File System Implementation

Take a shot every time you see 'block'

1 File-System Structure

- File Structure
  - Logical Storage Unit
  - Collection of related information
- File system resides on secondary storage (disks)
  - Provided user interface to storage, mapping logical to physical
  - Provides efficient and convenient access to disk by allowing data to be stored, located retrived easily
- Disk provides in-place rewrite and random access
  - I/O transfers performed in **blocks** of **sectors** (usually 512 bytes)
- File control block - storage structure consisting of information about a file
- Device driver controls the physical device
- File system organized into layers

2 Layered File System

Application Programs

⇓

Logical File System

⇓

File Organization Module

⇓

Basic File System

⇓

I/O System

⇓

Devices

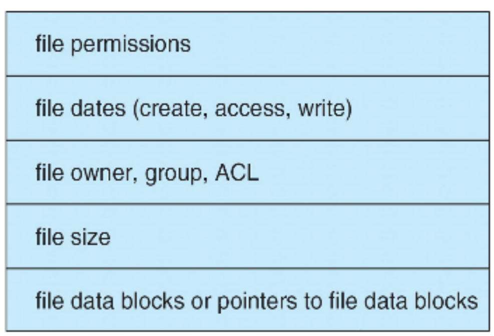
3 File System Layers

- Device drivers manage I/O devices at the I/O control layer
- Basic file system given command like "retrieve block 123" translates to device driver
- Also manages memory buffers and caches (allocation, freeing, replacement)
  - Buffers hold data in transit
  - Caches hold frequently used data
- File organisation module understands files, logical address, and physical blocks
  - Translates logical block # to physical block #
  - Manages free space, disk allocation

- Logical file system manages metadata information
  - Translates file name into file number, file handle, location by maintaining file control blocks
  - Directory management
  - Protection
- Layering useful for reducing complexity and redundancy, but adds overhead and can decrease performance. Translates file name into file number, file handle, location by maintaining file control blocks
  - Logical layers can be implemented by any coding method according to OS designer

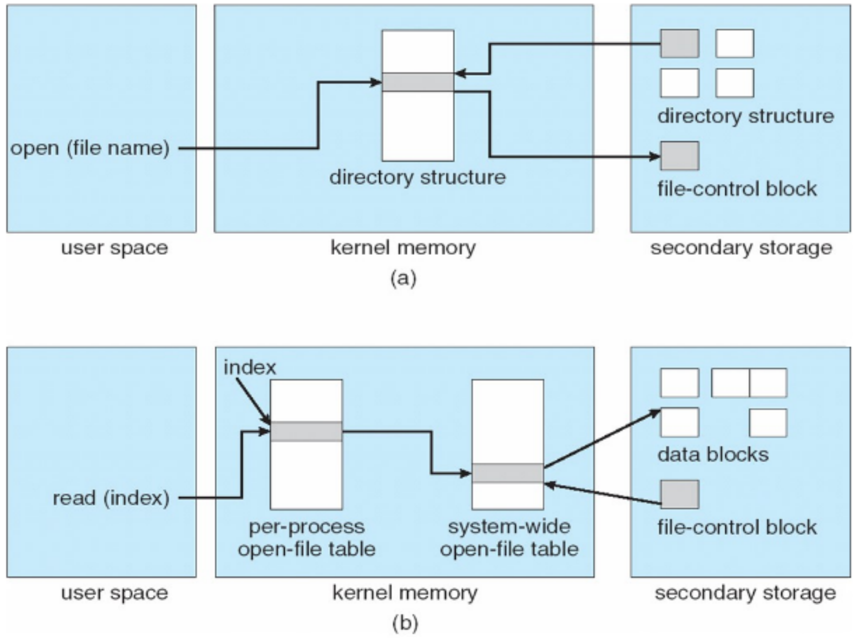
4 File-System Implementation

- We have system calls at the API level, but how do we implement their functions
  - On disk and in memory structures
- Boot control block contains info needed by system to boot OS from that volume
  - Needed if volume contains OS, usually the first block of the volume
- Volume control block (superblock, master file table) contains volume details
  - Total # of blocks, # of free blocks, block size, free block pointers or array
- Directory structure organizes the files
  - Names and inode numbers, master file table
- Per-file **File Control Block (FCB)** contains many details about the file
  - Inode number, permissions, size, dates
  - NTFS stores into a master file table using relational DB structures



