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
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
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
Journaling in Ext3 (part 2)

File system

/  
  ├── dir1
  │    └── dir2
  │         └── file1 data
  │         └── dir2 inode
  ├── file2 inode
  │    └── another inode
  │         └── another inode
  │                     └── another inode

Journal

file1 inode
another inode

dir1 data

dir2 data

Free vector block x

Free vector block y

end of transaction

File-system block cache

new file2 data

another cached block

dir1 inode
another inode

dir2 inode
another inode

file1 inode
another inode

file1 data

dir1 data

dir2 data

file1 data

file2 data

file2 data
Journaling in Ext3 (part 3)

File system

```
/  
  |  
  v  
dir1  
  |  
  v  
file1  
  |  
  v  
file2  
  |  
  v  
dir2  
```

Journal

```
file1
  inodes
  another
  inode

file2
  inodes
  another
  inode

dir1
  inodes
  another
  inode

dir2
  inodes
  another
  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
  inodes
  another
  inode

dir2
  inodes
  another
  inode

file1
  inodes
  another
  inode

file2
  inodes
  another
  inode

file1
data

file2
data

file1
data

file2
data

file1
data

file2
data
```

Operating Systems In Depth

Copyright © 2010 Thomas W. DuBois. All rights reserved.
Journaling in Ext3 (part 4)

File system

/  
  
  dir1  
  
  file1  
  
  dir2  
  
  file2

Journal

file1
inode
another
inode
dir1
data

file2
inode
another
inode
dir2
data

dir1
inode
another
inode
dir2
inode
another
inode

free vector block
x
free vector block
y
end of transaction

File-system block cache

new
file2
data

another cached block
dir1
inode
another
inode
dir1
data

dir2
inode
another
inode
dir2
data

free vector block
x
free vector block
y
file1
inode
another
inode
file1
data

free vector block
x
free vector block
y
file2
inode
another
inode
file2
data

Operating Systems In Depth

XV–14

Copyright © 2016 Thomas W. Davenport. All rights reserved.
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 …

- Eli deletes file A
  - A’s data block x added to free vector
- Jan creates file B
- Jan 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 …)
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)
Shadow-Page Tree

Root

Inode file indirect blocks

Inode file data blocks

Regular file indirect blocks

Regular file data blocks
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.
Desired Properties of Directories

- No restrictions on names
- Fast
- Space-efficient
## S5FS Directories

<table>
<thead>
<tr>
<th>Component Name</th>
<th>Inode Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>1</td>
</tr>
<tr>
<td>..</td>
<td>1</td>
</tr>
<tr>
<td>unix</td>
<td>117</td>
</tr>
<tr>
<td>etc</td>
<td>4</td>
</tr>
<tr>
<td>home</td>
<td>18</td>
</tr>
<tr>
<td>pro</td>
<td>36</td>
</tr>
<tr>
<td>dev</td>
<td>93</td>
</tr>
</tbody>
</table>
FFS Directory Format

Directory Block

<table>
<thead>
<tr>
<th>117</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>lux</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>tct</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>18</td>
</tr>
<tr>
<td>484</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>usr</td>
</tr>
<tr>
<td>10</td>
</tr>
</tbody>
</table>

Free Space
Extensible Hashing (part 1)

$h_2$

Indirect buckets

insert(Fritz)

$h_2(Fritz) = 2$

<table>
<thead>
<tr>
<th>Buckets</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>_inode index</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ralph</td>
<td></td>
</tr>
<tr>
<td>Lily</td>
<td></td>
</tr>
<tr>
<td>Joe</td>
<td></td>
</tr>
<tr>
<td>Belinda</td>
<td></td>
</tr>
<tr>
<td>George</td>
<td></td>
</tr>
<tr>
<td>Harry</td>
<td></td>
</tr>
<tr>
<td>Betty</td>
<td></td>
</tr>
</tbody>
</table>
Extensible Hashing (part 2)

Indirect buckets

Buckets

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Ralph</td>
<td>inode index</td>
</tr>
<tr>
<td>Lilly</td>
<td>inode index</td>
</tr>
<tr>
<td>Joe</td>
<td>inode index</td>
</tr>
<tr>
<td>Belinda</td>
<td>inode index</td>
</tr>
<tr>
<td>George</td>
<td>inode index</td>
</tr>
<tr>
<td>Harry</td>
<td>inode index</td>
</tr>
<tr>
<td>Betty</td>
<td>inode index</td>
</tr>
</tbody>
</table>

$h_3$
B+ Trees (part 1)
The result of adding entries for Igor and Lucy to the example of the previous slide.
The directory after removing Paula and Otto.