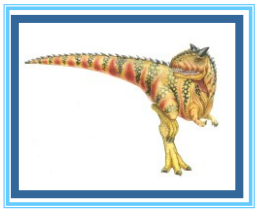
# Chapter 11: File System Implementation

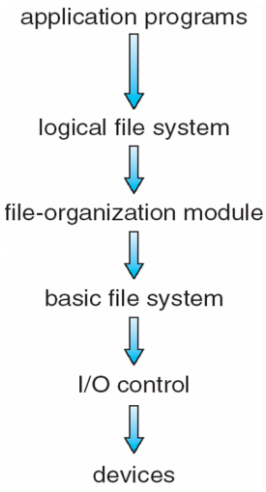





## File-System Structure

- File structure
  - Logical storage unit
  - Collection of related information
- File system organized into layers
- File system resides on secondary storage (disks)
  - Provides efficient and convenient access to disk by allowing data to be stored, located retrieved easily
- File control block – storage structure consisting of information about a file
- Device driver controls the physical device

# Layered File System






# File-System Implementation

- Boot control block contains info needed by system to boot OS from that volume
- Volume control block contains volume details
- Directory structure organizes the files
- Per-file File Control Block (FCB) contains many details about the file:

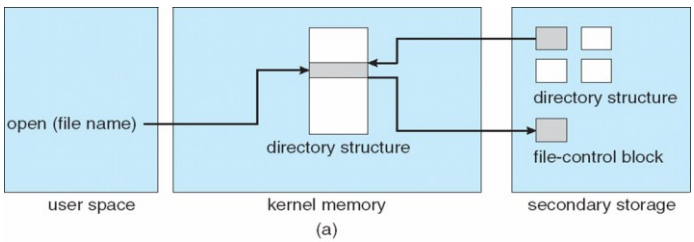
| file permissions |
|---|
| file dates (create, access, write) |
| file owner, group, ACL |
| file size |
| file data blocks or pointers to file data blocks |

## A Typical File Control Block

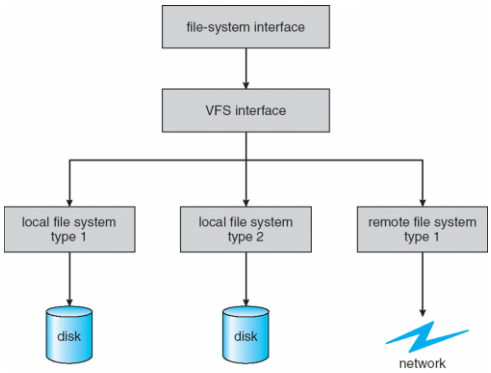
| |
|---|
| file permissions |
| file dates (create, access, write) |
| file owner, group, ACL |
| file size |
| file data blocks or pointers to file data blocks |



## In-Memory File System Structures




# Schematic View of Virtual File System




- Virtual File Systems (VFS) 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.
- The API is to the VFS interface, rather than any specific type of file system.



# Directory Implementation

- **Linear list** of file names with pointer 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

# Allocation Methods

- An allocation method refers to how disk blocks are allocated for files:
- Contiguous allocation
- Linked allocation
- Indexed allocation

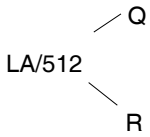


# Contiguous Allocation

- 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

# Contiguous Allocation

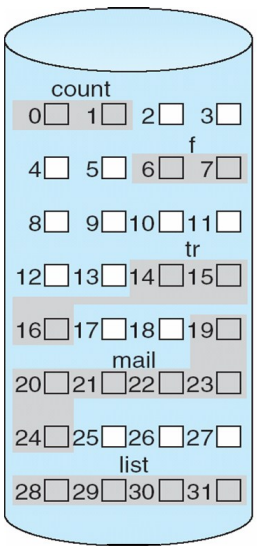
- Mapping from logical to physical




Block to be accessed = ! + starting address
Displacement into block = R



# Contiguous Allocation of Disk Space

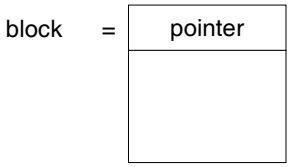



directory

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

# Linked Allocation

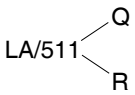
- Each file is a linked list of disk blocks: blocks may be scattered anywhere on the disk.






