CSCI315 – Operating Systems Design

Department of Computer Science
Bucknell University

File System Implementation 3

Ch 14.5-14.6

This set of notes is based on notes from the textbook authors, as well as L. Felipe Perrone, Joshua Stough, and other instructors.
Xiannong Meng, Fall 2021.

Review

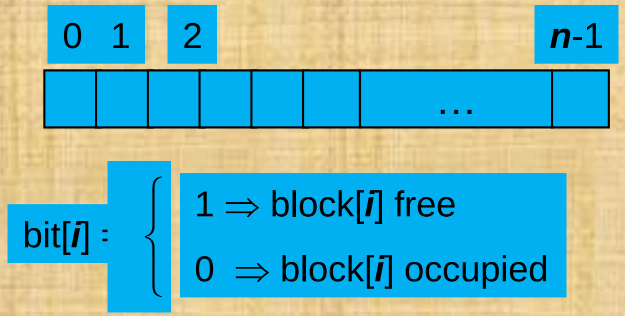
- We discussed different ways of allocating blocks for file data
  - **contiguous allocation** in which data blocks for a file are contiguous
  - **linked allocation** in which the data blocks are “chained” together, similar to a linked list
  - **indexed allocation** in which one index block points all data blocks
  - **hybrid** (or combination) of the above such as what is used in Linux systems.

Performance

- Best method depends on file access type
  - Contiguous great for sequential and random access
- Linked good for sequential, not random
- Declare access type at creation -> select either contiguous or linked
- Indexed more complex
  - Single block access could require 2 index block reads then data block read
  - Clustering can help improve throughput, reduce CPU overhead

Free-Space Management

- File system maintains **free-space list** to track available blocks/clusters
  - (Using term “block” for simplicity)
- **Bit vector** or **bit map** (*n* blocks)



Free block number calculation

(number of bits per word) * (number of 0-value words) + offset of first 1 bit

CPUs have instructions to return offset within word of first “1” bit

Free-Space Management (Cont.)

- Bit map requires extra space
  - Example:

    block size = 4KB = 2^{12} bytes

    disk size = 2^{40} bytes (1 terabyte)

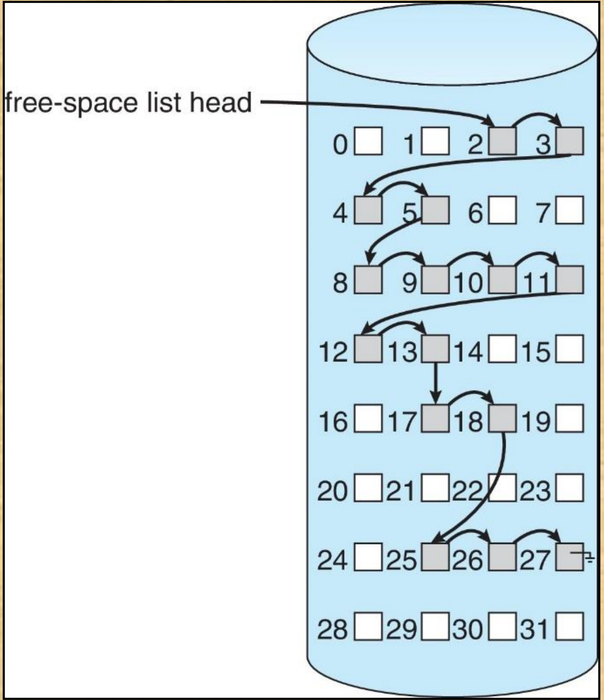
    $\boldsymbol{n} = 2^{40}/2^{12} = 2^{28}$ bits (or 32MB)

    if clusters of 4 blocks -> 8MB of memory

- Easy to get contiguous files

Linked Free Space List on Disk

- Linked list (free list)
  - Cannot get contiguous space easily
  - No waste of space
  - No need to traverse the entire list if # free blocks recorded



Free-Space Management (Cont.)

- Grouping
  - Modify linked list to store address of next *n-1* free blocks in first free block, plus a pointer to next block that contains free-block-pointers
- Counting
  - Because space is frequently contiguously used and freed, with contiguous-allocation allocation, extents, or clustering
    - Keep address of first free block and count of following free blocks
    - Free space list then has entries containing addresses and counts

Free-Space Management (Cont.)

