CSE2008: Operating Systems

L34 L35 & L36: File System Management

VIT-AP
UNIVERSITY

Dr. Subrata Tikadar
SCOPE, VIT-AP University

Recap

- Introductory Concepts
- Process Fundamentals
- IPC
- CPU Scheduling Algorithms
- Multithreading Concepts
- Synchronization
- Deadlock
- Memory Management
- Virtual Memory Concepts
- Mass-Storage Management

Outline

- File Concept
- Access Methods
- Directory Structure
- File-System Structure
- File-System Operations
- Directory Implementation
- Allocation Methods
- Free-Space Management
- Efficiency and Performance

File Concept

- System's Perspective:

  The OS provides a uniform logical view of stored information, i.e., the OS abstracts from the physical properties of its storage devices to define a logical storage unit, the **file**.

- User's Perspective:

  A file is the smallest allotment of logical secondary storage; that is, data cannot be written to secondary storage unless they are within a file.

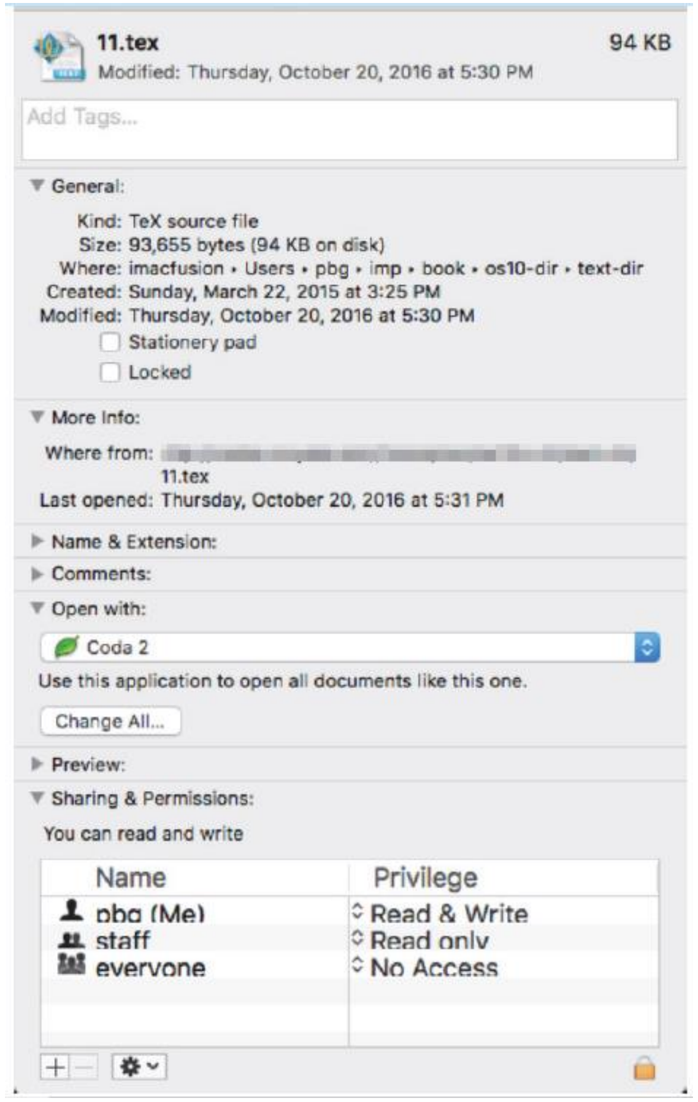
➢ In detail, we will discuss the concept of
  - File Attributes
  - File Operations
  - File Types
  - File Structure
  - Internal File Structure

File Concept

- File Attributes
  - Generally consist of
    - **Name.** The symbolic file name is the only information kept in humanreadable form.
    - **Identifier.** This unique tag, usually a number, identifies the file within the file system; it is the non-human-readable name for the file.
    - **Type.** This information is needed for systems that support different types of files.
    - **Location.** This information is a pointer to a device and to the location of the file on that device.
    - **Size.** The current size of the file (in bytes, words, or blocks) and possibly the maximum allowed size are included in this attribute.
    - **Protection.** Access-control information determines who can do reading, writing, executing, and so on.
    - **Timestamps and user identification.** This information may be kept for creation, last modification, and last use. These data can be useful for protection, security, and usage monitoring.

File Concept

- File Attributes (cont…)
  - Example - file info window on macOS


11.tex 94 KB
Modified: Thursday, October 20, 2016 at 5:30 PM

Add Tags…

General:

Kind: TeX source file
Size: 93,655 bytes (94 KB on disk)
Where: imacfusion ▸ Users ▸ pbg ▸ imp ▸ book ▸ os10-dir ▸ text-dir
Created: Sunday, March 22, 2015 at 3:25 PM
Modified: Thursday, October 20, 2016 at 5:30 PM
☐ Stationery pad
☐ Locked

More Info:

Where from: [redacted] 11.tex
Last opened: Thursday, October 20, 2016 at 5:31 PM

Name & Extension:

Comments:

Open with:

Coda 2
Use this application to open all documents like this one.
Change All…

Preview:

Sharing & Permissions:

You can read and write

Name	Privilege
pbg (Me)	Read & Write
staff	Read only
everyone	No Access

File Concept

- File Operations
  - Seven basic operations
    1. **Creating a file.**

       Two steps are necessary to create a file. First, space in the file system must be found for the file. Second, an entry for the new file must be made in a directory
    2. **Opening a file.**

       Rather than have all file operations specify a file name, causing the operating system to evaluate the name, check access permissions, and so on, all operations except create and delete require a file open() first. If successful, the open call returns a file handle that is used as an argument in the other calls
    3. **Writing a file.**

       To write a file, we make a system call specifying both the open file handle and the information to be written to the file. The system must keep a write pointer to the location in the file where the next write is to take place if it is sequential. The write pointer must be updated whenever a write occurs.

File Concept

- File Operations
  - Seven basic operations (cont…)

4. Reading a file.

To read from a file, we use a system call that specifies the file handle and where (in memory) the next block of the file should be put. Again, the system needs to keep a read pointer to the location in the file where the next read is to take place, if sequential. Once the read has taken place, the read pointer is updated. Because a process is usually either reading from or writing to a file, the current operation location can be kept as a per-process current-file-position pointer. Both the read and write operations use this same pointer, saving space and reducing system complexity.

