Operating Systems & Systems Programming
Module 6
File System and Input Output Management

Dr. Vikash

Jaypee Institute of Information Technology, Noida

Overview

1. File Concepts
2. File Structure
3. Access Methods
4. Directory Structure
5. Protection
6. File-System Structure
7. Allocation Methods
8. Free Space Management

Module Objective

- To explain the function of file systems
- To describe the interfaces to file systems
- To discuss file-system design tradeoffs, including access methods, file sharing, file locking, and directory structures
- To explore file-system protection

File Concept

- Contiguous logical address space
- Types:
  - Data (numeric, character, binary)
  - Program
- Contents defined by file's creator
  - Many types

File Attributes

- **Name**- only information kept in human-readable form
- **Identifier**- unique tag (number) identifies file within file system
- **Type**- needed for systems that support different types
- **Location**- pointer to file location on device
- **Size**- current file size
- **Protection**- controls who can do reading, writing, executing
- **Time, date, and user identification**- data for protection, security, and usage monitoring
- Information about files are kept in the directory structure, which is maintained on the disk
- Many variations, including extended file attributes such as file checksum
- Information kept in the directory structure

File Operations

- File is an abstract data type
- **Create**
- **Write** - at write pointer location
- **Read** - at read pointer location
- **Reposition within file** - seek
- **Delete**
- **Truncate**
- **Open(Fi)**- search the directory structure on disk for entry Fi, and move the content of entry to memory
- **Close (Fi)**- move the content of entry Fi in memory to directory structure on disk

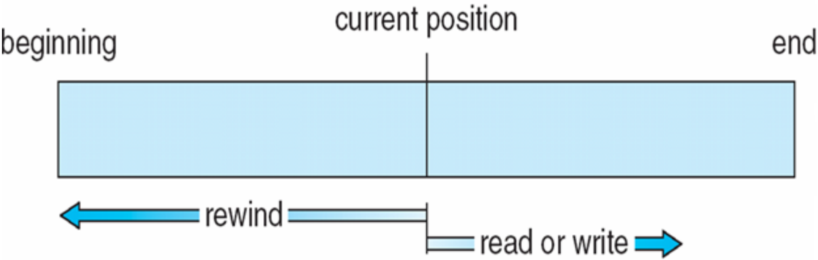
File Types - Name, Extension

file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rtf, doc	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	ps, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

File Structure

- None - sequence of words, bytes
- Simple record structure
  - Lines
  - Fixed length
  - Variable length
- Complex Structures
  - Formatted document
  - Relocatable load file
- Can simulate last two with first method by inserting appropriate control characters
- Who decides:
  - Operating system
  - Program

Sequential-access File



- **Sequential Access**
  - read next
  - write next
  - reset
  - no read after last write (rewrite)
- **Direct Access**- file is fixed length **logical records**
  - read n
  - write n
  - position to n
    - read next
    - write next
  - rewrite n

  n = relative block number
- Relative block numbers allow OS to decide where file should be placed

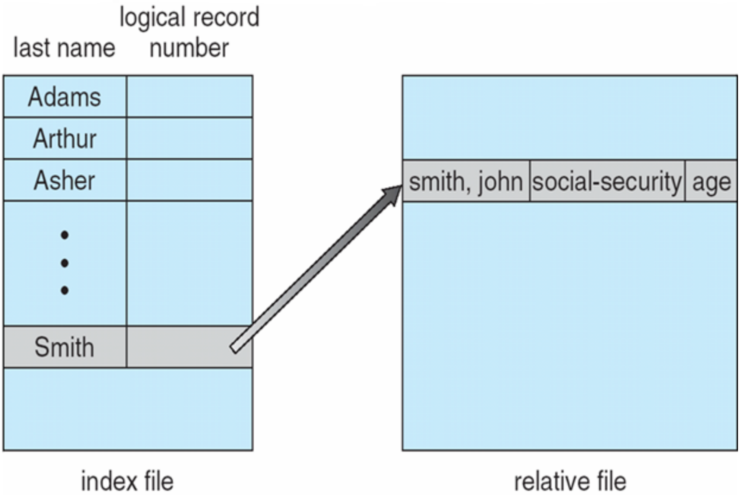
Simulation of Sequential Access on Direct-access File

sequential access	implementation for direct access
reset	$cp = 0;$
read next	*read cp*; $cp = cp + 1;$
write next	*write cp*; $cp = cp + 1;$

