

File Services

Topics in this Chapter:

- Understanding Windows File Systems
- Understanding Linux File Systems
- Understanding Permissions Management (Access Control)
- Understanding File Backup, Restore, and Replication Options
- AMANDA
- Designing Linux-based File Services
- Migrating File Services to Linux

- ☑ Summary
- ☑ Solutions Fast Track
- ☑ Frequently Asked Questions

Introduction

File services is a term that is used when talking about accessing files on a remote system. Regardless of computer platform, most companies use a networked repository for commonly accessed data, including home directories, department files, and company resource files. Access to networked data must be protected from unauthorized access using access control, must be protected from loss or damage with regular backups, must be served with adequate performance for end user usage requirements, and must be organized in a manageable way that can handle growth. While many types of file services are available in Linux, Samba is the most tangible solution for supporting Windows clients with a Linux server. The challenge for this conversion is to have a smooth migration of the data, shares, and access control lists (ACLs), while maintaining performance and ensuring data security.

Understanding Windows File Systems

File services offer network access to data stored on file systems. Each type of file system has features and limitations that affect the way file services can share the data. The Modern Microsoft Operating System (OS) offers local compatibility with FAT16, FAT32, NTFS4, and NTFS5 file systems, and the ability to share files on these file systems with the network using a protocol called Common Internet File Services (CIFS). This section discusses the details regarding each of these file systems, pointing out the benefits and differences of each, and also covers the history behind the file system evolution Microsoft has followed.

Why do we need file systems? Disk drive surfaces are divided into address locations (sectors). Partitions can be created on the disk by grouping a number of sectors. When a partition is formatted with a file system, the file system attempts to organize how the data will be organized, written, found, accessed, and checked within the sectors of the partition.

Understanding Windows File Allocation Table File Systems

A File Allocation Table (FAT)-formatted partition groups the sectors into clusters and each file occupies one cluster. All FAT file systems have a maximum limit of clusters that can be in a partition. The cluster limit is based on the number of bits used to address sections of the cluster minus a few reserved bits. As hard drives

increase in size there are more sectors on the disk. To create a larger partition (across an entire disk), a cluster must contain more sectors. A lot of space is wasted if a file is less than the size of a cluster (sectors on the disk remain unusable).

Tools and Traps

FAT Bits

- FAT12 uses 12 bits, or 2^{12} bits or about 4,086 clusters.
- FAT16 uses 16 bits, or 2^{16} bits or about 65,526 clusters.
- FAT32 uses 28 bits, or 2^{28} bits or about 268 million clusters.

Additionally, there is a limit on the size of each cluster. In order to use an entire disk on a large hard drive, you must create many partitions and remember which partition contains the stored data.

Tools and Traps

FAT Partitions

- FAT12 can create 16MB partitions.
- FAT16 can create 2GB partitions in DOS 4.0 and up, Windows 9x, and ME, or 4GB partitions with NT.
- FAT32 can create 8 terabytes (TB) partitions, but only a 32GB partition is possible with versions of Windows up to XP.

Tools and Traps

Maximum Files in a Directory

- FAT - 512 files or folders per folder
- FAT32 - 65,534 files or folders per folder
- NT File System (NTFS) - 4,294,967,295 files or folders per folder

By default, FAT stores a backup copy of the FAT table in the partition header; with Fat32, this can be disabled to speed up access. All FAT versions lack the concept of user ownership and user level access control, file locking, and redundancy and can fragment easily. FAT stores some basic file attributes such as *read only*, *hidden*, *system file*, *volume label*, *subdirectory*, and *archive*. There are three naming rules for files stored on a Fat12 or Fat16 partition.

1. The name is limited to an 8-character prefix, followed by a period and a 3-character suffix (i.e., *my_files.txt*).
2. The name must begin with a letter or number.
3. The case will not be preserved.

VFAT/FAT32 has the same 2.0GB maximum file system size limit that FAT16 has and adds three improvements:

1. Long file name support (255 characters long including spaces and multiple periods; preserves the case but is not case-sensitive).
2. Improved performance using “32-bit protected mode access.”
3. Better management capabilities; disk locking for “exclusive mode” disk access by a program to a file.

