CIS3110 Lecture 11 Summary - File Systems

Directory Structure and Organization

(File-System Interface - 13.1, 13.2, 13.3)

Root Directory Structure

- **Unix/Linux**: The root directory is identified by an inode number, stored as a file
- **FAT/NTFS**: The root directory is in a fixed location on disk, contains a fixed-size array of directory entries
- **Key difference**: The fixed-size root directory in FAT/NTFS limits the number of entries (typically 1024), whereas Unix can have an unlimited number of entries since the root directory is stored as a file that can grow

Directory Entries

- **Unix/Linux**: Directory entries contain only the filename and inode number
- **FAT/NTFS**: Directory entries contain filename plus all file attributes (size, date, permissions, etc.)
- **Unix advantage**: More efficient storage for directory entries, especially with many files

Directory Navigation

- Both systems have the concept of "." (current directory) and ".." (parent directory)
- Both systems organize directories in a hierarchical tree structure
- Unix maintains these as actual directory entries, Windows handles parent directory references differently

File Structure and Block Organization

(File-System Structure - 14.1, 14.2, 14.3, 14.4)

Unix File System Design

- **Inode structure**: Contains all file metadata except the name
- **Block access optimization**:
  - Direct blocks for small files (most common case)
  - Single indirect blocks for medium files
  - Double indirect blocks for larger files
  - Triple indirect blocks for very large files

- **Asymmetrical tree design**: Optimized for small files, which are statistically more common

NTFS Design

- Uses a B-tree structure for block tracking
- Consistent approach for all file sizes
- Requires balancing but handles changes well
- No special optimization for small files

FAT Design

- Simple linked list structure
- Each entry points to the next block
- Simple to implement but inefficient for random access
- Still used for small drives (USB sticks, etc.) due to simplicity

Space Efficiency Strategies

(File-System Structure - 14.3, 14.5)

Block Size and Waste

- Larger block sizes improve performance but increase waste
- Example: 4KB blocks can waste up to 45% of disk space, larger blocks (64KB) can waste significantly more

Fragment Block Technique

- McKusick's solution to waste: fragment blocks
- Collects partial blocks at the end of files into a shared block
- Small table at beginning of fragment block indicates offsets
- Can achieve up to 97% disk utilization
- Very small files might entirely reside in fragment blocks

Free Space Management

- **FAT**: Flags entries in the FAT table as free
- **Unix/NTFS**: Use a linked list of free blocks, with each list entry containing multiple block references
- Unix/NTFS approach reduces seek operations when allocating new blocks

Disk Performance Optimization

(File-System Structure - 14.6, I/O Systems - 12.4)

Extent-Based Allocation

- Store file blocks in physically adjacent locations
- Maximizes disk bandwidth, minimizes seek time
- Used heavily in database systems
- Challenge: hard to maintain as files grow or are modified

Cylinder Groups

- Organize disk into "neighborhoods" of multiple cylinders
- Keep related data in the same cylinder group
- Less rigid than extent-based allocation but provides similar benefits
- New files placed in same cylinder group as their parent directory
- New directories placed in different cylinder groups to distribute load

Allocation Strategies

- Regular files: placed in same cylinder group as parent directory
- Directories: placed in cylinder groups with few directories and many free inodes
- Large files: striped across multiple cylinder groups to prevent saturation

Physical Disk Structure

Terminology

- **Sector**: Smallest addressable unit on disk (typically 512 bytes)
- **Block**: Collection of sectors, consistent size across filesystem (4KB, 8KB, 16KB)
- **Track**: All sectors accessible by one head position
- **Cylinder**: All tracks at same distance from disk center (accessible without moving heads)
- **Platter**: Physical disk containing two surfaces (top and bottom)

Disk Vendor Considerations

- Vendors use base-10 for capacity (1TB = 1,000,000,000,000 bytes)
- Computing uses base-2 (1TB = 1,099,511,627,776 bytes)
- This discrepancy explains why formatted disks show less capacity than advertised

File System Buffer Management

(I/O Systems - 12.3, 12.4, 12.5, File-System Internals - 15.5)

Virtual File System Layer

- Abstracts different file systems behind common interface
- Allows transparent access to both local and remote file systems
- Uses vnodes (virtual nodes) for common attributes and file-system specific extensions

Buffer Caches

- **Physical block buffer cache**: Holds metadata and file blocks from physical disks
- **Logical block buffer cache**: Holds file data that could be from local or remote systems
- Allows sharing of file data between systems (Network File System)

Memory Management for I/O

- Kernel uses mbufs (memory buffers) for general-purpose storage
- Mbufs can be dynamically allocated between frame storage and I/O buffers
- Avoids unnecessary paging when memory is available

File I/O Operations

Read Process

1. Determine logical block number from file offset
2. Check if block is in buffer cache
3. If found, copy data to user space
4. If not found:
   - Allocate space in buffer cache
   - Translate logical block to physical block using metadata (FAT or inode)
   - Issue read request to disk controller
   - Block process until I/O completes
   - When data available, copy to user space

Sparse Files

- Files with "holes" where data has never been written

- System doesn't allocate blocks for holes
- Reading from holes returns zeros
- Saves space for certain applications

Reliability Considerations

- Multiple copies of critical structures (FAT tables, superblocks)
- Ordering of operations (commit inode before directory entry)
- Trade-offs between performance (delayed writes) and reliability (synchronous writes)
- File system recovery after crashes

Implementation Differences

Feature	Unix/Linux	FAT	NTFS
Directory entries	Name + inode	Name + all attributes	Name + all attributes
File access	Asymmetric tree	Linked list	B-tree
Block management	Direct/indirect blocks	FAT table	B-tree
Free space	Free block lists	FAT table entries	Free block lists
Permissions	User/group/other	Read-only bit	ACLs
Space efficiency	Fragment blocks	Poor	Better than FAT
Small file optimization	Yes	No	No
Directories	Growable files	Fixed size	Growable

Key Concepts to Remember

1. File systems have two distinct structures:
    - The directory tree that users see and navigate
    - The block management system that organizes data on disk
2. The primary challenge in file system design is balancing:
    - Space efficiency
    - Performance (minimizing seeks)
    - Reliability
    - Simplicity of implementation
3. Unix file systems optimize for small files and random access
4. System calls like read/write are abstracted from the physical storage details through layers of indirection

5. Buffer caches are critical for performance by reducing physical disk operations