processes, thus allowing them to send and receive messages to and from each other.

Unix and Ubuntu use what is known as a System V IPC Package (IPC is short for inter-process communication) which consists of three mechanisms (“Ubuntu Manpage: Svipc - System V Interprocess Communication Mechanisms”). Messages allow processes to send formatted data streams to arbitrary processes (in other words, memory passing). Shared memory allows processes to share parts of their virtual address space. Semaphores allow processes to synchronize execution. In addition, Unix also offers a more primitive version of inter-process communication for tracing (or debugging) processes (Bach 356-359).

Android uses a memory passing methodology called Binder. In contrast with Unix, Android does not support System V, as it lacks the ability to clean up buggy or malicious applications. Binder itself is comprised of three layers: the kernel module, the object-oriented userspace API, and the programming model. The kernel module itself is what implements the inter-processing communications. The object-oriented userspace API allows applications to interact with the inter-process communications. Lastly, the programming model is where applications declare their inter-process communication interfaces (Tanenbaum 816).

Windows supports a number of different inter-process communications techniques which include pipes, sockets, remote procedure calls, shared objects, and mailslots. Pipes can be separated into two categories: byte and message. Message-mode pipes preserve the message boundaries of a message while byte-mode focuses on the bites themselves, just like in Unix. Mailslots are much like pipes, although it only has one category. However, mailslots can be used to send a message to multiple receivers, as opposed to just one, and a successful delivery is not guaranteed. Sockets are also similar to pipes, although sockets are used to connect processes from different machines. A remote procedure call (RPC) is used by a process to request another process to perform some procedure and to store the results back into the original process. Lastly, shared objects can be placed into the virtual memory addresses of multiple processes at once. In addition, when one process makes a change to the object, all other instances of said object changes as well. Windows also support multiple synchronization methods including semaphores, mutual exclusion, critical region, and events (Tanenbaum 916-917).

Mac OS X’s and iOS’s synchronization methods are limited to locks and semaphores. Locks (or mutual exclusion) protect resources by only allowing one thread to use it at a time. Semaphores are much like locks, however multiple threads can use a certain resource, though there is still a limited amount of “locks” for said resource (“Kernel Programming Guide – Synchronization Primitives”).

IBM’s OS 360 rules for inter-process communications are defined that restrict what communications to certain parameters. These rules can certainly be bypassed, but this can cause negative side effects that will be difficult to spot (“IBM Operating System/360 Concepts and Facilities”).

With Multics, data exchanges must be done within a shared database, also referred to as a “mailbox”. The ability to share databases in Multics is the major reason why effective inter-process communications is made possible. Using this method, cooperating processes simply need to agree the segment to be shared. Then, they simply need to reference the segment via its unique file name. In addition, mutual exclusion (locks) can be used to guard against unwanted references (Spier 84-85).

While process scheduling is very important for a computer, memory management is just as, if not more important. Memory management is responsible for tracking and managing the allocated and freed memory in the memory hierarchy. Some of the responsibilities may include deciding how much memory a process needs, determining when a process is to be allocated, and much more.

There are several different techniques involved with memory management. Swapping is a common technique that involves transferring entire processes from main memory to disk and back. Unix in its original incarnation used swapping heavily. When the BSD system was created, Unix used demand paging policy. This technique involved moving memory pages rather than entire processes (Bach 271). Ubuntu also uses swapping and demand paging policy in its operating system. Overlapping is another technique used by Unix ("Memory Management In Unix Operating System Computer Science Essay."). This technique involves splitting a program into pieces, and later replacing old pieces with new pieces, similar to page replacement.

More techniques in memory management revolve around partitioning and allocations. Single-partitioning allocation uses a relocation-register scheme. OS 360/370 originally used single-partitioning in response to its batch system. OS 360/370 can also use multiple-partitioning allocation (Lunny 8). Multiple-partitioning allocation divides main memory into partitions to hold a single process each. Multiple-partitioning allocation can be partitioned using fixed or variable partitions. Fixed partitioning divides memory into an unchanging partition size. This technique is quite simple, however it can lead to internal fragmentations, which is when memory partitions are larger than is what is requested. Variable partitioning can solve internal fragmentations. This allows memory to be dynamically split into different sizes. The downside to this is that external fragmentations can be caused. This is when holes of free memory are scattered throughout the main memory. Non-contiguous allocation is a solution for this.

