NTFS Basics

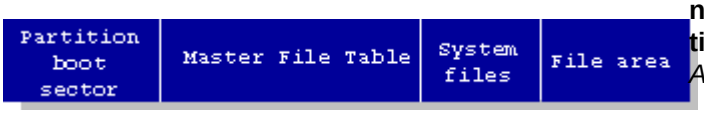
The Windows NT file system (NTFS) provides a combination of performance, reliability, and compatibility not found in the FAT file system. It is designed to quickly perform standard file operations such as read, write, and search - and even advanced operations such as file-system recovery - on very large hard disks.

Formatting a volume with the NTFS file system results in the creation of several system (*metadata*) files such as *$MFT - Master File Table, $Bitmap*, $LogFile and others, which contains information about all the files and folders on the NTFS volume.

The first information on an NTFS volume is the Partition Boot Sector (*$Boot* metadata file), which starts at sector 0 and can be up to 16 sectors long. This file describes the basic NTFS volume information and a location of the main metadata file - $MFT.

The following figure illustrates the layout of an NTFS volume when formatting has finished.

Figure 5-1 Formatted NTFS Volume



What's New in NTFS ver. 3.0

Encryption The Encrypting File System (EFS) provides the core file encryption technology used to store encrypted files on NTFS volumes. EFS keeps files safe from intruders who might gain unauthorized physical access to sensitive, stored data.

Disk Quotas Windows 2000 supports disk quotas for NTFS volumes. You can use disk quotas to monitor and limit disk-space use.

Reparse Points Reparse points are new file system objects in NTFS that can be applied to NTFS files or folders. A file or folder that contains a reparse point acquires additional behavior not present in the underlying file system. Reparse points are used by many of the new storage features in Windows 2000, including volume mount points.

Volume Mount Points Volume mount points are new to NTFS. Based on reparse points, volume mount points allow administrators to graft access to the root of one local volume onto the folder structure of another local volume.

Sparse Files Sparse files allow programs to create very large files but consume disk space only as needed.

Distributed Link Tracking NTFS provides a link-tracking service that maintains the integrity of shortcuts to files as well as OLE links within compound documents.

so the bad sector is not reused.

The Ext4 Linux file system

by Dr. Oliver Diedrich

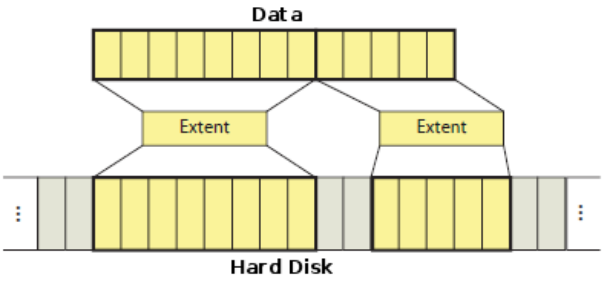
Ext3, the default Linux file system for many years, is definitely starting to show its age. Modern mass storage devices are approaching its limits and block-based data management is no longer adequate for modern file sizes. High time for an update!

Appeared in c't 10/09, p. 180

In the face of rapidly rising data volumes, it is increasingly clear that Ext3, the current default Linux file system, is reaching its limits. A maximum file system, and thus volume size, of 16 TB can already be a tight squeeze for large RAID arrays; Ext3's 32-bit block numbers and 4 KB data blocks mean, however, that there's no way around this limit. A major refurbishment is therefore due.

Development of Ext4 started in 2006 with two changes to the Ext3 file system: block number size was increased to 48 bits and indirect block addressing – in which the data blocks making up a file are stored in a long list made up of individual block numbers – was replaced by extents, consisting of ranges of data blocks. Because this involved changing the structure of the data stored on the disk, the programmers decided that rather than introducing these patches into Ext3, it was

time to create a new version of the file system – Ext4 – based on the Ext3 code.



Extents map parts of a file to areas on the hard drive

The result of three years of Ext4 development has been significant advances from Ext3 which increase the volume limit to 1024 PB. This should be sufficient for many years to come. Extents, long implemented in other file systems such as XFS, should improve the efficiency of managing large files. There are also a whole range of under-the-bonnet changes intended to improve Ext4 performance compared to Ext3.

The kernel development team adopted the Ext4 code in version 2.6.19 to give it the opportunity to come to maturity in the kernel. Ext4 was marked as experimental in versions up to and including 2.6.27, but since Linux 2.6.28 the new file system is now considered stable. Not that this rules out the odd bug or other unpleasant surprise. The latest **Ubuntu 9.04[1]** can already be installed on Ext4 and the forthcoming Fedora 11 release will use Ext4 as its default file system.

Large volumes

Ext4 works with 48-bit block numbers, whilst the default block size remains 4 KB. This allows file system sizes composed of up to 2^{48} 4 KB blocks – equivalent to an exabyte (1024 PB) – compared to the 16 TB maximum in Ext3. Why not go straight to 64-bit block numbers? An **article[2]** by the development team offers a very pragmatic reason: 1 EB is going to be more than enough storage for a very long time – indeed a complete e2fsck run on a file system of this size would (on current hardware) take more than 100 years. Before we even begin to approach this limit, a whole other set of problems, which will necessitate much more substantial file system changes than 64-bit block numbers, will need to be addressed. Plus there's also the fact that 48-bit block numbers fit better into the old Ext3 data structure.

