Austin Group Defect Tracker

Aardvark Mark III

Anonymous | Login | Signup for a new account *2015-03-04 04:25 UTC* Project: All Projects Switch RSS

Main | My View | View Issues | Change Log | Docs Issue # Jump

Viewing Issue Simple Details [Jump to Notes] [Issue History] [Print]

ID	Category	Severity	Type	Date Submitted	Last Update
0000672	[1003.1(2008)/Issue 7] System Interfaces	Editorial	Omission	2013-03-19 15:45	2013-08-01 10:25

Reporter	xroche	**View Status**	public
Assigned To	ajosey		
Priority	normal	**Resolution**	Open
Status	Under Review		
Name	Xavier Roche		
Organization			
User Reference			
Section			
Page Number			
Line Number			
Interp Status	---		
Final Accepted Text			

Summary 0000672: Necessary step(s) to synchronize filename operations on disk

Description

POSIX documents a way of ensuring data is actually sync'ed on permanent storage through fsync(), fdatasync() and aio_fsync().

This way, previously written data, and/or modified meta-data, are guaranteed to be actually protected against a reasonably unexpected situation (system crash, power outage ...)

However, when dealing with file entry handling, such as:
  * file creation (open(O_CREAT))
  * file renaming (rename())
  * symlinking (symlink())
  * hard-linking (link())
  * etc.
there is no documented way to actually give the same guarantee.

Some implementations (such as the Linux glibc) have a somewhat (badly) documented way:
  * open the container directory in read-only (O_RDONLY)
  * apply fsync() or fdatasync() on it

Please refer to the "fsync()'ing a directory file descriptor" thread on the austin-group-l mailing list for insightful comments on this issue.

Several points were discussed, and these (possibly not fully correct) observations were made:
  * directory entries are not attributes of the files they point to, and can not expect to be synchronized [when fsync'ing the file]
  * tracking relationship between directory entries and file descriptors would be cumbersome (a file may be hard-linked in another directory, then have its initial entry being deleted, for example, or renamed to another location)
  * it is not clear whether a directory can be opened at all using open() (readdir() may be the only allowed interface), and what would be the open flags
  * it is not clear what fsync() on a directory file descriptor would do

Desired Action

Clarify that file meta-data have no relationship with directory entry(ies) on the POSIX side.
Clarify how synchronizing a filename operation can be achieved.

Tags No tags attached.

Attached Files

Relationships

Notes

(0001497)
geoffclare
(manager)
2013-03-19 16:49

It is clear that this bug will need an interpretation, but I'm not quite sure how to handle that, given that it raises several issues and on some the standard is clear whereas on one it is not clear. The standard response templates don't seem to cater for this kind of mixed answer.

I believe it is clear from the descriptions of fsync(), open(), dirfd() and various associated definitions in XBD chapter 3 that:

a. Directory entries are data contained in directory files, not attributes of the files they link to, and therefore an fsync() call on a file is not required to have any effect on any directory entries that exist for that file.

b. A file descriptor for a directory can be obtained by using open() with O_RDONLY or O_SEARCH, or by using dirfd() on a directory stream.

c. Since the fsync() description and associated definitions make no mention of specific file types, the requirements apply to all file types, including directories.

The one thing that is not clear is how the definition of Synchronized I/O Data Integrity Completion applies to modifications to directories, since it is worded in terms of "write" operations. I suggest that we reword the first two paragraphs as:

> For read operations, when the operation has been completed or diagnosed if unsuccessful. The operation is complete only when an image of the data has been successfully transferred to the requesting process. If there were any pending write requests or (if the file is a directory) directory modifications affecting the data to be read at the time that the synchronized read operation was requested, these write requests are successfully transferred prior to reading the data.
>
> For write operations and directory modifications, when the operation has been completed or diagnosed if unsuccessful. The operation is complete only when the written data or (if the file is a directory) modified directory entries have been successfully transferred to storage and all file system information required to retrieve them is successfully transferred.

We should also add something to the APPLICATION USAGE section on the fsync() page.

(0001499)
Konrad_Schwarz
(reporter)
2013-03-20 08:03

The paper "Soft Updates: A Technique for Eliminating Most Synchronous Writes in the Fast Filesystem" (http://www.usenix.org/publications/library/proceedings/usenix99/full_papers/mckusick/mckusick.pdf) [^] makes it clear that traditional Unix file systems have no need the interface proposed here, as directory operations are atomic and durable by design.

