

Design of Filesystems in Modern Operating Systems

How Filesystems Effectively Manage Storage

Brian S. Stephan

Submitted by: Brian S. Stephan
Submitted to: Dr. Chris Taylor
Submitted for: CS-384
Submitted on: February 3, 2003

Table Of Contents

Purpose	3
Introduction to File Systems	3
FAT32/16	5 ^{ooo}
NTFS	7
Linux VFS	10
Extended 2/3	11 ^{oooooooo}
ReiserFS	14
XFS	16
Even More Filesystems	17
Conclusion	18
Bibliography	19

Purpose

The purpose of this report is to enumerate and expand upon a variety of filesystems currently in use in today's modern computers. Its goal is to serve as a introduction to both filesystem concepts present in modern systems, and to provide a solid foundation for understand how the filesystem that may be on your computer operates in every day activities, and how filesystems are effective storage managers. This report is also to give examples of how filesystems can contrast and achieve the same means using different methods, allowing for diversity in specific issues solved best by one filesystem compared to another.

For purposes of scope, the filesystems mentioned here are found in use in Windows and Linux operating systems, both of which primarily use x86 processors (although Linux runs on other architectures). Naturally, other operating systems such as Mac OS exist, but this document makes no mention of these systems and their possible filesystems.

Introduction To Filesystems

One of the major necessities of computing, since the introduction of a computer system that could be loaded with different data and separate processes, has been storage. Early on, stored programs and data were placed on analog tape drives, or stored in static memory chips that could not be altered. By now, however, the common method of storage for everyday computing is the hard disk. Not only can data be read from and written to these nonvolatile disks, allowing for dynamic changes in storage, but disk architecture is such that it can be systematically formatted by an operating system such that optimizations can be reached. This allows for many variations in the common vein of operating systems and their chosen filesystems used for formatting the

disk layout and organizing its structure as seen by the operating system. To bring context to this discussion of various filesystems, a slight description of hard disk architecture is in order.

A hard disk is a collection of multiple circular platters, or discs, stacked on top of one another such that it creates a cylinder. Each disc is physically divided into circular tracks, which are then each divided further by radial lines, producing sectors that are roughly rectangular in shape. These sectors are how the hard drive is physically divided, and a magnetic head that is in contact with each disc is manipulated to search for a specified sector, from which it reads or writes data as instructed. A hard disk, also commonly known as a hard drive, is a physical unit which contains many of these discs.

Furthermore, a hard disk is physically divided into partitions, ranging from only one partition across the entire disk to a multitude of partitions as necessary. There is a limit to the number of physical partitions possible on a hard disk, but the details of such are not necessary to understand filesystems. The file system, managed by the operating system, formats the partition such that the partition can be written to and read from by the operating system. Despite any limitations on partitions on a hard disk, there is no restriction upon the number of or types of different filesystems on a disk. The only limit on available filesystems for a computer is placed by the operating system being used, as ultimately the operating system can only read filesystem formats that it supports.

An important concept in filesystems is that of the cluster, which is a logical unit of file storage on a hard disk, as managed by the operating system and dependent on the filesystem being used. A cluster is not a physical division on the hard disk, instead it is a collection (or cluster, hence the name) of physical sectors that are combined by the filesystem into acting as

one unit. For example, a 4 kilobyte cluster consists of a number of physical sectors, whose combined size is 4 kilobytes. This organization potentially yields internal fragmentation. If a filesystem is using 4 kilobyte (4,096 bytes) clusters, a file that is only 8 bytes long takes up one entire cluster, leaving the other 4,088 bytes on the hard disk unused and inaccessible, because as far as the operating system is concerned, the entire cluster has been allocated.

It should be noted that in addition to hard disks, filesystems are implemented on all forms of storage used in modern computing. For example, floppy disks are formatted with filesystems just as hard disks, and CDRoms use a format called ISO9660, with the Joliet extension for long filenames.