# Linked Allocation (Cont.)

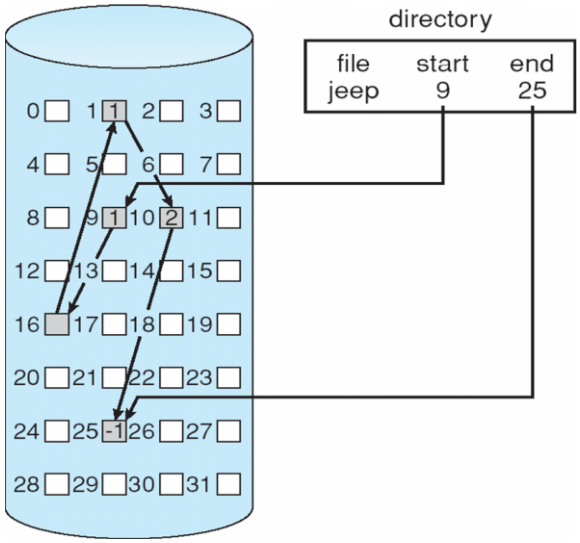
- Simple – need only starting address
- Free-space management system – no waste of space
- No random access
- Mapping




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

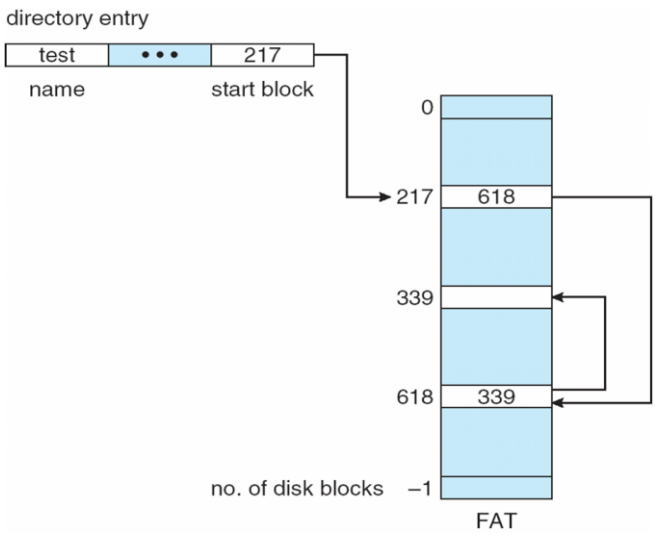
File-allocation table (FAT) – disk-space allocation used by MS-DOS and OS/2.

# Linked Allocation



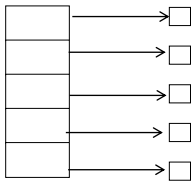



# File-Allocation Table




# Indexed Allocation

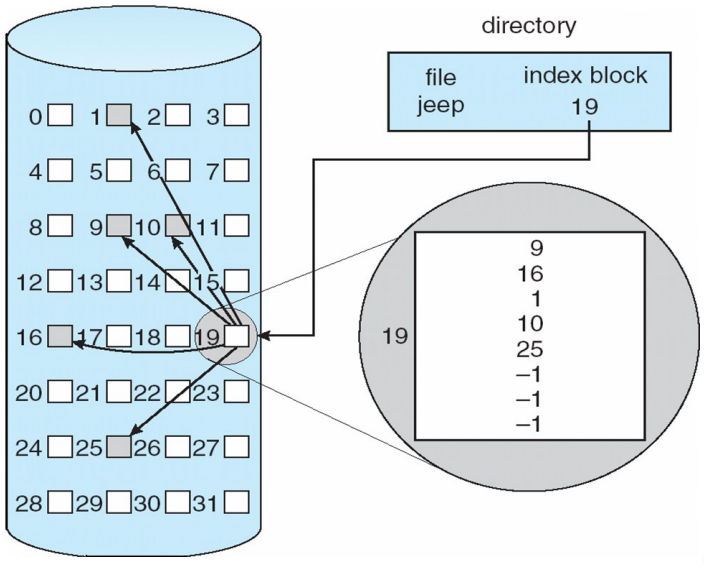
- Brings all pointers together into the index block
- Logical view



