CS415 Module 8 Part C - File System Internals and Implementation

Athens State University

The Six Types of Files We Must Support

Ordinary Arbitrary data in zero or more data blocks

Directory List of file names and pointers to associated inodes

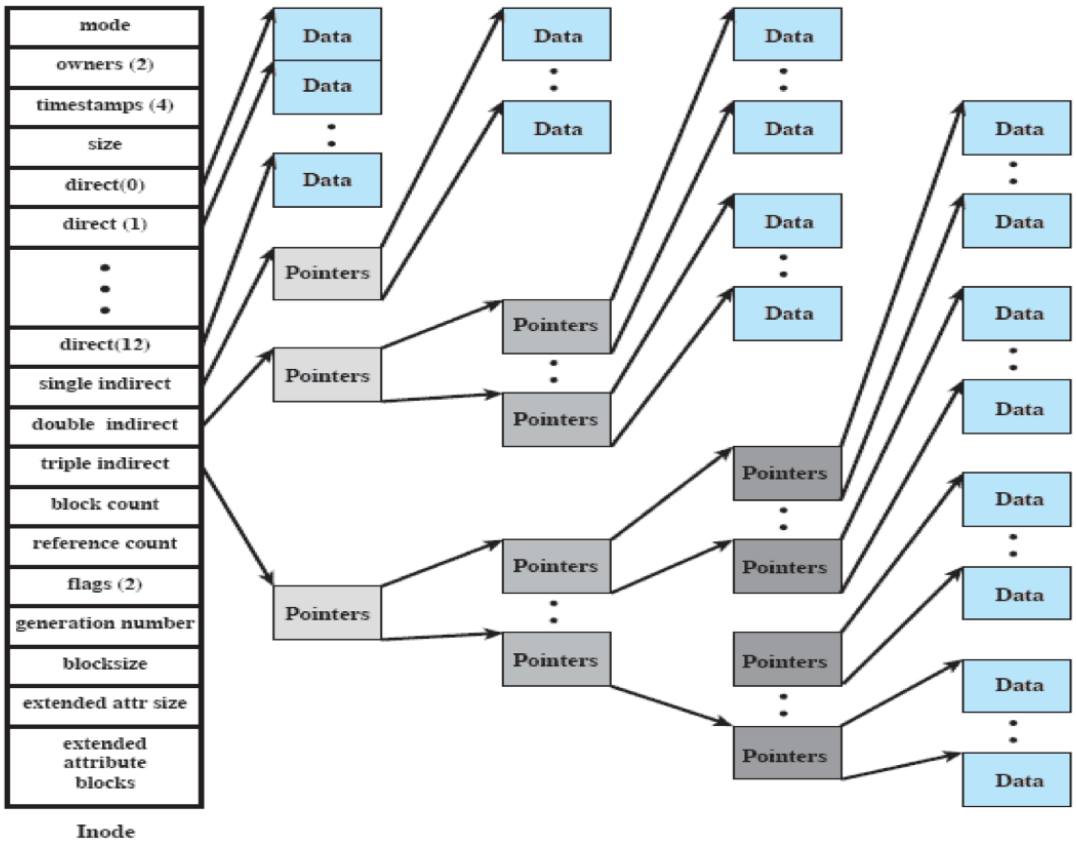
Special Contains no data but provides a means to link physical devices to file names

Named pipes Used for IPC

Links An alias for an existing file

Symbolic link Data file containing the name of the file it is linked to

Case Study: FreeBSD I-node and File Structure



The attributes of the file as well as its permissions and other control information are stored in the inode. The exact inode structure varies from one UNIX implementation to another. The FreeBSD inode structure includes the following data elements:

-
- The type and access mode of the file
- The file's owner and group-access identifiers
- The time that the file was created, when it was most recently read and written, and when its inode was most recently updated by the system
- The size of the file in bytes
- A sequence of block pointers, explained in the next subsection
- The number of physical blocks used by the file, including blocks used to hold indirect pointers and attributes
- The number of directory entries that reference the file
- The kernel and user-settable flags that describe the characteristics of the file
- The generation number of the file (a randomly selected number assigned to the inode each time that the latter is allocated to a new file; the generation number is used to detect references to deleted files)
- The blocksize of the data blocks referenced by the inode (typically the same as, but sometimes larger than, the file system blocksize) The size of the extended attribute information
- Zero or more extended attribute entries

The blocksize value is typically the same as, but sometimes larger than, the file system blocksize. On traditional UNIX systems, a fixed blocksize of 512 bytes was used. FreeBSD has a minimum blocksize of 4,096 bytes (4 Kbytes); the blocksize can be any power of 2 greater than or equal to 4,096. For typical file systems, the blocksize is 8 Kbytes or 16 Kbytes. The default FreeBSD blocksize is 16 Kbytes.