FAT32/16

One filesystem that is commonly used by many operating systems (predominately, Windows operating systems) is the FAT32 and older FAT16 filesystem. FAT, which stands for File Allocation Table, is an older filesystem supported in one form or another by MS-DOS, all Windows versions, OS/2, and Linux. Designed in 1977, the FAT filesystem was designed for Microsoft Stand-Alone Disk Basic and used to read and write floppy disks. The first version of FAT to be designed with hard disks in mind, FAT16, was developed for MS-DOS and was used up until the Windows 95 operating system. The successor to FAT16, aptly-named FAT32, improved on partition size limitations inherent in the system, and was supported in Windows 95 with a patch, and is still the default filesystem for Windows 9x operating systems (Windows 98 and ME). Additionally, the filesystem is supported in the Windows NT, 2000, and XP operating systems and Linux kernels, which makes it a viable filesystem to use when a partition might need

to be read and written by a number of operating systems present on the computer. For example, the primary partition on the latest MSOE laptops is formatted in FAT32 so crucial data can be accessed from both the Windows 2000 and Linux operating systems.

The FAT system works by utilizing a list of linked lists to manage files, which will generally be spread over many clusters. The file allocation table contains the beginning of a chain of allocation units for the file, which it then traverses in order. This behavior can be problematic when the hard disk is suffering from external fragmentation. External fragmentation is when one file or object, for any reason, is split into many pieces and scattered across the memory medium - in this case, the hard disk. In addition to this possible problem in design, the FAT filesystems have critical limitations that hamper performance of the system, mainly on larger drives.

One of the larger problems facing FAT partitions is that FAT16, which is an outdated filesystem, only supports volume sizes of 2 gigabytes, far less than what is commonly available in current systems. The FAT32 system increases this limit to 2 terabytes, which exceeds current hard disk sizes, but it is possible that the limitation will again be an issue of the filesystem is still in use when drives of beyond this size become available.

Another potential issue with FAT32 is that the cluster size is variable and can become very large. The following is a chart displaying possible cluster sizes on a FAT32 disk, and how it relates to disk size:

Disk Size	Cluster Size	Efficiency
<= 260 MB	4 KB	96.6% °4 KB
<= 8 GB	8 KB	92.9 % °8 KB
<= 60 GB	16 KB	85.8% °16 KB
<= 2 TB	32 KB	73.8% °32 K

As the partition increases, two situations are possible for arranging the disk clusters. Smaller clusters obviously lead to less internal fragmentation as more clusters are necessary to

compose a file, which means higher precision, but this benefit comes at the price of there being more clusters for the filesystem to manage, leading to an increased overhead and more seek time as more clusters have to be accessed, possibly not at concurrent places on the physical disk. This scenario is actually restricted from happening on the FAT32 filesystem, and is described below. On the other hand, applying the larger cluster size makes the overhead more bearable, but results in a much greater risk for internal fragmentation. Any system that has a large number of files ranging from 1 to 4 KB in size, for example, would do very poorly on a drive larger than 60 GB in size, as it would result in severe internal fragmentation for any of these smaller files. To represent this dilemma in cluster size on FAT32 partitions, the efficiency entry on the table above shows the overall drop in filesystem effectiveness as the size of the partition increases.

The reason for this increasing cluster size is that the maximum number of clusters possible on a FAT32 filesystem partition is 268,435,445 clusters. Because of this, as the size of the hard disk partition increases, in order for the FAT32 cluster limit to be maintained, the size of each individual cluster must be increased.

In closing the discussion of the FAT filesystem, it is an aged filesystem that has not been able to keep its effectiveness as technology progresses, mainly due to the increase in average hard disk size. The further this progression goes on, the less realistic FAT becomes for being used on hard disks. However, this filesystem performs well on smaller volumes, and still has plenty of application in such scenarios, such as use on floppy disks and other portable storage devices.

