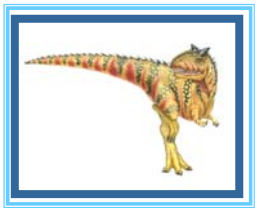
# Chapter 11:
# File System Implementation

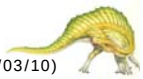


(slides improved by R. Doemer, 06/03/10)



## Chapter 11: File System Implementation

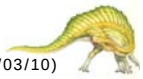
- File-System Structure
- File-System Implementation
- Directory Implementation
- Allocation Methods
- Free-Space Management
- Efficiency and Performance
- Recovery
- Log-Structured File Systems
- NFS
- Example: WAFL File System



(slide modified by R. Doemer, 06/03/10)

# Objectives

- To describe the details of implementing
  - local file systems and
  - directory structures
- To discuss
  - block allocation and
  - free-block algorithms and
  - trade-offs
- To describe the implementation of remote file systems



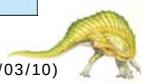
(slide modified by R. Doemer, 06/03/10)



# File-System Structure

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

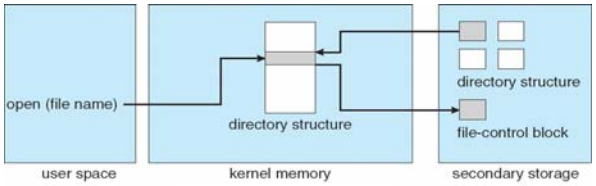
| file permissions |
| --- |
| file dates (create, access, write) |
| file owner, group, ACL |
| file size |
| file data blocks or pointers to file data blocks |



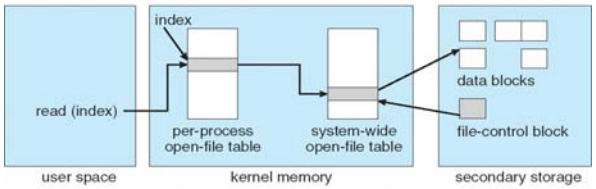
(slide modified by R. Doemer, 06/03/10)

# In-Memory File System Structures

- Necessary file system structures provided by the operating system:




(a) Opening a file.



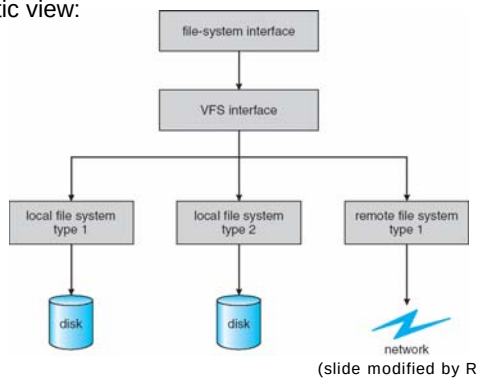

(a) Reading a file.

(slide modified by R. Doemer, 06/03/10)



# Virtual File Systems

- **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.
- Schematic view:




(slide modified by R. Doemer, 06/03/10)

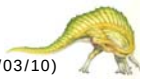
# Directory Implementation

- **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



# File Allocation Methods

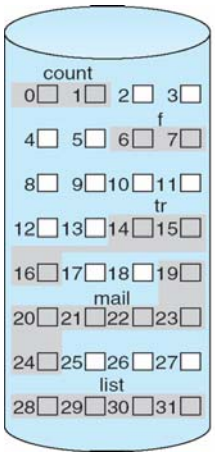
- An **allocation method** refers to how disk blocks are allocated for files:
  - **Contiguous Allocation**
  - **Linked Allocation**
  - **Indexed Allocation**



(slide modified by R. Doemer, 06/03/10)

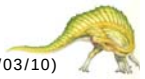
# Contiguous Allocation

- Each file occupies a set of contiguous blocks on the disk




directory

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

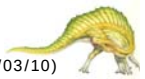


(slide modified by R. Doemer, 06/03/10)



# Contiguous Allocation

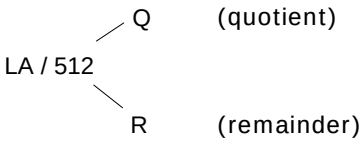
- Contiguous allocation of disk space is:
- Simple
  - only starting location (block #) and
  - length (number of blocks) are required
- Supports random access
  - see address mapping on next page
- Wasteful of space
  - dynamic storage-allocation problem
- Files cannot grow
  - Subsequent blocks may be occupied by other files



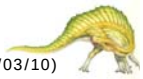
(slide modified by R. Doemer, 06/03/10)