Other Access Methods

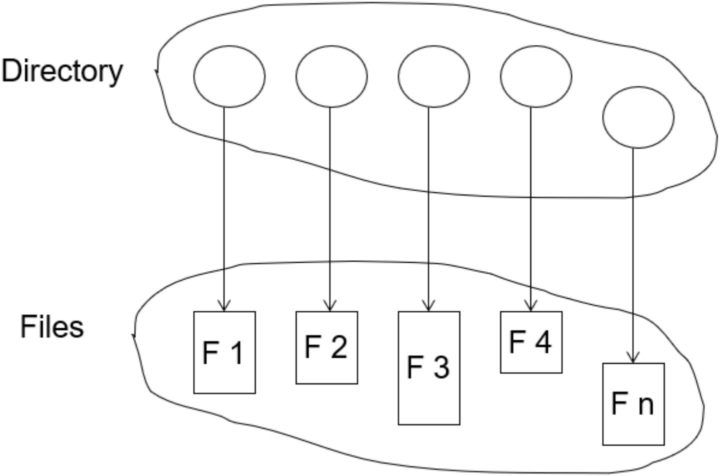
- Can be built on top of base methods
- General involve creation of an index for the file
- Keep index in memory for fast determination of location of data to be operated on (consider UPC code plus record of data about that item)
- If too large, index (in memory) of the index (on disk)
- IBM indexed sequential-access method (ISAM)
  - Small master index, points to disk blocks of secondary index
  - File kept sorted on a defined key. All done by the OS
- VMS operating system provides index and relative files as another example

Example of Index and Relative Files



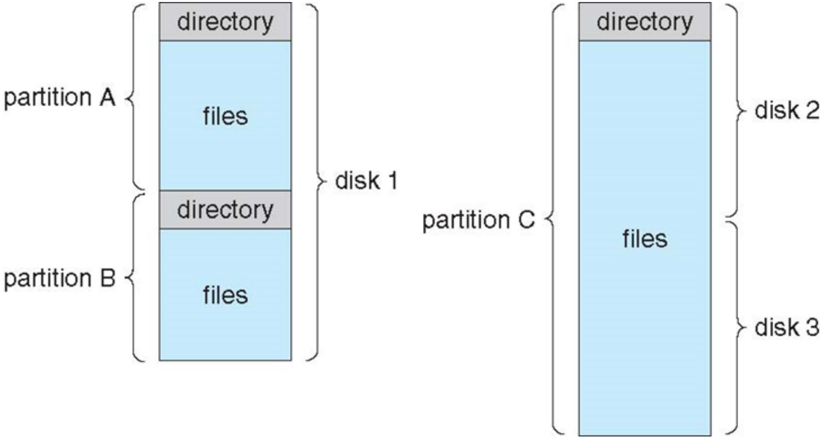
Directory Structure

- A collection of nodes containing information about all files



- Both the directory structure and the files reside on disk

A Typical File-system Organization



Types of File Systems

- We mostly talk of general-purpose file systems
- But systems frequently have may file systems, some general- and some special- purpose
- Consider Solaris has
  - tmpfs - memory-based volatile FS for fast, temporary I/O
  - objfs - interface into kernel memory to get kernel symbols for debugging
  - ctfs - contract file system for managing daemons
  - lofs - loopback file system allows one FS to be accessed in place of another
  - procfs - kernel interface to process structures
  - ufs, zfs - general purpose file systems

Operations Performed on Directory

- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

Directory Organization

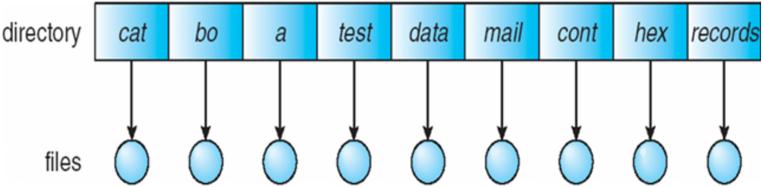
The directory is organized logically to obtain

- Efficiency - locating a file quickly
- Naming - convenient to users
  - Two users can have same name for different files
  - The same file can have several different names

  Grouping - logical grouping of files by properties, (e.g., all Java programs, all games, ...)