NTFS

The other predominant filesystem used by Windows operating systems is the aptly-named NTFS, which stands for (Windows) NT filesystem. Originally designed for the Windows NT operating system, the filesystem is also supported by Windows 2000 and XP. Some of the easily noticeable gains brought by using NTFS is that while it has the same 2 TB limit on volume sizes

just as in FAT32, there is no practical limit to the number of clusters that can be used by the filesystem, which eliminates the massive ballooning of cluster size that is found on large FAT32 partitions. Additionally, NTFS has other inherent features not found in FAT32 such as file compression, object permissions (comparable to those in UNIX/Linux systems), and in the Windows 2000 and XP operating systems, encryption and disk quotas. Again, due to the nature of the filesystem, NTFS performs poorly on small disk volumes but much better on larger volumes than FAT32, which makes it a better choice for the future as security becomes more and more an issue, along with the ever-increasing hard disk size.

The organization of NTFS filesystems relies heavily upon the MFT, or master file table. This table does not need to be in any specific location on the hard disk, therefore its size is not under any constraints (other than the size of the hard disk!). The first 16 entries are reserved for usage by the filesystem. The first record entry in the table describes the MFT, and the second is a link to a MFT mirror file created in case the first is corrupted. Both of these locations are recorded into the boot sector of the hard disk. For even more security and fault-tolerance, a copy of the boot sector is made at the logical center of the disk. The third record in the MFT is the log file, which is used for file recovery in case of faults. The seventeenth and following records begin where the table starts managing files and directories, which are also viewed as files by NTFS, on the volume.

Each MFT entry allocates some of its space for file attributes, the rest belonging to data or the file's index, depending on the size of the file. The attributes stored in the entry are standard information are as follows: the file's timestamp and link count (the number of references to this file), a list of locations storing attribute records that don't fit in the MFT, the filename of this

entry (maximum name is 255 Unicode characters) stored in both long and shorted 8.3 (eight characters for filename and allowance of a three character extension) formats, security descriptors of who owns the file and how it can be accessed, and various other identifiers that may or may not be in use to track index root, allocation, and other such data. The rest of the allocated space in the entry is file data or an index. If the file is small enough to fit entirely inside the entry of the MFT, it is stored there, otherwise the location of the file is stored in the remainder of the entry.

One of the key qualities of the NTFS filesystem is that inherent in the filesystem is transaction logging and recovery behavior, which makes the filesystem recoverable and almost guarantees the volume's consistency. Transactions, or reads and writes to the disk, are logged by the filesystem to a file mentioned above. While generally not important, in the event of a disk failure the NTFS filesystem restores consistency on the disk by executing recovery procedures, accessing information stored in the log file. A recover might be necessary in a situation where only half of the file was written before disk failure, in which case the recovery system would "roll back" to the previous state of the file. Additionally, this logging requires very little overhead to implement. This recovery procedure is done the first time a program attempts to access a NTFS volume after a system reboot following a failure. Note that this logging system does not ensure the loss of user data during a system failure, it just guarantees that the volume structure is not corrupted by mismatching entries in the MFT.

Another recovery technique implemented in NTFS is cluster remapping, which occurs when a bad-sector error results from attempting to access a physical sector on the hard disk. When this type of error is detected by the OS, NTFS dynamically remaps the cluster containing the bad sector to a new location on the disk, allocating a new cluster for the data at the new

location. On a read, however, there is no possible recovery (for NTFS or any other filesystem, as the physical sector is damaged), and the data is lost. In both circumstances, NTFS places the address of the cluster containing a bad sector in a file containing bad clusters, so that the bad sector is not reused. It should be noted that this behavior does not at all serve as a backup or repair. Any attempt to minimize negative effects in the filesystem caused by a damage in the physical hard disk, such as the methods taken by NTFS, cannot reduce the possibility of further physical damage.