"Traditionally, filesystem consistency has been maintained across system failures either by using synchronous writes to sequence dependent metadata updates or by using write-ahead logging to atomically group them." (First sentence of the Abstract.)

Before burdening writers of portable applications with the interface proposed by this Defect Report, I think it would be worthwhile to find out which file systems break with tradition in this regard.

As the proposed change breaks existing applications -- durability of file system modifications now requires synchronization of directories in addition to synchronization of file system data -- I think it worthwhile to consider an alternative resolution, namely that directory changes are always atomic & durable ("synchronous") on a POSIX file system, without any action required by the programmer.

Finally, in light of the research that has gone into this topic, if the committee decides to introduce an interface for synchronizing directory modifications, a pathconf() constant for determining whether the interface is actually necessary for a given path should be added.

By the way, I suspect http://pubs.opengroup.org/onlinepubs/9699919799/xrat/V4_port.html [^] misspells _POSIX_SYNC_IO as _POSIX_SYNCHRONIZED_IO.

(0001542)
geoffclare
(manager)
2013-04-19 15:27

In the April 18 teleconference it was agreed that the standard should mandate that directory operations are always synchronized on conforming file systems, and should include warnings about non-conforming configurations. The proposed changes are as follows.

Changes to XBD...

At page 94 line 2581-2588 section 3.376 change:

> For read, when the operation has been completed or diagnosed if unsuccessful. The read is complete only when an image of the data has been successfully transferred to the requesting process. If there were any pending write requests affecting the data to be read at the time that the synchronized read operation was requested, these write requests are successfully transferred prior to reading the data.
>
> For write, when the operation has been completed or diagnosed if unsuccessful. The write is complete only when the data specified in the write request is successfully transferred and all file system information required to retrieve the data is successfully transferred.

to:

> For read operations, when the operation has been completed or diagnosed if unsuccessful. The operation is complete only when an image of the data has been successfully transferred to the requesting process. If there were any pending write requests or (if the file is a directory) directory modifications affecting the data to be read at the time that the synchronized read operation was requested, these requests are successfully transferred prior to reading the data.
>
> For write operations and directory modification operations, when the operation has been completed or diagnosed if unsuccessful. The operation is complete only when the written data or (if the file is a directory) modified directory entries have been successfully transferred to storage and all file system information required to retrieve them is successfully transferred.

At page 107 line 2859 add a new XBD 4.2 section (and renumber the current 4.2 and all later 4.x sections):

> 4.2 Directory Operations
>
> All file system operations that read a directory or that modify the contents of a directory (for example creating, unlinking, or renaming a file) shall be completed as defined for synchronized I/O data integrity completion (see section 3.376).
>
> < small>Note: Although conforming file systems are required to perform all directory modifications as synchronized I/O operations, some file systems may support non-conforming configurations (for example via mount options) where directory modifications are not synchronized. Applications that rely on directory modifications being synchronized should only be used with such file systems in their conforming configuration(s).</small>

Changes to XSH...

At page 574 line 19833 section aio_fsync() change the APPLICATION USAGE section from:

> None.

to:

> Refer to fdatasync() and fsync().

At page 815 line 27215 section fdatasync() append to the first paragraph:

> If the file is a directory, an implicit fdatasync() is already performed on every I/O operation (see XBD 4.2) and consequently if fdatasync() is called explicitly it shall take no action and shall return the value 0.

At page 815 line 27232 section fdatasync() change the APPLICATION USAGE section from:

None.

to:

Although conforming file systems are required to complete all directory modifications as defined for synchronized I/O data integrity completion, some file systems may support non-conforming configurations (for example via mount options) where directory modifications are not synchronized. When the file system is configured in this way, calls to fdatasync() on directories may cause I/O operations to be synchronized, rather than being a no-op.

At page 954 line 31987 section fsync() add a new paragraph to the APPLICATION USAGE section:

Since conforming file systems are required to complete all directory modifications as defined for synchronized I/O data integrity completion (see XBD 4.2), calling fsync() on a directory only synchronizes the file attributes such as timestamps. However, some file systems may support non-conforming configurations (for example via mount options) where modifications to directory contents are not synchronized. When the file system is configured in this way, calls to fsync() on directories may cause directory contents to be synchronized in addition to file attributes.

