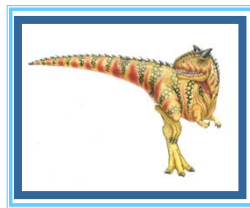


Chapter 11: File System Implementation



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

Operating System Concepts – 8th Edition,

Silberschatz, Galvin and Gagne ©2009



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)

Operating System Concepts – 8th Edition

11.2

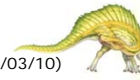
Silberschatz, Galvin and Gagne ©2009





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



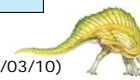
(slide modified by R. Doemer, 06/03/10)



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

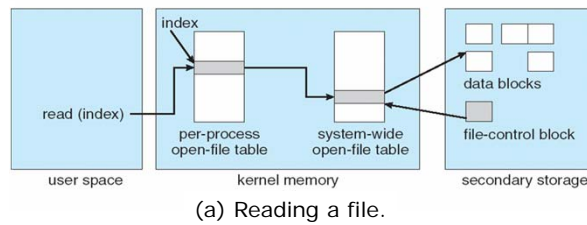
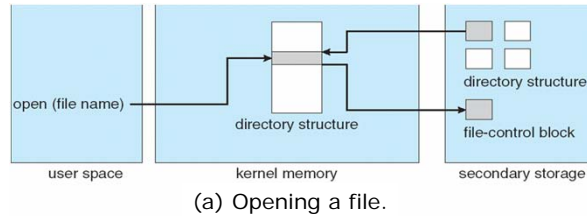


(slide modified by R. Doemer, 06/03/10)

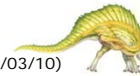


In-Memory File System Structures

- Necessary file system structures provided by the operating system:

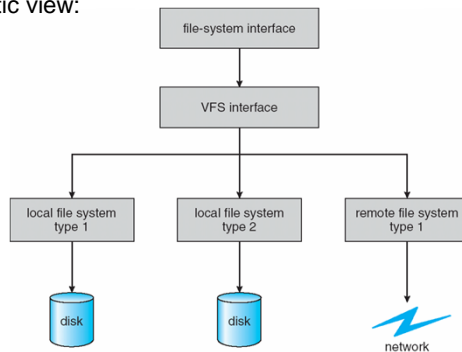


(slide modified by R. Doemer, 06/03/10)



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

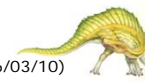
- **Linear list** of file names with pointer to the data blocks.
 - simple to program
 - time-consuming to execute

- **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

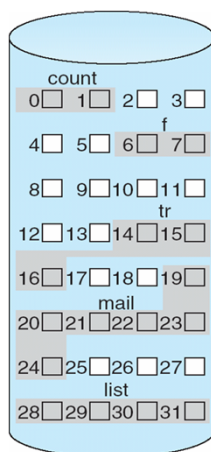
- An **allocation method** refers to how disk blocks are allocated for files:
 - **Contiguous Allocation**
 - **Linked Allocation**
 - **Indexed Allocation**





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)



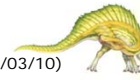


Contiguous Allocation

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

$$\begin{array}{rcl} & & Q \quad (\text{quotient}) \\ & \swarrow & \\ LA / 512 & & \\ & \searrow & \\ & & R \quad (\text{remainder}) \end{array}$$

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



(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)

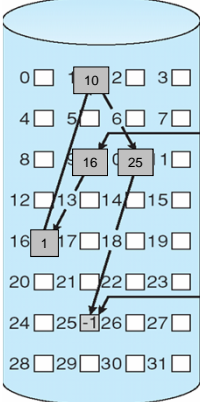


Linked Allocation

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


pointer
data

pointer
data




directory

file	start	end
jeep	9	25

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

Operating System Concepts – 8th Edition
11.13
Silberschatz, Galvin and Gagne ©2009




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

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

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

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

Operating System Concepts – 8th Edition
11.14
Silberschatz, Galvin and Gagne ©2009

File Allocation Table (FAT)

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

test	...	217
name		start block

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

FAT

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

Operating System Concepts – 8th Edition
11.15
Silberschatz, Galvin and Gagne ©2009

Indexed Allocation

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

Index table

file	index block
jeep	19

9
16
1
10
25
-1
-1
-1

- Simple
- Efficient
- Random access
- Limited file size

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

Operating System Concepts – 8th Edition
11.16
Silberschatz, Galvin and Gagne ©2009

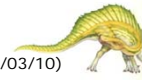


Indexed Allocation

- 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

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

- 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)

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


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

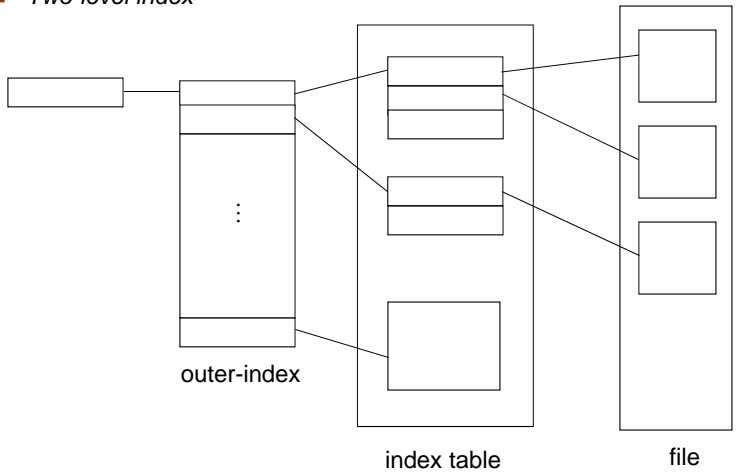


(slide modified by R. Doemer, 06/03/10)



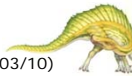

Indexed Allocation

- *Two-level index*



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

Operating System Concepts – 8th Edition 11.19 Silberschatz, Galvin and Gagne ©2009

Indexed Allocation

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

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

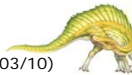
Q_1 = displacement into outer-index
 R_1 is used as follows:

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

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

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

Operating System Concepts – 8th Edition 11.20 Silberschatz, Galvin and Gagne ©2009



Indexed Allocation

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

The diagram illustrates the structure of a UNIX inode. It consists of several fields: mode, owners (2), timestamps (3), size, block count, direct blocks, single indirect, double indirect, and triple indirect. The direct blocks field points to a series of data blocks. The single indirect field points to an indirect block, which in turn points to a series of data blocks. The double indirect field points to a double indirect block, which points to a series of indirect blocks, each of which points to a series of data blocks. The triple indirect field points to a triple indirect block, which points to a series of double indirect blocks, each of which points to a series of indirect blocks, each of which points to a series of data blocks.

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

Operating System Concepts – 8th Edition 11.21 Silberschatz, Galvin and Gagne ©2009

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**
 - ▶ (details omitted)

The diagram shows a linked free space list on disk. It consists of a vertical cylinder representing the disk, divided into blocks numbered 0 to 31. A 'free-space list head' points to block 0. Arrows indicate the sequence of blocks in the list: 0 → 1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9 → 10 → 11 → 12 → 13 → 14 → 15 → 16 → 17 → 18 → 19 → 20 → 21 → 22 → 23 → 24 → 25 → 26 → 27 → 28 → 29 → 30 → 31.

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

Operating System Concepts – 8th Edition 11.22 Silberschatz, Galvin and Gagne ©2009