Since NTFS has no realistic limit on the number of clusters it can manage, it does not suffer from the same constraints on cluster size as FAT32. Cluster size is variable, starting from 512 bytes and doubling up to a maximum cluster size of 64 KB. This allows for the user setting up his or her NTFS drive to determine what cluster size is best for the system: small clusters for smaller files, and vice versa for larger multimedia files. This user-determined cluster size, when used properly, results in much less internal fragmentation as the cluster size is better suited for the files on the volume.

Linux VFS

Before continuing to filesystems present mainly in the Linux operating system, clarification should be made concerning the Linux Virtual File System. The Linux VFS acts as a interface layer, implementing generic filesystem actions. This is built into the kernel, and then other filesystems attach itself to the VFS. Working as a mediator between the filesystems existing on the hard disks and the Linux kernel, the VFS keeps track of available filesystem types and

which devices are associated with a certain filesystem and perform basic, generic operations involving file input/output.

Extended 2/3 (Ext2/3fs)

While the filesystems detailed thus far have been Windows-predominant filesystems, there are of course a large number of filesystems unique to the UNIX/Linux operating system. One such filesystem, and the most popular and supported for Linux systems, is known as Extended 2. Additionally, Extended 3 consists of further modifications to the filesystem that expand functionality and error tolerance.

Just as in Windows filesystems, Ext2fs makes logical cluster divisions for the hard disk, with the only difference being that these are generally called blocks for UNIX/Linux systems. Otherwise, they are the same concept as clusters. The size of these blocks can be 1, 2, or 4 KB, chosen when creating the filesystem on the volume. Furthermore, Ext2fs groups together a fixed number of these blocks, in sequential order into a group block. Because of this, the filesystem is actually managed as a series of group blocks, used to keep related information physically close on the disk, reducing the management necessary at one time by working in a more organized context. The maximum volume size supported by Ext2fs is 2 TB, making it very similar to Windows filesystems from a block and maximum size perspective.

UNIX/Linux filesystems use a method called indexed allocation, where a collection of pointers is managed by the operating system, through use of the VFS and its interaction with various filesystems. Access is done by looking up a block in the array, which returns the location of a physical location on the hard disk. While this may seem like a limiting structure, similar to

that employed in other filesystems, the array is expanded by using indirection. Indirection works by reserving some of the entries in the array to further arrays. While the top entries, for example, lead directly to locations on the hard disk, latter entries will lead to an entirely new array, which has entries that in turn can lead to either physical locations on the disk, or more arrays, resulting in directions up to triple indirection.

This crucial aspect of the Ext2fs system is implemented through the use of inodes. Inodes resources collected by the filesystem required for management and kept apart from their corresponding data. Each file and directory (which is treated as a file in the Linux operating system anyway) has a separate inode structure, and is described by one and only one inode. The inode limit for the volume is determined while creating the filesystem, and inodes are stored in the superblock, the most system-critical of blocks created by the filesystem.

Inodes, grouping together series of blocks, provide storage of up to 12 block numbers by design, which are direct blocks. These are the links directly to the physical portions of the drive, resulting in no further disk access to reach the intended data. The thirteenth entry in the inode is the indirect block, as in the indirection mentioned above. This block points to a list of direct blocks. As an example, assuming the size of a block is 1 KB, and each block number is 4 bytes long, there is space for 256 indirect blocks, meaning that blocks 13 through 286 in the file for this inode are accessed by an indirect block method. The only penalty from this method of the filesystem's reference to physical data is the necessity of an additional access to reach the desired block. And as eluded to above, blocks fourteen and fifteen of the inode are the double and triple indirection blocks, respectively, which add appropriate levels of block layers.

Ext2fs's superblock, mentioned while discussing inodes, is a block which contains

information on the state of the filesystem, and is located 1024 bytes into the volume the filesystem is on. The superblock is 1 KB in size, and contains three types of information: filesystem parameters which are determined when the filesystem was formatted and cannot be changed further such as total number of available inodes, blocks, and the operating system making use of the filesystem; filesystem parameters that can be changed, such as blocks reserved for the 'root' user (the administrator on a Linux system); and the filesystem state which contains important information for daily use, like the count of free blocks and inodes.

