

Module 11: File-System Implementation

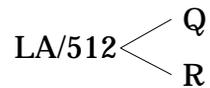
- File-System Structure
- Allocation Methods
- Free-Space Management
- Directory Implementation
- Efficiency and Performance
- Recovery

File-System Structure

- File structure
 - Logical storage unit
 - Collection of related information
- File system resides on secondary storage (disks).
- File system organized into layers.
- *File control block* – storage structure consisting of information about a file.

Contiguous Allocation

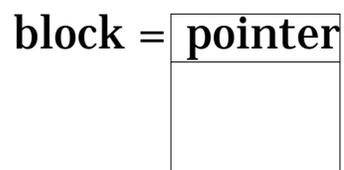
- Each file occupies a set of contiguous blocks on the disk.
- Simple – only starting location (block #) and length (number of blocks) are required.
- Random access.
- Wasteful of space (dynamic storage-allocation problem).
- Files cannot grow.
- Mapping from logical to physical.



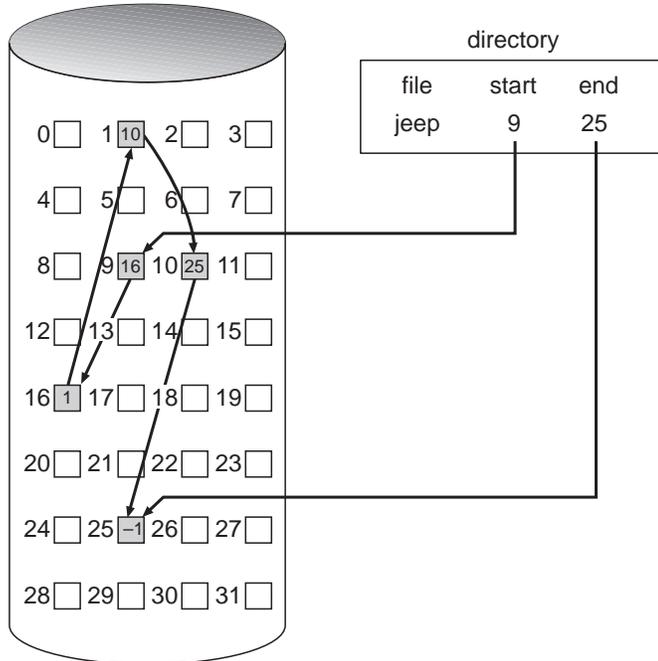
- Block to be accessed = $Q + \text{starting address}$
- Displacement into block = R

Linked Allocation

- Each file is a linked list of disk blocks; blocks may be scattered anywhere on the disk.



- Allocate as needed, link together; e.g., file starts at block 9



Linked Allocation (Cont.)

- Simple – need only starting address
- Free-space management system – no waste of space
- No random access
- Mapping

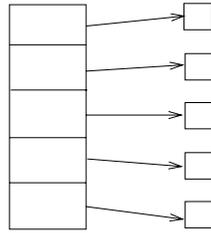
$$LA/511 \begin{cases} Q \\ R \end{cases}$$

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

- *File-allocation table (FAT)* – disk-space allocation used by MS-DOS and OS/2.

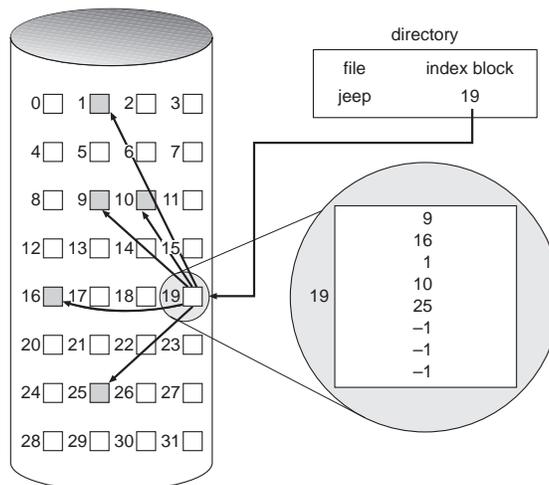
Indexed Allocation

- Brings all pointers together into the *index block*.
- Logical view



index table

Example of Indexed Allocation



Indexed Allocation (Cont.)

- 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 words and block size of 512 words. We need only 1 block for index table.

$$LA/512 \begin{cases} Q \\ R \end{cases}$$

- Q = displacement into index table
- R = displacement into block

Indexed Allocation – Mapping (Cont.)

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

$$LA/(512 \times 511) \begin{cases} Q_1 \\ R_1 \end{cases}$$

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

$$R_1/512 \begin{cases} Q_2 \\ R_2 \end{cases}$$

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

Indexed Allocation – Mapping (Cont.)

- Two-level index (maximum file size is 512^3)

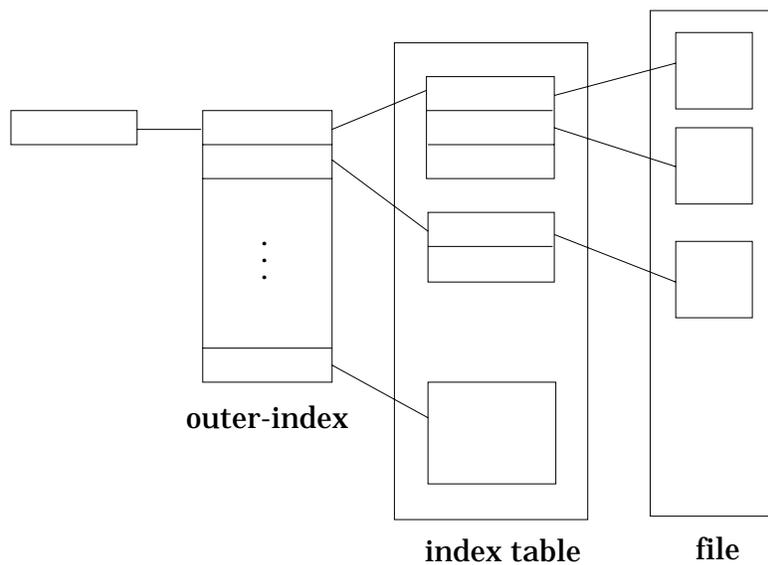
$$LA / (512 \times 512) \begin{cases} Q_1 \\ R_1 \end{cases}$$