Extended attribute entries are variable-length entries used to store auxiliary data that are separate from the contents of the file. The first two extended attributes defined for FreeBSD deal with security. The first of these support access control lists. The second defined extended attribute supports the use of security labels, which are part of what is known as a mandatory access control scheme.

On the disk, there is an inode table, or inode list, that contains the inodes of all the files in the file system. When a file is opened, its inode is brought into main memory and stored in a memory-resident inode table.

- File allocation is done on a block basis
- Allocation is dynamic, as needed, rather than using preallocation
- An indexed method is used to keep track of each file, with part of the index stored in the inode for the file
- In all UNIX implementations the inode includes a number of direct pointers and three indirect pointers (single, double, triple)

File allocation is done on a block basis. Allocation is dynamic, as needed, rather than using preallocation. Hence, the blocks of a file on disk are not necessarily contiguous. An indexed method is used to keep track of each file, with part of the index stored in the inode for the file. In all UNIX implementations, the inode includes a number of direct pointers and three indirect pointers (single, double, triple).

The FreeBSD inode includes 120 bytes of address information that is organized as fifteen 64-bit addresses, or pointers. The first 12 addresses point to the first 12 data blocks of the file. If the file requires more than 12 data blocks, one or more levels of indirection is used as follows:

- The thirteenth address in the inode points to a block on disk that contains the next portion of the index. This is referred to as the single indirect block. This block contains the pointers to succeeding blocks in the file.
- If the file contains more blocks, the fourteenth address in the inode points to a double indirect block. This block contains a list of addresses of additional single indirect blocks. Each of single indirect blocks, in turn, contains pointers to file blocks.
- If the file contains still more blocks, the fifteenth address in the inode points to a triple indirect block that is a third level of indexing. This block points to additional double indirect blocks.

Total Number of Data Blocks In A File

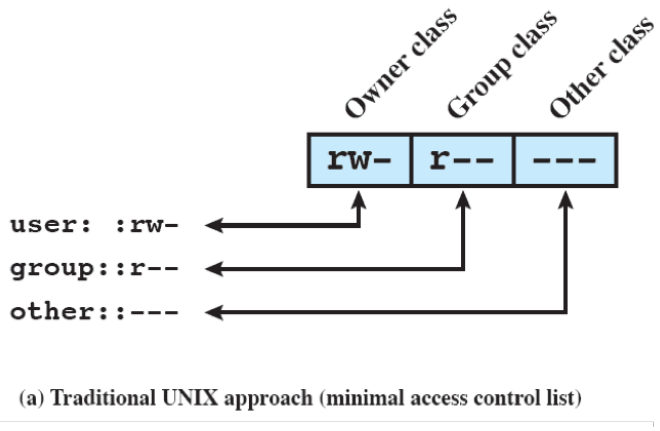
Level	Number of Blocks	Number of Bytes
Direct	12	48K
Single Indirect	512	2M
Double Indirect	512 × 512 = 256K	1G
Triple Indirect	512 × 256K = 128M	512G

The total number of data blocks in a file depends on the capacity of the fixed-size blocks in the system. In FreeBSD, the minimum block size is 4 Kbyte, and each block can hold a total of 512 block addresses. Thus, the maximum size of a file with this block size is over 500 GB.