Virtual File Allocation Table (VFAT)/FAT32 offers the following advances:

- Uses 32 bits per FAT entry and a smaller cluster size to increase the maximum file system size to 32GB.
- Each individual file occupies a single cluster; a small file with a smaller cluster size wastes less space.

- An extension to the VFAT that shares the VFAT file name limitation.
- Utilities were made available to convert VFAT to FAT32 in a one-way operation.
- FAT32 partitions are larger than 32GB, are prepared by other OS', and are usable by windows.

Windows NT can use a 64 Kbits cluster and a 64 Kbits cluster, extending the maximum file system size to 4GB for a VFAT partition.

With an FAT file system, network file-sharing ACLs are limited to *full control*, *change*, and *read*.

In 1988, FAT16 became available, and is used with DOS 4.0 and up, Windows 3.x, Windows 95 OEM SR1, Windows ME, and Windows NT. By 1995, VFAT became available, and can be used with Windows 3.x, Windows 95 OEM SR1, Windows ME, and Windows NT. Around 1998, FAT32 became available, and can be used with Windows 95 OEM SR2, Windows 98, Windows ME, Windows 2000, and Windows XP.

Understanding Windows NTFS File Systems

With Windows NT4, Microsoft included a more complex file system called NTFS. NTFS uses a Master File Table (MFT) to track each file in the partition, with a security descriptor (security and permissions) for each file. Each security descriptor contains a System Access Control List (SACL) for auditing and a Discretionary Access Control List (DACL) affecting access to the file. Each access control entry is composed of an ID for the user or group and the permission granted to that ID. The ID can be for a user in the local or trusted domain, or a host account. The NTFS permissions directly affect local file system access, and are compared with share permissions in the file sharing ACL for determining effective access to a share. “No Access” permission blocks all access to a file or directory, trumping any other permissions (local or shared). When granting users “Full Control” permissions, make sure they understand how to calculate effective share permissions:

1. Check for a “NO ACCESS” trump.
2. Check the most permissive SHARE ACL (they are a member of all groups).
3. Check the most permissive FILE ACL (they are a member of all groups)

Effective permission over a network is the most restrictive of the FILE and SHARE permissions. NTFS Version 1.1 (or NTFS Version 4.0) can be used by NT4, Windows 200x, and XP. NTFS 4.0 offers the following advantages over FAT:

- A more efficient allocation of space
- An ACL (multiple users and groups and multiple levels of access)
- Six permission groups - *no access*, *list*, *read*, *add*, *add and read*, *change*, and *full control*
- A static ACL inheritance model in which new files inherit from the folder they are in
- Support for very large files (a single file can use the entire partition)
- A journal of changes to the file system for auditing.
- Support for Redundant Array of Inexpensive Disks (RAID).
- Reduced fragmentation (compared to FAT), but it still occurs.
- Supports file names up to 255 characters and is case-sensitive.

NTFS 1.2 (or NTFS 5.0) became available with Windows 2000, and can be used with NT 4.0 SP4b, Windows 2000, and XP. The Security Configuration Manager (SCM) program must be installed in order for NT4 SP4b to use the new security features of NTFS 5.0. If a NTFS 4.0 partition is attached to a Windows 2000 system, it is converted to a NTFS 5.0 partition with the following enhancements:

- **Reparse Points** An object composed of metadata and an application filter (a program) stored with an associated file or directory on the file system. When the file is accessed, the metadata is passed through the filter, modifying how the file is accessed. A single file or directory can have multiple reparse points. There are some internal reparse points, and any NTFS 5.0-aware application can create them:
 - **Symbolic Links** Points to the real path of a file
 - **Junction Points** Symbolic links for the directory
 - **Volume Mount Points** Symbolic link for mount points
 - **Remote Storage Server** Rules to move files to offline or near-line storage; leaves a reparse point to get the file back.

