Previous Chapter - Table of Contents - Next Chapter

Chapter 20 The Ext2/Ext3/Ext4 Filesystems - Notes

20.2 Introduction

ext family of filesystems native to Linux since its earliest days, have been the most widely used of choices available. Until very recently, **ext4** most frequent default choice of Linux distributions, due to its excellent combination of performance, integrity, stability.

20.3 Learning Objectives:

- Describe the main features of the **ext4** filesystem and how it is laid out on disk.
- Explain the concepts of **block groups**, **superblock**, **data blocks and inodes**.
- Use the **dumpe2fs** and **tune2fs** utlities.
- List the **ext4** filesystem enhancements.

20.4 Ext4 history and Basics

ext2 filesystem: Linux's original native variety, is available on all Linux systems, but rarely used today.

ext3 filesystem: first **journalling** extension of **ext2**. Had same on-disk layout as **ext2** except for presence of journal file.

ext4 filesystem: first appeared in kernel version 2.6.19, experimental designation removed in version 2.6.28. Included significant enhancements, eg. use of **extents** for large files, instead of lists of file blocks.

extent: simply a group of contiguous blocks. Can improve large file performance + reduce fragmentation by using them.

ext4 -> default filesystem on mos enterprise Linux distributions, although RHEL 7 adopted **XFS** as default.

20.5 Ext4 Filesystem Features

Close correspondence of features between VFS and ext2/3/4 filesystems, as each designed with other in mind.

Block size is selected when filesystem built. Can be 512, 1024, 2048, or 4096 bytes. By default, 4096 usually used for hard disk systems, unless partition very small. Unless very many small files, larger block sizes can be more efficient in terms of minimizing hard disk access.

Linux kernel memory management system requires only an integral number of blocks must fit into page of memory. Thus, cannot have 8 KB blocks on x86 platform where pages of memory 4 KB in size.

Number of **inodes** on filesystem can also be tuned, which may save disk space.

Feature called **inode reservation** consists of reserving several inodes when directory is created, expecting them to be used in the future. Improves performance, because, when new files created, directory already has inodes allocated.

If pathname of symbolic link less than 60 characters, so-called *fast symbolic link* created, stored directly in inode, obviating need to read data block.

20.6 Ext4 Filesystem Enhancements

ext4 filesystem:

- Backwards compatible with **ext3** and **ext2**
- Increases maximum filesystem size to 1 EB (from 16 TB), and maximum file to 16 TB (from 2 TB). These limits arise from 48-bit addressing that is used; full 64-bit addressing may be in future, but not really needed at this point
- Increases without limit the number of subdirectories, which was limited to 32K in ext3
- Splits large files into the largest possible extents instead of using indirect block mapping. This can improve large file performance and reduce fragmentation
- Uses **multiblock allocation** which can allocate space all at once, instead of one block at a time. In addition, delayed allocation can also increase performance
- Can **pre-allocate** disk space for a file. Allocated space is usually both guaranteed and contiguous
- Uses **allocate-on-flush**, a performance technique which delays block allocation until data is written to disk
- Uses fast **fsck** which can make the speed of a filesystem check go up by an order of a magnitude or more
- Uses **checksums** for journal which improves reliability. Can also safely avoid a disk I/O wait during journalling, resulting in slight performance boost
- Uses improved timestamps measured in nanoseconds

20.7 Ext4 Layout

All fields written to disk in **little-endian** order, except the journal.

Disk blocks partitioned into **block groups**, each of which contains inode and data blocks stored adjacently, thereby lowering access time.

Layout of standard block group is simple. For block group 0, first 1024 byte unused (to allow for boot sectors, etc.). Superblock will start at first block, except for block group 0. This is followed by group descriptors and number of **GDT** (**G**roup **D**escriptor **T**ables) blocks. These followed by data block bitmap, inode bitmap, inode table, and data blocks.

Data blocks **pre-allocated** to files before actually used. Thus, when file's size increased, adjacent space already reserved and file fragmentation reduced.

Filesystem **superblock** contains bit-fields which are used to ascertain whether or not filesystem requires checking when first mounted during system initialization. Status can be:

- **clean**: filesystem cleanly unmounted previously
- **dirty**: usually means mounted
- **unknown**: not cleanly unmounted, such as when there is a system crash

In latter two cases, **fsck** will be run to check filesystem and fix any problems before it is mounted.