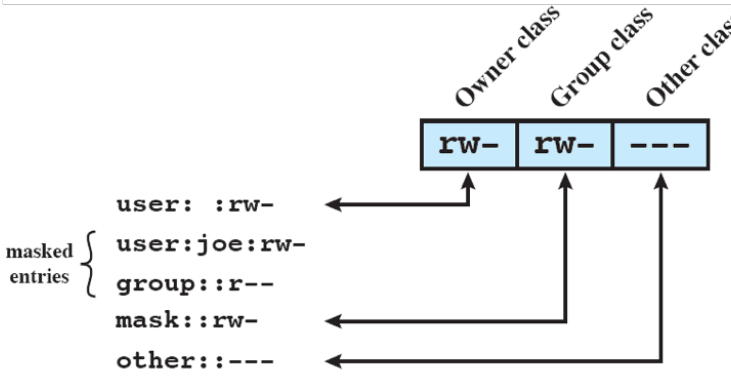
This scheme has several advantages:

1. The inode is of fixed size and relatively small and hence may be kept in main memory for long periods.
2. Smaller files may be accessed with little or no indirection, reducing processing and disk access time.
3. The theoretical maximum size of a file is large enough to satisfy virtually all applications.

File Access Control



(a) Traditional UNIX approach (minimal access control list)



(b) Extended access control list

Most UNIX systems depend on, or at least are based on, the file access control scheme introduced with the early versions of UNIX. Each UNIX user is assigned a unique user identification number (user ID). A user is also a member of a primary group, and possibly a number of other groups, each identified by a group ID. When a file is created, it is designated as owned by a particular user and marked with that user's ID. It also belongs to a specific group, which initially is either its creator's primary group, or the group of its parent directory if that directory has SetGID permission set. Associated with each file is a set of 12 protection bits. The owner ID, group ID, and protection bits are part of the file's inode.

Nine of the protection bits specify read, write, and execute permission for the owner of the file, other members of the group to which this file belongs, and all other users. These form a hierarchy of owner, group, and all others, with the highest relevant set of permissions being used. Figure (a) shows an example in which the file owner has read and write access; all other members of the file's group have read access, and users outside the group have no access rights to the file. When applied to a directory, the read and write bits grant the right to list and to create/rename/delete files in the directory. The execute bit grants the right to search the directory for a component of a filename.

The remaining three bits define special additional behavior for files or directories. Two of these are the "set user ID" (SetUID) and "set group ID" (SetGID) permissions. If these are set on an executable file, the operating system functions as follows. When a user (with execute privileges for this file) executes the file, the system temporarily allocates the rights of the user's ID of the file creator, or the file's group, respectively, to those of the user executing the file. These are known as the "effective user ID" and "effective group ID" and are used in addition to the "real user ID" and "real group ID" of the executing user when making access control decisions for this program. This change is only effective while the program is being executed. This feature enables the creation and use of privileged programs that may use files normally inaccessible to other users. It enables users to access certain files in a controlled fashion. Alternatively, when applied to a directory, the SetGID permission indicates that newly created files will inherit the group of this directory. The SetUID permission is ignored.

The final permission bit is the "Sticky" bit. When set on a file, this originally indicated that the system should retain the file contents in memory following execution. This is no longer used. When applied to a directory, though, it specifies that only the owner of any file in the directory can rename, move, or delete that file. This is useful for managing files in shared temporary directories.

One particular user ID is designated as "superuser." The superuser is exempt from the usual file access control constraints and has systemwide access. Any program that is owned by, and SetUID to, the "superuser" potentially grants unrestricted access to the system to any user executing that program. Hence, great care is needed when writing such programs.

This access scheme is adequate when file access requirements align with users and a modest number of groups of users. For example, suppose a user wants to give read access for file X to users A and B and read access for file Y to users B and C. We would need at least two user groups, and user B would need to belong

to both groups in order to access the two files. However, if there are a large number of different groupings of users requiring a range of access rights to different files, then a very large number of groups may be needed to provide this. This rapidly becomes unwieldy and difficult to manage, even if possible at all. 6 One way to overcome this problem is to use access control lists, which are provided in most modern UNIX systems.

A final point to note is that the traditional UNIX file access control scheme implements a simple protection domain structure. A domain is associated with the user, and switching the domain corresponds to changing the user ID temporarily.