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

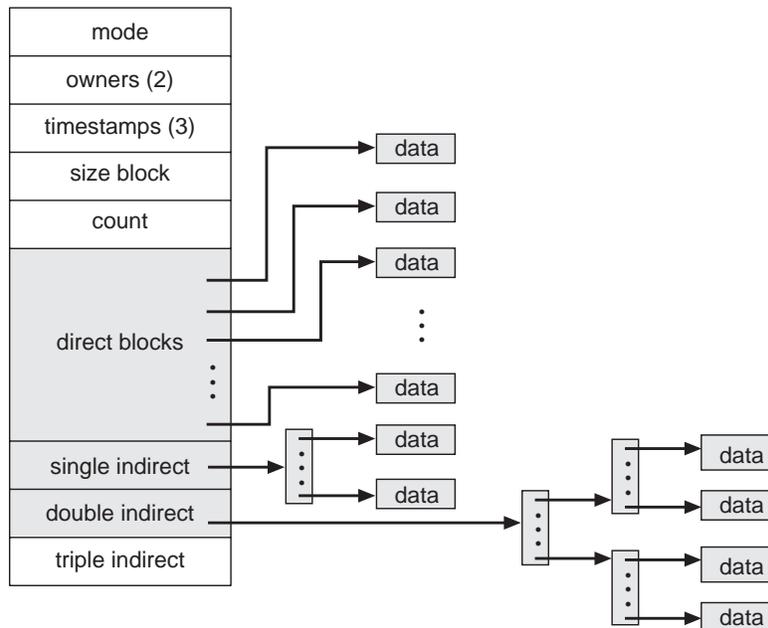
$$R_1 / 512 \begin{cases} Q_2 \\ R_2 \end{cases}$$

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

Indexed Allocation – Mapping (Cont.)



Combined Scheme: UNIX (4K bytes per block)



Free-Space Management

- Bit vector (n blocks)



$$\text{bit}[i] = \begin{cases} 0 \Rightarrow \text{block}[i] \text{ free} \\ 1 \Rightarrow \text{block}[i] \text{ occupied} \end{cases}$$

- Block number calculation

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

Free-Space Management (Cont.)

- Bit map requires extra space. Example:

block size = 2^{12} bytes

disk size = 2^{30} bytes (1 gigabyte)

$n = 2^{30}/2^{12} = 2^{18}$ bits (or 32K bytes)

- Easy to get contiguous files
- Linked list (free list)
 - Cannot get contiguous space easily
 - No waste of space
- Grouping
- Counting

Free-Space Management (Cont.)

- Need to protect:
 - Pointer to free list
 - Bit map
 - * Must be kept on disk.
 - * Copy in memory and disk may differ.
 - * Cannot allow for block[i] to have a situation where bit[i] = 1 in memory and bit[i] = 0 on disk.
 - Solution:
 - * Set bit[i] = 1 in disk.
 - * Allocate block[i].
 - * Set bit[i] = 1 in memory.

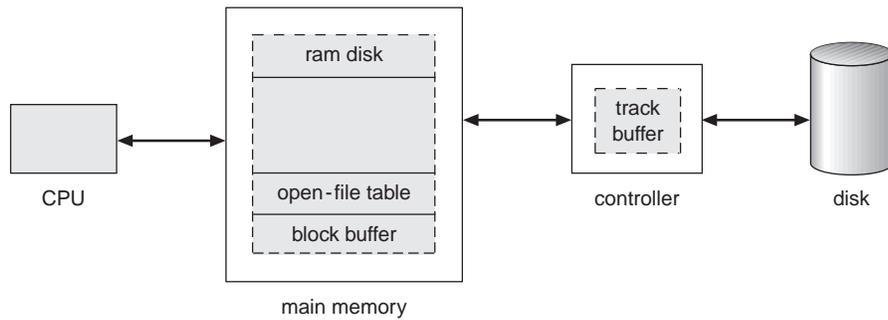
Directory Implementation

- Linear list of file names with pointers to the data blocks.
 - simple to program
 - time-consuming to execute
- Hash Table – linear list with hash data structure.
 - decreases directory search time
 - *collisions* – situations where two file names hash to the same location
 - fixed size

Efficiency and Performance

- Efficiency dependent on:
 - disk allocation and directory algorithms
 - types of data kept in file's directory entry
- Performance
 - *disk cache* – separate section of main memory for frequently used blocks
 - *free-behind* and *read-ahead* – techniques to optimize sequential access
 - improve PC performance by dedicating section of memory as *virtual disk*, or *RAM disk*

Various Disk-Caching Locations



Recovery

- Consistency checker – compares data in directory structure with data blocks on disk, and tries to fix inconsistencies.
- Use system programs to *back up* data from disk to another storage device (floppy disk, magnetic tape).
- Recover lost file or disk by *restoring* data from backup.