Changes to XRAT...

At page 3444 line 115531 add a new XRAT A.4.2 section (and renumber the current A.4.2 and all later A.4.x sections):

A.4.2 Directory Operations

Earlier versions of this standard did not make clear that all directory modifications are performed as synchronized I/O operations, although that is the historical behavior and was always intended. Applications have no need to call fdatasync() or fsync() on a directory unless they want to synchronize the file attributes (using fsync()), provided the directory is on a conforming file system. However, since applications may wish to use fdatasync() or fsync() to synchronize directory modifications on non-conforming file systems, implementations are required to support fdatasync() on directories as a no-op on conforming file systems.

(0001545)
eggert (reporter)
2013-04-25 02:22

The proposed changes have caused some consternation on the Glibc developer list (see the thread containing < http://sourceware.org/ml/libc-alpha/2013-04/msg00630.html>), [^] and this suggests that the proposal needs further clarification and/or rewording.

I expect that many modern POSIX implementations fail to conform to the proposed change. Systems with soft updates (as described in the 1999 McKusick & Ganger paper), for example, typically don't guarantee that rename operations are durable: all that's guaranteed is that the file system is consistent after a reboot. Certainly Ext4, XFS, and BTRFS fail to conform to the proposed wording when operating in the default mode, and I expect that many other modern file systems are similar.

If it's really the intent to disallow many (most?) reasonably-high-performance file systems when operating in their default mode, the standard should clearly say so, at least in the rationale somewhere. I'm hoping that this is not the intent, though, as it would create an uncomfortably wide gap between what POSIX requires and what most modern systems actually do.

Instead, I suggest that the POSIX standard and/or rationale be rewritten to more clearly distinguish between atomicity and durability, to say that directory operations must be atomic but not necessarily durable, and to say that fdatasync and fsync can be used to establish the durability of directory operations.

My source for the abovementioned info about Ext4, XFS, and BTRFS is: Ren K, Gibson G. TableFS: enhancing metadata efficiency in the local file system. CMU Parallel Data Laboratory, CMU-PDL-12-110 (September 2012) < http://www.istc-cc.cmu.edu/publications/papers/2012/CMU-PDL-12-110.pdf>. [^]

(0001548)
geoffclare
(manager)
2013-04-25 09:09
edited on: 2013-04-25 15:10

Suggested Interpretation response

The standard does not speak to the issue of filename synchronization, and as such no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

Rationale:

Although applications can call fsync() to synchronize a directory, historically this was not necessary (except to synchronize attributes of the directory such as timestamps) and there are a large number of existing applications that call fsync() or fdatasync() on files they create but not on the directories those files are created in. These applications should not have to change to explicitly synchronize directories.

In answer to the specific points raised in this request:

a. Directory entries are data contained in directory files, not attributes of the files they link to, and therefore an fsync() call on a file is not required to have any effect on any directory entries that exist for that file.

b. A file descriptor for a directory can be obtained by using open() with O_RDONLY or O_SEARCH, or by using dirfd() on a directory stream.

c. Since the fsync() description and associated definitions make no mention of specific file types, the requirements apply to all file types, including directories.

d. It is not clear how the definition of Synchronized I/O Data Integrity Completion applies to modifications to directories, since it is worded in terms of "write" operations.

Notes to the Editor (not part of this interpretation):

Make the changes described in Note: 0001542.

(0001550)
joerg (reporter)
2013-04-25 11:49

The historical UNIXv6 filesystem did not have rename(), but UFS from *BSD that started in 1981 did and even the original implementaion did not grant a rename() to be synchronous but rather just atomic. In case of a crash, either the old or the new state was recovered.

The concept at that time already was to order the disk operations in a way that grants a consistent state but not necessarily a specific operation to be on the background storage.

Newer concepts like ZFS write copies of meta data that forms a new tree and a new state appears as the stable state on the background storage after the related new suberblock (ZFS calls it Überblock) has been written.

(0001567)
ajosey (manager)
2013-05-03 09:14

[Proposal from Paul Eggert 2 May 2013]

Well, here's a first cut. I'm sure it could use improvement. In particular, I don't like that last paragraph....

Changes to XBD

After XBD page 94 line 2613, insert:

> For the purpose of this definition, an operation that reads or searches a directory is considered to be a read operation, an operation that modifies a directory is considered to be a write

operation, and a directory's entries are considered to be the data
read or written.