5 In-Memory File system structures

- Mount table storing file system mounts, mount points, file system types
- The following figure illustrates the necessary file system structures provided by the operating systems
- Figure (a) refers to opening a file
- Figure (b) refers to reading a file
- Plus buffers hold data blocks from secondary storage
- Open returns a file handle for subsequent use
- Data from read eventually copied to specified user process memory address

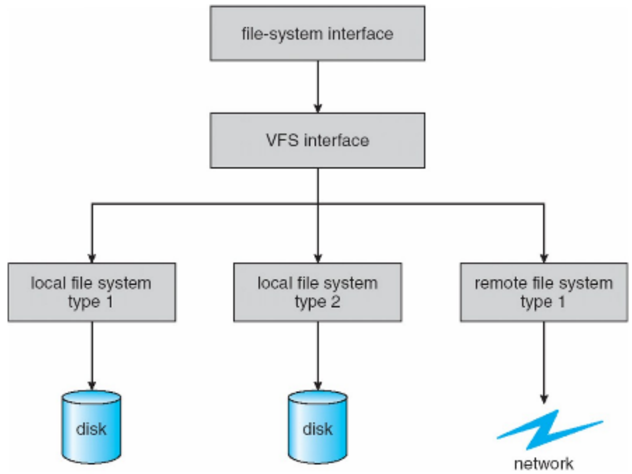


6 Partitions and Mounting

- Partition can be a volume containing a file system ("cooked") or **raw** - just a sequence of blocks with no file system
- Boot block can point to boot volume or boot loader set of blocks that contain enough code to know how to load the kernel from the file system, or a boot management program for multi-os booting
- Root partition contains the OS, other partitions can hold other OSes, other file systems, or be raw
  - Mounted at boot time
  - Other partitions can mount automatically or manually
- At mount time, file system consistency is checked if the metadata is correct
  - If not, fix it, try again
  - If yes, add to mount table, allow access

7 Virtual File systems

- Virtual File Systems (VFS) on Unix provide an object-oriented way of implementing file systems
- VFS allows the same system call interface (the API) to be used for different types of file systems
  - Separates file-system generic operations from implementation details
  - Implementation can be one of many file systems types, or network file system
    - Implements vnodes which hold inodes or network file details
  - Then dispatches operation to appropriate file system implementation routines
- The API is to the VFS interface, rather than any specific type of file system

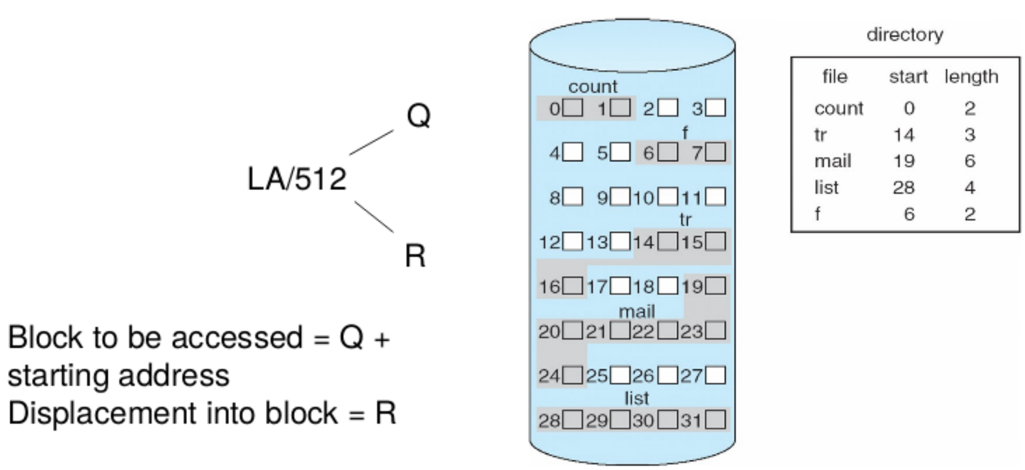


8 Directory Implementation

- Linear list of file names with pointer to the data blocks
  - Simple to program
  - Time consuming to execute
    - Linear search time
    - Could keep ordered alphabetically via linked list or use B+ tree
- Hash table - linear list with hash data structure
  - Decreases directory search time
  - Collisions - situations where two file names hash to the same location

9 Allocation Methods - Contiguous

- An allocation method refers to how disk blocks are allocates for files
- Contiguous allocation - each file occupies set of contiguous blocks
  - Best performance in most cases
  - Simple - only starting location (block #) and length (number of blocks) are required
  - Problems include finding space for file, knowing file size, external fragmentqation, need for compaction off-line (downtime) or on-line
- Mapping from logical to physical