5. Repositioning within a file.

The current-file-position pointer of the open file is repositioned to a given value. Repositioning within a file need not involve any actual I/O. This file operation is also known as a file seek.

File Concept

- File Operations
  - Seven basic operations (cont…)

6. Deleting a file.

To delete a file, we search the directory for the named file. Having found the associated directory entry, we release all file space, so that it can be reused by other files, and erase or mark as free the directory entry. Note that some systems allow hard links—multiple names (directory entries) for the same file. In this case the actual file contents is not deleted until the last link is deleted.

7. Truncating a file.

The user may want to erase the contents of a file but keep its attributes. Rather than forcing the user to delete the file and then recreate it, this function allows all attributes to remain unchanged—except for file length. The file can then be reset to length zero, and its file space can be released.

File Concept

- File Operations
  - Associated information examples (e.g., for opening a file)
    - **File pointer.**

      On systems that do not include a file offset as part of the read() and write() system calls, the system must track the last read– write location as a current-file-position pointer. This pointer is unique to each process operating on the file and therefore must be kept separate from

      the on-disk file attributes.
    - **File-open count.**

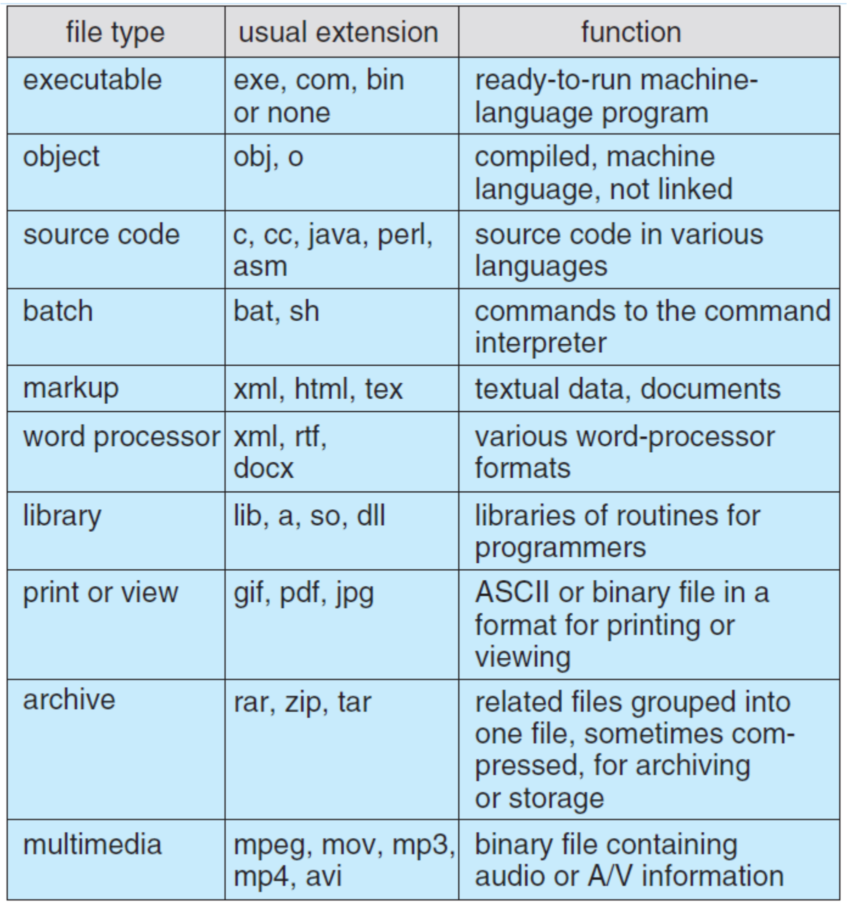
      As files are closed, the operating system must reuse its open-file table entries, or it could run out of space in the table. Multiple processes may have opened a file, and the system must wait for the last file to close before removing the open-file table entry. The file-open count tracks the number of opens and closes and reaches zero on the last close. The system can then remove the entry.
    - **Location of the file.**

      Most file operations require the system to read or write data within the file. The information needed to locate the file (wherever it is located, be it on mass storage, on a file server across the network, or on a RAM drive) is kept in memory so that the system does not have to read it from the directory structure for each operation.
    - **Access rights.**

      Each process opens a file in an access mode. This information is stored on the per-process table so the operating system can allow or deny subsequent I/O requests.

File Concept

- File Types



file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, perl, asm	source code in various languages
batch	bat, sh	commands to the command interpreter
markup	xml, html, tex	textual data, documents
word processor	xml, rtf, docx	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	gif, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	rar, zip, tar	related files grouped into one file, sometimes com-pressed, for archiving or storage
multimedia	mpeg, mov, mp3, mp4, avi	binary file containing audio or A/V information

File Concept

- File Structure
  - File types can also be used to indicate the internal structure of the file
    - Source and object files have structures that match the expectations of the programs that read them.
    - Certain files must conform to a required structure that is understood by the operating system.

    Example:

    The operating system requires that an executable file have a specific structure so that it can determine where in memory to load the file and what the location of the first instruction is. Some operating systems extend this idea into a set of system-supported file structures, with sets of special operations for manipulating files with those structures.

File Concept

- Internal File Structure
  - Internally, locating an offset within a file can be complicated for the operating system. Disk systems typically have a well-defined block size determined by the size of a sector. All disk I/O is performed in units of one block (physical record), and all blocks are the same size. It is unlikely that the physical record size will exactly match the length of the desired logical record. Logical records may even vary in length. Packing a number of logical records into physical blocks is a common solution to this problem.

    Example: The UNIX operating system defines all files to be simply streams of bytes. Each byte is individually addressable by its offset from the beginning (or end) of the file. In this case, the logical record size is 1 byte. The file system automatically packs and unpacks bytes into physical disk blocks— say, 512 bytes per block—as necessary.

