5.10. Filesystems

5.10.1. What are filesystems?

A *filesystem* is the methods and data structures that an operating system uses to keep track of files on a disk or partition; that is, the way the files are organized on the disk. The word is also used to refer to a partition or disk that is used to store the files or the type of the filesystem. Thus, one might say ``I have two filesystems'' meaning one has two partitions on which one stores files, or that one is using the ``extended filesystem'', meaning the type of the filesystem.

The difference between a disk or partition and the filesystem it contains is important. A few programs (including, reasonably enough, programs that create filesystems) operate directly on the raw sectors of a disk or partition; if there is an existing file system there it will be destroyed or seriously corrupted. Most programs operate on a filesystem, and therefore won't work on a partition that doesn't contain one (or that contains one of the wrong type).

Before a partition or disk can be used as a filesystem, it needs to be initialized, and the bookkeeping data structures need to be written to the disk. This process is called *making a filesystem*.

Most UNIX filesystem types have a similar general structure, although the exact details vary quite a bit. The central concepts are *superblock*, *inode* , *data block*, *directory block* , and *indirection block*. The superblock contains information about the filesystem as a whole, such as its size (the exact information here depends on the filesystem). An inode contains all information about a file, except its name. The name is stored in the directory, together with the number of the inode. A directory entry consists of a filename and the number of the inode which represents the file. The inode contains the numbers of several data blocks, which are used to store the data in the file. There is space only for a few data block numbers in the inode, however, and if more are needed, more space for pointers to the data blocks is allocated dynamically. These dynamically allocated blocks are indirect blocks; the name indicates that in order to find the data block, one has to find its number in the indirect block first.

UNIX filesystems usually allow one to create a *hole* in a file (this is done with the `lseek()` system call; check the manual page), which means that the filesystem just pretends that at a particular place in the file there is just zero bytes, but no actual disk sectors are reserved for that place in the file (this means that the file will use a bit less disk space). This happens especially often for small binaries, Linux shared libraries, some databases, and a few other special cases. (Holes are implemented by storing a special value as the address of the data block in the indirect block or inode. This special address means that no data block is allocated for that part of the file, ergo, there is a hole in the file.)

5.10.2. Filesystems galore

Linux supports several types of filesystems. As of this writing the most important ones are:

minix

The oldest, presumed to be the most reliable, but quite limited in features (some time stamps are missing, at most 30 character filenames) and restricted in capabilities (at most 64 MB per filesystem).

xia

A modified version of the minix filesystem that lifts the limits on the filenames and filesystem sizes, but does not otherwise introduce new features. It is not very popular, but is reported to work very well.

ext3

The ext3 filesystem has all the features of the ext2 filesystem. The difference is, journaling has been added. This improves performance and recovery time in case of a system crash. This has become more popular than ext2.

ext2

The most featureful of the native Linux filesystems. It is designed to be easily upwards compatible, so that new versions of the filesystem code do not require re-making the existing filesystems.

ext

An older version of ext2 that wasn't upwards compatible. It is hardly ever used in new installations any more, and most people have converted to ext2.

reiserfs

A more robust filesystem. Journaling is used which makes data loss less likely. Journaling is a mechanism whereby a record is kept of transaction which are to be performed, or which have been performed. This allows the filesystem to reconstruct itself fairly easily after damage caused by, for example, improper shutdowns.

jfs

JFS is a journaled filesystem designed by IBM to to work in high performance environments>

xfs

XFS was originally designed by Silicon Graphics to work as a 64-bit journaled filesystem. XFS was also designed to maintain high performance with large files and filesystems.

