Operating Systems
[14B. File-System Implementation]

Chung-Wei Lin

cwlin@csie.ntu.edu.tw

CSIE Department

National Taiwan University

From Operating System Concepts (10th Edition)

Outline

- File-System Structure, File-System Operations
- Directory Implementation, Allocation Methods
- **Free-Space Management**
  - Bit Vector
  - Linked List
  - Grouping
  - Counting
  - Space Maps
  - TRIMing Unused Blocks
- Efficiency and Performance
- Recovery
- Example: WAFL File System

Free-Space Management

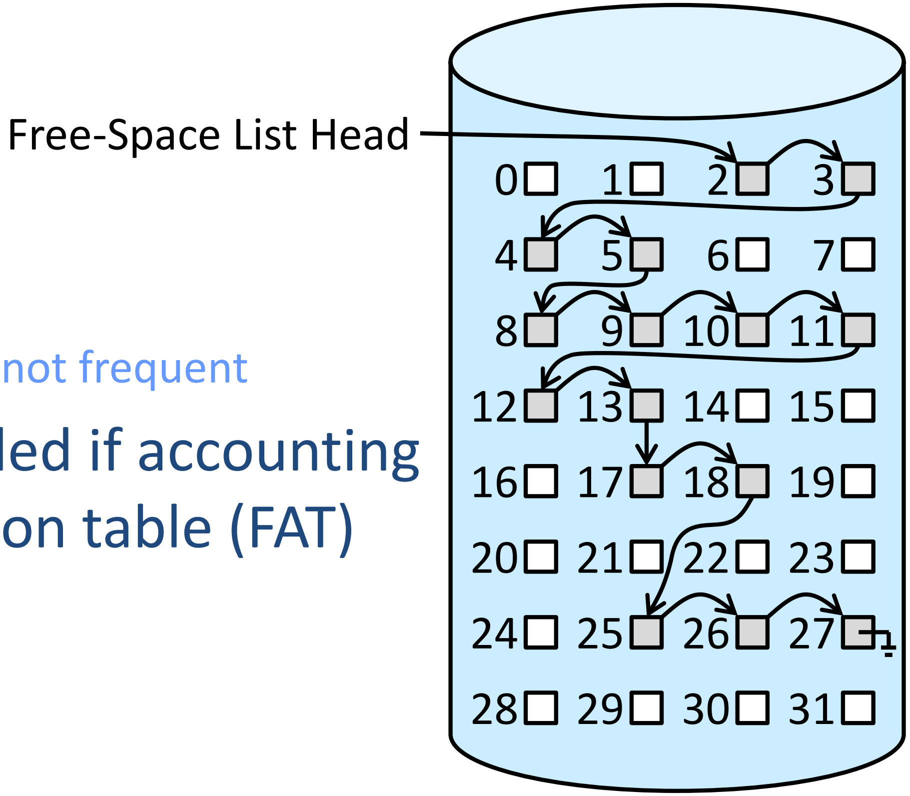
- To keep track of free disk space, the system maintains a **free-space list** which records all free device blocks
  - To create a file, we search the free-space list for the required amount of space and allocate that space to the new file
    - This space is then removed from the free-space list
  - When a file is deleted, its space is added to the free-space list
- The free-space list, despite its name, is not necessarily implemented as a "list"

Bit Vector (or Bitmap)

- If a block is free, its bit is 1; if the block is allocated, its bit is 0
- Calculation of the first free block
  - (# of bits per word) * (# of first 0-value words) + offset of first 1 bit
    - Bit-manipulation instructions can be used effectively
- Advantage
  - Simple
- Disadvantage
  - Inefficient unless the entire vector is kept in main memory
  - Example
    - 1-TB (2^{40} bytes) disk with 4-KB (2^{12} bytes) blocks
    - The number of blocks = $2^{40}/2^{12} = 2^{28}$
    - The size of bit vector = 2^{28} bits (32 MB)
    - If we use 4-block clusters instead of blocks, we still need 8 MB

Linked List

- Link together all the free blocks
- Keep a pointer to the first free block in a special location in the file system and cache it in memory
- Advantage
  - No waste of space
- Disadvantage
  - Inefficient to traverse the list
    - Fortunately, traversing the list is not frequent
- No separate method is needed if accounting free-block into a file-allocation table (FAT)



Grouping

- A modification of the linked-list (free-list) approach
  - Stores the addresses of n free blocks in the first free block
    - The first n-1 of these blocks are actually free
    - The last block contains the addresses of another n free blocks, and so on
- Advantage
  - The addresses of a large number of free blocks can be found faster than the standard linked-list approach