Some more memory management techniques include segmentation and memory management. Segmentation is the dividing of a logical address space into segments and memory mapping is the act of treating data as if it is in the main memory when it is not. Multics uses segmentation and memory mapping together, while Android use memory mapping with demand paging (Tanenbaum 813). Windows uses the same memory management methods as Android and does not support segmentation in any form (Tanenbaum 933). In addition, Windows utilizes file mapping which involves representing addressable objects as files (Tanenbaum 873).

Paging is a major technique used for memory management that was touched on slightly earlier. Paging allows the physical address space of a process to be non-contiguous by splitting the process into pages. Like overlapping, there must be a way to replace older pages with newer pages if there is no more memory available. The algorithms for this are known as page replacement algorithms. There are three notable page replacement algorithms, but only two of which are used by operating systems. The first one to mention is the usual first in first out scheduling. This algorithm simply removes the oldest page and adds the newest page. Least recently used is another page replacement algorithm that (as the name suggests) replaces the page that has not been used the longest. Lastly, there is Belady’s optimal page replacement algorithm, which is the algorithm that is not used by operating systems. This algorithm came in response to, what is known as, Belady’s anomaly. This anomaly refers to the first in first out and is when the number of page frames actually increases the number of page faults, instead of decreasing it. Belady’s optimal algorithm replaces the page that will not be used for the longest time. The reason that this algorithm is not used is somewhat obvious. An operating system has no way of knowing what future requests will be, so this algorithm is more theoretical than practical.

As operating systems that uses paging, Unix and Ubuntu would need some page replacement algorithm to use. In response, the Unix, Linux, and Ubuntu operating systems use the least recently used replacement algorithm ("Memory Management In Unix Operating System Computer Science Essay").

Android has some very interesting capabilities when it comes to getting rid of unnecessary memory. The two functionalities used by Android are known as the Out-of-Memory Killer and the Garbage Collection. Garbage collection is when pieces of memory that has not been used for a while is freed back into the heap. This is done without the input of the user and it keeps the memory use of the Android device optimal ("Overview of Android Memory Management."). The Out-of-Memory Killer takes action when the device is low on RAM space. In response, the OOMK kills processes that are the least critical in order to free more space (Tanenbaum 813).

Another operating system with interesting memory management features is Mac OS X and iOS. They utilize virtual memory system that stems from an older design called Mach VM. The new and improved virtual memory utilizes a Universal Page List. This list allows communication with the virtual memory system. Specifically, it can be used to request certain paging behaviors. Pagers are used to get data into VM entries. Three types of pagers include vnode pagers, default pagers, and device pagers. The default moves “normal” data in and out of storage. Vnode pagers are generally used for memory-mapped I/O files. Lastly, device pagers allows the user to map non-general purpose with cache characters needed for the that memory ("Memory and Virtual Memory.").

Another major functionality of any operating system is the ability to manage input and output devices. A common piece of I/O systems is drivers. Drivers manage particular devices or buses. Windows I/O system consists of four components. Plug-and-play services detects when new devices are enabled and searches for a suitable driver to load the system into. The device power manager adjusts power states and reduces power consumption when the device is not in use. The I/O manager itself supports the manipulation of I/O kernel objects. Lastly, the device driver is the kernel module that controls all of the devices (Tanenbaum 944).

Unix also utilizes device drivers while also specifying two different types of devices. Block devices act as random access storage devices. Raw or character devices act as everything else, including terminal or network devices. Unix also supports “software devices” which are devices that are not physical (Bach 948).

Mac OS X’s I/O kit is serves as a framework for driver development. This framework is actually a complete redesign from Mac OS 9. With the new I/O kit, much of the driver code is already written and all hardware support is provided directly to them. This code is much more reusable than prior operating systems created by Apple.

Mac OS X’s I/O kit has three important elements. “Families” defines a collection of high-level abstractions common to all devices and provides services for several different types of devices. Drivers is the second element, which has been explained prior. Lastly, nubs act as points of connections between drivers (“Kernel Programming Guide”).