After page 107 line 2885 add a new XBD 4.2 section (and renumber the current 4.2 and all later 4.x sections):

4.2 Directory Operations

All file system operations that read or search a directory or that modify the contents of a directory (for example creating, unlinking, or renaming a file) shall operate atomically. That is, each operation shall either have its entire effect and succeed, or shall not affect the file system and shall fail. Furthermore, these operations shall be serializable, that is, the state of the file system and of the results of each operation shall always be values that would be obtained if the operations were executed one after the other.

Changes to XSH

At page 579 line 19973 section aio_fsync() change the APPLICATION USAGE section from:

None.

to:

Refer to fdatasync() and fsync().

At page 821 line 27587 section fdatasync() change the APPLICATION USAGE section from:

None.

to:

An application that modifies a directory, e.g., by creating a file in the directory, can invoke fdatasync() on the directory to ensure that the directory's entries are synchronized.

After page 963 line 32522 section fsync() add a new paragraph to the APPLICATION USAGE section:

An application that modifies a directory, e.g., by creating a file in the directory, can invoke fsync() on the directory to ensure that the directory's entries and metadata are synchronized.

Changes to XRAT

After page 3472 line 117096 add a new XRAT A.4.2 section (and renumber the current A.4.2 and all later A.4.x sections):

A.4.2 Directory Operations

Earlier versions of this standard did not make clear that directory modifications are performed atomically and serially, although that is the historical behavior and was always intended. Earlier versions also did not specify the behavior of fdatasync() or fsync() on directories.

Although directory operations are atomic and serializable, they are not necessarily durable. An application that require a directory modification to be durable should use fdatasync() or fsync() on the directory.

It is unspecified whether a directory modification results in a consistent data structure in the storage device associated with the file system, even if fdatasync() or fsync() is used. Some operations, such as rename(), can affect more than one directory, whereas fdatasync() and fsync() can affect at most one directory at a time.

(0001600)
geoffclare
(manager)

Latest proposal, based on Note: 0001567 and email sequence 19026.
Page and line numbers are for the 2013 edition.

Changes to XBD...

2013-05-10 11:08

After page 94 line 2613, insert a new paragraph:

For the purpose of this definition, an operation that reads or searches a directory is considered to be a read operation, an operation that modifies a directory is considered to be a write operation, and a directory's entries are considered to be the data read or written.

After page 107 line 2884 add a new XBD 4.2 section (and renumber the current 4.2 and all later 4.x sections):

4.2 Directory Operations

All file system operations that read or search a directory or that modify the contents of a directory (for example creating, unlinking, or renaming a file) shall operate atomically. That is, each operation shall either have its entire effect and succeed, or shall not affect the file system and shall fail. Furthermore, these operations shall be serializable; that is, the state of the file system and of the results of each operation shall always be values that would be obtained if the operations were executed one after the other. If the file system is accessed via a memory cache, these requirements shall apply both to the file system state in the cache and to the file system state on the underlying storage.

If an application creates a regular file, writes to it, and then calls fdatasync(), fsync(), or aio_fsync() on it, the directory entry for the filename used to create the file shall be transferred to storage no later than the file contents.

< small>Note: Although conforming file systems are required to perform all directory modifications as described above, some file systems may support non-conforming configurations (for example via mount options) for which this is not the case. Applications that synchronize regular files but do not explicitly synchronize directories after modifying them should only be used with such file systems in their conforming configuration(s).</small>

Changes to XSH...

At page 579 line 19973 section aio_fsync() change the APPLICATION USAGE section from:

None.

to:

Refer to fdatasync() and fsync().

At page 821 line 27587 section fdatasync() change the APPLICATION USAGE section from:

None.

to:

An application that modifies a directory, e.g., by creating a file in the directory, can invoke fdatasync() on the directory to ensure that the directory's entries are synchronized, although for most applications this should not be necessary (see XBD 4.2).

After page 963 line 32522 section fsync() add a new paragraph to the APPLICATION USAGE section:

An application that modifies a directory, e.g., by creating a file in the directory, can invoke fsync() on the directory to ensure that the directory's entries and file attributes are synchronized. For most applications, synchronizing the directory's entries should not be necessary (see XBD 4.2).