Single-Level Directory

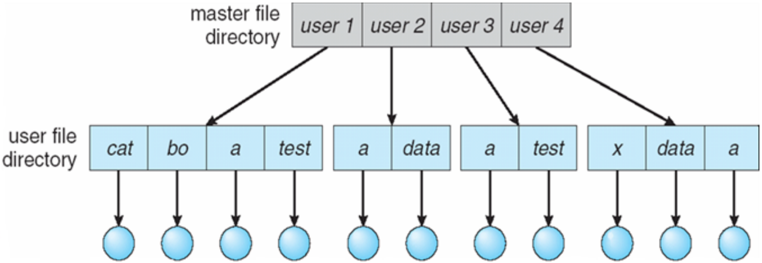
- A single directory for all users



- Naming problem
- Grouping problem

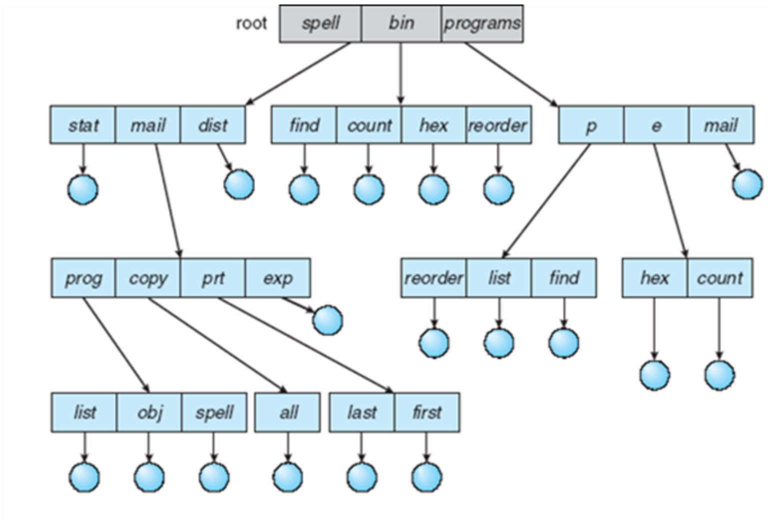
Two-Level Directory

- Separate directory for each user



- Path name
- Can have the same file name for different user
- Efficient searching
- No grouping capability

Tree-Structured Directories

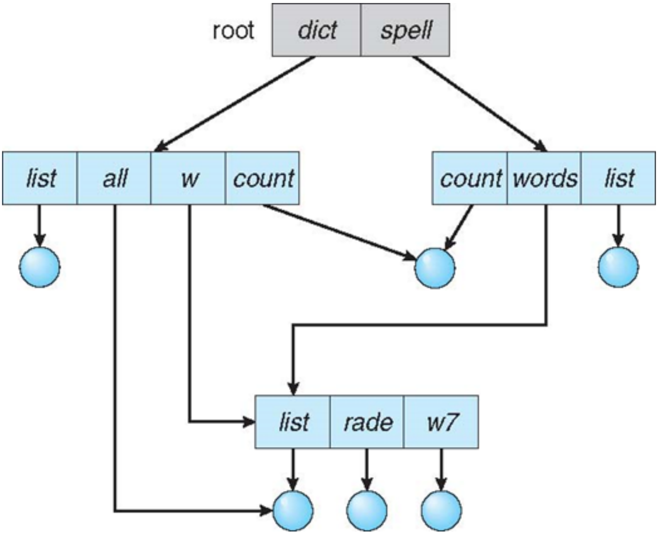


Tree-Structured Directories

- Efficient Searching
- Grouping Capability
- Current directory (working directory)
  - **cd /spell/mail/prog**
  - **type list**