Multics utilizes a early software for I/O control known as the input/output control system. However, unlike operating systems before it, Multics is able to use IOCS in a more secondary fashion. This is because some of its functionalities is usable in other parts of the operating systems, such as Multics’s file system and traffic controller. Multics’s I/O system also utilizes dynamic switching for its I/O devices by creating interfaces for all of its I/O devices. Each I/O device is also given a symbolic name which can be dynamically modified (Feiertag).

OS 360’s I/O architecture makes use of one of its model-independent interfaces known as the Channel-to-Control-Unit Electrical Specification and Signaling Protocols. This interface allows peripheral devices to work across the entire range of models. It allows I/O devices to interchange between different systems. A compatible system for I/O control is made possible since commands and status indications are made consistent no matter what (Gifford 294).

Another important piece of an operating system is its security. Security provides protections to an operating system’s resources and the information stored in the computer. For an old operating system like OS 360, the safeguarding of data was limited to passwords due to the lack of user management. Passwords were stored into a control table which itself required a master password to access. With OS 360, data was able to be flagged as protected which would require a password to access (Lunny 11).

Multics uses tables to list users and their access to data. This table is known as the Access Control List. However, the table is vulnerable to many security exploitations. The hardware architecture and software utilization of the table can create exploitative errors that can leave them open to trap doors. The table itself can also be circumvented implementation errors (“Multics Data Security”).

Unix and Android use user ID’s to protect the user’s files (group ID’s are also available to use). When the user creates a file, the file is given the user’s ID. The user then has the ability to set permissions on the file. In other words, user can decide who can read, write, and execute certain files (Bach 798).

Unix, in particular, has special types of accounts. Administrative accounts searches for and disables dormant accounts and checks that all accounts have passwords. Special accounts disables accounts that have no passwords as well as checks that there are no shared accounts. Lastly, root accounts restrict the number of people who know the root password (“Unix Security Guidelines” 1-2). Mac OS X and iOS actually follow Unix’s security model (Halvorsen 33).

To expand on Mac OS X and iOS, two particular security models are used to authenticate users, the user-level security model and the kernel-level security model. The kernel security model uses two mechanisms, user credentials and ACL (Access Control List) permissions. There are also other security measures such as solving certain paging problems. With the kernel, it is possible to “wire down” memory to prevent pages from being written out to the backing store (“Security Considerations”).

Windows uses a security reference monitor which meets all of the requirements of the “Common Criteria”. The Common Criteria is an international standard for computer security. As a bonus, Window’s new and improved security model now limits the privileges of the administrator when he or she is performing non-administrative tasks. This is known as User Access Control and no longer allows users to navigate with unnecessary abilities that can be maliciously exploited. In addition, a new Admin Approval Mode now requires both regular users and administrators to enter the credentials anytime they attempt to perform something that requires administrator privileges, although this is an optional feature ("The Windows Security Model.").

Ubuntu’s security has a lot to digest. First of all, administrative root accounts are disabled by default. Instead, the user is encouraged to use the “sudo” commands for administrative duties. Also, firewalls are available using Linux’s Netfilter system and Ubuntu’s own Iptables. The netfilter monitors and controls the network traffic while iptables assigns rules created by the userspace. Cryptography methods such as eCryptfs and Certifications are available and AppArmor helps to protect individual programs. Lastly, passwords require a minimum of six characters and the Ctrl+Alt+Delete functionality can be disabled (“Ubuntu Documentation – Security”).

In this day-and-age, computer systems have become less stationary. It is now a common to have a computer system in a person’s pocket at any given time. For this reason, power management has become a crucial necessity in the operating system. Take Window’s operating system. Like most operating systems, it puts the computer into a low-power sleep mode to conserve power. The computer will enter sleep mode when the computer is idle of the power button is pressed. Hardware interrupts, such as the internal clock, will release the computer from its sleep mode.

Android uses wake locks to help with power management. Wake locks all the android computer to enter sleep mode. In order for the computer system to be “awake”, the wake lock must be held.

In Mac OS X and iOS operating systems, power management is a responsibility if its I/O kit. Its drivers has the ability to transition between different power states in the same way that the system can request that a device change power states. The I/O kit also maintains a tree of power dependencies known as the “power plane”. The power plane displays all of the devices that support power managements as nodes connected to a parent node (Halvorsen 205-207).