The next "version" of the Ext2fs is Ext3fs, which does nothing more than add a concept called journaling to the filesystem. The ideology behind this is founded in necessary behavior in the Ext2fs when the system goes down unexpectedly. Upon reboot, Ext2fs filesystems force a filesystem consistency check by running a standard disk utility in Linux called "fsck," which checks and repairs faults on the filesystem. The unfortunate fact of the matter is that some people have developed extremely large Ext2fs volumes, file systems going well over 500 GB in size that would take three to four hours to create, let alone check for errors every time the system unexpectedly crashed. This lead to serious downtime, which could be problematic depending on the purpose of the machine. Because of this, Ext3fs added journaling, which for all conceptual purposes acts just as the transaction logging in NTFS. With journaling, Ext3fs can roll back to a coherent version of the filesystem, right before the disk writes that occurred during the crash. This journaling system makes running fsck at system boot unnecessary in all but the most catastrophic of hard disk failures. In order to keep compatibility with Ext2fs, this is the only feature added to Ext3fs. Because of this, it is almost no work to convert an existing Ext2fs system

to Ext3fs by running a utility to create a journal on the existing filesystem, and making sure your Linux kernel supports Ext3fs.

ReiserFS

Ext2/3fs is considered to be the "standard" Linux filesystem, used by a large majority of systems, mainly due to its established stability and fault protection. However, along the vein of Linux being an alternative to other popular operating systems, there are other filesystems that while not being as widely used, offer noticeable performance gains over Ext2/3fs. One of these is ReiserFS, another popular Linux filesystem.

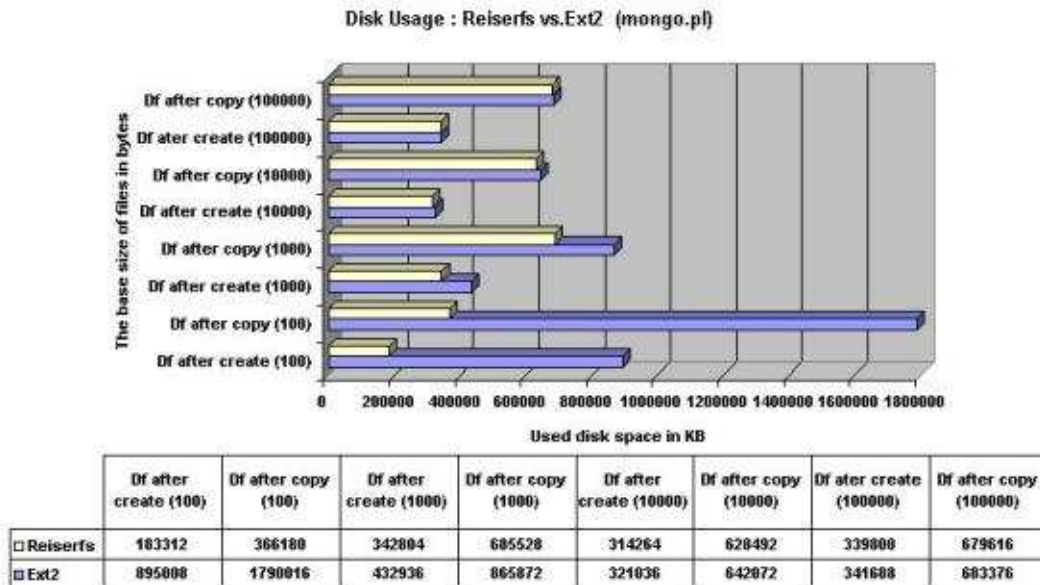
Just as Ext3fs, ReiserFS uses journaling so it isn't necessary to fsck every time there is an unexpected system crash. Additionally, ReiserFS uses balanced trees as the foundation for its filesystem, leading to fewer worst-case scenarios when looking up physical disk locations when the tree is properly managed. On the storage level, ReiserFS allows for a much larger volume size, listing its maximum size as 17.6 terabytes.

