Operating Systems

Filesystem Examples and Case Studies

Filesystem Examples

- Ext2
- Ext3
- Ext4
- XFS
- JFFS
- WAFL
- ZFS

Ext2

- The Second Extended Filesystem
  - Default Linux filesystem for 7 years
  - Written in 1993 to replace the "Extended Filesystem"
  - Borrows heavily from the Berkeley Fast File System

Ext2 Features

- Variable Block Size(.5k, 1k, 2k and 4k)
- 2TB maximum file size
- 16TB maximum filesystem size

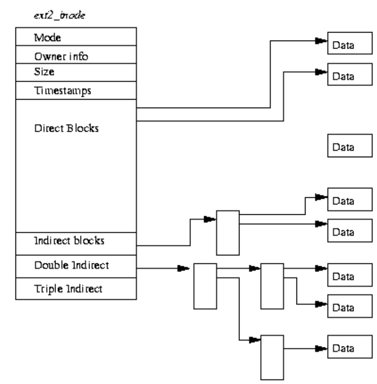
Ext2 On-Disk Layout

- Block Groups
  - Disk broken up into groups to keep related data "close"
  - Contain copies of "filesystem control information"

Super Block	Group Desc	Block Bitmap	Inode Bitmap	Inodes	Data

Ext2 On-Disk Layout

- Inodes



Ext2 On-Disk Layout

- Directories
  - Linked list of directory entries
  - Directory entries can be variable length
    - allows for long filenames

Inode Number	entry length	filename length	filename

Ext2 Performance Optimizations

- Allocates data blocks in same block group as the inode that refers to them
- Preallocates blocks on write

Ext2 Limitations

- No filesystem reliability
  - file system checks unwieldy with larger drives
- Directory scalability
  - O(n) time to find a file in the directory

Ext3

- Adds journaling capabilities to ext2
  - removes the need to perform file system checks if one shuts down improperly
- Maintains backward compatible with Ext2

Ext3 – Journaling

- Adds a file called .journal to the ext2 filesystem
  - Allows for easy conversion from ext2 to ext3 and vice versa
- When the filesystem is mounted as ext3, the .journal file is used for the transaction log
- When the filesystem is unmounted properly, it looks just like an ext2 filesystem.
- When not unmounted properly, it is marked so that ext2 knows not to mount it.

Ext3 Features

- Multiple types of journaling
  - Data Journaling – Data blocks are written to the journal so data is written twice. High overhead, high reliability
  - Ordered – Data updates are flushed before metadata transactions are committed
  - Writeback – Data updates are left to the “sync” mechanism so files can contain garbage after a crash
  - Allows administrators to tune the reliability of their filesystem

Ext3 – Performance Improvements

- Directory Scalability
  - HTree – similar to B-tree with low depth and high fanout
  - Directories are viewed as a hash table instead of linked list
  - To find a directory entry, simply hash the name and look at that location
  - Backwards compatible with ext2
  - Creating 100,000 files
    - 38 minutes with htree disabled
    - 11 seconds with htree enabled

Ext3 – Performance Improvements

- Improved Directory Allocation
  - Borrowed from the BSD world
  - Allocate subdirectories in the same block group as their parent
  - Allocate space for children when a directory is allocated
  - Put unrelated directories in different block groups (if possible)
  - Significantly improves data locality
  - As always, backwards compatible

Ext3 – Limitations

- Journaling means data gets written to disk twice
  - slower than ext2
- Backward compatibility with ext2 means that it can't make use of a number of newer filesystem features (extents, tail packing, etc)

Ext4

- Forked from Ext3 to address
  - 16TB filesystem limit due to 32-bit block numbers
  - 32K entry limit on subdirectories
  - timestamp resolution (seconds)
  - performance

Ext4

- 48-bit block numbers -> 1EB filesystem
- Indirect blocks replaced with extents
  - A single descriptor for a range of contiguous blocks
  - Logical, Length, Physical

```
struct ext4_extent {
 __le32 ee_block;      /* first logical block extent covers */
 __le16 ee_len;        /* number of blocks covered by extent */
 __le16 ee_start_hi;   /* high 16 bits of physical block */
 __le32 ee_start;      /* low 32 bits of physical block */
};
```

- 16TB file (32 bit logical block number)
- Max extent 128MB (15 bits)

Ext4

- Persistent pre-allocation – space is reserved and likely contiguous
  - fallocate() call
- Nanosecond timestamps
- Journal checksumming

XFS

- Written in 1996 as a replacement for EFS on Irix (The Silicon Graphics UNIX system)
  - EFS was not supporting the I/O rates that SGI customers needed
  - EFS didn't support filesystems that were greater than 8GB in size, or files greater than 2GB in size

XFS Features

- 9EB filesystems
- 9EB files
- Journaled
- Variable block size
- Good sparse file support

XFS Features

- Implementing Files
  - Inodes contain links to extents
    - If more extents are needed than fit in the inode, a B-tree is used to manage the list of extents
    - Permits efficient implementation of sparse files
- Pre-allocation – file space allocated, but nothing written (like Ext4)