Ubuntu has specialized software for handling its power management needs. ACPI allows Ubuntu to configure power management for hardware devices. ACPI has different states of power management including S1, S2, S3, and S4. S1 maintains power to the CPU and RAM but all processor caches are flushed, CPU stops all executions, and other hardware devices are powered down. In S2, CPU is powered down and in S3, the RAM is put into sleep mode but is still powered. Lastly, in S4, the RAM is transferred into the hard drive and the system is sent into hibernation mode (i.e. powered down) (“Ubuntu Documentation – Power Management”).

Ubuntu also contains a software package called “gnome-power-manager”. This is graphical user interface that shows battery life and shuts down, hibernates, or dims the screen under certain conditions (“Ubuntu Documentation – Power Management”). Ubuntu also has an API known as “Upower”. It serves as an abstraction for enumerating power devices, listening to device events, and querying power history and stats (“Power Management”).

Arguably the most important functionality of any software (let alone operating system) is the ability to manage and store data. In particular, an operating system requires the ability to store and manage files. A file is a collection of related of information recorded. The tool for managing these files is known as a file system.

A very common type of file system is a hierarchal storage system. These involves directories containing other directories, similar to a tree data structure. The first operating system to use such a file system is Multics (Vleck F). Since then, many later operating systems such as Unix, Ubuntu, Mac OS X, Android, Windows, and others. This type of system utilizes directories, which act as a special type of file that contains a list of entries. Entries themselves require a unique name and can other point to a file (known as a branch) or point to other directories (known as a link). When the user is operating a directory (then known as a working directory), he or she now may access a file located in the working directory. However, as mentioned in the security section, Multics has an Access Control List that monitors and controls the user permissions of different files (Daley & Neumann).

Windows has access to three notable file systems: FAT 16, FAT 32, and NTFS (NT File System). FAT 16 supports 16-bit disk addresses, which keeps disk partitions to a limit of 2 GB. This file system is mainly used for accessing floppy disks. FAT 32 supports 32-bit addresses, which can keep disk partitions to a limit of 2 TB. Due to lack of security, this is mainly used for flash drives. Lastly, NTFS supports 64-bit addresses, which can theoretically keep disk partitions up to 2^64 bytes.

Ubuntu also uses the three file systems listed in Windows. NTFS is actually Ubuntu’s default file system. However, being an open source operating system, Ubuntu has access to much more file systems. These include ext2, ext3, ext4, reiserFS, JFS, and XFS (Simon).

Android’s share of file systems can be split into different categories. exFAT, F2FS, JFFS2, and YAFFS2 are common flash file systems that Android devices can utilize. Media-based file systems that Android supports include some of Window’s notable file systems (FAT 12, FAT16, and FAT32) and VFAT, an extension of the FAT file systems (commonly used for formatting SD cards. Lastly, pseudo file systems are supported by Android as well. These include cgroup, which provides a means to access and define kernel parameters. Rootfs serves as a root point for mount file systems. Procfs reflects the amount of kernel data structures and is mounted on the proc directory. Sysfs is an object-oriented structure reflects the devices known by the kernel and is mounted on the sys directory. Lastly, tmpfs, is mounted on the dev directory and its data is lost when the device is rebooted (Anderson).

Mac OS X supports a vast amount of different file systems including: HFS+, UFS, NFS, ISO 9660, MS-DOS, SMB, AFP, and UDF. HFS+ is the standard file system for Mac OS X and iOS. It gained popularity due to its journaling capabilities, which allows it to record transactions before said transaction is performed. This allows data to be replayed after a reboot, which makes the file system very reliable. HFS+ can also support very large files (up to 8 EiB) (Halvorsen 35). HFS, an older Mac OS X file system, is still compatible, though it can no longer be booted by OS X. UFS is the BSD standard file format that is used by Unix. NFS is the industry standard for network file system and ISO 9660 is the file system of choice for CD’s and DVD’s. SMB is the file sharing standard for Windows and AFP is the file sharing standard for Mac OS. In addition, OS X Virtual File System (VFS) can be used to extend support to other file systems (“Kernel Programming Guide – File Systems Overview”).