1 File Systems in Windows

The NTFS File System

- Early version of MS-DOS and Windows supported a 16-bit file system format known as the File Allocation Table (FAT)
- This evolved into the FAT-32 32-bit file system used in the Windows-9x operating system
  - We suffer from today as most operating systems support this file system format; thus, most USB flash disks use this file system type
- Eventually, the Windows NT develops designed a new file system, the New Technology File System (NTFS) to address the issues with FAT-32

NTFS Volume and File Structure

Sectors

- Smallest physical storage unit on the disk
- Size in bytes must be a power of 2, usually 512 bytes

Cluster

- One or more contiguous sectors
- The size must also be a power of 2

Volume

- Logical partition on a disk, consisting of one or more clusters used by a file system to allocate space
- Can be all or a portion of a single disks or extend across multiple disks

NTFS Partition and Cluster Sizes

Volume Size	Sectors per Cluster	Cluster Size
≤ 512 Mbyte	1	512 bytes
512 Mbyte - 1 Gbyte	2	1K
1 Gbyte - 2 Gbyte	4	2K
2 Gbyte - 4 Gbyte	8	4K
4 Gbyte - 8 Gbyte	16	8K
8 Gbyte - 16 Gbyte	32	16K
16 Gbyte - 32 Gbyte	64	32K
> 32 Gbyte	128	64K

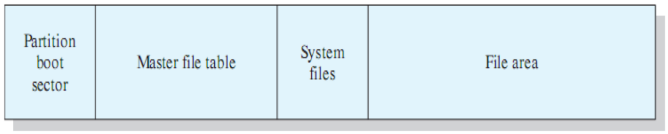


Figure 12.19 NTFS Volume Layout

- Every element on a volume is a file and every file consists of a collection of attributes
- Even the data contents of a file are treated as an attribute

NTFS uses a remarkably simple but powerful approach to organizing information on a disk volume. Every element on a volume is a file, and every file consists of a collection of attributes. Even the data contents of a file is treated as an attribute. With this simple structure, a few general-purpose functions suffice to organize and manage a file system.

The diagram shows the layout of an NTFS volume, which consists of four regions. The first few sectors on any volume are occupied by the partition boot sector (although it is called a sector, it can be up to 16 sectors long), which contains information about the volume layout and the file system structures as well as boot startup information and code. This is followed by the master file table (MFT) , which contains information about all of the files and folders (directories) on this NTFS volume. In essence, the MFT is a list of all files and their attributes on this NTFS volume, organized as a set of rows in a table structure.

Following the MFT is a region containing system files . Among the files in this region are the following:

- MFT2: A mirror of the first few rows of the MFT, used to guarantee access to the volume in the case of a single-sector failure in the sectors storing the MFT.
- Log file: A list of transaction steps used for NTFS recoverability.
- Cluster bit map: A representation of the space on the volume, showing which clusters are in use.
- Attribute definition table: Defines the attribute types supported on this volume and indicates whether they can be indexed and whether they can be recovered during a system recovery operation.

Master File Table

- The heart of the Windows file system is the Master File Table (MFT)
- The MFT is organized as a table of 1,024-byte rows, called records
- Each row describes a file on this volume, including the MFT itself, which is treated as a file
- Each record in the MFT consists of a set of attributes that serve to define the file (or folder) characteristics and the file contents

The heart of the Windows file system is the MFT. The MFT is organized as a table of 1,024-byte rows, called records. Each row describes a file on this volume, including the MFT itself, which is treated as a file. If the contents of a file are small enough, then the entire file is located in a row of the MFT. Otherwise, the row for that file contains partial information and the remainder of the file spills over into other available clusters on the volume, with pointers to those clusters in the MFT row of that file. Each record in the MFT consists of a set of attributes that serve to define the file (or folder) characteristics and the file contents.

MFT Record Attributes

Attribute Type	Description
Standard information	Includes access attributes (read-only, read/write, etc.); time stamps, including when the file was created or last modified; and how many directories point to the file (link count).
Attribute list	A list of attributes that make up the file and the file reference of the MFT file record in which each attribute is located. Used when all attributes do not fit into a single MFT file record.
File name	A file or directory must have one or more names.
Security descriptor	Specifies who owns the file and who can access it.
Data	The contents of the file. A file has one default unnamed data attribute and may have one or more named data attributes.
Index root	Used to implement folders.
Index allocation	Used to implement folders.
Volume information	Includes volume-related information, such as the version and name of the volume.
Bitmap	Provides a map representing records in use on the MFT or folder.