index table

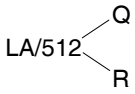


# Example of Indexed Allocation




# 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

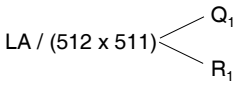



Q = displacement into index table
R = displacement into block

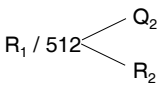


# Indexed Allocation – Mapping (Cont.)

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



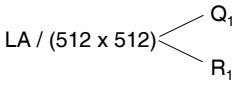

$Q_1$ = block of index table
$R_1$ is used as follows:



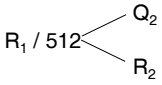

$Q_2$ = displacement into block of index table
$R_2$ displacement into block of file:

# Indexed Allocation – Mapping (Cont.)

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



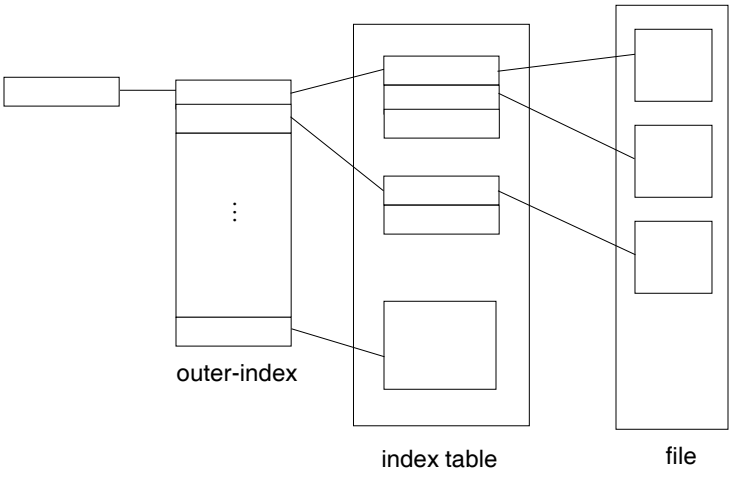

$Q_1$ = displacement into outer-index
$R_1$ is used as follows:



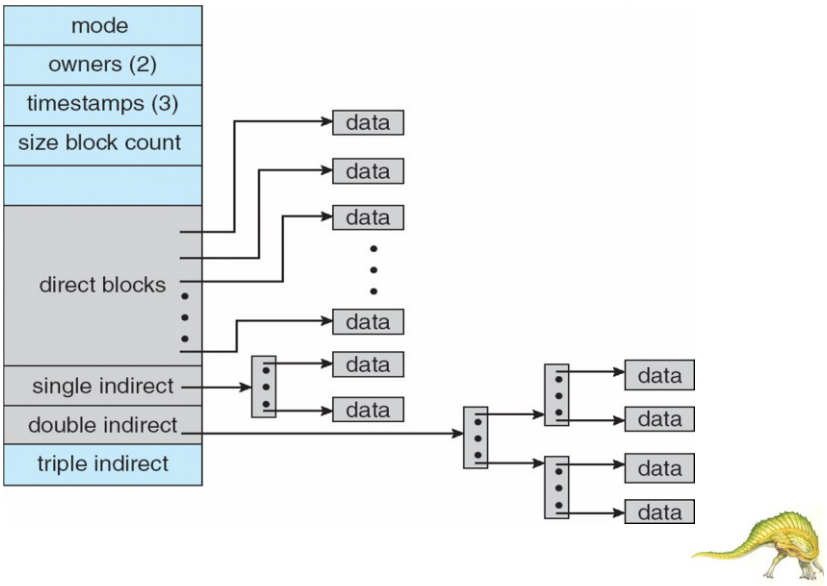

$Q_2$ = displacement into block of index table
$R_2$ displacement into block of file:



# Indexed Allocation – Mapping (Cont.)




## Combined Scheme: UNIX UFS (4K bytes per block)








## 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

# End of Chapter 11