Counting

- Each entry consists of an address and a count
  - Keep the address of the first free block and the number (n) of free contiguous blocks that follow the first block
    - Not keeping a list of n free block addresses
    - Similar to the extent method of allocating blocks
- Advantage
  - Shorter list, as long as the count is generally greater than 1
    - Several contiguous blocks may be allocated or freed simultaneously, particularly with the contiguous-allocation algorithm or through clustering
- These entries can be stored in a balanced tree for efficient lookup, insertion, and deletion
  - Rather than a linked list

Space Maps

- Oracle's **ZFS** file system was designed to encompass huge numbers of files, directories, and even file systems
  - On these scales, metadata I/O can have a large performance impact
    - Need to control the size of data structures and minimize the metadata I/O
- Divide the space into **metaslabs** of manageable size
  - A given volume may contain hundreds of metaslabs
  - Each metaslab has an associated space map
- Use log-structured file-system techniques
  - A space map is a log of all block activity, in time order, in counting format
  - When ZFS decides to allocate or free space from a metaslab
    - Load the space map into memory in a balanced-tree structure indexed by offset
    - Replay the log into that structure

TRIMing Unused Blocks

- Some storage devices (e.g., HDD) allowing blocks to be overwritten need only the free list for managing free space
  - A block does not need to be treated specially when freed
    - A freed block typically keeps its data until the data are overwritten
- Some storage devices (e.g., NVM) must be erased before they can again be written to
  - A new mechanism is needed to allow the file system to inform the storage device that a page is free and can be considered for erasure
    - For ATA-attached drives, it is TRIM
    - For NVMe-based storage, it is the **`unallocate`** command
  - With them, the garbage collection and erase steps can occur before the device is nearly full

Outline

- ❑ File-System Structure
- ❑ File-System Operations
- ❑ Directory Implementation
- ❑ Allocation Methods
- ❑ Free-Space Management
- ❑ **Efficiency and Performance**
  - ➢ Efficiency
  - ➢ Performance
- ❑ Recovery
- ❑ Example: WAFL File System

Efficiency and Performance

- Disks tend to be a major bottleneck in system performance
  - They are the slowest main computer component
- Even NVM devices are slow compared with CPU and main memory

Some Factors

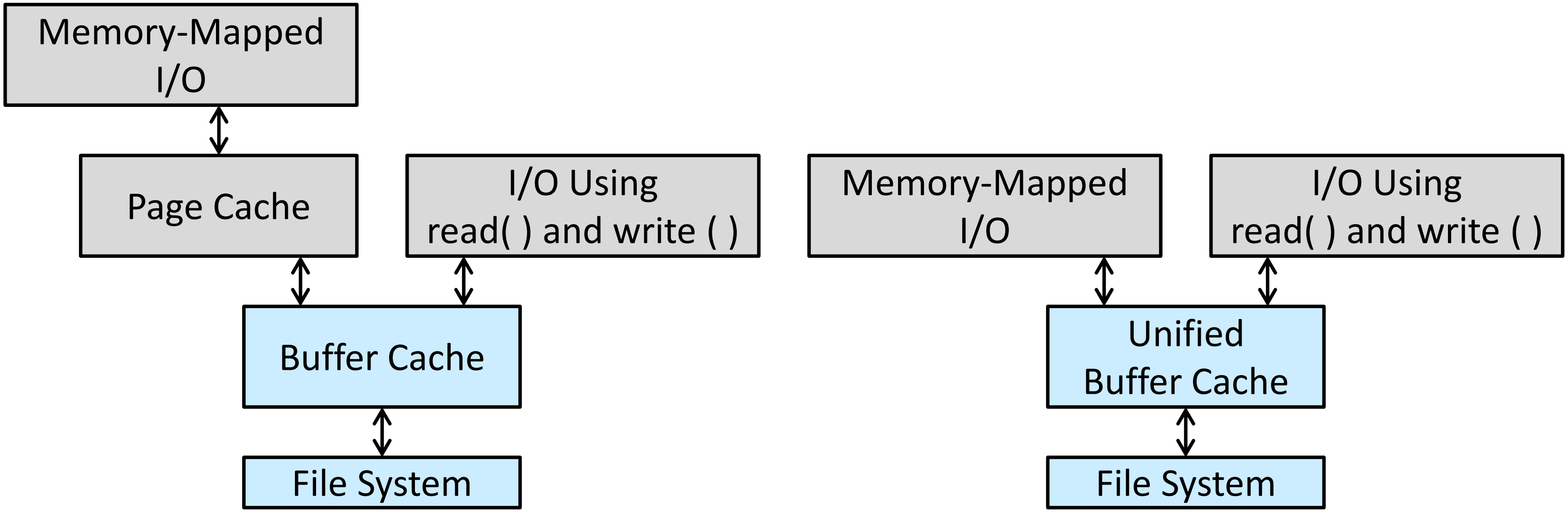
- Allocation and directory algorithms
  - Example: UNIX tries to keep a file's data blocks near that file's inode block to reduce seek time
- Types of data kept in a file's directory
  - What if keeping the "last access date" of a file?
- Size of the pointers used to access data
- Fixed-size or varying-size (dynamically-allocated) data structures