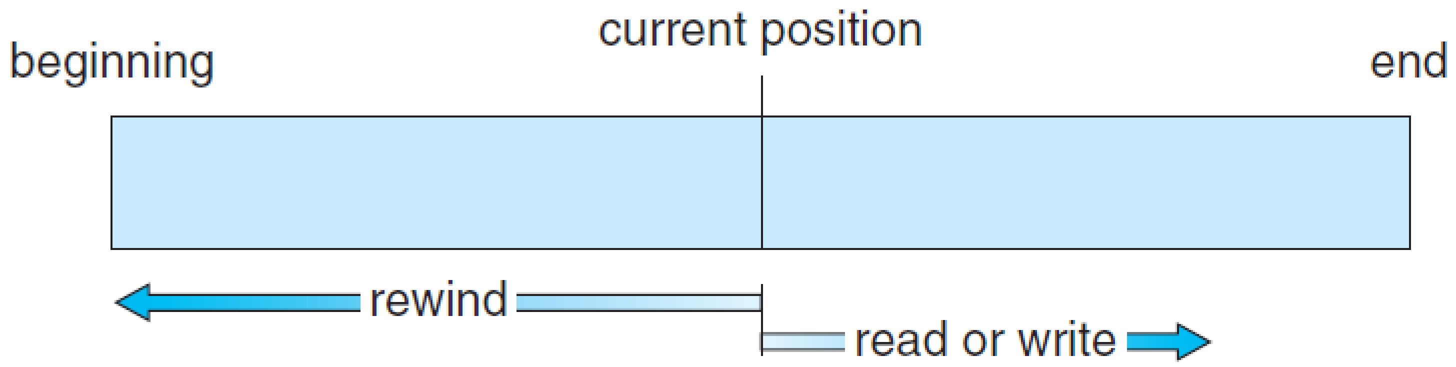
Access Methods

- Sequential Access
- Direct Access
- Other Access Methods

Access Methods

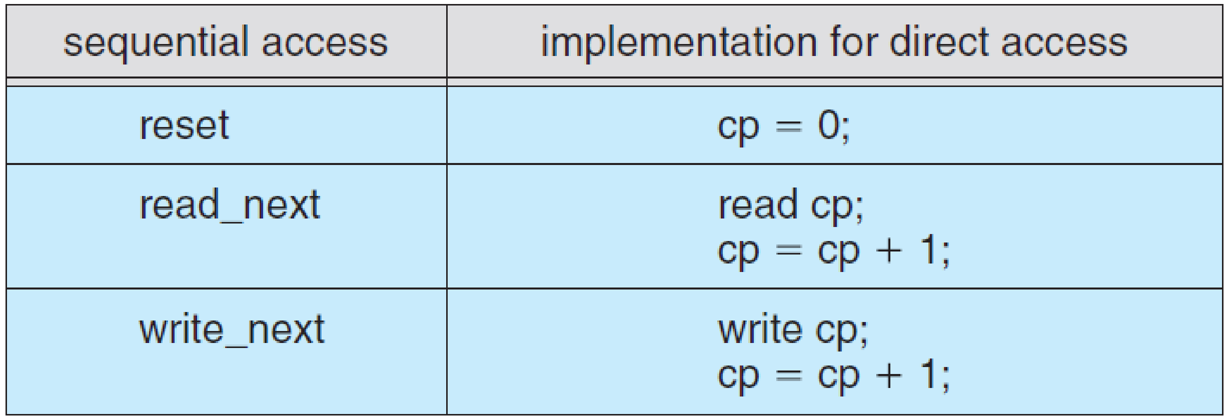
- Sequential Access
  - Information in the file is processed in order, one record after the other.
  - By far the most common

    Example: Editors and compilers usually access files in this fashion.



Access Methods

- Direct Access
  - A file is made up of fixed-length logical records that allow programs to read and write records rapidly in no particular order.
  - The file is viewed as a numbered sequence of blocks or records – we may read block 14, then read block 53, and then write block 7; there are no restrictions on the order of reading or writing for a direct-access file.



sequential access	implementation for direct access
reset	cp = 0;

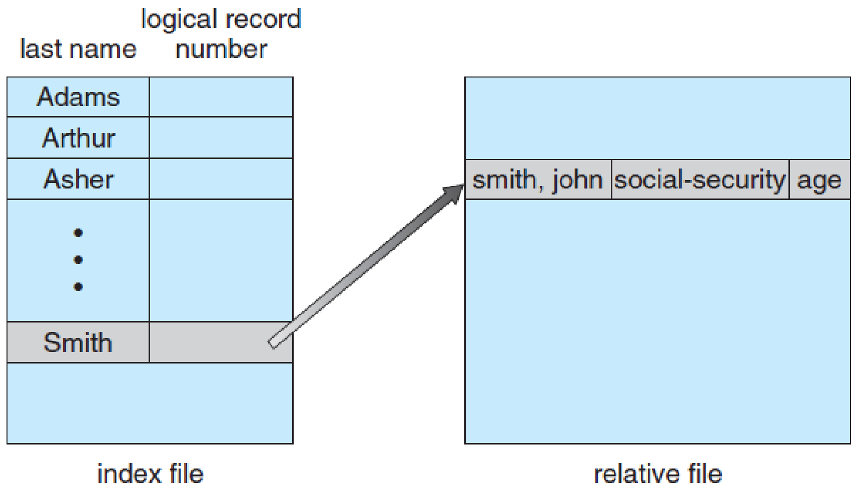
read_next	read cp;
	cp = cp + 1;

write_next	write cp;
	cp = cp + 1;

Access Methods

- Other Access Methods
  - Can be built on top of a direct-access method.
  - Generally involve the construction of an index for the file
  - The index, like an index in the back of a book, contains pointers to the various blocks.

➢ To find a record in the file, we first search the index and then use the pointer to access the file directly and to find the desired record.



Directory Structure

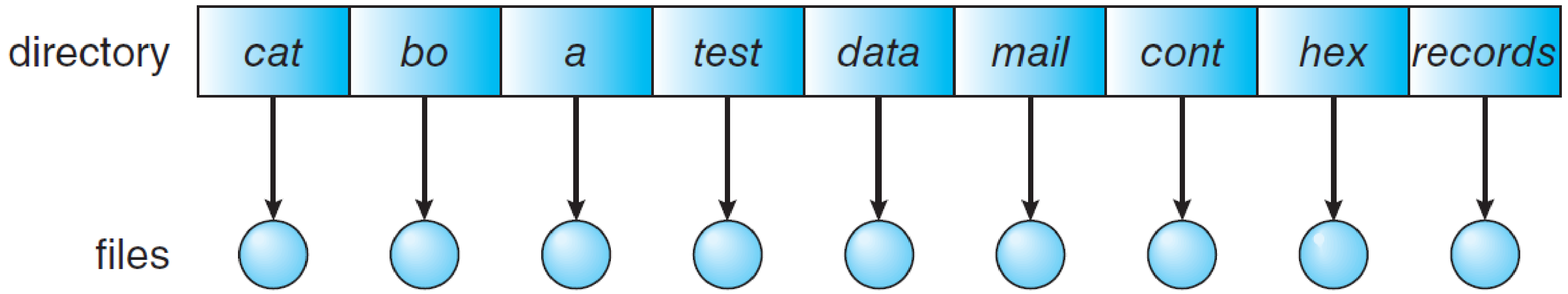
- List of operations that are to be performed on a directory
  - Search for a file
  - Create a file
  - Delete a file
  - List a directory
  - Rename a file
  - Traverse the file system