XFS Features

- Support for Large Numbers of Files
  - Inodes are generated on the fly
  - A B-Tree keeps track of the locations of allocated inodes
  - Use 64-bit inode numbers
  - Much more complex than the simple methods used by filesystems like ext2 and ext3

XFS Features

- Implementing Directories
  - Entries are stored in a B-tree indexed on the filename
  - Filenames are hashed to make the B-tree implementation faster

XFS Features

- Managing Free Space
  - 2 B-trees used
    - One uses the extent location as the index
    - The other uses the length of the extent as the index
    - Allows for free space to be found quickly

XFS Features

- Delayed Allocation (Allocate-on-Flush)
  - Don't allocate space for files until you actually write them out
  - Much more likely to have contiguous allocation
  - Temporary files will never go to the disk

JFFS

- Journaling Flash File System
- Written by Axis Communications AB in Sweden
- A filesystem for embedded systems
  - Intended to be used on Flash memory devices

JFFS

- Flash Characteristics
  - Divided into blocks of a given size
  - Writes to a block require writing the entire block
  - Lifetime of Flash is measured in “erase” cycles
    - Hundreds of thousands

JFFS

- Configured as a Log-Structured Filesystem
  - Modifications write nodes to a Flash block
  - When the block fills, the FS consults a list of unused blocks to find the next block to write to
  - When the unused blocks list hits a threshold, garbage collection is performed

JFFS

- Garbage Collection
  - Selects a block from the dirty list or the clean list
  - Writes any valid nodes in it to the tail of the transaction log
  - Ensures an even distribution of writes to blocks

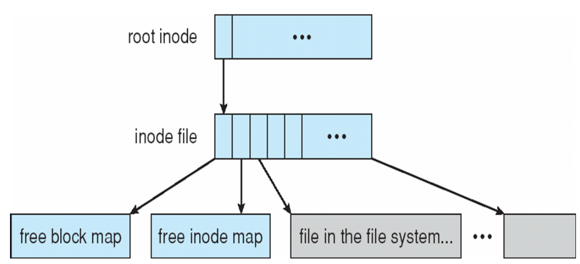
JFFS

- How do reads occur?
  - On boot, the entire medium is scanned
  - The most recent version of each inode is brought into memory

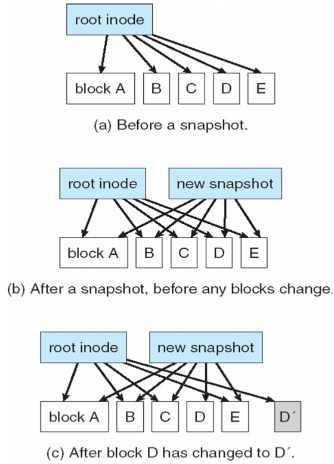
WAFL File System

- Used on Network Appliance's distributed file system appliances
- “Write-anywhere file layout”
- Serves NFS, CIFS, http, ftp
- Random I/O optimized, write optimized
  - NVRAM for write caching
- Similar to Berkeley Fast File System, with extensive modifications
- Uses novel approach for writing data to disk
  - Ensures that the filesystem is always consistent
  - Allows for easy creation of snapshots