The most interesting optimization made to ReiserFS is that it manages to be much more space efficient than other filesystems such as Ext3fs as it attempts to place as many small files into one block as possible, instead of allocating a block for each small file on the system. Additionally, due to the dynamic nature of the balanced tree used in ReiserFS, there is no fixed space left allocated for inodes, which saves disk space for other usage.

ReiserFS, therefore, shows a very significant gain when working with very small files. Comparisons of ReiserFS and Ext2fs were made using filesystems composed of files of various sizes and looking at the total disk usage by using a standard Linux utility called 'df', which

returns the physical disk space used and available (in opposition to a total of file sizes on the system). When using files with a base size of 100,000 bytes, 699 files and 16 directories placed on the filesystem showed ReiserFS to be slightly favorable over Ext2fs, with Ext3fs taking up 341,688 KB and ReiserFS only 339,800 KB. This slight advantage in using ReiserFS is consistent down to a base file size of 1,000 bytes. However, the storage impact is extremely evident when using a smaller file base size. Employing a base size of 100 bytes, the tests showed ReiserFS performing much more favorably than Ext2fs; with 201,885 files and 373 directories, Ext2fs allocated 895,008 KB while ReiserFS used a scant 183,312 KB. These numbers prove without a doubt that for systems employing large amounts of small (less than 1000 byte) files, ReiserFS has an over four-fold storage gain over the commonly used Ext2/3fs.

Table 1: ReiserFS vs. Ext2fs (source: http://www.namesys.com/disk_usage.html)



Because of this impressive performance in storage, ReiserFS's combination of effective file storage along with a journaling system makes ReiserFS a prime choice for Linux filesystems. The only drawback is there is no way to convert an existing Ext2/3fs system to ReiserFS, which

could be problematic for systems that have lots of critical data and little place to temporarily store it.

XFS

While Ext2/3fs and ReiserFS are two filesystems that have been for many years and proven themselves as worthy filesystems, this filesystem, XFS, has only recently been introduced into the Linux kernel (as of 2.4.x) and while still being in active development, already shows impressive performance.

Another journaling filesystem, XFS is able to quickly restart its input/output operations after it is unexpectedly interrupted, just as in Ext2/3fs and ReiserFS. One speciality of XFS is being able to reduce the overhead of journaling systems through algorithms which allow for the data reliability of using a journaling system while keeping the performance drops due to logging the transactions to a minimum. XFS is organized using a table structure, which allows for fast lookups for searching and space allocation.

Certainly the most interesting feature of XFS, however, and the one that makes it an likely candidate for the filesystem of choice as it matures, is the fact that XFS is a full 64-bit filesystem. Because of this, the filesystem is able to handle files as large as a million terabytes, as shown in this example arithmetic by the developers of XFS, Silicon Graphics, Inc.:

$$2^{63} = 9 \times 10^{18} = 9 \text{ exabytes}$$

This is of course a considerably large filesize, when you take into consideration that a full DVD takes up 8 gigabytes, 1/128th of one terabyte. While this certainly isn't an issue in present day application (there isn't a consumer-level hard disk that even nears one single terabyte in storage),

this makes the filesystem a very good selection for future use. Currently, in the Linux 2.4 kernel, this storage is limited by the page size, as there is no necessity for such storage in current Linux kernels. Using a 4 KB page size, the maximum file offset (and therefore, the maximum space for one file) is 16 TB, and 64 TB on a system using 16 KB pages. Under similar conditions, the maximum volume size supported by XFS is 2 TB, but as the Linux kernel begins to move towards a 64-bit layer for block devices (such as hard drives), this limit will increase as well.

Interestingly, despite the larger filesize supported, this support doesn't create any unnecessary overhead. While being faster than both ReiserFS and Ext3fs in accessing and manipulating large files, it would also perform comparably to ReiserFS and commonly bested Ext3fs.