Changes to XRAT...

After page 3472 line 117096 add a new XRAT A.4.2 section (and renumber the current A.4.2 and all later A.4.x sections):

A.4.2 Directory Operations

Earlier versions of this standard did not make clear that directory modifications are performed atomically and serially, although that is the historical behavior and was always intended. Earlier versions also did not specify the behavior of aio_fsync(), fdatasync() or fsync() on directories.

Although directory operations are atomic and serializable, they are not necessarily durable. An application that requires a directory modification to be durable should use fdatasync() or fsync() (or aio_fsync()) on the directory. However, the intention of the requirements for directory modifications is that most applications should not need to do this. For example, a common method of updating a file is to create a new temporary file, call fdatasync() or fsync() to synchronize the new file, and then use rename() to replace the old file with the new file. If a crash occurs after the rename(), then the file being updated will have either its old contents or its new contents on the storage device when the system is rebooted. An application need only synchronize the directory if it wants to be sure the updated file will have its new contents on the storage device.

It is unspecified whether a directory modification results in a consistent data structure in the storage device associated with the file system, even if aio_fsync(), fdatasync() or fsync() is used on directories. Some operations, such as rename(), can affect more than one directory, whereas these synchronization calls can affect at most one directory at a time. If the file system is inconsistent after a crash it is usually automatically checked and repaired when the system is rebooted, or can be repaired manually using a utility such as fsck.

(0001603)
eggert (reporter)
2013-05-14 16:15

In looking at the latest proposed change, I still see problems, and have further suggestions.

1. After discussing atomicity and serializability, the proposed 4.2 now says:

If the file system is accessed via a memory cache, these requirements shall apply both to the file system state in the cache and to the file system state on the underlying storage.

This additional requirement on underlying storage appears to be too strong. If I understand it correctly, it would require that the underlying storage is always the consistent result of serial atomic operations, and would require that sequences of operations be serializable even if the operations apply to different file systems. I doubt whether modern operating systems behave this way.

This sentence was apparently prompted by Geoff Clare's comment in < http://article.gmane.org/gmane.comp.standards.posix.austin.general/7348> [^] that the proposed 4.2 could all be read as applying only to the cache. But that is the intent of proposed 4.2 -- the proposal should not be read as applying to underlying storage. The discussion of underlying storage should be orthogonal to the proposed 4.2.

I suggest rewording this sentence to make this clear, perhaps something like the following:

If the file system is accessed via a memory cache, these requirements shall apply to the file system state in the cache. These requirements are in addition to the requirements for synchronized input and output.

If there is also a need to clarify how synchronized input and output work, perhaps we should add a new paragraph about it.

2. quoting text added to the proposed change:

If an application creates a regular file, writes to it, and then calls fdatasync(), fsync(), or aio_fsync() on it, the directory entry for the filename used to create the file shall be transferred to storage no later than the file contents.

This quoted text has several issues:

2A. I'm not sure that practical POSIXish file systems guarantee this property. Has this been checked?

2B. The quoted text seems to be catering to the following scenario (error-checking omitted):

```
  int fd = open ("temp", O_CREAT|O_WRONLY|O_EXCL|..., ...);
  write (fd, buf, sizeof buf);
  fsync (fd);
  close (fd);
  rename ("temp", "permanent");
```

But the quoted text doesn't suffice for this scenario, as the application must invoke fsync/fdatasync/etc on the parent directory for the 'rename' to be synchronized and for the change to be committed. And if the application does synchronize the parent directory, then the above-quoted requirement won't help the application -- so why is the requirement helpful?

2C. Even if some sort of requirement along these lines is needed, the quoted text is too strong. Surely the application doesn't need the directory entry to be transferred to storage before the file contents; it could be transferred afterwards. All that'd be needed is that the directory entry be transferred to storage before fsync() returns.

2D. The quoted text doesn't clearly state what happens when other operations intervene. For example, suppose the directory entry is unlinked or renamed after the file is created but before it is written or fsynced. Surely there's no intent that the creating directory entry must be synchronized to storage in these cases.

A simple way to work around these problems is to omit the quoted text.

3. In the <small>Note:

> Applications that synchronize regular files but do not explicitly synchronize directories after modifying them should only be used with such file systems in their conforming configuration(s).