Other fields in superblock contain information about when filesystem was last checked, both in data + number of mounts, and automatic checking triggered when tunable limits exceeded for these quantities.

First block on filesystem: so-called **boot block**, reserved for partition boot sector.

20.8 Superblock Information

Superblock contains global information about filesystem:

- Mount count and maximum mount count (Mount count is incremented every time disk is successfully mounted, and its value is number of times this has been done since last **fsck**.) One can also force filesystem check after 180 days by default, or another time that can be changed by **tune2fs** utility
- Block size (Cannot be greater than page of memory, set by **mkfs**.)
- Blocks per group
- Free block count

- Free inode count
- Operating System ID

As discussed earlier, superblock redundantly stored in several block groups.

20.9 Block Groups

After boot block, there is a series of **block groups**, each of which is same size. Layout of each block group given below:

ext2/3/4 Filesystem Layout

ext2 filesystem

First and second block groups same in every block group, comprise **Superblock** and **Group Descriptors**. Under normal circumstances, only those in first block group used by kernel; duplicate copies only referenced when filesystem being checked. If everything OK, kernel merely copies them over from the first block group. If there is problem with master copies, goes to next and so on until healthy one found and filesystem structure is rebuilt. This redundancy makes it very difficult to thoroughly fry **ext2/3/4** filesystem, as long as filesystem checks run periodically.

In early incarnations of **ext** filesystem family, each block group contained group descriptors for every block group + copy of superblock. As optimization today, not all block groups have copy of superblock + group descriptors. Too see what you have, could examine it as shown below (putting in appropriate device node) to see precise locations. Happens when filesystem created with `sparse-super` option, which is default.

dumpe2fs

Number of block groups constrained by fact that the **block bitmap**, which identifies used and free blocks within group, has to fit in single block. Thus, if block 4096 bytes in size, a block group can contain no more than 32 K blocks, or 128 MB. If we take pargest possible block group size, 10 GB partition would thus have to have at least 80 block groups.

Block allocator tries to keep each file's blocks within same block group to reduce seek times.

20.10 Data Blocks and Inodes

Data block and **inode** bitmaps are blocks whose bits contain 0 for each free blocks or inode, and 1 for each used one. Only one of these bitmaps per block group.

Inode table contains as many consecutive blocks as necessary to cover number of inodes in block group. Each inode requires 128 bytes. Thus, 4 KB block can contain 32 inodes.

Note: there is space reserved for some operating system dependent information; different OSs could possibly mount **ext2/3/4** filesystem, much as Linux can mount many kinds of non-native filesystems.

Note: inode number number itself *not* stored on disk in this structure. Calue can be quickly calculated from block group number and offse within inode table.

ext2 and **ext3** filesystems did not yet incorporate use of extents to organize larger files. Instead, `i_block[]` array of pointers to data blocks, of length `EXT_N_BLOCKS=15` described by inode. The way this is handled is somewhat complex:

- First 12 elements in array simply point to first 12 data blocks in file
- 13th element points to block that represents a second-order array of block numbers; 14th to a third-order array, 15th to fourth-order array

Algorithm makes addressing small files go the fastest, as it should. For instance, with 4 KB block size, 48 KB file can be directly addressed, a 2 MB file requires a second-order process, a 1 GB a third-order, a 4 GB file a fourth-order.

20.11 dumpe2fs

Can use the utility **dumpe2fs** to scan filesystem information (limits, capabilities and flags, other attributes). Eg.:

```
$ sudo dumpe2fs /dev/sda1
```

20.12 tune2fs

tune2fs used to change filesystem parameters.

To change maximum number of mounts between filesystem checks (max-mount-count):

```
$ sudo tune2fs -c 25 /dev/sda1
```

To change time internal between checks (interval-between-checks):

```
$ sudo tune2fs -i 10 /dev/sda1
```

To list contents of superblock, including current values of parameters which can be changed:

```
$ sudo tune2fs -l /dev/sda1
```

##

Back to top

Previous Chapter - Table of Contents - Next Chapter