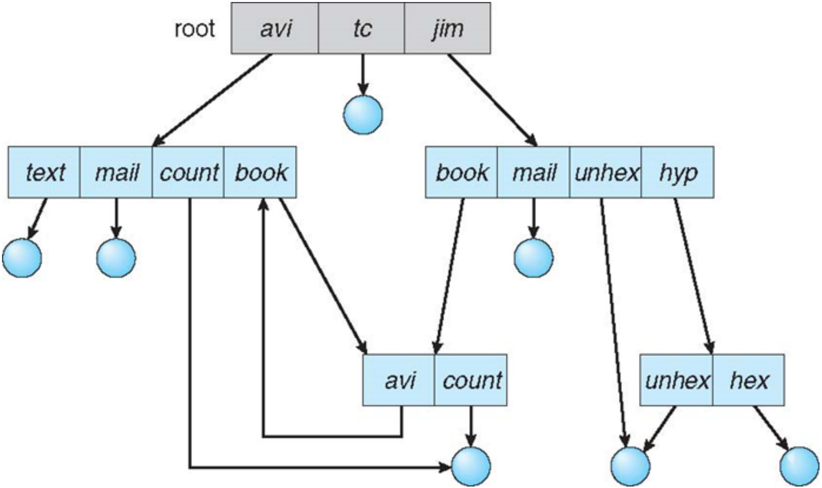
Acyclic-Graph Directories

Have shared subdirectories and files



Acyclic-Graph Directories

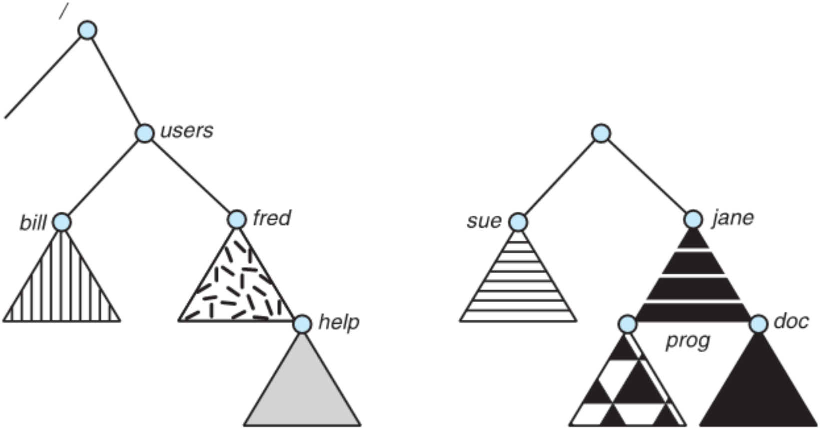
- Two different names (aliasing)
- If **dict** deletes **list** $\rightarrow$ dangling pointer
  Solutions:
  - Backpointers, so we can delete all pointers
  - Backpointers using a daisy chain organization
  - Entry-hold-count solution


root
avi
tc
jim
text
mail
count
book
book
mail
unhex
hyp
avi
count
unhex
hex

General Graph Directory (Cont.)

- How do we guarantee no cycles?
  - Allow only links to file not subdirectories
  - **Garbage collection**
  - Every time a new link is added use a cycle detection algorithm to determine whether it is OK

- A file system must be **mounted** before it can be accessed
- A unmounted file system (i.e., Fig. 11-11(b)) is mounted at a **mount point**.



(a) (b)

File Sharing

- Sharing of files on multi-user systems is desirable
- Sharing may be done through a **protection** scheme
- On distributed systems, files may be shared across a network
- Network File System (NFS) is a common distributed file-sharing method
- If multi-user system
  - **User IDs** identify users, allowing permissions and protections to be per-user
    Group IDs allow users to be in groups, permitting group access rights
  - Owner of a file / directory
  - Group of a file / directory

File Sharing - Remote File Systems

- Uses networking to allow file system access between systems
  - Manually via programs like FTP
  - Automatically, seamlessly using **distributed file systems**
  - Semi automatically via the **world wide web**
- **Client-server** model allows clients to mount remote file systems from servers
  - Server can serve multiple clients
  - Client and user-on-client identification is insecure or complicated
  - **NFS** is standard UNIX client-server file sharing protocol
  - **CIFS** is standard Windows protocol
  - Standard operating system file calls are translated into remote calls
- Distributed Information Systems (**distributed naming services**) such as LDAP, DNS, NIS, Active Directory implement unified access to information needed for remote computing

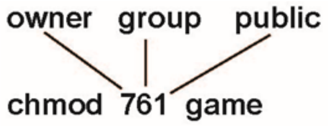
- All file systems have failure modes
  - For example corruption of directory structures or other non-user data, called **metadata**
- Remote file systems add new failure modes, due to network failure, server failure
- Recovery from failure can involve **state information** about status of each remote request
- **Stateless** protocols such as NFS v3 include all information in each request, allowing easy recovery but less security