According to Ext4 head developer Ted T'so, extending block numbers to 64 bits shouldn't be too big a deal, and may even be tackled during ongoing Ext4 development. Indeed some structures, such as super-blocks, block group descriptors and the new JBD2 journaling layer – developed in tandem with Ext4 – are already set up for 64-bit block numbers.

The i_blocks value in the inode, which records the number of blocks occupied by a file and in Ext3 is 32 bits long, has been adapted to the larger block numbers in two ways. Firstly, it no longer counts in terms of 512 byte hard drive sectors (as was the case in Ext3), but instead counts in terms of the file system's block size – generally 4 KB. A flag in the inode indicates how this value should be interpreted, something which is very important when upgrading from Ext3 to Ext4, where old Ext3 inodes which count by sector may still be present.

Secondly, two previously unused bytes in the inode are now used to store the high 16 bits of the 48-bit block number. The file system feature `huge_file` indicates that the file system is working with 48-bit block numbers and that inodes can count in file system blocks. Despite 48-bit block numbers, individual files cannot at present be larger than 16 TB, as the current extents structure does not allow management of larger files (on which more below).

Of the other file systems in the kernel, the main competitor to Ext4 is XFS – IBM's JFS has to date failed to find many fans within the Linux community and Reiser4 has still not been integrated into the kernel. In expounding the advantages over XFS, Ext4 developers cite the leaner code base (around 30,000 lines totalling 900 KB, compared to 100,000 lines totalling 3.2 MB for XFS), the ability to convert an Ext3 file system to Ext4 and the large proportion of code imported from the mature and extremely well-tested Ext3.

Linux file systems		
File system	Maximum file size	Maximum file system size
Ext4	16 Tbyte	1024 Pbyte
Ext3	2 Tbyte	16 Tbyte

Reference:

http://www.h-online.com/open/features/The-Ext4-Linux-file-system-746579.html

THE FAT FILE SYSTEMS. FAT32 FAT16 FAT12

The File Allocation Table (FAT) file system is a simple file system originally designed for small disks and simple folder

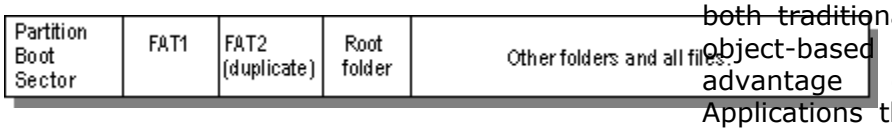
structures. The FAT file system is named for its method of organization, the file allocation table, which resides at the beginning of the volume. To protect the volume, two copies of the table are kept, in case one becomes damaged. In addition, the file allocation tables and the root folder must be stored in a fixed location so that the files needed to start the system can be correctly located.

A volume formatted with the FAT file system is allocated in clusters. The default cluster size is determined by the size of the volume. For the FAT file system, the cluster number must fit in 16 bits and must be a power of two.

Structure of a FAT Volume

The figure below illustrates how the FAT file system organizes a volume.

Figure 4-1



File System Specifications

FAT32 is a derivative of the File Allocation Table (FAT) file system that supports drives with over 2GB of storage. Because FAT32 drives can contain more than 65,526 clusters, smaller clusters are used than on large FAT16 drives. This method results in more efficient space allocation on the FAT32 drive.

The largest possible file for a FAT32 drive is 4GB minus 2 bytes.

The FAT32 file system includes four bytes per cluster within the file allocation table. Note that the high 4 bits of the 32-bit values in the FAT32 file allocation table are reserved and are not part of the cluster number.

Reference:
http://www.ntfs.com/fat-systems.htm

WinFS Overview

WinFS is the code name of a Windows storage subsystem, being developed by Microsoft for use on its future Windows (c) Operating System. WinFS is a relational database located on NTFS and representing itself to the operating system as a file storage subsystem. The codename WinFS stands for Windows Future Storage.

WinFS intends to link the worlds of traditional relational databases, objects, XML, and file systems of unstructured documents with the concept of metadata over files. Instead of representing a file solely by directory path and filename, WinFS represents individual domain objects - e.g. images, e-mails, address book entries, and any kind of regular file - with indexed and searchable context and keyword information.

The underlying system is based on Microsoft SQL Server (c) database engine. WinFS provides access to data through both traditional file-based APIs, and new object-based approaches that take advantage of the new features. Applications that are not written to take advantage of WinFS can access the contents of a WinFS Store through a regular UNC path.

Why WinFS?

A traditional file system, such as FAT or NTFS, has its contents organized in a hierarchal directory structure and is relatively slow in searching the content by particular attributes. If you organize your pictures in folders “by Dates” – there would be no way to access them “by Persons”, “by Events” etc… You are to use third party custom software like Adobe PhotoAlbum (c) to perform this task.

WinFS overcomes the hierarchy and "flattens" the storage of individual files (i.e. there is no "hierarchy" based on directory and file names), and it enables searching for items by their attributes (like date the photograph was taken, who or what is in the picture, what camera was used to take the picture, etc).