10 Extent-Based systems

- Many newer file systems (i.e., Veritas File System) use a modified contiguous allocation scheme
- Extent-based file systems allocate disk blocks in extents
- An extent is a contiguous chunk of blocks
  - Extents are allocated for file allocation
  - A file consists of one or more extents

11 Allocation methods - linked

Linked allocation - each file a linked list of blocks

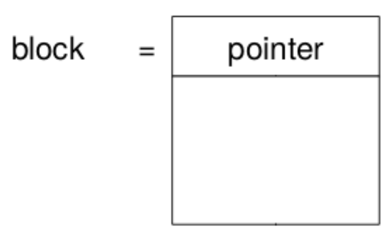
- File ends at nil pointer
- No external fragmentation
- Each block contains pointer to next block
- No compaction, external fragmentation
- Free space management system called when new block needed
- Improve efficiency by clustering blocks into groups but increases internal fragmentation
- Reliability can be a problem
- Locating a block can take many I/Os and disk seeks

FAT (File Allocation Table) variation - CHONKY AF

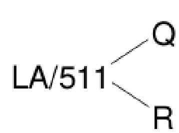
- Beginning of volume has table, indexed by block number
- Much like a linked list, but faster on disk and cacheable
- New block allocation simple

12 Linked allocation

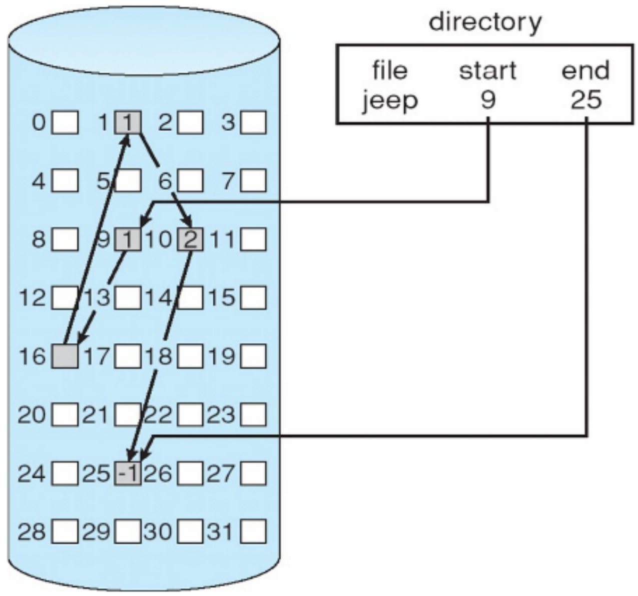
Each file is a linked list of disk blocks: blocks may be scattered anywhere on the disk



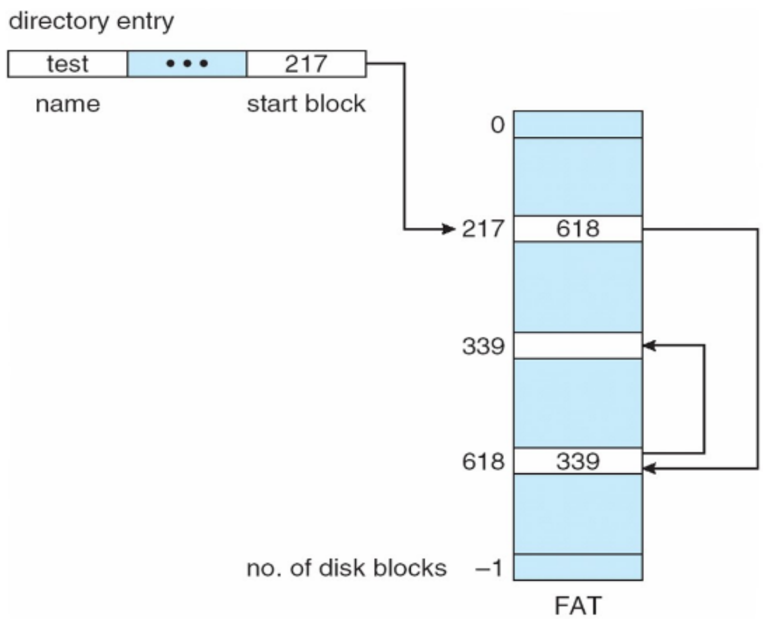
Mapping



Block to be accessed is the Qth block in the linked chain of blocks representing the file
Displacement into block = $R + 1$