Protection

- File owner/creator should be able to control:
  - what can be done
  - by whom
- Types of access
  - **Read**
  - **Write**
  - **Execute**
  - **Append**
  - **Delete**
  - **List**

Access Lists and Groups

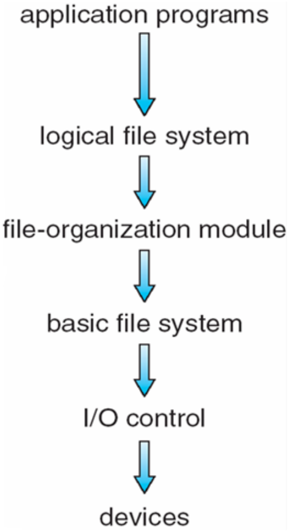
- Mode of access: read, write, execute
- Three classes of users on Unix / Linux
  a. owner access 7 $\rightarrow$ 1 1 1
  b. group access 6 $\rightarrow$ 1 1 0
  c. public access 1 $\rightarrow$ 0 0 1
- Ask manager to create a group (unique name), say G, and add some users to the group.
- For a particular file (say game) or subdirectory, define an appropriate access.



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 retrieved 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

Layered File System



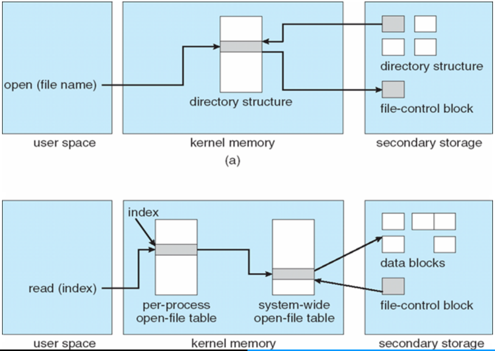
File System Layers

- **Device drivers** manage I/O devices at the I/O control layer
  - Given commands like "read drive1, cylinder 72, track 2, sector 10, into memory location 1060" outputs low-level hardware specific commands to hardware controller
- **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 organization module** understands files, logical address, and physical blocks
- Translates logical block # to physical block #
- Manages free space, disk allocation

File System Layers

- **Logical file system** manages metadata information
  - Translates file name into file number, file handle, location by maintaining file control blocks (inodes in UNIX)
  - Directory management
  - Protection
- Layering useful for reducing complexity and redundancy, but adds overhead and can decrease performanceTranslates file name into file number, file handle, location by maintaining file control blocks (inodes in UNIX)
  - Logical layers can be implemented by any coding method according to OS designer
- Many file systems, sometimes many within an operating system
  - Each with its own format (CD-ROM is ISO 9660; Unix has UFS, FFS; Windows has FAT, FAT32, NTFS as well as floppy, CD, DVD Blu-ray, Linux has more than 40 types, with extended file system ext2 and ext3 leading; plus distributed file systems, etc.)
  - New ones still arriving ZFS, GoogleFS, Oracle ASM, FUSE

- 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
- 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*

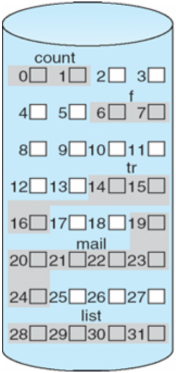


Allocation Methods

- An allocation method refers to how disk blocks are allocated 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 fragmentation, need for
  - compaction off-line (downtime) or on-line

Contiguous Allocation

Mapping from logical to physical



directory

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

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

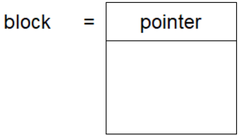
Extent-Based Systems

- 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 block of disks
  - Extents are allocated for file allocation
  - A file consists of one or more extents

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

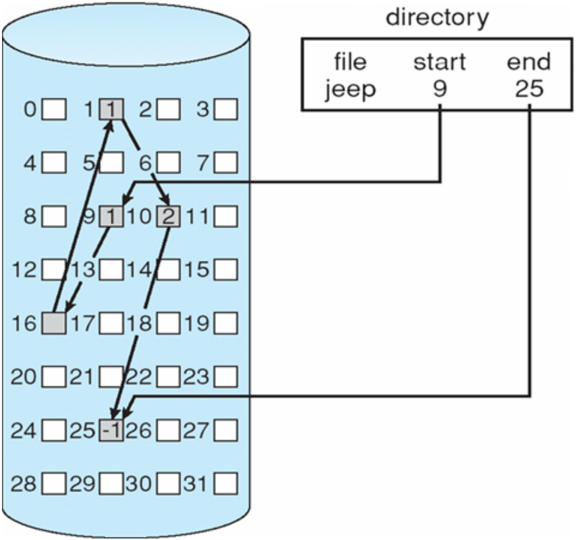
Each file is a linked list of disk blocks: blocks may be scattered anywhere on the disk