A unique feature in the Unix operating system is its use of what is called an inode. In essence, an inode is a unique identification of a file in Unix. A single inode contains all of the attributes of a given file including its file ownership, access rights, access times, file type, number of links to the file, file size, and a table of contents for the data’s disk addresses. These inodes are accessed by their inode number and file system (Bach 60-64). Also, thanks to the use of inodes, Unix (and Ubuntu) is capable of changing files while they are open (Simon).

In the front-end of operating system, the user interface becomes quite prominent. Simply, the user interface allows the user to enter and receive information to and from the computer system. Some operating systems, like Unix, utilize a basic, keyboard-oriented interface. This interface is purely text-based, and does not have any widgets (Tanenbaum 726). Other operating systems utilize a graphical user interface. A graphical user interface is a more visually appealing interface that utilizes icons and windows to make the interface more user friendly. Most modern operating systems like Windows, Mac OS X, iOS, and Android uses a graphical user interface, largely to create a commercially successful product. With Windows, the graphical user interface is actually a part of the kernel (Tanenbaum 719).

OS 360 actually has two model-independent interfaces created for its usage, the first simply being the instruction set. The second, however, being the Channel-to-Control Unit Electrical Specification and Signing Protocols (not the catchiest of names…). Simply, this interface monitors I/O devices, as addressed earlier in the paper (Gifford 294).

Evidently, the realm of operating systems is quite vast. There are so many different operating systems that have been created. Some operating system attributes tend to be quite similar, like how many operating systems utilize Round Robin with Multilevel Feedback or Completely Fair Scheduling, or how most operating systems use a hierarchal file system. However, there are always exceptions to the status quo, either because the operating system in question is quite old and was unable to follow the accepted trends, or for other reasons. Multics and OS 360 were very unique due to being two of the first major operating systems ever made. Many operating systems spawn from other operating systems, like Ubuntu and Android from Linux, and Linux from Unix. It’s possible that an operating system will arrive that will change the status quo again, leaving the Windows 10, Mac OS X, and Android operating systems appearing as ancient as Multics and OS 360. Only time will tell.

References

Bach, M. J. (1986). *The Design of the Unix Operating System*. Englewood Cliffs, NJ: Prentice-Hall of India.

Kayande, D., & Shrawankar, U. (2011, October). Priority Based Pre-emptive Task Scheduling for Android Operating System. *International Journal of Computer Science and Telecommunications*, *2*(7), 17-21.

Yaghmour, K. (2013). *Embedded Android Porting, Extending, and Customizing*(1st ed.). Oreilly & Associates.

Singh, A. (2006). *Mac OS X Internals: A Systems Approach*. Upper Saddle River, NJ: Addison-Wesley.

Scheduling Priorities. (n.d.). Retrieved March 18, 2018, from <https://msdn.microsoft.com/en-us/library/windows/desktop/ms685100(v=vs.85).aspx>

Lunny, E. (2003). *IBM OS/360: An Overview of the First General Purpose Mainframe*.

Halvorsen, O. H., & Clarke, D. (n.d.). *Mac OS X and iOS Internals*. Apress.

Vyssotsky, V., Corbató, F., & Graham, R. (1965). *Structure of the Multics Supervisor*. Fall Joint Computer Conference.

"Ubuntu 8.10." 4. Virtual Memory Management. Accessed March 22, 2018. http://ubuntu-version-8.blogspot.com/2009/05/4-virtual-memory-management.html.

*Unix Security Guidelines*. (2006). MN: Minnesota State Colleges & Universities.

Bensoussan, A., Clingen, C., & Daley, R. (1972). *The Multics Virtual Memory: Concepts and Design*(Vol. 15). Association for Computing Machinery.

Tanenbaum, A. S., Hernandez, J., Joly, R., & Dupuy, M. (2008). *Modern Operating System*. Paris: Pearson education France.

"Thread." Android Developers. March 07, 2018. Accessed March 27, 2018. https://developer.android.com/reference/java/lang/Thread.html.

"Kernel Programming Guide." About This Document. August 08, 2013. Accessed March 27, 2018. <https://developer.apple.com/library/content/documentation/Darwin/Conceptual/KernelProgramming/>.

