Edith Cowan University

CSG1102
Operating Systems

Assignment 2

Implementation and operation of the ext3 filesystem

Martin Ponce
Student 10371381

Tutor: Peter Hannay

September 23, 2015

Contents

1 Executive summary **3**

2 Introduction to journaled filesystems **3**

3 Roots in ext2 **4**

4 Ext3 journal mechanics **4**
4.1 Journaling Block Device (JBD) 4
4.2 Transactions 5
4.3 Journal approach 5
4.4 Ext3 journal modes 5
4.4.1 Writeback mode 5
4.4.2 Ordered mode 6
4.4.3 Data journaling mode 6

5 Ext3 operations **6**
5.1 Creating a file in ext3 6
5.2 Updating a file in ext3 6
5.3 Recovery in ext3 6

6 Conclusion **6**

References **7**

1 Executive summary

2 Introduction to journaled filesystems

One of the most important parts of an operating system is the filesystem. It contains and manages critical data on disk drives, such as user configuration, user data and applications and of course, the operating system itself. The filesystem ensures that what is read from storage is identical to what was originally written. In other words, it ensures data integrity. A filesystem achieves data integrity by managing and storing not only the actual data, but also information about the data, or files in storage. It also stores and manages information about the filesystem itself. This information is known as metadata. User's trust the storage of data to the reliability and performance of the filesystem (Best, 2002).

But what happens to data when a system is improperly shutdown during file operations due to a crash or abrupt power loss? When Linux detects an improper shutdown, it signals a non-journaled filesystem (such as ext2) to perform a consistency check with `fsck` before booting into the operating system. This function scans the entire filesystem and attempts to fix any detected consistency issues that can be safely resolved. The `fsck` utility may take a considerable amount of time, depending on size of the filesystem, which can equate to serious downtime for the user (Jones, 2008). According to the developer of the ext3 filesystem, Tweedie (2000), Linux filesystems are growing in size, causing longer `fsck` process times.

Journaled filesystems negate the need for `fsck` (or equivalent in non *nix operating systems) to verify filesystem integrity, therefore greatly reducing downtime and increasing system availability in the event of an improper shutdowns (Best, 2002; Bost, 2012; Jones, 2008; Prabhakaran, Arpaci-Dusseau, & Arpaci-Dusseau, 2005; Tweedie, 1998, 2000). This is achieved by implementing a *log* or *journal* that records changes destined for the filesystem. The changes recorded in the journal are committed to the filesystem periodically, and when an improper shutdown occurs, the journal is referenced as a checkpoint to recover unsaved information to avoid corrupted filesystem metadata (Jones, 2008). Therefore, a `fsck` scan of the entire filesystem is no longer required to determine consistency after an improper shutdown, as the journal is used for reference to determine integrity.

3 Roots in ext2

In addition to creating a journaled filesystem for Linux, Tweedie (2000) describes one of the main goals in the design of the ext3 filesystem is the compatibility with the existing ext2 filesystem. This goal was justified, as much of the Linux user base was using ext2 at the time, and greater compatibility between the two filesystems would ensure seamless transition for users. This goal was successfully achieved, and an ext3 filesystem that is cleanly unmounted can easily be remounted as an ext2 filesystem. Conversely, an ext2 filesystem can easily be upgraded to ext3 *in-place*, while the volume is mounted (Bovet & Cesati, 2006; Tweedie, 2000; **?**).

This backwards and forwards compatibility was achieved by using the same source code base for ext3 as ext2. Therefore, ext2 and ext3 share many common properties. For example, both filesystems use the same on-disk and metadata format. Ext3 also inherits ext2's `fsck` (Robbins, 2001a; Tweedie, 2000). Although the desired effect of ext3 is to avoid the use of `fsck`, there are still times when it may be required, in the event that metadata is corrupted. Availability of `fsck`, with years of development further enhances the reliability of ext3, when compared to other journaled filesystems which do not have a similar capability (Robbins, 2001a).

4 Ext3 journal mechanics

With backwards/fowards compatibility in mind, journaling functionality has been added to ext2 to create the ext3 filesystem by implementing two main concepts, the Journaling Block Device, and transactions.

The ext3 journal itself is stored in an inode using a circular buffer data structure (Jones, 2008; Prabhakaran et al., 2005; Robbins, 2001a; Tweedie, 2000). An inode is an area on the disk where all the information about a file is stored, however, the actual file itself is not stored in an inode. Best (2002, p. 2) uses the analogy of a "bookkeeping file for a file", and asserts that an inode is in fact a file itself. The decision to store the journal in an inode contributes to the goal of backwards/forwards compatibility with ext2, and avoids the use of incompatible extensions to the ext2 metadata (Robbins, 2001a). It is common for the journal to be stored within the filesystem itself, however it is also possible to store the journal on a separate device or partition (Prabhakaran et al., 2005; Tweedie, 2000).

Once the process of journaling is complete, data and metadata are eventually placed into an ext2 structured fixed location on the disk (Prabhakaran et al., 2005).

4.1 Journaling Block Device (JBD)

In order to implement journaling in ext3, an API external to the filesystem was developed, called the *Journaling Block Device* (JBD). Its main purpose is to implement "a journal on any kind of block device" (Robbins, 2001a, p. 8). Tweedie (2000) asserts that the JBD is completely encapsulated from ext3. It does not know anything about how the filesystem works, and conversely, the filesystem does not know anything about journaling. The extent of ext3's knowledge of journaling is through *transactions*.