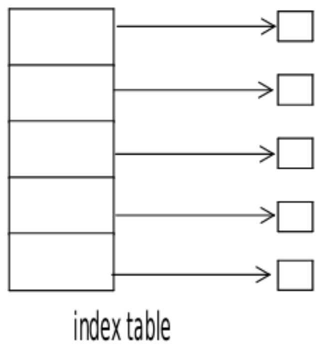
13 File-Allocation Table

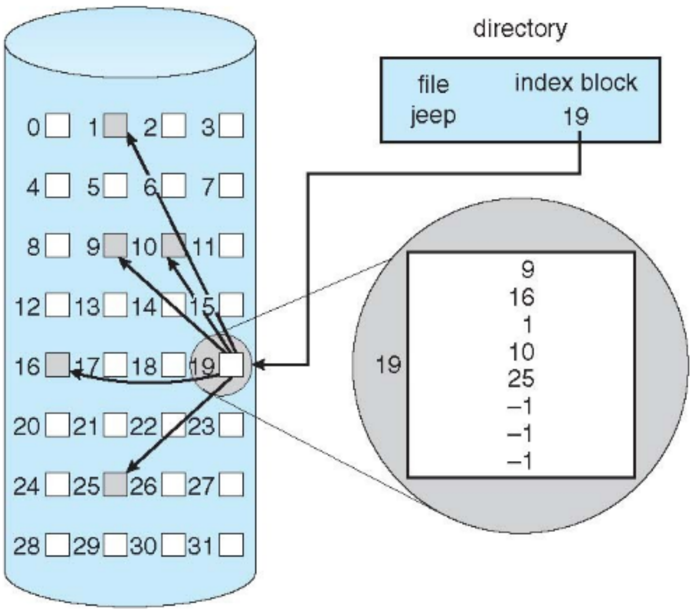


14 Allocation Methods - Indexed

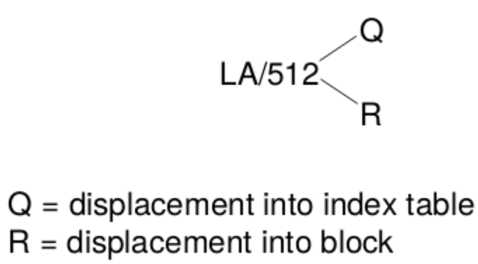
Indexed allocation - each file has its own index block(s) of pointers to its data blocks

Logical view





- Need index table
- Random access
- Dynamic access without external fragmentation, but have overhead of index block
- Mapping from logical to physical in a file of maximum size of 256K bytes and block size of 512 bytes. We need only 1 block for index table



- Mapping from logical to physical in a file of unbounded length (block size of 512 words)
- Linked scheme - Link blocks of index table (no limit on size)

$LA / (512 \times 511)$ → Q_1, R_1

Q_1 = block of index table
R_1 is used as follows:

$R_1 / 512$ → Q_2, R_2

Q_2 = displacement into block of index table
R_2 displacement into block of file:

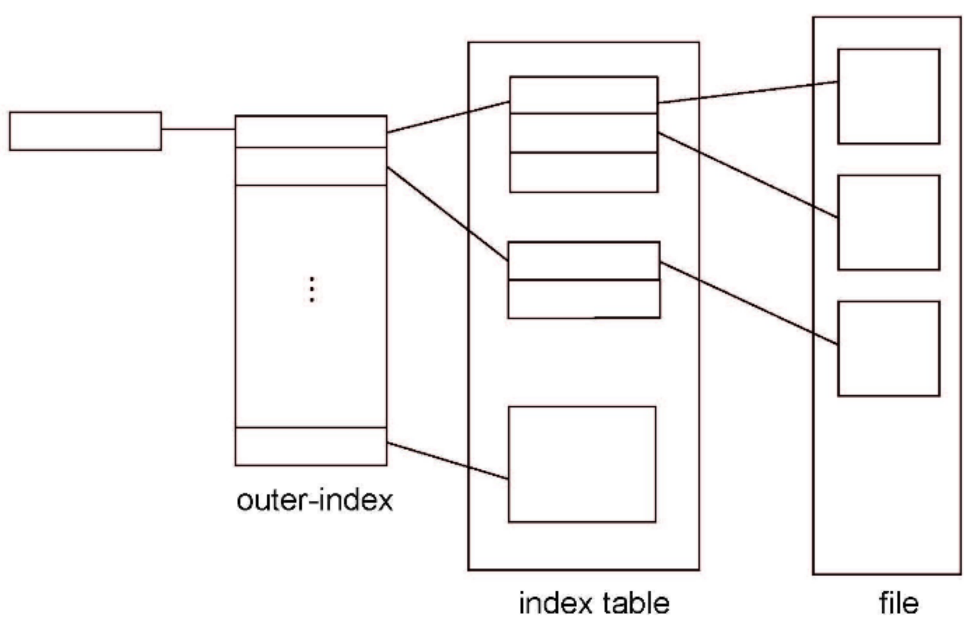
- Two-level index (4K blocks could store 1,024 four-byte pointers in outer index → 1,048,567 data blocks and file size of up to 4GB)