# Contiguous Allocation

- Mapping from logical to physical address in a file
  - LA = Logical Address




  - Q + starting address = block to be accessed
  - R = displacement into block

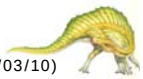


(slide modified by R. Doemer, 06/03/10)



# Extent-Based Systems

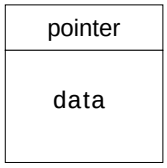
- **Extend-based File System**:
  - Modified contiguous allocation scheme
  - Used by many newer file systems
    - i.e. Veritas File System
- Extent-based file systems allocate disk blocks in **extents**
- A file consists of one or more extents
- An extent is a contiguous block of disks

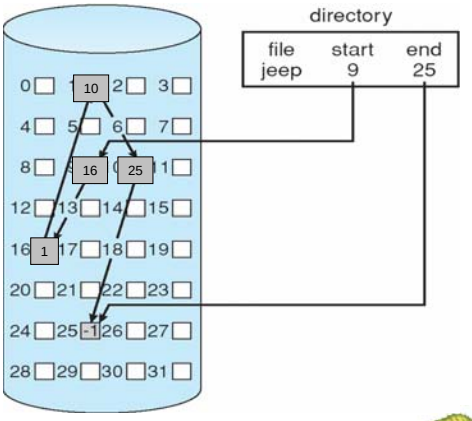


(slide modified by R. Doemer, 06/03/10)

# Linked Allocation

- Each file is a **linked list of disk blocks**:
  - Blocks may be scattered anywhere on the disk
  - Block =




(slide modified by R. Doemer, 06/03/10)



# Linked Allocation

- Simple
  - need only starting address and
  - follow next block pointers
- Efficient free-space management system
  - no waste of space
- No random access
  - Need to read blocks sequentially
- Mapping

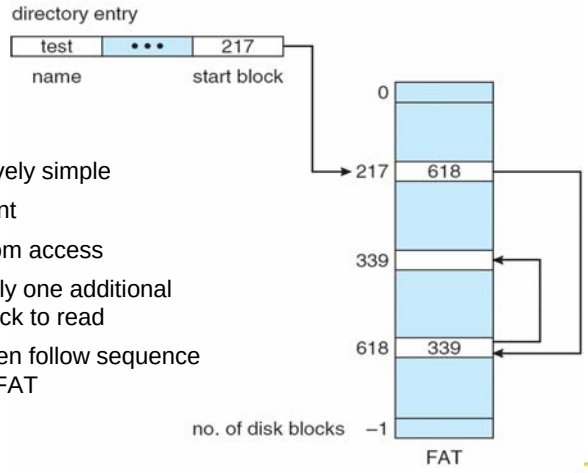
LA / 511 → Q, R

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

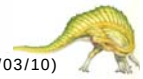
(slide modified by R. Doemer, 06/03/10)

# File Allocation Table (FAT)

- **File Allocation Table** (**FAT**)
  - Disk-space allocation used by MS-DOS and OS/2




  - Relatively simple
  - Efficient
  - Random access
    - Only one additional block to read
    - Then follow sequence in FAT

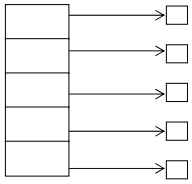


(slide modified by R. Doemer, 06/03/10)

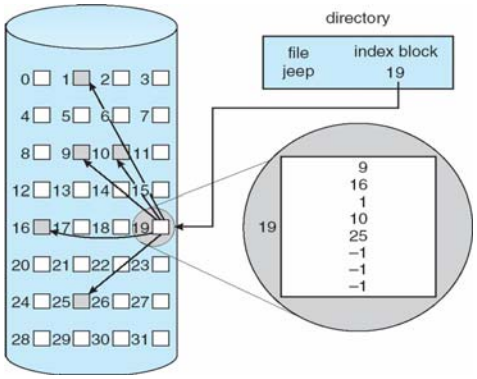


# Indexed Allocation

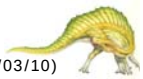
- Brings all block pointers together into the **index block**
- Logical view:



Index table



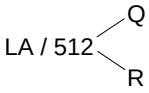

  - Simple
  - Efficient
  - Random access
  - Limited file size



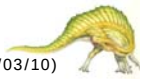
(slide modified by R. Doemer, 06/03/10)

# Indexed Allocation

- Dynamic access without external fragmentation, but have overhead of index block.
- Mapping from logical to physical address 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