Directory Structure

- Common schemes for defining the logical structure of a directory
  - Single-Level Directory
  - Two-Level Directory
  - Tree-Structured Directories
  - Acyclic-Graph Directories
  - General Graph Directory

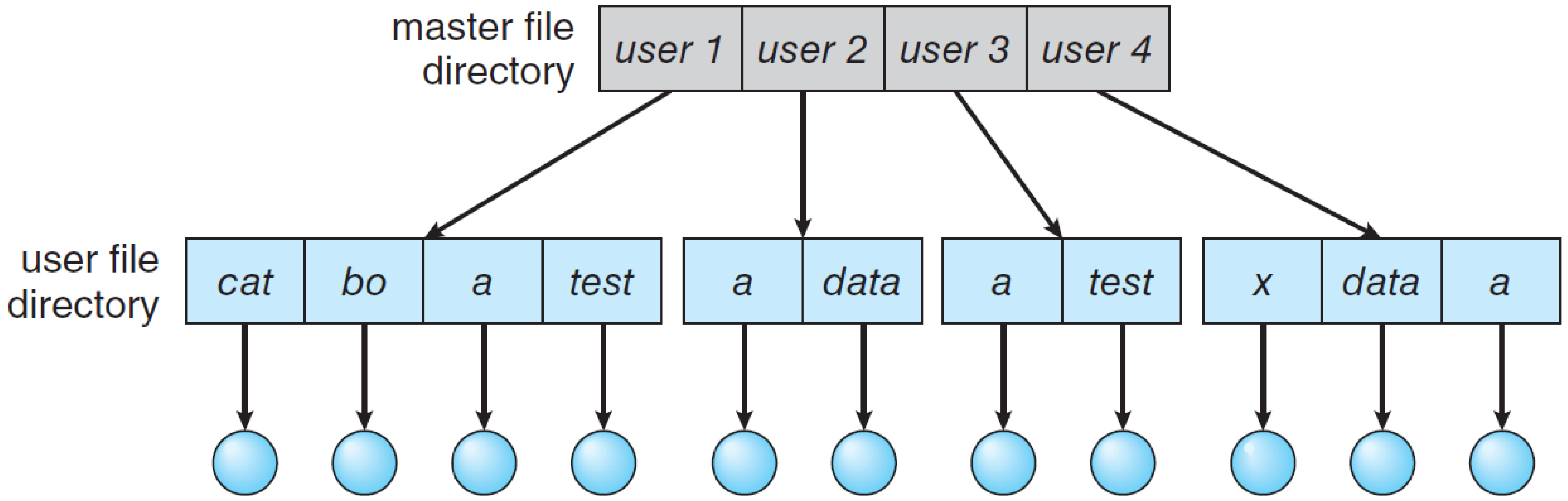
Directory Structure

- Common schemes for defining the logical structure of a directory
  - Single-Level Directory



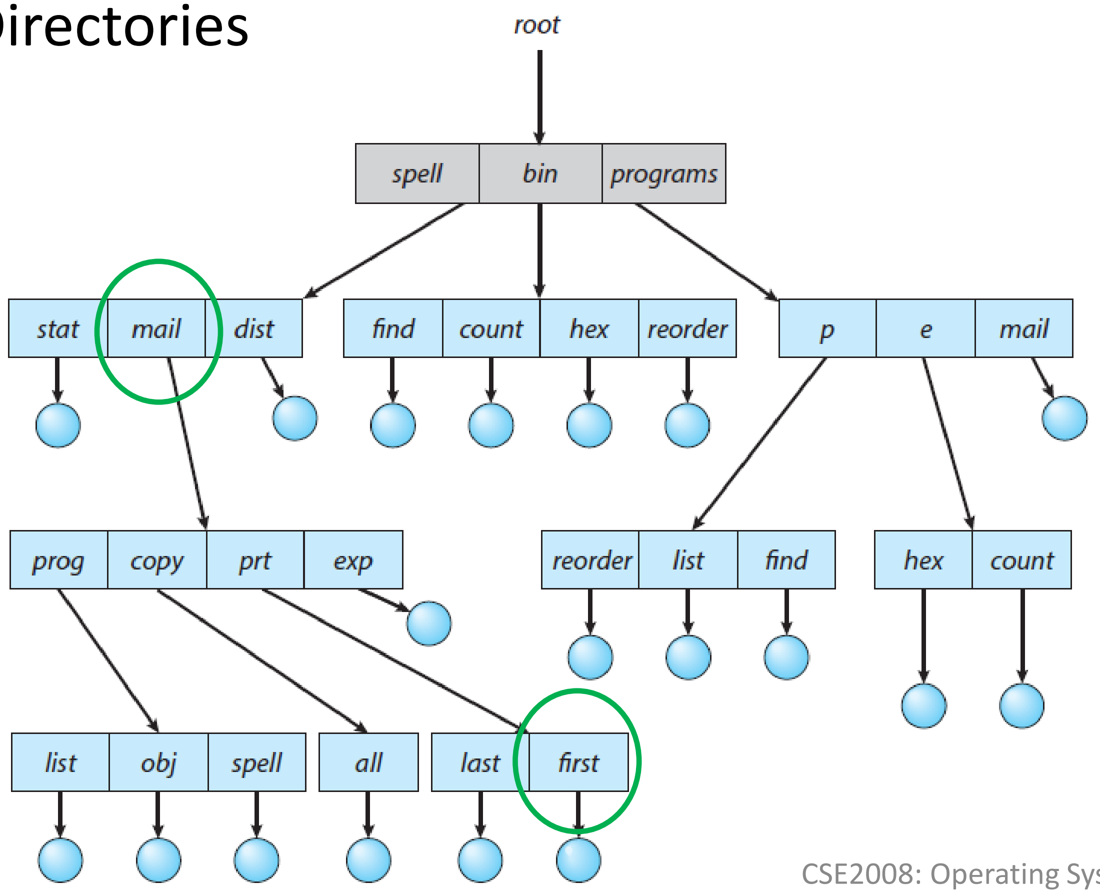
Directory Structure

- Common schemes for defining the logical structure of a directory
  - Two-Level Directory