Another interesting aspect to XFS is its concept of delayed allocation, in which it essentially waits until the last moment to write data to disk. For example, when the data to be written to disk is sent to the filesystem, it reserves space for the new data, but makes no decision as where to put it until it knows under what circumstances it is writing to disk. If it is appending data to an existing file, XFS can allocate a single contiguous region on disk to use in writing. This decision might not have been made had XFS decided on a location to write to the moment it was given the data to write. By waiting, XFS makes more intelligent decisions as to where to place relevant data. This example is a perfect example of how filesystems can reduce external fragmentation on a hard disk.

Even More Filesystems

Naturally, this isn't where filesystems end. Many other filesystems exist, either not in

common use, in outdated forms, or currently in development. An example is VFAT, an alteration on the FAT file system available in older Windows operating systems and in Linux, which allows for filenames longer than the 8.3 used in early FAT filesystems. Another filesystem is JFS, another Linux journaled file system designed and used by IBM for their enterprise servers. Devfs is a device filesystem that allows kernel device drivers to register devices by name, and have them appear in /dev in a Linux system as the actual devices.

Conclusion

In modern computers, there are a multitude of filesystems available for different operating systems with different attributes, ranging from those that are well-supported and have wide usage, to newer or even still being developed filesystems that offer increased performance over older filesystems. For those looking for the end-all question as to which filesystem is "the best," there is no universal answer, as filesystems have evolved to the point where they can be finely tuned to handle specific situations better than others, giving computer owners many options when making the selection for their system.

Personally, the author uses NTFS (with 4 KB clusters) for his Windows XP operating system, and Ext3fs for Linux. However, as a direct result from research done for this paper, he is strongly considering switching to ReiserFS when possible.

Bibliography

- Aivazian, Tigran. *Linux Kernel 2.4 Internals, 3. Virtual Filesystem (VFS)*. August 7, 2000.
<http://www.moses.uklinux.net/patches/lki-3.html>
- Johnson, Michael K. *A tour of the Linux VFS*. 1996.
<http://digilander.libero.it/blaad/tuts/vfstour.html>
- Microsoft Knowledge Base. *Description of the FAT32 File System*. February 22, 2002.
<http://support.microsoft.com/default.aspx?scid=kb;EN-US;154997>
- NTFS.com. *NTFS File System General Information*. Date Unknown. <http://www.ntfs.com/>
- Oxman, Gadi. *The extended-2 filesystem overview*. August 3, 1995.
<http://buzco.nyct.net/Linux/ext2fs/Ext2fs-overview-0.1.html>
- Pitt, Andrew. *The FAT32 Resource Page*. Date Unknown. <http://www.project9.com/fat32/>
- ReiserFS Development Team. *Hard Disk usage, ReiserFS and Ext2fs*. Date Unknown.
http://www.namesys.com/disk_usage.html
- ReiserFS Development Team. *[ReiserFS] FAQ*. Date Unknown. <http://www.reiserfs.org/faq.html>
- Robbins, Daniel. *Advanced filesystem implementor's guide, part 9*. January 2002. <http://www-106.ibm.com/developerworks/opensource/library/l-fs9.html?dwzone=opensource>
- Silberschatz/Galvin, J. Wiley. *Operating Systems Concepts, 6th edition*. 2003.
- Silicon Graphics, Inc. *XFS for Linux Features*. Date Unknown.
<http://oss.sgi.com/projects/xfsl/features.html>
- Solomon, David A. *Inside Windows NT*, 2nd edition. 1998.
- Tweedie, Dr. Stephen. *EXT3, Journaling Filesystem*. July 20, 2000.
<http://olstrans.sourceforge.net/release/OLS2000-ext3/OLS2000-ext3.html>
- Virtual File System Switch*.
<http://www.csee.umbc.edu/courses/graduate/CMSC691X/summer99/kerch06/vfs.htm>