$LA / (512 \times 512)$ → Q_1, R_1

Q_1 = displacement into outer-index
R_1 is used as follows:

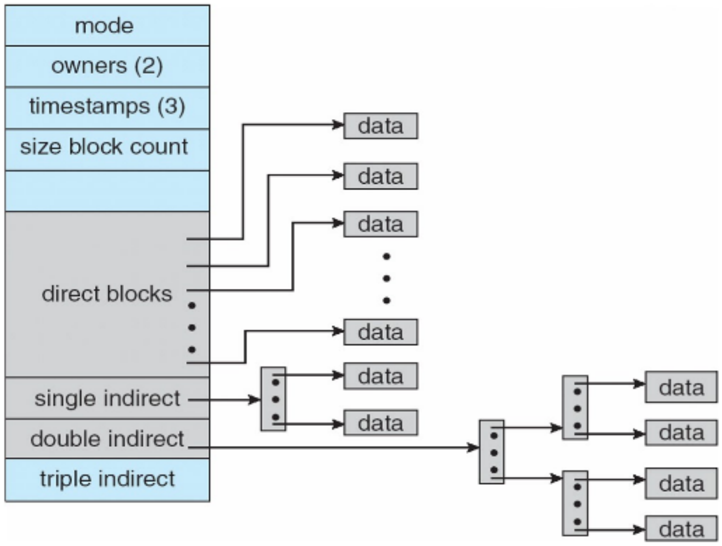
$R_1 / 512$ → Q_2, R_2

Q_2 = displacement into block of index table
R_2 displacement into block of file:



15 Combined Scheme: UNIX UFS

4K bytes per block, 32-bit addresses



More index blocks than can be addressed with 32-bit file pointer

16 Performance

- Best method depends on file access type
  - Contiguous great for sequential and random
- 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
- Adding instructions to the execution path to save one disk I/O is reasonable

17 Free space management

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

bit[*i*] = 1 ⇒ block[*i*] free
0 ⇒ block[*i*] occupied

Block number calculation

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

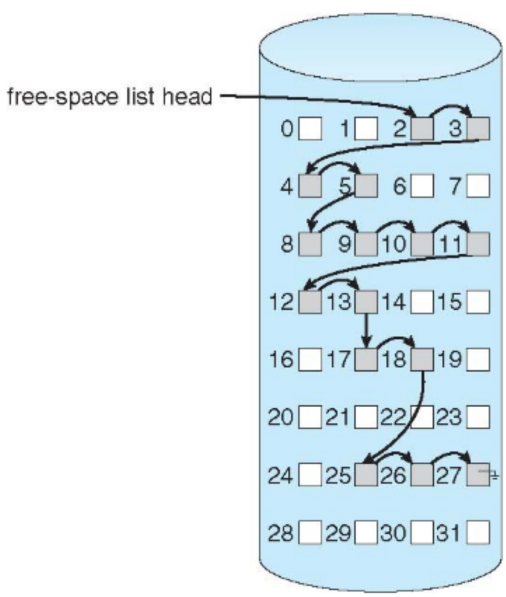
CPUs have instructions to return offset within word of first “1” bit

- Bit map requires extra space
- Easy to get contiguous files

18 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 # of free blocks recorded)



19 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 data structures

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
- Reads slower than writes

20 Recovery

- Consistency checking - compares data in directory structure with data blocks on disk, and tries to fix inconsistencies. Can be slow and sometimes fails
- Use system programs to back up data from disk to another storage device (magnetic tape, other magnetic disk, optical)
- Recover lost file or disk by restoring data from backup

21 Log structured file systems

- Log structured (or Journaling) file systems record each metadata update to the file system as a transaction
- All transactions are written to a log
  - A transaction is considered committed once it is written to the log (sequentially)
  - Sometimes to a separate device or section of disk
  - However, the file system may not be updated
- The transactions in the log are asynchronously written to the file system structures
  - When the file system strucrures are modified, the transaction is removed from the log
- If the file system crashes, all remaining transactions in the log must still be performed
- Faster recovery from crash, removes chance of inconsistency of metadata