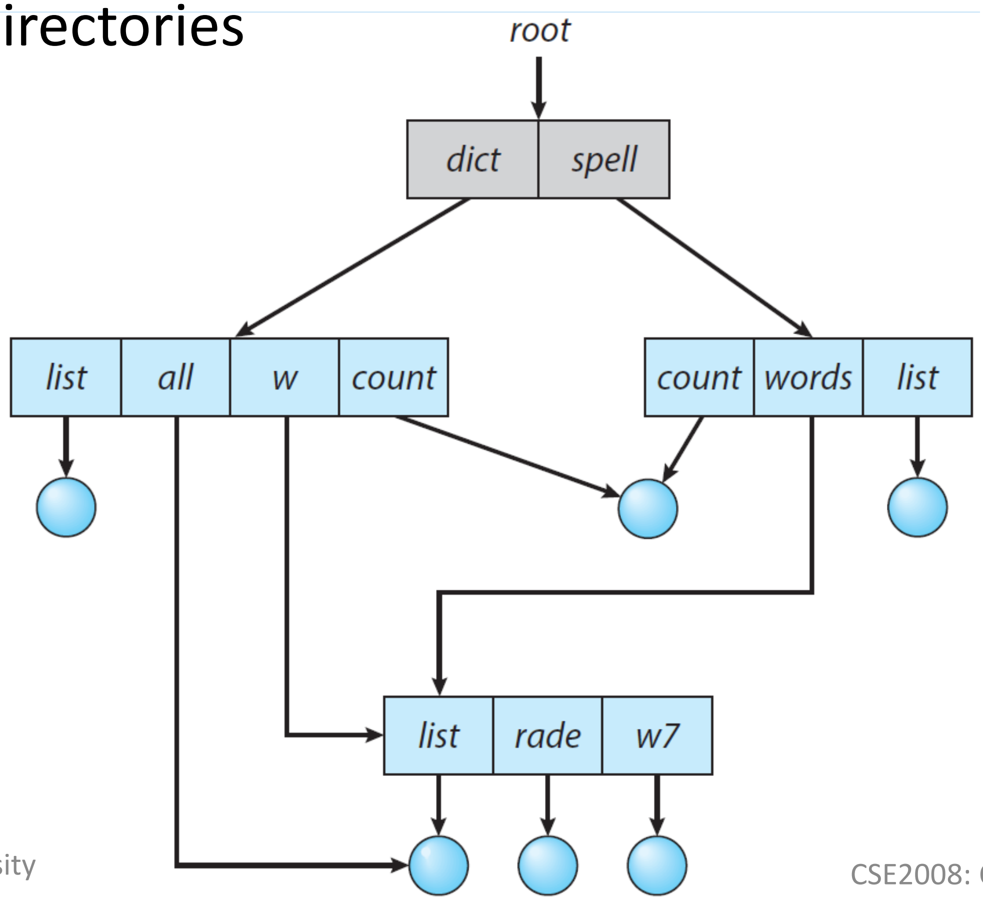
Directory Structure

- Common schemes for defining the logical structure of a directory
  - Tree-Structured Directories



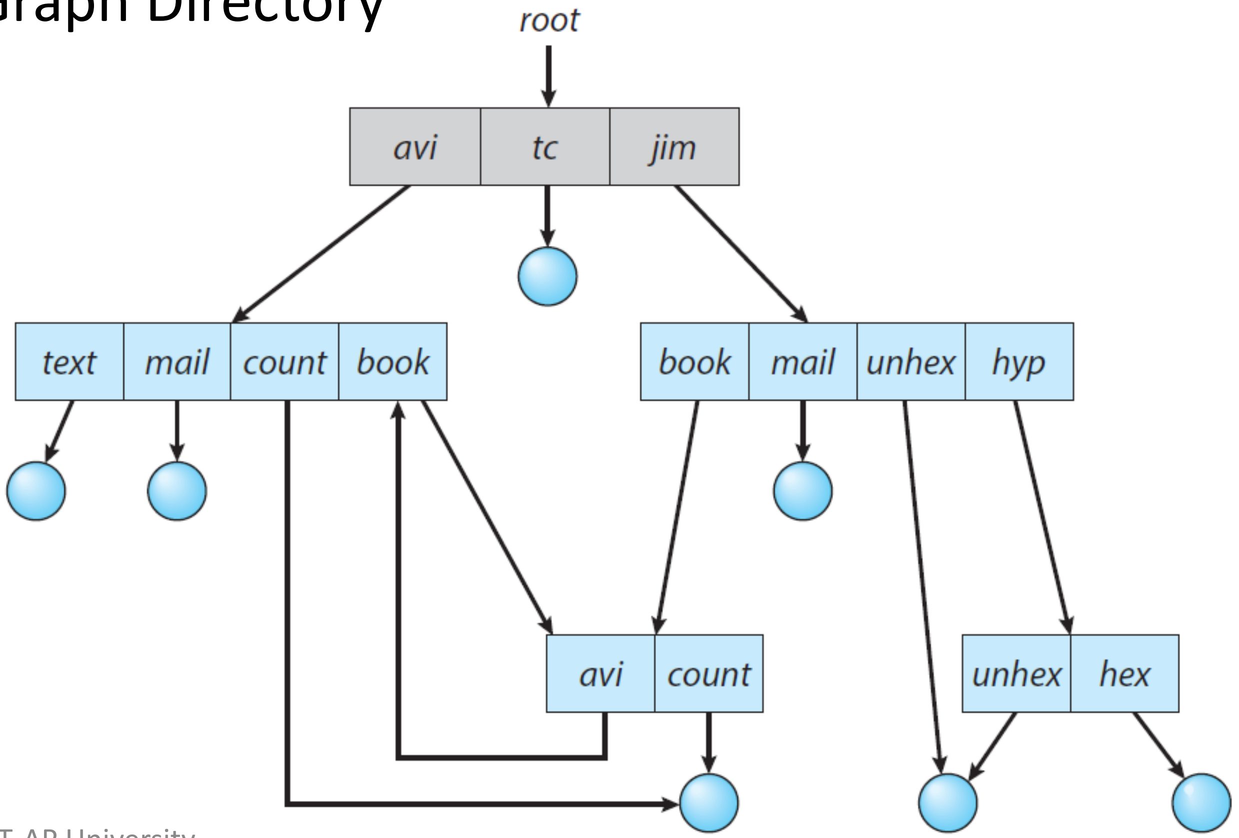
Directory Structure

- Common schemes for defining the logical structure of a directory
  - Acyclic-Graph Directories



Directory Structure

- Common schemes for defining the logical structure of a directory
  - General Graph Directory



File-System Structure

<Next Session>

<u>QUIZ – 12/05/2022</u>

Syllabus: All topics that were covered neither in CAT 1 nor in CAT 2.