If a file system has a non-conforming configuration, all bets are off: synchronization of any sort is unreliable. Unfortunately the above-quoted text could be misread to imply that although the mentioned synchronization doesn't work some other forms of synchronization do work. I suggest stating the point more generally instead. Perhaps something like this?

> Applications that are used on non-conforming file systems cannot rely on files being synchronized properly.

4. I thought of a new issue.

There's no way, even with this proposal, to synchronize symbolic links. Perhaps we should just clarify this by appending the following after XBD page 94 line 2613:

> The standard provides no way to synchronize the contents or attributes of a symbolic link.

Another option is to fix the symlink problem -- for example, I think GNU/Linux has a way to do this, with its O_PATH flag. But that's a bigger deal.

(0001618)
geoffclare
(manager)
2013-05-21 10:07
edited on: 2013-05-22 10:14

New proposal which is a modified version of Note: 0001600 based on subsequent email discussion.

Page and line numbers are for the 2013 edition.

Changes to XBD...

After page 94 line 2613, insert a new paragraph:

> For the purpose of this definition, an operation that reads or searches a directory is considered to be a read operation, an

operation that modifies a directory is considered to be a write operation, and a directory's entries are considered to be the data read or written.

The standard provides no way to synchronize the contents or attributes of a symbolic link.

After page 107 line 2884 add a new XBD 4.2 section (and renumber the current 4.2 and all later 4.x sections):

4.2 Directory Operations

All file system operations that read or search a directory or that modify the contents of a directory (for example creating, unlinking, or renaming a file) shall operate atomically. That is, each operation shall either have its entire effect and succeed, or shall not affect the file system and shall fail. Furthermore, these operations shall be serializable; that is, the state of the file system and of the results of each operation shall always be values that would be obtained if the operations were executed one after the other.

After page 107 line 2939 add a new XBD 4.8 section (and renumber the remaining 4.x sections):

4.8 File System Cache

If the file system is accessed via a memory cache, file-related requirements stated in the rest of this standard shall apply to the cache, except where explicitly stated otherwise: this includes directory atomicity and serializability requirements (see XBD 4.2), file times update requirements (see XBD 4.10), and read-write serializability requirements (see write()). Cache entries shall be transferred to the underlying storage as the result of successful calls to fdatasync(), fsync(), or aio_fsync(), and may be transferred to storage automatically at other times. Such transfers shall be atomic, with minimum units being directory entries (for directory contents), aligned data blocks of the fundamental file system block size (for regular-file contents; see <sys/statvfs.h>), and all attributes of a single file (for file attributes).

< small>Note: If the system crashes before the cache is fully transferred, later operations' effects may be present in storage with earlier effects missing.</small>

< small>Note: Operations that create or modify multiple directory entries, aligned data blocks, or file attributes (e.g., mkdir(), rename(), write() with large buffer size, open() with O_CREAT) may have only part of their effects transferred to storage, and after a crash these operations may appear to have been only partly done, with the parts not necessarily done in any order. For example, only the second half of a write() may be transferred; or rename("a","b") may result in "b" being created without "a" being removed.</small>

< small>Note: Although conforming file systems are required to perform all caching as described above, some file systems may support non-conforming configurations (for example via mount options) for which this is not the case. Applications that are used on non-conforming file systems cannot rely on files being synchronized properly.</small>

Changes to XSH...

At page 579 line 19973 section aio_fsync() change the APPLICATION USAGE section from:

None.

to:

Refer to fdatasync() and fsync().

At page 821 line 27587 section fdatasync() change the APPLICATION USAGE section from:

None.

to:

An application that modifies a directory, e.g., by creating a file in the directory, can invoke fdatasync() on the directory to ensure that the directory's entries are synchronized, although for most applications this should not be necessary (see XBD 4.8).

After page 963 line 32522 section fsync() add a new paragraph to the APPLICATION USAGE section:

An application that modifies a directory, e.g., by creating a file in the directory, can invoke fsync() on the directory to ensure that the directory's entries and file attributes are synchronized. For most applications, synchronizing the directory's entries should not be necessary (see XBD 4.8).

Changes to XRAT...

After page 3472 line 117096 add new XRAT A.4.2 and A.4.8 sections (and renumber the current A.4.2 and all later A.4.x sections).

A.4.2 Directory Operations

Earlier versions of this standard did not make clear that directory modifications are performed atomically and serially, although that is the historical behavior and was always intended.