- **New Security and Permissions Methodology (Dynamic Inheritance)** As the ACL of parent directories change, dynamic permission inheritance allows subdirectories to inherit the changes. Inherited permissions and manually set permissions are maintained separately. One drawback is that dynamic inheritance is more processor/memory intensive than static inheritance. “No Access” trump control has been removed since “allow” and “deny” controls were added. There are thirteen attributes to assigning permissions: *Traverse Folder/Execute File, List Folder/Read Data, Read Attributes, Read Extended Attributes, Create Files/Write Data, Create Folders/Append Data, Write Attributes, Write Extended Attributes, “Delete Subfolders and Files, Delete, Read Permissions, Change Permissions, and Take Ownership*. When determining effective permissions, integrate the following new rules:
 - “Manually set” wins over “inherited”
 - “Deny” wins over “allow”
 - “Parent inherited” wins over “grandparent inherited”
- **Change Journals** Logs all file system operations in a 64-bit update sequence number. The actual data changed is not stored.
- **Encryption** This option, called the Encrypting File System (EFS), stores data in an encrypted format. The OS creates 128-bit (or 40-bit) public/private keys; encryption is done with the public key and decryption is done with the private key. The encryption operations are part of the OS, not the file system. The EFS methods are part of the NTFS 5.0 hard and soft disk quotas based on users and groups and globally.
- **Sparse File Support** For data with long sequences of zeros, only non-zero data is written to disk, along with a record of where the zeros should be inserted. Once stored in this special way, the file is permanently changed and cannot be converted back to its original form.

Linux NTFS compatibility is limited to a *read-only* kernel driver or a *read-write* wine hack. The Linux NTFS kernel module has been reverse-engineered to allow a Linux system to mount and read an NTFS file system. At the time of this writing, the driver allows for writing the same length files as those already present on the disk, but a check disk (CHKDSK) is usually required by Windows NT, Windows 200, and Windows XP upon reboot to clean up damage caused by the driver. Alternatively, the Captive project offers modules for *read* and *write*

access to NTFS partitions using wine's Microsoft Windows NT kernel emulation, by reusing one of the original *ntoskrnl.exe* ReactOS parts and the Microsoft Windows *ntfs.sys* driver. This is a reliable method for accessing an NTFS file system that requires NTFSPROGS v1.8.0 (a group of NTFS utilities based around a shared library), which requires GLIBC v2.3 (a GNU/Hurd and GNU/Linux library). The project home page can be found at: www.jankra-tochvil.net/project/captive/CVS.html.pl.

Since most network file services migrations occur by copying files across the network to the new Linux server, there is little need for an NTFS driver for Linux. However, in cases where there is a large amount (terabytes) of disk data, the network-based copying of files may take more than a day to complete, even over a fast network. In these cases it is often much faster to temporarily install the NTFS-formatted disks into the Linux machine and perform the file copy locally.

Understanding Linux File Systems

In addition to support for FAT and NTFS (mostly *read-only*), Linux also supports a multitude of other file systems. The 2.6 kernel natively supports second and third extended file system (*ext2/3*) and ReiserFS among others. Since *ext2/3* and ReiserFS are the most commonly used file systems with Linux, they are the focus of this section.

Ext2/3 and ReiserFS have many common attributes. Every file and directory has an address on the disk that is referred to as an *inode*. They both support a concept of hard and soft links, where you can create a file name which points to another file name (similar in function to a Windows link) by using the inode value. A *soft link* is a file that contains the inode address of a file that stores the inode to the data. A *hard link* is a second file that contains the inode of the target data. *Ext2/3* and ReiserFS are hierarchical, where every file or directory is stored in a directory with “/” or “root” at the top of the tree.

Linux file */etc/fstab* (file system table) lists file systems, mount points, and mount options, and is parsed when the system boots. When the file system is mounted, a line is added to file */etc/mtab* (mount table) and a “clean” bit in the partition header is read and then set to “0.” When unmounted (as it is during shutdown), the clean bit is set to “1.” During the boot process, if the “clean” bit is not set to “1,” the partition is referred to as “dirty” and special programs are run against the file system to check for errors. These programs (*fsck*, *fsck.ext2*, and *fsck.reiserfs*) perform “consistency” checks of data and metadata on the disk. The file system is “consistent” if each data block is either allocated to a single inode