Mode: Online over MS Teams

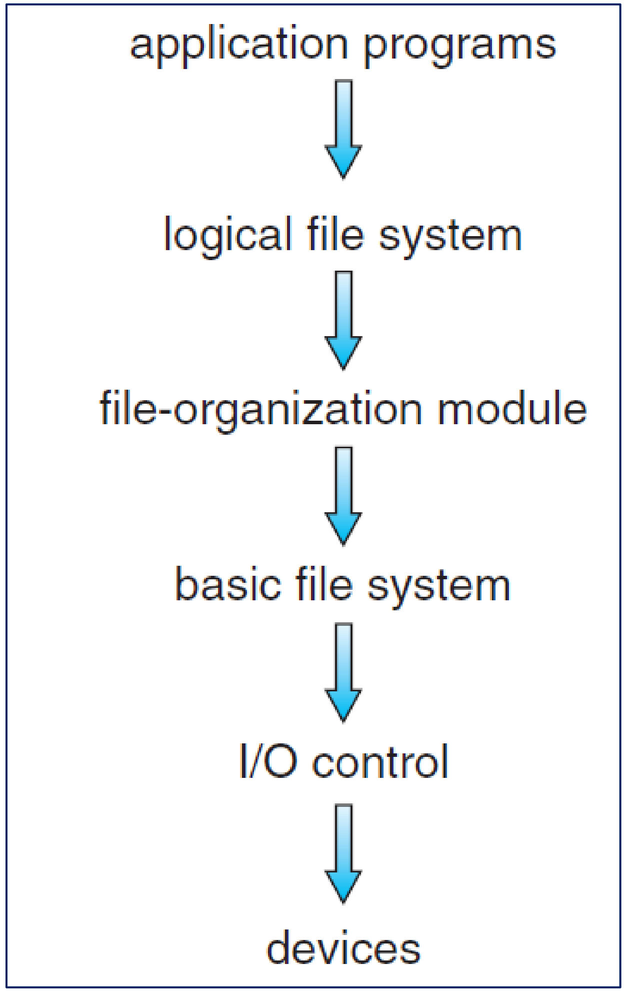
Question Type: Objective (MCQ, matching, fill in the blanks, true/false … … …)

File-System Structure

- Disks provide most of the secondary storage on which file systems are maintained. Two characteristics make them convenient for this purpose:

  1. A disk can be rewritten in place; it is possible to read a block from the disk, modify the block, and write it back into the same block.

  2. A disk can access directly any block of information it contains. Thus, it is simple to access any file either sequentially or randomly, and switching from one file to another requires the drive moving the read–write heads and waiting for the media to rotate.



File-System Operations

- Overview
  - On-storage structures
  - In-memory structures

File-System Operations

- Overview
  - On-storage structures
    - A boot control block (per volume) can contain information needed by the system to boot an operating system from that volume. If the disk does not contain an operating system, this block can be empty. It is typically the first block of a volume. In UFS, it is called the boot block. In NTFS, it is the partition boot sector.
    - A volume control block (per volume) contains volume details, such as the number of blocks in the volume, the size of the blocks, a free-block count and free-block pointers, and a free-FCB count and FCB pointers. In UFS, this is called a superblock. In NTFS, it is stored in the master fil table.
    - A directory structure (per file system) is used to organize the files. In UFS, this includes file names and associated inode numbers. In NTFS, it is stored in the master file table.
    - A per-file FCB contains many details about the file. It has a unique identifier number to allow association with a directory entry. In NTFS, this information is actually stored within the master file table, which uses a relational database structure, with a row per file.

File-System Operations

- Overview
  - In-memory structures
    - An in-memory mount table contains information about each mounted volume.
    - An in-memory directory-structure cache holds the directory information of recently accessed directories. (For directories at which volumes are mounted, it can contain a pointer to the volume table.)
    - The system-wide open-file table contains a copy of the FCB of each open file, as well as other information.
    - The per-process open-file table contains pointers to the appropriate entries in the system-wide open-file table, as well as other information, for all files the process has open.
    - Buffers hold file-system blocks when they are being read from or written to a file system.

File-System Operations

- Usage
  - The open() call passes a file name to the logical file system
  - The open() system call first searches the system-wide open-file table to see if the file is already in use by another process

    If opened

    A per-process open-file table entry is created pointing to the existing system-wide open-file table

    Else

    The directory structure is searched for the given file name

    [Once the file is found, the FCB is copied into a system-wide open-file table in memory]
  - An entry is made in the per-process open-file table, with a pointer to the entry in the system-wide open-file table and some other fields
  - When a process closes the file:

    The per-process table entry is removed, and the system-wide entry's open count is decremented. When all users that have opened the file close it, any updated metadata are copied back to the disk-based directory structure, and the system-wide open-file table entry is removed.