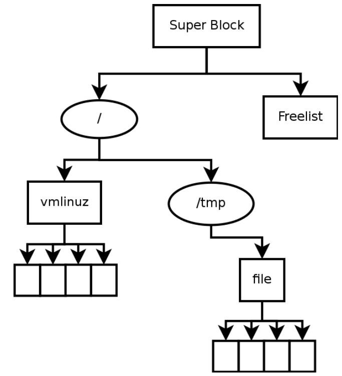
The WAFL File Layout



Snapshots in WAFL

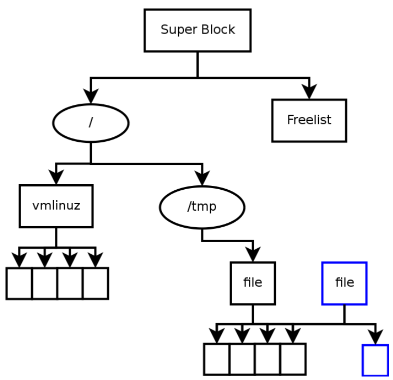


WAFL



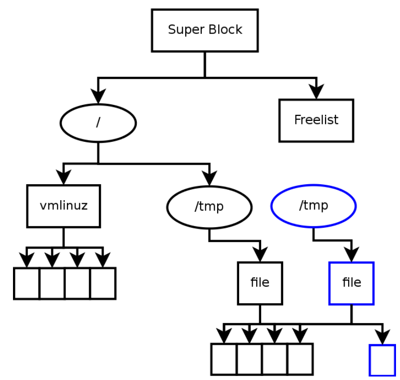
Initial Filesystem State

WAFL



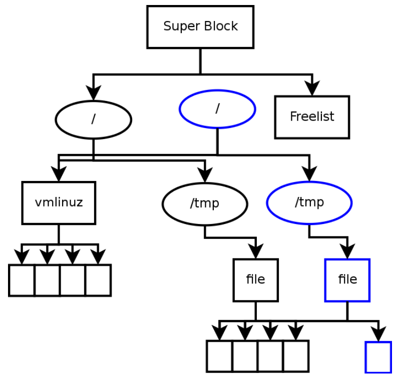
Create a new “file” inode

WAFL



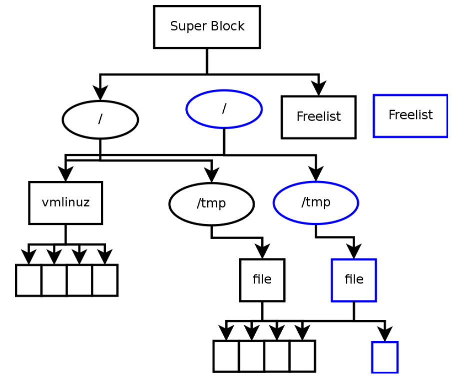
Create a new “/tmp” directory inode

WAFL



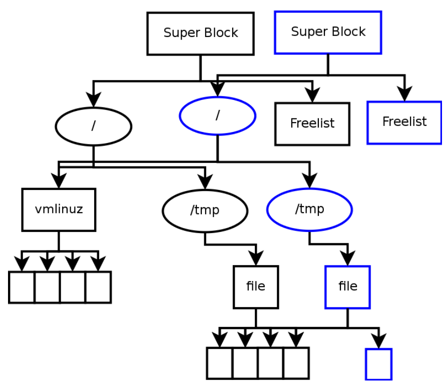
Create a new root directory inode

WAFL



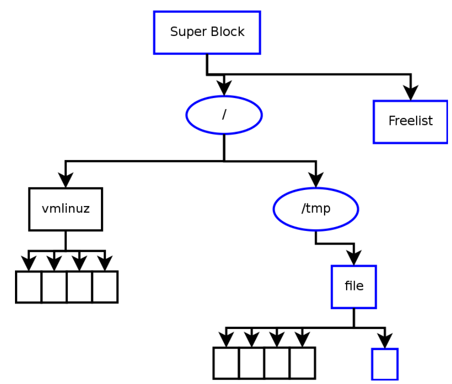
Update a copy of the freelist

WAFL



Create a new Super Block

WAFL



Overwrite the Old Superblock with the New One

ReiserFS4

- Attempted to unify the concepts of files and directories
  - Multics had directories as special types of file
  - ReiserFS4 defines files and directories as the same thing
    - A file can be accessed using open or opendir
    - File attributes implemented as “subfiles” of a given file
- Uses a plug-in architecture
  - allows new kinds of “files” to be implemented via a plug-in
  - very extensible

NTFS

- Allows files to have multiple data streams
  - Files are not just a sequence of bytes
- Case sensitive/insensitive filenames
- Per File/Directory Encryption
  - handled outside of the filesystem

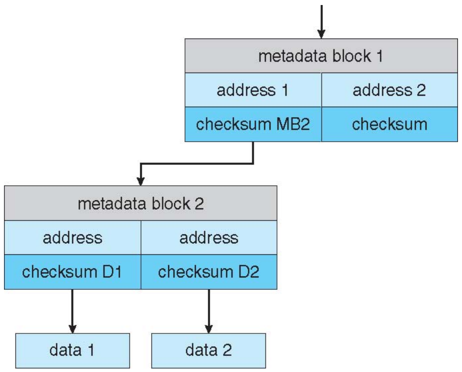
ZFS

- Developed by Sun
  - Open source
  - Originally stood for Zettabyte File System
- Performs copy on write similar to WAFL
  - Adds the ability to keep snapshots of filesystem state
- Includes an “intent log” to describe operations and aid in recovery
- Uses variable block sizes

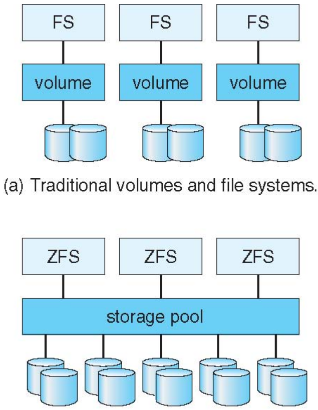
ZFS

- ZFS includes checksums of all data and metadata
  - Checksums kept with pointer to object, to detect if object is the right one and whether it changed
  - Can detect and correct data and metadata corruption
- ZFS also removes the notion of volumes, partitions
  - Disks allocated in pools
  - Filesystems with a pool share that pool, use and release space like “malloc” and “free” memory allocate / release calls

ZFS Checksums All Metadata and Data



Traditional and Pooled Storage in ZFS



(a) Traditional volumes and file systems.

(b) ZFS and pooled storage.

FUSE

- Filesystem in Userspace
- Loadable module that sends file I/O to a filesystem implementation in userspace