File Systems Part 3
Transactions

- “ACID” property:
  - atomic
    - all or nothing
  - consistent
    - take system from one consistent state to another
  - isolated
    - have no effect on other transactions until committed
  - durable
    - persists
How?

- Journaling
  - before updating disk with steps of transaction:
    - record previous contents: *undo journaling*
    - record new contents: *redo journaling*

- Shadow paging
  - steps of transaction written to disk, but old values remain
  - single write switches old state to new
Example Transactions (1)

- Create file
  - create inode
    - modify free vector/list
    - initialize inode
  - update directory
    - modify contents
      • possibly modify free vector
    - update directory inode
Example Transactions (2)

- Rename file
  - update new directory
    - update new directory inode
    - modify new directory
      - update free vector
  - update old directory
    - update old directory inode
    - modify old directory
      - update free vector
Example Transactions (3)

- Write to a file
  - for each block
    - update free vector
    - copy data to block
  - update inode
Example Transactions (4)

- Delete a file
  - for each block
    - update free vector
  - update inode free vector/list
  - update directory
    - update directory inode
    - modify directory
      - update free vector
Data vs. Metadata

• Metadata
  – system-maintained data pertaining to the structure of the file system
    - inodes
    - indirect, doubly indirect, triply indirect blocks
    - directories
    - free space description
    - etc.

• Data
  – data written via write system calls
Journaling

- Journaling options
  - journal everything
    - everything on disk made consistent after crash
    - last few updates possibly lost
    - expensive
  - journal metadata only
    - metadata made consistent after a crash
      - user data not
    - last few updates possibly lost
    - relatively cheap
Committing vs. Checkpointing

- Checkpointed updates
  - written to file system and are thus permanent
- Committed updates
  - not necessarily written to file system, but guaranteed to be written eventually (checkpointed), even if there is a crash
- Uncommitted updates
  - not necessarily written to file system (yet), may disappear if there is a crash
Ext3

- A journaled file system used in Linux
  - same on-disk format as Ext2 (except for the journal)
    - (Ext2 is an FFS clone)
  - supports both full journaling and metadata-only
Full Journaling in Ext3

- File-oriented system calls divided into subtransactions
  - updates go to cache only
  - subtransactions grouped together
- When sufficient quantity collected or 5 seconds elapsed, commit processing starts
  - updates (new values) written to journal
  - once entire batch is journaled, end-of-transaction record is written
- Cached updates are then checkpointed — written to file system
  - journal cleared after checkpointing completes
Quiz 1

You have a Linux system with an Ext3 file system with full journaling. You run a script that deletes some files, creates some new files and writes data to them, then renames the new files. This takes two seconds. Immediately after the script finishes, there’s a power failure and the system crashes. When it comes back up, after crash recovery:

a) it will definitely appear as if the script ran to completion

b) it will appear as if the script ran to some point and then terminated (this allows for both none of it and all of it)

c) some later parts of the script may have completed, even though earlier parts did not
Journaling in Ext3 (part 1)
Journaling in Ext3 (part 2)

File system

/ → dir1 → dir2

Journal

file1 inode
another inode

file2 inode
another inode

dir1 inode
dir2 inode

file1 data
file2 data

Free vector block x
Free vector block y

end of transaction

File-system block cache

new file2 data

another cached block

dir1 inode
dir2 inode

file2 inode

file1 data
Journaling in Ext3 (part 3)

File system

Journal

File-system block cache

end of transaction
Journaling in Ext3 (part 4)
Quiz 2

You have a Linux system with an Ext3 file system with metadata-only journaling. You run a script that deletes some files, creates some new files and writes data to them, then renames the new files. This takes two seconds. Immediately after the script finishes, there’s a power failure and the system crashes. When it comes back up, after crash recovery:

a) it will appear as if the script ran to some point and then terminated (this allows for both none of it and all of it)

b) the script may appear to have completed, though there’s no data written to the new files

c) there may be data written to the new files, but no files were deleted
Metadata-Only Journaling in Ext3

• It’s more complicated!
• Scenario (one of many):
  – you create a new file and write data to it
  – transaction is committed
    - metadata is in journal
    - user data still in cache
  – system crashes
  – system reboots; journal is recovered
    - new file’s metadata are in file system
    - user data are not
    - metadata refer to disk blocks containing other users’ data
Coping

- Zero all disk blocks as they are freed
  - done in “secure” operating systems
  - expensive
- Ext3 approach
  - write newly allocated data blocks to file system before committing metadata to journal
  - fixed?
Yes, but ...

- Mary deletes file A
  - A’s data block x added to free vector
- Ted creates file B
- Ted writes to file B
  - block x allocated from free vector
  - new data goes into x
  - system writes newly allocated x to file system in preparation for committing metadata, but …
- System crashes
  - metadata did not get journaled
    - A still exists; B does not
    - B’s data is in A
Fixing the Fix

• Don’t reuse a block until transaction freeing it has been committed
  – keep track of most recently committed free vector
  – allocate from it
Fixed Now?

• No …
Yet Another Problem (part 1)
Directory A and file X are created, then deleted. These operations become part of the same transaction, which is committed, but not checkpointed. However, since it was committed, the blocks occupied by A and X are added to the free vector and available for use.
File Y is created and written to. Its data block is the one that formerly held the contents of A. The transaction containing Y is committed separately from the other transaction and, perhaps because it’s shorter, is checkpointed first.
The system crashes and comes back up. As part of recovery, the contents of the journal are applied to the file system. The journaled contents of the block that originally contained the entries of directory A are copied back into their original location, which now is the first data block of Y. Thus Y’s previous contents are destroyed.
The Fix

- The problem occurs because metadata is modified, then deleted.
- Don’t blindly do both operations as part of crash recovery
  - no need to modify the metadata!
  - Ext3 puts a “revoke” record in the journal, which means “never mind…”
Fixed Now?

• Yes!
  − (or, at least, it seems to work ...)

Operating Systems in Depth  
XVI–29  
Copyright © 2019 Thomas W. Gosper. All rights reserved.
Ext4

- Latest Linux file system
  - used at Brown CS
- Retains much of Ext3
  - journaling
  - inodes
- Adds extents
  - four extents in inode
  - if more needed, B-tree is used
Shadow Paging

- Refreshingly simple
- Provides historical snapshots
- Examples
  - WAFL (Network Appliance)
  - ZFS (Sun)
A leaf node in a shadow-page tree is modified (step 1).
A leaf node in a shadow-page tree is modified (step 2). Copies are made of the leaf node and its ancestors all the way up to the root. A copy of the old root is maintained, pointing to a snapshot of the old version of the file system.
Quiz 3

When the shadow-page tree is updated:

a) all file-system data must be written to disk synchronously: writes may be cached for the sake of reads, but write system calls may not return until the data is on disk

b) file-system data may be cached (and written asynchronously) only if all lower parts of the tree are written to disk before upper parts

c) file-system data may be cached (and written asynchronously) as long as the root is written last