- Space Maps
  - Used in **ZFS** https://en.wikipedia.org/wiki/ZFS
  - Consider meta-data I/O on very large file systems
    - Full data structures like bit maps couldn't fit in memory -> thousands of I/Os
  - Divides device space into **meta-slab** units and manages meta-slabs
    - Given volume can contain hundreds of meta-slabs
  - Each meta-slab has associated space map
    - Uses counting algorithm
  - But records to log file rather than file system
    - Log of all block activity, in time order, in counting format
  - Meta-slab activity -> load space map into memory in balanced-tree structure, indexed by offset
    - Replay log into that structure
    - Combine contiguous free blocks into single entry

TRIMing Unused Blocks

- HDDS overwrite in place so need only free list
- Blocks not treated specially when freed
  - Keeps its data but without any file pointers to it, until overwritten
- Storage devices not allowing overwrite like **NVM** (non-volatile memory) suffer badly with same algorithm
  - Must be erased before written, erases made in large chunks (blocks, composed of pages) and are slow
  - TRIM is a newer mechanism for the file system to inform the NVM storage device that a page is free
    - Can be garbage collected or if block is free, now block can be erased

https://en.wikipedia.org/wiki/Trim_(computing)

Efficiency and Performance

- Efficiency dependent on:
  - Disk allocation and directory algorithms
  - Types of data kept in file’s directory entry
  - Pre-allocation or as-needed allocation of metadata structures
  - Fixed-size or varying-size data structures

Efficiency and Performance (Cont.)

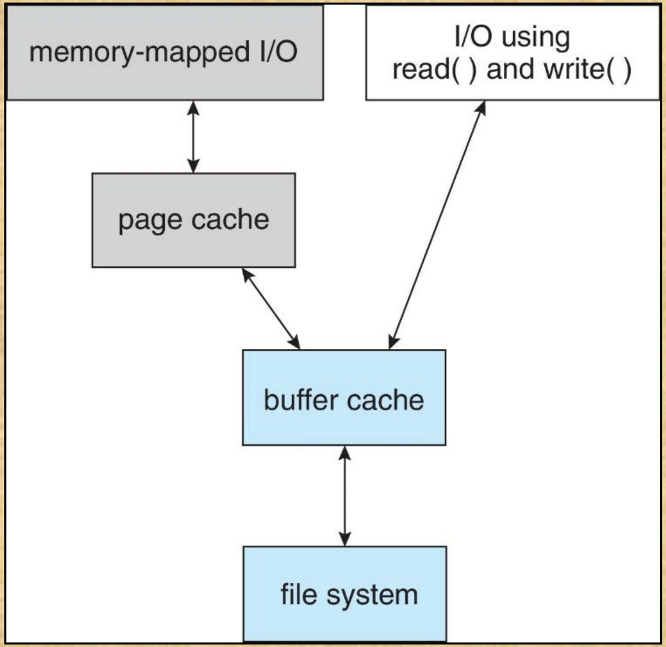
- Performance
  - Keeping data and metadata close together
  - **Buffer cache** – separate section of main memory for frequently used blocks
  - **Synchronous** writes sometimes requested by apps or needed by OS
    - No buffering / caching – writes must hit disk before acknowledgement
    - **Asynchronous** writes more common, buffer-able, faster
  - **Free-behind** and **read-ahead** – techniques to optimize sequential access
    - a) **free-behind technique** - free the memory of a block as soon as the next block is requested b) **read-ahead technique** - when a block is requested, read and cache several subsequent disk blocks to make use of spatial locality

http://www.cs.uni.edu/~fienup/courses/copy-of-operating-systems/lecture-notes/notes98f-17.lwp/odyframe.htm

Page Cache

- A **page cache** caches pages rather than disk blocks using virtual memory techniques and addresses
- Memory-mapped I/O uses a page cache
- Routine I/O through the file system uses the buffer (disk) cache
- This leads to the figure on next slide

I/O Without a Unified Buffer Cache



Unified Buffer Cache

- A **unified buffer cache** uses the same page cache to cache both memory-mapped pages and ordinary file system I/O to avoid **double caching**
- But which caches get priority, and what replacement algorithms to use?

I/O Using a Unified Buffer Cache