The JBD API allows the filesystem to communicate the modifications to be performed to the JBD as transactions, which in turn is recorded to the journal by

the JBD. Ext3 also requests permissions from the JBD before modifying certain data on the disk, providing the JBD the opportunity to manage the journal on behalf of the ext3 filesystem driver (Robbins, 2001a).

4.2 Transactions

Transactions are the main concept in journaled filesystems, which "corresponds to a single update of the filesystem" (Tweedie, 1998, p. 4). This concept has been implemented into ext2 to create the ext3 filesystem and provides a format for the filesystem to communicate with the JBD API (Tweedie, 2000).

Similar to database transactions, a journaled filesystem transaction deals with a sequence of changes to the filesystem as a "single, atomic operation" (Best, 2002, p. 4). In other words, the entire sequence of a transaction must be completed, or none at all. As Tweedie (2000, p. 4) explains, "exactly one transaction results from any single filesystem request made by an application, and contains all of the changed metadata resulting from that request".

4.3 Journal approach

Robbins (2001a, p. 3) describes two methods of journaling, *logical journaling* and *physical journaling*. Logical journaling refers to the method where the journal stores "spans of bytes that need to be modified on the host filesystem" and is found to be used in other journaled filesystems such as XFS. In other words, this approach only records individual bytes that require modification, and is efficient at storing many, smaller modifications to a filesystem (Robbins, 2001a).

On the other hand, physical journaling is where entire blocks of modified filesystem are recorded in the journal, and is the journaling approach used by ext3. This means that an unmodified piece of data may also be recorded in the journal, if it resides in a modified block. Although the journal will require more space, it requires less CPU overhead compared to logical journaling, since there is less complexity transferring a literal block from memory to disk (Robbins, 2001a).

4.4 Ext3 journal modes

Ext3 offers three modes of journal operation, which can be selected during the mounting of the filesystem. These modes are *writeback mode*, *ordered mode*, and *data journaling mode* (Jones, 2008; Prabhakaran et al., 2005). Each mode offers varying levels of data consistency guarantees and are interchangeable.

4.4.1 Writeback mode

In writeback mode, only metadata is journaled and data blocks are written directly to disk. While this mode preserves the filesystem structure and guarantees metadata consistency, it provides the "weakest consistency semantics of the three modes" (Prabhakaran et al., 2005, p. 108). In other words, it is still possible for data to be corrupted, because the order between journal and fixed location data writes are not enforced. For example, if a system crash occurs after metadata has been journaled, but before the data block is written, it is likely the data may contain garbage or

previously written data (Jones, 2008; Prabhakaran et al., 2005). However, this mode provides "the best ext3 performance under most conditions" (Robbins, 2001b, p. 2).

4.4.2 Ordered mode

Similar to writeback mode, ordered mode only journals metadata. However, order between journal and fixed location data writes are enforced. This is the default mode, if a user does not select one during the mounting of the ext3 filesystem. Metadata and data block writes are grouped logically as transactions, as mentioned in Section 4.2. When the time comes to *flush* the transaction (write metadata) to disk, data blocks must be written first before metadata is journaled (Robbins, 2001b). This ordering of writes effectively guarantees both metadata and data recovery consistency (Jones, 2008; Prabhakaran et al., 2005).

4.4.3 Data journaling mode

Data journaling mode offers the guarantee of data and metadata consistency, due to the journaling of both metadata and data. However, there are performance trade-offs with this mode since data is being written twice: once to the journal, and again to the fixed ex2 location. Data journaling mode is generally considered the slowest of all ext3 journaling modes, however Robbins (2001b, p. 3) references an experiment which shows it can perform well where "interactive performance IO needs to be maximized".

5 Ext3 operations

5.1 Creating a file in ext3

5.2 Updating a file in ext3

5.3 Recovery in ext3

6 Conclusion

References

Best, S. (2002). Journaling File Systems. *Linux Magazine*(October).

Bost, T. (2012). *Linux for Windows systems administrators: Managing and monitoring the extended file system.* Retrieved 2015-09-14, from http://www.ibm.com/developerworks/linux/library/l-filesystem-management/index.html

Bovet, D. P., & Cesati, M. (2006). *Understanding the Linux Kernel* (3rd ed.). Sebastopol, CA: O'Reilly.

Jones, T. M. (2008). *Anatomy of linux journaling file systems.* Retrieved 2015-09-14, from http://www.ibm.com/developerworks/linux/library/l-journaling-filesystems/index.html

Prabhakaran, V., Arpaci-Dusseau, A. C., & Arpaci-Dusseau, R. H. (2005). Analysis and Evolution of Journaling File Systems. In *Usenix annual technical conference* (pp. 105–120). Retrieved from https://www.usenix.org/legacy/publications/library/proceedings/usenix05/tech/general/full_papers/prabhakaran/prabhakaran_html/main.html

Robbins, D. (2001a). *Common threads: Advanced filesystem implementor's guide, Part 7.* Retrieved 2015-09-14, from http://www.ibm.com/developerworks/library/l-fs7/

Robbins, D. (2001b). *Common threads : Advanced filesystem implementor's guide, Part 8.* Retrieved 2015-09-23, from http://www.ibm.com/developerworks/library/l-fs8/index.html

Tweedie, S. C. (1998). Journaling the Linux ext2fs filesystem. In *Linuxexpo '98* (pp. 1–8). Retrieved from http://www.stanford.edu/class/cs240/readings/ext2-journal-design.pdf

Tweedie, S. C. (2000). *EXT3 , Journaling Filesystem.* Retrieved 2015-09-14, from http://olstrans.sourceforge.net/release/OLS2000-ext3/OLS2000-ext3.html