Buffer Cache and Page Cache

- Some systems maintain a separate section of main memory for a **buffer cache**
  - Keep blocks which are assumed to be used again shortly
- Other systems cache file data using a **page cache**
  - Use virtual memory techniques to cache file data as pages rather than as file-system-oriented blocks
    - More efficient than caching through physical disk blocks
  - Memory-mapped I/O uses a page cache
  - Routine I/O through the file system uses a buffer cache
- **Unified virtual memory**
  - Several systems, including Solaris, Linux, and Windows, use page caching to cache both process pages and file data

Unified Buffer Cache

- Use the same page cache for both memory-mapped I/O mapping and file system I/O to avoid **double caching**
  - Double caching
    - Waste memory
    - Waste significant CPU and I/O cycles
    - Result in corrupt files with inconsistencies between the two caches



Synchronous and Asynchronous Writes

- Synchronous write
  - The writes are not buffered
  - The calling routine must wait for the data to reach the drive
- Asynchronous write
  - The writes are stored in the cache
  - The calling routine proceeds
- Most writes are asynchronous
  - However, metadata writes can be synchronous
- Operating systems frequently allows a process to request writes to be synchronous
  - Example: databases use this feature for atomic transactions to assure that data reach stable storage in the required order

Free-Behind and Read-Ahead

- A file being read or written sequentially should not have its pages replaced in least recently used (LRU) order
  - The most recently used page will be used last, or perhaps never again
- Sequential access can be optimized
  - **Free-behind** removes a page from the buffer as soon as the next page is requested
    - The previous pages are not likely to be used again and waste buffer space
  - **Read-ahead** reads and caches a requested page and several subsequent pages
    - These pages are likely to be requested after the current page is processed

Outline

- ❑ File-System Structure
- ❑ File-System Operations
- ❑ Directory Implementation
- ❑ Allocation Methods
- ❑ Free-Space Management
- ❑ Efficiency and Performance
- ❑ **Recovery**
  - ➢ Consistency Checking
  - ➢ Log-Structured File Systems
  - ➢ Other Solutions
  - ➢ Backup an Restore
- ❑ Example: WAFL File System

Consistency Checking

- Consistency checking
  - Compare the data in the directory structure and other metadata with the state on storage
  - Try to fix any inconsistency it finds
- The allocation and free-space-management algorithms dictate
  - What types of problems the checker can find
  - How successful it will be in fixing them

Log-Structured File Systems

- **Log-based transaction-oriented** (or **journaling**) file systems
  - All metadata changes are written sequentially to a log
    - The log may be in a separate section of the file system or even on a separate storage device
  - Once the changes are written (committed) to this log, the system call can return to the user process
  - These log entries are replayed across the actual file-system structures
  - When the file-system structures are modified, the **transaction** is removed from the log
    - Transaction: each set of operations for performing a specific task
- If the system crashes, any transactions which were not completed to the file system must now be completed
  - The only problem occurs when a transaction was not committed before the system crashed

Other Solutions

- **Snapshot**: a view of the file system at a specific point in time
  - Some systems let a transaction write all data and metadata changes to new blocks
    - They never overwrite blocks with new data
  - When the transaction is complete, the metadata structures that pointed to the old blocks are updated to point to the new blocks
  - The file system can then remove the old pointers and the old blocks and make them available for reuse
    - If the old pointers and the old blocks are kept, a snapshot is created
- ZFS further provides checksumming of all metadata and data blocks