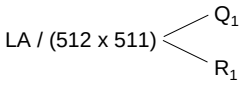


(slide modified by R. Doemer, 06/03/10)

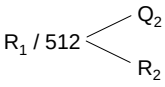


# Indexed Allocation

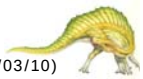
- Mapping from logical to physical address in a file of *unbounded length* (block size of 512 words)
  - Linked scheme
  - Link blocks of index table (no limit on size)




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



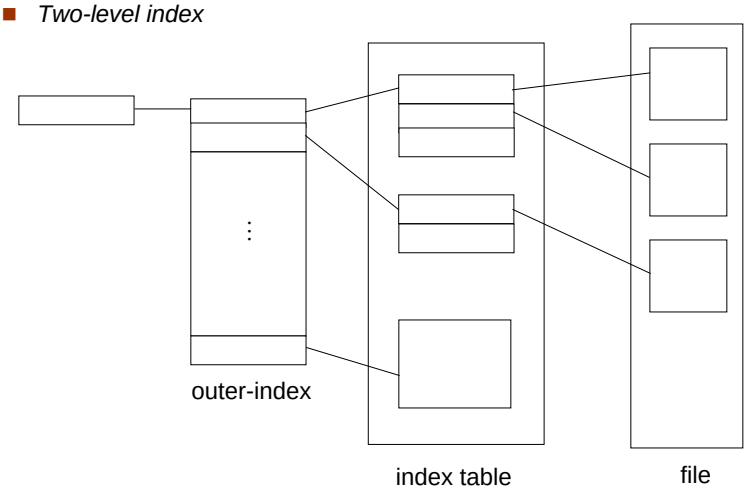

$Q_2$ = displacement into block of index table
$R_2$ = displacement into block of file



(slide modified by R. Doemer, 06/03/10)

# Indexed Allocation

- *Two-level index*

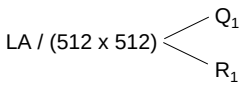



(slide modified by R. Doemer, 06/03/10)

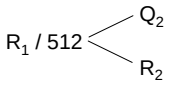


# Indexed Allocation

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




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

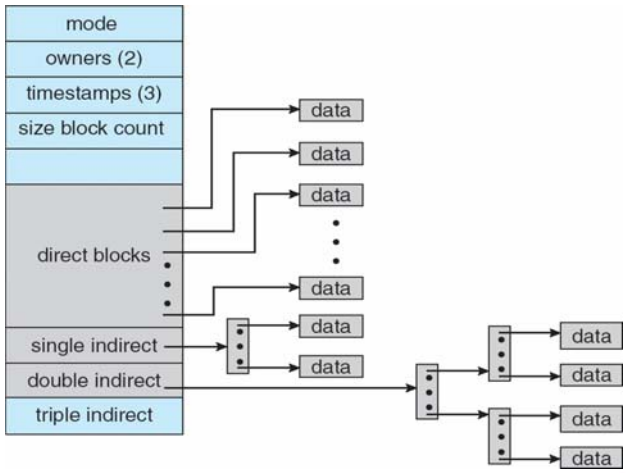



$Q_2$ = displacement into block of index table
$R_2$ = displacement into block of file

(slide modified by R. Doemer, 06/03/10)

# Indexed Allocation

- *Combined Scheme:* UNIX inode (with 4K bytes per block)

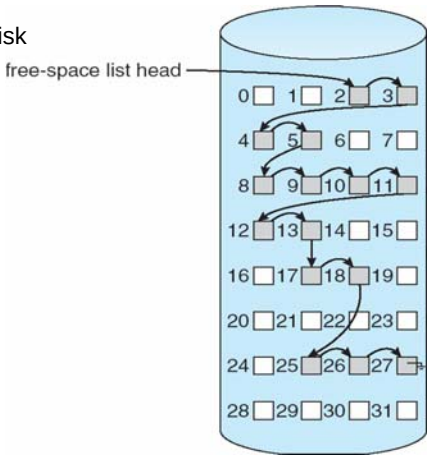



(slide modified by R. Doemer, 06/03/10)



# Free-Space Management

- Many schemes are possible
  - **Linked free space** list on disk
    - See figure to the right
    - No wasted space!
    - Can be extended
      - Grouping
      - Counting
  - Special "**free space file**"
    - Allocation takes blocks from this file
    - Often used in indexed or FAT allocation
  - **Bit maps**
    - (omitted for EECS111)




(slide modified by R. Doemer, 06/03/10)

# End of Chapter 11