A.4.8 File System Cache

Earlier versions of this standard did not specify the behavior of aio_fsync(), fdatasync() or fsync() on directories, nor did they specify constraints on the underlying storage in the absence of calls to aio_fsync, fdatasync() or fsync().

Although directory operations are atomic and serializable, they are not necessarily durable. An application that requires a directory modification to be durable should use fdatasync() or fsync() (or aio_fsync()) on the directory. However, the intention of the requirements for directory modifications is that most applications should not need to do this. For example, a common method of updating a file is to create a new temporary file, call fdatasync() or fsync() to synchronize the new file, and then use rename() to replace the old file with the new file. If a crash occurs after the rename(), then the file being updated will have either its old contents or its new contents on the storage device when the system is rebooted. An application needs to synchronize the directory only if it wants to be sure the updated file will have its new contents on the storage device.

Some operations, such as rename(), can affect more than one directory, whereas synchronization calls such as fsync() can affect at most one directory at a time. Two calls to fsync() may be needed after a rename() to ensure its durability.

If the file system is inconsistent after a crash it is usually automatically checked and repaired when the system is rebooted, or can be repaired manually using a utility such as fsck.

If an unrecoverable I/O error occurs when cache is transferred to storage, this standard provides no way for applications to discover the error reliably. Implementations are encouraged to report such errors on subsequent reads of the storage.

(0001624)
geoffclare
(manager)
2013-05-23 15:26
edited on: 2013-05-23 15:27

In the May 23 teleconference we agreed that we should ask for input from file system developers (specifically, Mark B. to ask the AIX devs, Jim P. to ask the Solaris devs, and Andrew to contact Apple and HP for their input; input from other file system developers would also be welcome).

The questions for the file system developers are:

1. Do you believe your file system(s) conform to the proposed requirements in Note: 0001618?

2. These requirements currently allow for the possibility that on rename("a/b", "c/d"), the removal of "b" from directory "a"

could be transferred to storage before the creation of "d" in directory "c" is transferred. Thus, if a crash occurs between these two transfers, this would result in neither "a/b" nor "c/d" existing on recovery. Is this correct for your file system(s), or do you handle renames in such a way as to ensure that this can't happen?

(0001691)
jim_pugsley
(manager)
2013-08-01 10:25

1. Solaris devs believe that UFS conforms to the proposed requirements.

2. UFS rename() is atomic, either both directories are modified or neither is.

Issue History

Date Modified	Username	Field	Change
2013-03-19 15:45	xroche	New Issue	
2013-03-19 15:45	xroche	Status	New => Under Review
2013-03-19 15:45	xroche	Assigned To	=> ajosey
2013-03-19 15:45	xroche	Name	=> Xavier Roche
2013-03-19 16:49	geoffclare	Note Added: 0001497	
2013-03-20 01:28	nialldouglas	Note Added: 0001498	
2013-03-20 01:30	nialldouglas	Note Deleted: 0001498	
2013-03-20 08:03	Konrad_Schwarz	Note Added: 0001499	
2013-04-09 06:51	xroche	Issue Monitored: xroche	
2013-04-19 15:27	geoffclare	Note Added: 0001542	
2013-04-25 02:22	eggert	Note Added: 0001545	
2013-04-25 03:40	eggert	Issue Monitored: eggert	
2013-04-25 09:09	geoffclare	Note Added: 0001548	
2013-04-25 09:14	geoffclare	Note Edited: 0001548	
2013-04-25 11:49	joerg	Note Added: 0001550	
2013-04-25 15:10	geoffclare	Note Edited: 0001548	
2013-05-03 09:14	ajosey	Note Added: 0001567	
2013-05-10 11:08	geoffclare	Note Added: 0001600	
2013-05-14 16:15	eggert	Note Added: 0001603	
2013-05-21 10:07	geoffclare	Note Added: 0001618	
2013-05-22 10:13	geoffclare	Note Edited: 0001618	
2013-05-22 10:14	geoffclare	Note Edited: 0001618	
2013-05-23 15:26	geoffclare	Note Added: 0001624	
2013-05-23 15:27	geoffclare	Note Edited: 0001624	
2013-07-16 10:08	sascha_silbe	Issue Monitored: sascha_silbe	
2013-08-01 10:25	jim_pugsley	Note Added: 0001691	