Directory Implementation

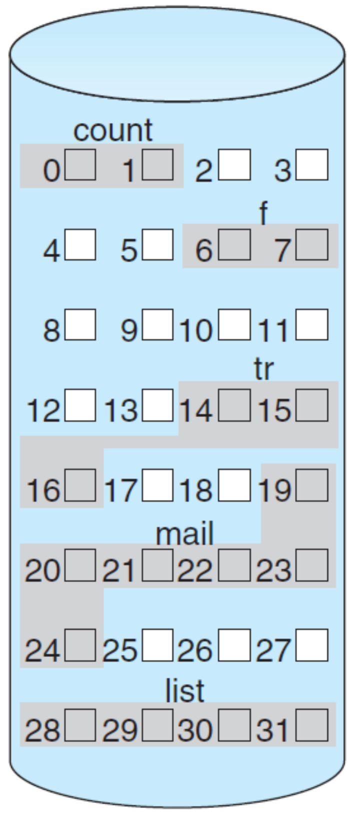
- Linear List
  - Simple to program but time-consuming to execute.
  - Finding a file requires a linear search
- Hash Table
  - Collisions
  - Generally fixed size and the dependence of the hash function on that size

Allocation Methods

- Contiguous Allocation
- Linked Allocation
- Indexed Allocation

Allocation Methods

- Contiguous Allocation

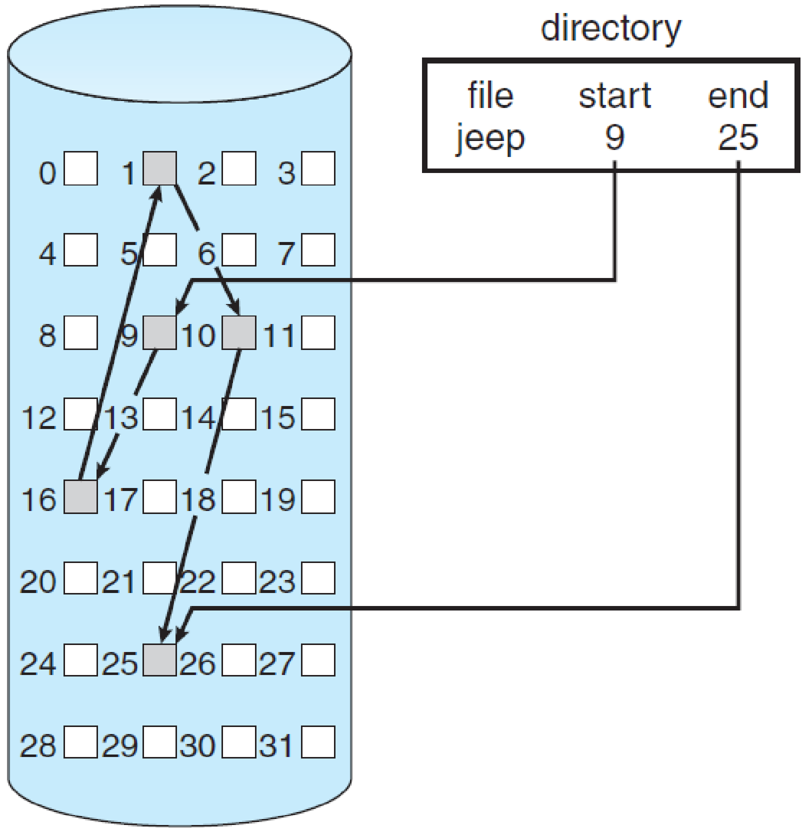


directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

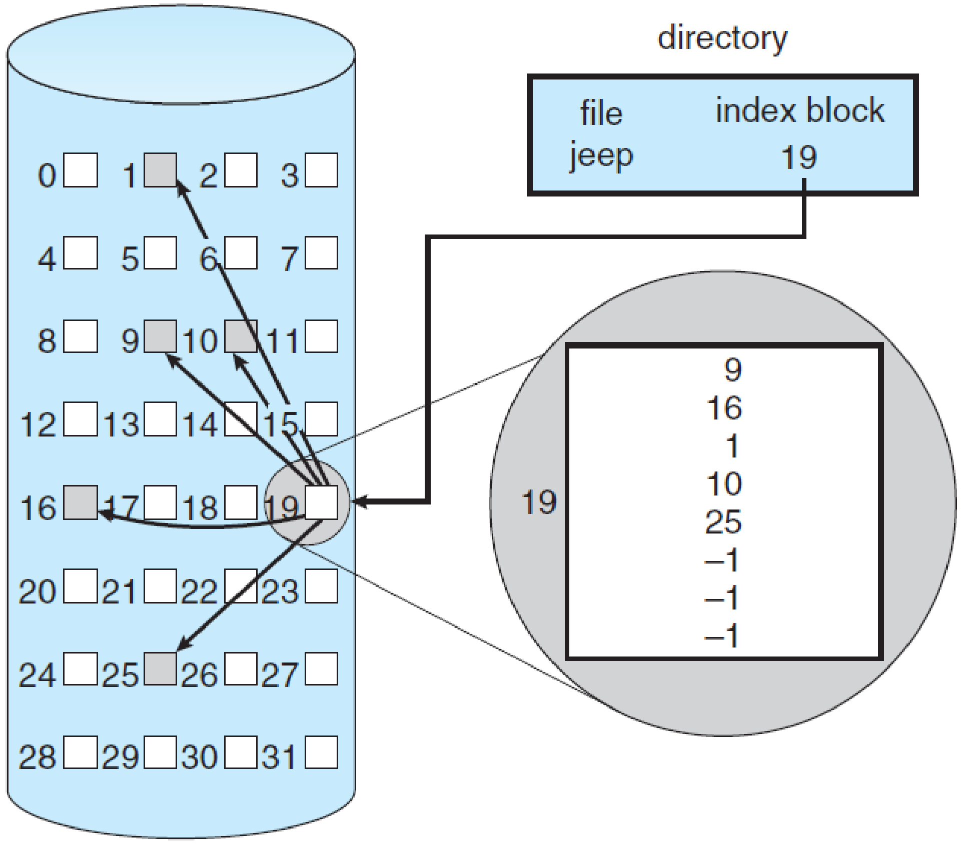
Allocation Methods

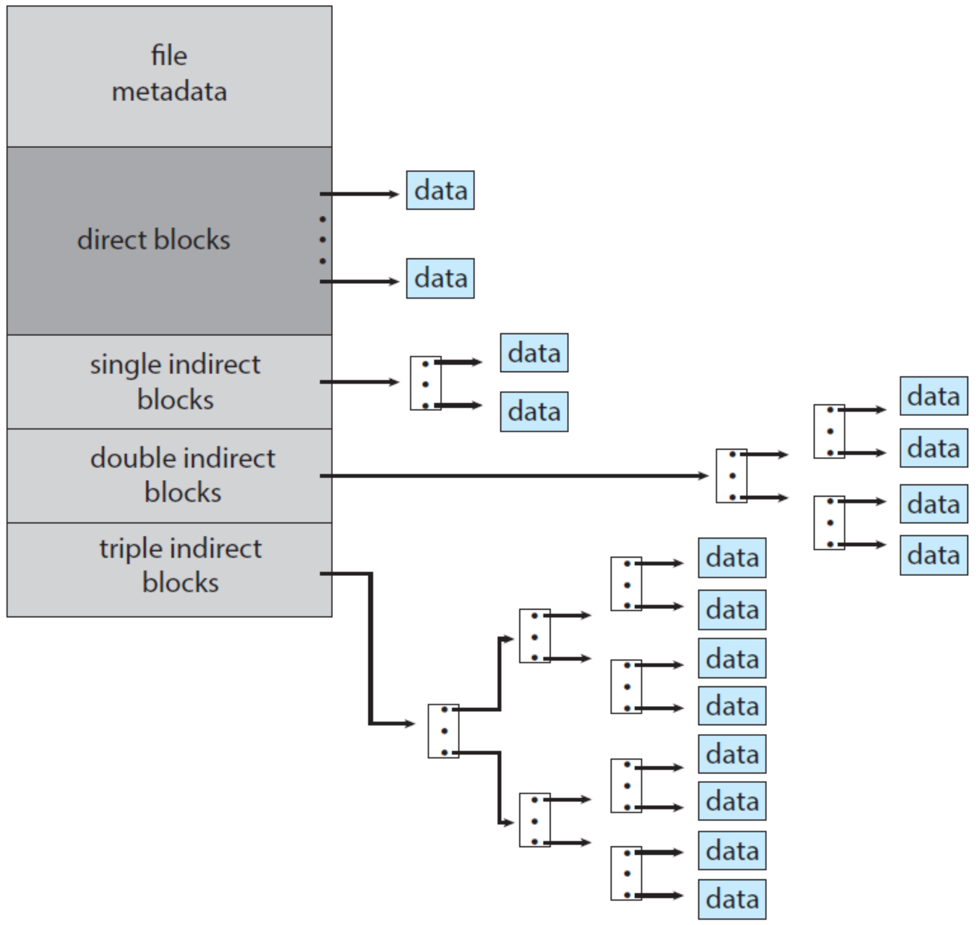
- Linked Allocation



Allocation Methods

- Indexed Allocation





Free-Space Management

- Bit Vector
- Linked List
- Grouping
- Counting
- Space Maps
- TRIMing Unused Blocks

Free-Space Management

- Bit Vector
  - Each block is represented by 1 bit. If the block is free, the bit is 1; if the block is allocated, the bit is 0.

    Example:

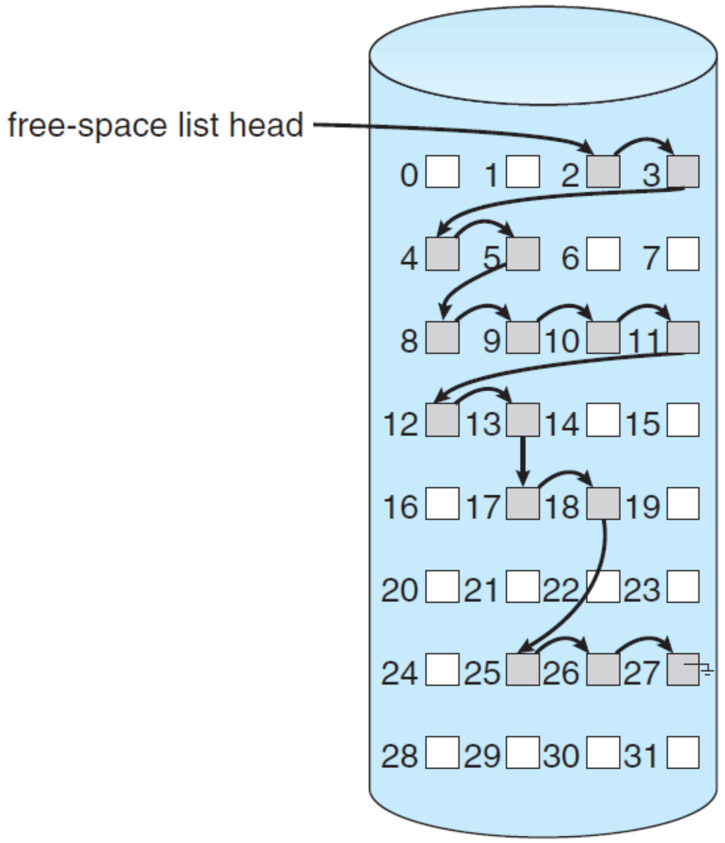
    Consider a disk where blocks 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, and 27 are free and the rest of the blocks are allocated.

    In this case, the free-space bitmap would be

    001111001111110001100000011100000 ...

Free-Space Management

- Linked List



Free-Space Management

- Grouping
  - A modification of the 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.

Free-Space Management

- Counting