In addition, support for several foreign filesystems exists, to make it easier to exchange files with other operating systems. These foreign filesystems work just like native ones, except that they may be lacking in some usual UNIX features, or have curious limitations, or other oddities.

msdos

Compatibility with MS-DOS (and OS/2 and Windows NT) FAT filesystems.

umsdos

Extends the msdos filesystem driver under Linux to get long filenames, owners, permissions, links, and device files. This allows a normal msdos filesystem to be used as if it were a Linux one, thus removing the need for a separate partition for Linux.

vfat

This is an extension of the FAT filesystem known as FAT32. It supports larger disk sizes than FAT. Most MS Windows disks are vfat.

iso9660

The standard CD-ROM filesystem; the popular Rock Ridge extension to the CD-ROM standard that allows longer file names is supported automatically.

nfs

A networked filesystem that allows sharing a filesystem between many computers to allow easy access to the files from all of them.

smbfs

A networks filesystem which allows sharing of a filesystem with an MS Windows computer. It is compatible with the Windows file sharing protocols.

hpfs

The OS/2 filesystem.

sysv

SystemV/386, Coherent, and Xenix filesystems.

NTFS

The most advanced Microsoft journaled filesystem providing faster file access and stability over previous Microsoft filesystems.

The choice of filesystem to use depends on the situation. If compatibility or other reasons make one of the non-native filesystems necessary, then that one must be used. If one can choose freely, then it is probably wisest to use ext3, since it has all the features of ext2, and is a journaled filesystem. For more information on filesystems, see Section 5.10.6. You can also read the Filesystems HOWTO located at http://www.tldp.org/HOWTO/Filesystems-HOWTO.html

There is also the proc filesystem, usually accessible as the `/proc` directory, which is not really a filesystem at all, even though it looks like one. The proc filesystem makes it easy to access certain kernel data structures, such as the process list (hence the name). It makes these data structures look like a filesystem, and that filesystem can be manipulated with all the usual file tools. For example, to get a listing of all processes one might use the command

```
$ ls -l /proc
total 0
dr-xr-xr-x   4 root     root           0 Jan 31 20:37 1
dr-xr-xr-x   4 liw      users          0 Jan 31 20:37 63
dr-xr-xr-x   4 liw      users          0 Jan 31 20:37 94
dr-xr-xr-x   4 liw      users          0 Jan 31 20:37 95
dr-xr-xr-x   4 root     users          0 Jan 31 20:37 98
dr-xr-xr-x   4 liw      users          0 Jan 31 20:37 99
-r--r--r--   1 root     root           0 Jan 31 20:37 devices
-r--r--r--   1 root     root           0 Jan 31 20:37 dma
-r--r--r--   1 root     root           0 Jan 31 20:37 filesystems
-r--r--r--   1 root     root           0 Jan 31 20:37 interrupts
-r--------   1 root     root     8654848 Jan 31 20:37 kcore
-r--r--r--   1 root     root           0 Jan 31 11:50 kmsg
-r--r--r--   1 root     root           0 Jan 31 20:37 ksyms
-r--r--r--   1 root     root           0 Jan 31 11:51 loadavg
-r--r--r--   1 root     root           0 Jan 31 20:37 meminfo
-r--r--r--   1 root     root           0 Jan 31 20:37 modules
dr-xr-xr-x   2 root     root           0 Jan 31 20:37 net
dr-xr-xr-x   4 root     root           0 Jan 31 20:37 self
```

```
-r--r--r--   1 root     root            0 Jan 31 20:37 stat
-r--r--r--   1 root     root            0 Jan 31 20:37 uptime
-r--r--r--   1 root     root            0 Jan 31 20:37
version
$
```

(There will be a few extra files that don't correspond to processes, though. The above example has been shortened.)

Note that even though it is called a filesystem, no part of the proc filesystem touches any disk. It exists only in the kernel's imagination. Whenever anyone tries to look at any part of the proc filesystem, the kernel makes it look as if the part existed somewhere, even though it doesn't. So, even though there is a multi-megabyte `/proc/kcore` file, it doesn't take any disk space.