Backup an Restore

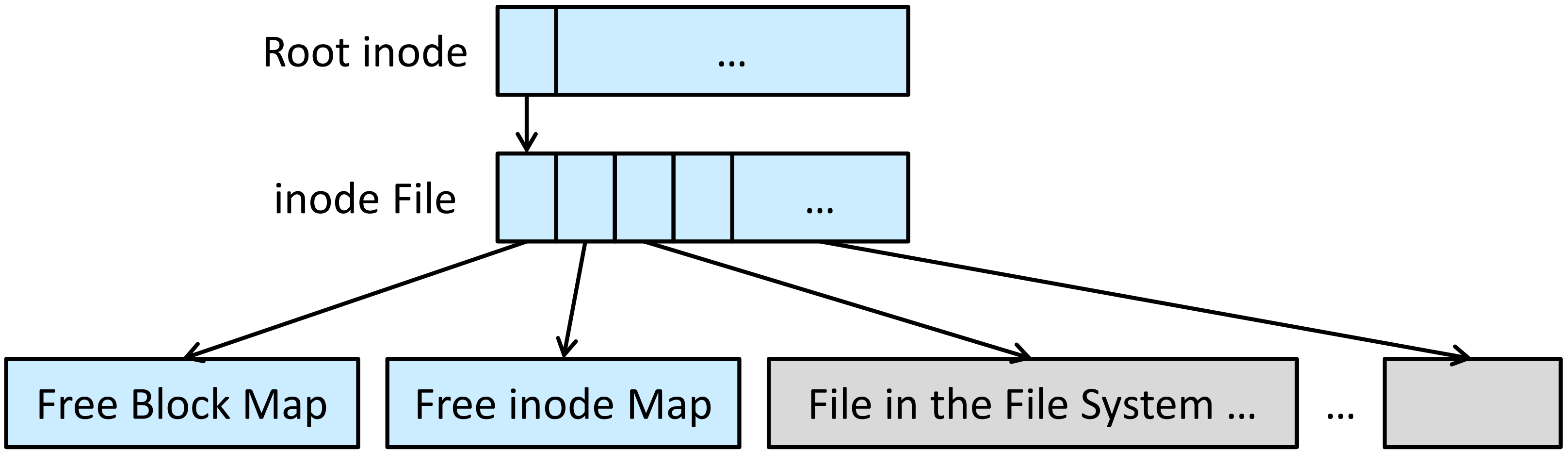
- System programs can be used to **back up** data from one storage device to another
- Recovery from the loss of an individual file or an entire device may then be a matter of **restoring** the data from backup
- A typical backup schedule
  - A **full backup** + multiple **incremental backup**

Outline

- File-System Structure
- File-System Operations
- Directory Implementation
- Allocation Methods
- Free-Space Management
- Efficiency and Performance
- Recovery
- **Example: WAFL File System**

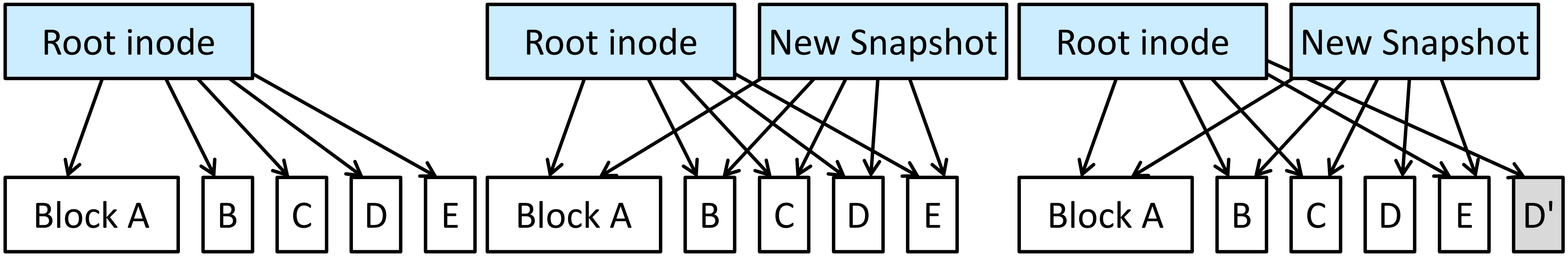
Example: WAFL File System

- The **write-anywhere file layout** (WAFL) is a file system optimized for random writes
  - Used exclusively on network file servers produced by NetApp
  - Meant for use as a distributed file system
- A file server may see a very large demand for random reads and an even larger demand for random writes
  - The NFS and CIFS protocols cache data from read operations, so writes are of the greatest concern to file-server creators



Snapshots in WAFL

- Many snapshots can exist simultaneously
  - The snapshot facility is useful for backups, testing, versioning
  - The snapshot facility is also very efficient
    - Not even require that copy-on-write copies be taken before the block is modified
- Other features
  - **Clone**: read-write snapshots
  - **Replication**: the duplication and synchronization of a set of data over a network to another system



Before a snapshot

After a snapshot and before any block changes

After block D has changed to D'

Objectives

- Describe the details of implementing local file systems and directory structures
- Discuss block allocation and free-block algorithms and trade-offs
- Explore file system efficiency and performance issues
- Look at recovery from file system failures
- Describe the WAFL file system as a concrete example

Q&A