    - Rather than keeping a list of n free block addresses, we can keep the address of the first free block and the number (n) of free contiguous blocks that follow the first block. Each entry in the free-space list then consists of a device address and a count. Although each entry requires more space than would a simple disk address, the overall list is shorter, as long as the count is generally greater than 1.

Free-Space Management

- Space Maps
  - A log of all block activity (allocating and freeing), in time order, in counting format. When ZFS decides to allocate or free space from a metaslab, it loads the associated space map into memory in a balanced-tree structure (for very efficient operation), indexed by offset, and replays the log into that structure. The in-memory space map is then an accurate representation of the allocated and free space in the metaslab.

Free-Space Management

- TRIMing Unused Blocks
  - Keeps storage space available for writing.
  - Without such a capability, the storage device gets full and needs garbage collection and block erasure, leading to decreases in storage I/O write performance (known as “awrite cliff”).

Efficiency and Performance

- Efficiency and Performance
  - The efficient use of storage device space depends heavily on the allocation and directory algorithms in use.
  - Even after the basic file-system algorithms have been selected, we can still improve performance in several ways, such as buffer cache, page cache, and so on.

Reference

- Abraham Silberschatz , Peter B. Galvin, Greg Gagne, “Operating System Concepts”, Addison Wesley, 10th edition, 2018
  - Chapter 13: Section 13.1 – 13.3
  - Chapter 14: Section 14.1 – 14.6

Next

- Security & Protection

Thank You