5.10.3. Which filesystem should be used?

There is usually little point in using many different filesystems. Currently, ext3 is the most popular filesystem, because it is a journaled filesystem. Currently it is probably the wisest choice. Reiserfs is another popular choice because it to is journaled. Depending on the overhead for bookkeeping structures, speed, (perceived) reliability, compatibility, and various other reasons, it may be advisable to use another file system. This needs to be decided on a case-by-case basis.

A filesystem that uses journaling is also called a journaled filesystem. A journaled filesystem maintains a log, or journal, of what has happened on a filesystem. In the event of a system crash, or if your 2 year old son hits the power button like mine loves to do, a journaled filesystem is designed to use the filesystem's logs to recreate unsaved and lost data. This makes data loss much less likely and will likely become a standard feature in Linux filesystems. However, do not get a false sense of security from this. Like everything else, errors can arise. Always make sure to back up your data in the event of an emergency.

See Section 5.10.6 for more details about the features of the different filesystem types.

5.10.4. Creating a filesystem

Filesystems are created, i.e., initialized, with the **mkfs** command. There is actually a separate program for each filesystem type. **mkfs** is just a front end that runs the appropriate program depending on the desired filesystem type. The type is selected with the `-t fstype` option.

The programs called by **mkfs** have slightly different command line interfaces. The common and most important options are summarized below; see the manual pages for more.

`-t fstype`

Select the type of the filesystem.

`-c`

Search for bad blocks and initialize the bad block list accordingly.

-l filename

Read the initial bad block list from the name file.

There are also many programs written to add specific options when creating a specific filesystem. For example **mkfs.ext3** adds a **-b** option to allow the administrator to specify what block size should be used. Be sure to find out if there is a specific program available for the filesystem type you want to use. For more information on determining what block size to use please see Section 5.10.5.

To create an ext2 filesystem on a floppy, one would give the following commands:

```
$ fdformat -n /dev/fd0H1440
Double-sided, 80 tracks, 18 sec/track. Total capacity
1440 KB.
Formatting ... done
$ badblocks /dev/fd0H1440 1440 $>$
bad-blocks
$ mkfs.ext2 -l bad-blocks
/dev/fd0H1440
mke2fs 0.5a, 5-Apr-94 for EXT2 FS 0.5, 94/03/10
360 inodes, 1440 blocks
72 blocks (5.00%) reserved for the super user
First data block=1
Block size=1024 (log=0)
Fragment size=1024 (log=0)
1 block group
8192 blocks per group, 8192 fragments per group
360 inodes per group

Writing inode tables: done
Writing superblocks and filesystem accounting information:
done
$
```

First, the floppy was formatted (the `-n` option prevents validation, i.e., bad block checking). Then bad blocks were searched with **badblocks**, with the output redirected to a file, `bad-blocks`. Finally, the filesystem was created, with the bad block list initialized by whatever **badblocks** found.

The `-c` option could have been used with **mkfs** instead of **badblocks** and a separate file. The example below does that.

```
$ mkfs.ext2 -c
/dev/fd0H1440
mke2fs 0.5a, 5-Apr-94 for EXT2 FS 0.5, 94/03/10
360 inodes, 1440 blocks
72 blocks (5.00%) reserved for the super user
First data block=1
Block size=1024 (log=0)
Fragment size=1024 (log=0)
1 block group
8192 blocks per group, 8192 fragments per group
360 inodes per group

Checking for bad blocks (read-only test): done
Writing inode tables: done
Writing superblocks and filesystem accounting information:
done
$
```

The `-c` option is more convenient than a separate use of **badblocks**, but **badblocks** is necessary for checking after the filesystem has been created.

The process to prepare filesystems on hard disks or partitions is the same as for floppies, except that the formatting isn't needed.

5.10.5. Filesystem block size

The block size specifies size that the filesystem will use to read and write data. Larger block sizes will help improve disk I/O performance when using large files, such as databases. This happens because the disk can read or write data for a longer period of time before having to search for the next block.

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | **sales@docketalarm.com** | **1-866-77-FASTCASE**

fastcase® Smarter legal research.