



# **Advanced Operating Systems**

## **Summer Semester 2023/2024**

**Martin Děcký**

# 5

## File Systems



# Classification of File Systems

- **Traditional**
  - **Examples:** ext4, XFS, NTFS, UFS (latest variants), BFS, JFS2, etc.
  - Universal set of features
  - Distinction between directory entries and i-nodes
  - On-disk layout affected by rotational media and traditional partitioning
  - Typically use of somewhat sophisticated data structures
  - Typically larger constant overhead
    - Not usable for small media
  - Reliability via journaling of changes
    - Soft updates as an alternative

# Classification of File Systems

- **Simpler traditional**
  - **Examples:** FAT, exFAT, etc.
    - Historical examples (with some advanced features): HPFS, HFS
  - Somewhat limited set of features
    - Typically missing permissions, ownership and other metadata, limited directory entry types, limited file names, limited file sizes, size of some data structures fixed, etc.
  - Frequently no distinction between directory entries and i-nodes
  - On-disk layout could be affected by slow / removable rotational media
  - Typically not so sophisticated data structures
  - Limited reliability

# Classification of File Systems

- **Optical**
  - **Examples:** ISO 9660, UDF
  - Compact, continuous structures to minimize seeking
    - Path tables, directories, files
  - Additional sessions referencing previous sessions
    - Keeping / adding / removing files
    - Wear leveling and block remapping for rewritable media
      - As opposed to hardware abstractions (e.g. Mount Rainier)
  - Hybrid media

# Classification of File Systems

- **Log-structured**
  - **Examples:** JFFS2, NILFS2, YAFFS, UBIFS, F2FS
  - **Idea:** Instead of keeping a journal for consistency, why not use the journal as the data storage?
  - Suits well zoned media (flash, SMR)
    - Block subdivision and GC more efficient than basic appending
  - Stale data can be accessed as snapshots (versions)
  - Inherently always consistent
  - Initial scan optimizations (persistent indexes)

# Classification of File Systems

- **Copy-on-write**
  - **Examples:** ZFS, btrfs, HAMMER2, APFS, ReFS
  - **Idea:** Flexible on-disk layout, but no overwrites
  - Stale data can be accessed as snapshots (versions)
  - Multiple mountable roots
  - Other advanced features (not strictly specific to COW)
    - Data checksums (separately stored, Merkle tree), data redundancy, deduplication, integration with logical volume management, hierarchical caching, wandering intent logs, replication
  - Inherently always consistent
  - Initial scan issues avoided, but GC still needed (also serves as defragmentation)

# Classification of File Systems

- **Read-only**
  - **Examples:** SquashFS, cramfs, EROFS, AXFS
  - Efficient storage of seed images (boot images, container images, thin provisioning, etc.)
    - Often coupled with union mounts for read/write support
  - Low overhead, no fragmentation, compression
  - Easy caching, execute-in-place (adaptive compression)



# Classification of File Systems

- **Shared-disk**
  - **Examples:** CXFS, GPFS, GFS2, OCFS, HAMMER2
  - Support for underlying block modifications from independent sources
    - Via iSCSI, ATA over Ethernet, Fibre Channel, InfiniBand, NVMe over fabric
  - In between regular file systems and network file systems
  - Distributed lock manager vs. metadata broker

# 5.1

## File System Curiosities



# Traditional File Systems with Bonuses

- **AdvFS, NSS**
  - Fairly traditional file systems, but supporting multiple block devices
- **HFS+**
  - Hard links to directories
- **RaiserFS**
  - Tail packing (sub-allocation of blocks)

# Traditional File Systems with Bonuses

- **NTFS**
  - Reparse points, file system filters
  - Caching i-node size in directory entry (non-consistent among hard links)
  - Hard links for 8.3 file names
  - Per-directory case sensitivity
    - Case insensitivity is not trivial [1][2]
  - Transactional NTFS
    - Integrated with Kernel Transaction Manager
    - Transaction-Safe FAT

# Traditional File Systems with Bonuses

- **XFS**
  - Allocation groups (concurrency)
  - Multiple devices, COW, snapshots, deduplication, striping
    - Controlled by Stratis
- **ext4**
  - Journal checksums
- **StegFS**
  - Steganographic extension to ext2
    - Undetectable, hidden layer of files on a regular file system

# Less Traditional File Systems

- **btrfs**
  - Integrated support for union mounting (read-only seeding)
- **Linear Tape File System (LTFS)**
- **NOVA**
  - Targeting byte-addressable persistent memory (NVRAM)
  - Log structured for metadata per i-node (concurrency)
    - Log is append-only, but non-continuous (linked list)
    - Replication and checksums
  - Data blocks managed as copy-on-write
  - Global journaling for reliability of non-atomic operations

# SPECIFYING FILE PATHS



**UNIX**

**WINDOWS**

Source: DALL-E 3 via ChatGPT 4



# Other File Systems Remarks

- **Resource forks, extended attributes**
  - Multiple streams associated with a single file
- **Forward and backward compatibility**
  - Feature sets, feature bitmaps
  - Allowed and required features
- **File system semantics are not trivial [3]**
- **Path lengths, valid path characters**
- **Path separator**
  - The history of slash / backslash is complicated [4][5]



# References

[1] <https://lwn.net/Articles/784041/>

[2] <https://www.youtube.com/watch?v=yVIEZKiMGJU>

[3] <https://danluu.com/deconstruct-files/>

[4] <https://www.os2museum.com/wp/why-does-windows-really-use-backslash-as-path-separator/>

[5] <https://learn.microsoft.com/en-us/archive/blogs/larryosterman/why-is-the-dos-path-character>



**Thank you!**

**Questions?**