Previous Chapter - Table of Contents - Next Chapter

Chapter 16 Title - Notes

16.3 Learning Objectives:

- Explain the basic filesystem organization.
- Understand the role of the VFS.
- Know what filesystems are available in Linux and which ones can be used on your actual system.
- Grasp why journalling filesystems represent significant advances.
- Discuss the use of special filesystems in Linux.

16.4 Filesystem Basics

Application programs -> read/write files, rather than dealing with physical locations on actual hardware (where files are stored).

Files (and their names) -> abstraction camouflaging physical I/O layer. Directly writing to disk in **raw** fashion (ignoring **filesystem** layer)-> *very dangerous*, only done by low-level operating system software, never by user application.

Local filesystems generally reside within disk partition (which can be physical partition on disk, or logical partition controlled by **Logical Volume Manager** (LVM)). Filesystems can also be part of network nature, true physical embodiment completely hidden to local system across network.

16.5 Inodes

Inode: data structure on disk that describes and stores file attributes, including location. Every file associated with own inode. Information stored in inode:

- Permissions
- User and group ownership
- Size
- Timestamps (nanoseconds)
  - Last access time
  - Last modification time
  - Change time

Note: File names **not** stored in file's inode, but instead stored in **directory** file.

All I/O activity concerning file usually also involves file's inode, as information must be updated.

data_storage_small

Data storage in an inode vs. data storage in a directory file

16.6 Hard and Soft Links

Directory file: particular type of file used to associate file names and inodes. Two ways to associate (or link) file name with inode:

- **Hard** links point to an inode
- **Soft** (or **symbolic**) links point to a file name which has an associated inode

Link: each association of directory file contents and inode. Additional links can be created using **ln**.

Because possible (+ quite common) for two or more directory entries to point to same inode (hard links), a file can be known by multiple names, each of which has its own place in directory structure. However, can only have one inode, no matter which name being used.

When process refers to pathname, kernel searched directories to find corresponding inode number. After name converted to inode number, inode loaded into memory + used by subsequent requests.

hard_and_soft_links

16.7 Filesystem Tree organization

All Linux systems use inverted tree hierarchy branching off the root (`/`) directory. While entire tree may be contained in one local filesystem in one partition, usually there are multiple partitions (or network filesystems) joined together at **mount points**. Can also include removable media, eg. USB drives, optical drives, etc.

In addition, certain **virtual pseudo filesystems** (useful abstractions which exist only in memory) will be mounted within tree. Include `/proc`, `/sys`, `/dev`, maybe `/tmp` and `/run`.

Each element mounted within tree *may* have own filesystem variety. But, to applications and operating system, all appears in one unified tree structure.

16.8 Virtual File System (VFS)

Linux implements Virtual File System (VFS), as do all modern operating systems. When application needs to access file, interacts with VFS abstraction layer, which then translates all I/O system calls (reading, writing, etc.) into specific code relevant to particular filesystem.

Thus, applications do not need to consider neither specific actual filesystem or physical media/hardware on which it resides, and network filesystems (such as NFS) can be handled transparently.

This permits Linux to work with more filesystem varieties than any other operating system. Democratic attribute has been large factor in success.

Most filesystems have full read/write access, while few have only read access, perhaps experimental write access. Some filesystem types, especially non-UNIX based ones, may require more manipulation in order to be represented in VFS.

Variants such as **vfat** do not have distinct read/write/execute permissions for owner/group/world fields. VFS has to make assumption about how to specify distinct permissions for the three types of user, such behavior can be influenced by mounting operations. There are non-kernel filesystem implementations, eg. read/write ntfs_3g, which are reliable but have weaker performance than in-kernel filesystems.

16.9 Available Filesystems

Linux supports many filesystem varieties, most with full read/write access:

- **ext4**: Linux native filesystem (and earlier ext2 and ext3)
- **XFS**: high-performance filesystem originally created by SGI
- **JFS**: high-performance filesystem originally created by IBM
- Windows-nativesL **FAT12**, **FAT16**, **FAT32**, **VFAT**, **NTFS**
- **Pseudo-filesystem**s resident only in memoery, including **proc**, **sysfs**, **devfs**, **debugfs**
- Network filesystems such as **NFS**, **coda**, **afs**
- etc.

Democratic flexibility large factor in success. Most filesystems have full read/write access, while few have read only access.

Commonly used filesystems include **ext4**, **xfs**, **btrfs**, **squashfs**, **nfs**, **vfat**. A list of currently supported filesystems at `/proc/filesystems`.

16.10 Current Filesystem Types

Can see list of filesystem types currently registered + understood by current running Linux kernel by doing:

```
$ cat /proc/filesystems
```

Note: additional filesystem types may have their code loaded only when system tries to access a partition that uses them.

filesystems

16.12 Journaling Filesystems

Number of newer, high performance filesystems include full **journalling** capability.

Journalling filesystems recover from system crashes/ungrateful shutdowns with little or no corruption + very rapidly. Comes at price of having some more operations to do, but additional enhancements can more than offset price.

In journalling filesystem, operations grouped into **transactions**. Transaction must be completed without error, **atomically**. Otherwise, filesystem not changed. Log file maintained of transactions. When error occurs, usually only last transaction needs to be examined.

Following journalling filesystems freely available under Linux:

- **ext3**: extension of earlier non-journalling ext2 filesystem
- **ext4**: vastly enhanced outgrowth of ext3. Features include extents, 48-bit block numbers, and up to 16TB size. Most linux distributions have used ext4 as default filesystem for quite a few years
- **reiserfs**: first journalling implementation used in Linus, but lost its leadership and further development was abandoned
- **JFS**: originally product of IBM, was ported from IBM's AIX operating system
- **XFS**: originally product of SGI, was ported from SGI's IRIX operating systems. RHEL 7 has adopted XFS as default filesystem
  btrfs: newest of journalling filesystems, still under rapid development

16.13 Special Filesystems

Linux widely employs use of **special filesystems** for certain tasks. Particularly useful for accessing various kernel data structures + tuning kernel behavior, or for implementing particular functions. Note: some of these special filesystems have no mount point. This means user applications don't interact with them, but kernel uses them, taking advantage of VFS layers + code.

Special Filesystems

Filesystem	Mount Point	Purpose
rootfs	None	During kernel load, provides an empty root directory
hugetlbfs	Anywhere	Provides an extended page access (2 or 4 MB on **X86**)
bdev	None	Used for block devices
proc	`/proc`	Pseudo filesystem access to many kernel structures and

		subsystems
sockfs	None	Used by **BSD Sockets**
tmpfs	Anywhere	RAM disk with swapping, re-sizing
shm	None	Used by System V IPC Shared Memory
pipefs	None	Used for pipes
binfmt_misc	Anywhere	Used by various executable formats
devpts	`/dev/pts`	Used by **Unix98** pseudo-terminals
usbfs	`/proc/bus/usb`	Used by **USB** sub-system for dynamical devices
sysfs	`/sys` (or elsewhere)	Used as a **device tree**
debugfs	`/sys/kernel/debug` (or elsewhere)	Used for simple debugging file access

##

Back to top

Previous Chapter - Table of Contents - Next Chapter