"Ubuntu Documentation." PowerManagement/Overview - Community Help Wiki. Accessed March 28, 2018. <https://help.ubuntu.com/community/PowerManagement/Overview>.

"Power Management." Power Management | Ubuntu Core Documentation. Accessed March 28, 2018. <https://docs.ubuntu.com/core/en/stacks/power/>.

"Ubuntu Documentation." Security. Accessed March 28, 2018. <https://help.ubuntu.com/lts/serverguide/security.html>.

"Ubuntu Manpage: Svipc - System V Interprocess Communication Mechanisms." Ubuntu Manuals. Accessed March 30, 2018. <http://manpages.ubuntu.com/manpages/trusty/man7/svipc.7.html>.

"Multics Data Security." Multics. 1983. Accessed March 31, 2018. <http://multicians.org/multics-data-security.html>.

"IBM Operating System/360 Concepts and Facilities." *IBM Systems Reference Library*. Accessed March 31, 2018. <http://bitsavers.trailing-edge.com/pdf/ibm/360/os/R01-08/C28-6535-0_OS360_Concepts_and_Facilities_1965.pdf>.

"The Multics Scheduler." Multics. August 21, 1995. Accessed March 31, 2018. <http://multicians.org/mult-sched.html>.

Simon, Anthony J. "LinuxFilesystemsExplained - Community Help Wiki." Ubuntu Documentation. August 11, 2015. Accessed April 01, 2018. <https://help.ubuntu.com/community/LinuxFilesystemsExplained>.

Feiertag, R. J., and E. I. Organick. The Multics Input/Output System. 1971. Accessed April 01, 2018. <http://multicians.org/rjf.html>.

"Operating System | Process Management | Deadlock Introduction." GeeksforGeeks. May 28, 2017. Accessed April 02, 2018. <https://www.geeksforgeeks.org/operating-system-process-management-deadlock-introduction/>.

*Unix Security Guidelines*. Saint Paul, MN: Minnesota State Colleges & Universities, 2006.

Daley, R. C., and P. G. Neumann. A General-Purpose File System For Secondary Storage. Accessed April 05, 2018. <http://multicians.org/fjcc4.html>.

Vleck, Tom Van. "Glossary - F." Multics. Accessed April 05, 2018. <http://multicians.org/mgf.html>.

Anderson, Bill. "Understanding the Android File Hierarchy." Android News for Costa Rica. January 13, 2013. Accessed April 5, 2018. <http://www.all-things-android.com/content/understanding-android-file-hierarchy>.

O'Sullivan, Bryan. "The History of Threads." Teideal Glic Deisbhéalach. June 9, 2005. Accessed April 05, 2018. <http://www.serpentine.com/blog/threads-faq/the-history-of-threads/>.

Gifford, David, and Alfred Spector. *Case Study: IBM's System/360-370 Architecture*. Vol. 30.

Spier, Michael J., and Elliot I. Organick. "THE MULTICS INTERPROCESS COMMUNICATION FACILITY \*." 83-91.

"Memory Management In Unix Operating System Computer Science Essay." UKEssays. March 23, 2015. Accessed April 09, 2018. <https://www.ukessays.com/essays/computer-science/memory-management-in-unix-operating-system-computer-science-essay.php>.

"Overview of Android Memory Management." Android Developers. February 21, 2018. Accessed April 09, 2018. <https://developer.android.com/topic/performance/memory-overview.html#SharingRAM>.

"Memory and Virtual Memory." Kernel Programming Guide. August 08, 2013. Accessed April 09, 2018. <https://developer.apple.com/library/content/documentation/Darwin/Conceptual/KernelProgramming/vm/vm.html#//apple_ref/doc/uid/TP30000905-CH210-BEHJDFCA>.

"Security Considerations." Kernel Programming Guide. August 08, 2013. Accessed April 09, 2018. <https://developer.apple.com/library/content/documentation/Darwin/Conceptual/KernelProgramming/security/security.html#//apple_ref/doc/uid/TP30000905-CH206-TPXREF108>.

"The Windows Security Model." Developers Network. Accessed April 09, 2018. <https://msdn.microsoft.com/en-us/library/bb385791.aspx>.