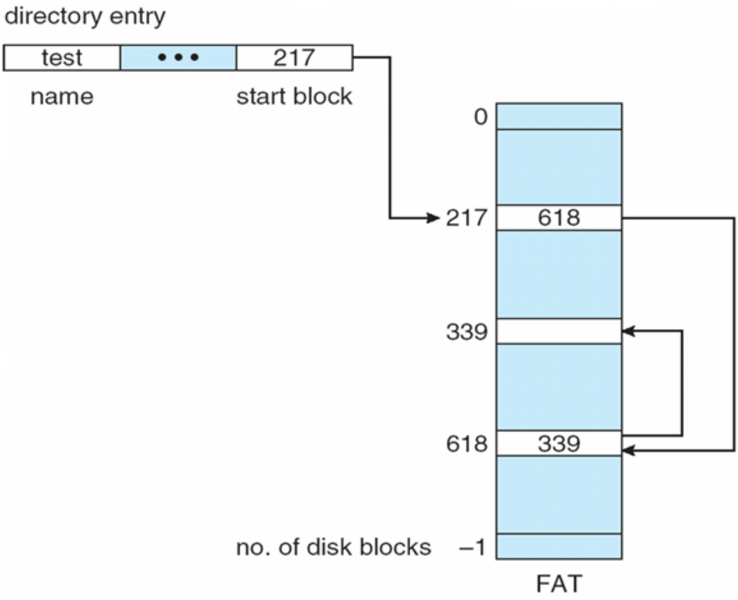
Block to be accessed is the Qth block in the linked chain of blocks representing the file.

Displacement into block = R + 1

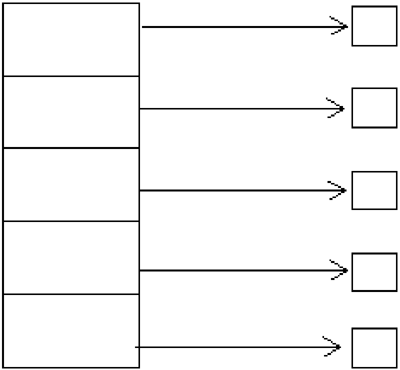
Linked Allocation



File-Allocation Table

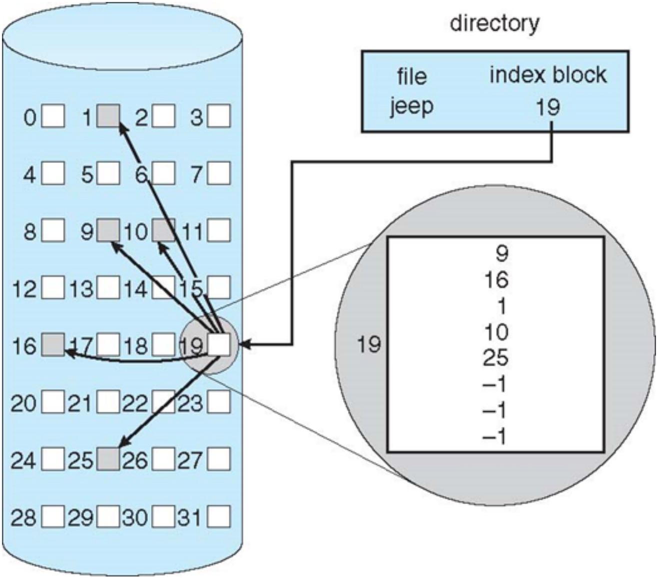


- **Indexed allocation**
  - Each file has its own **index block(s)** of pointers to its data blocks
- Logical view



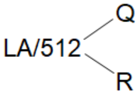
index table

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



Q = displacement into index table
R = displacement into block

Indexed Allocation (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)

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

Q1 = block of index table
R1 is used as follows:

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

Q2 = displacement into block of index table
R2 displacement into block of file:

Indexed Allocation (Cont.)

- 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)

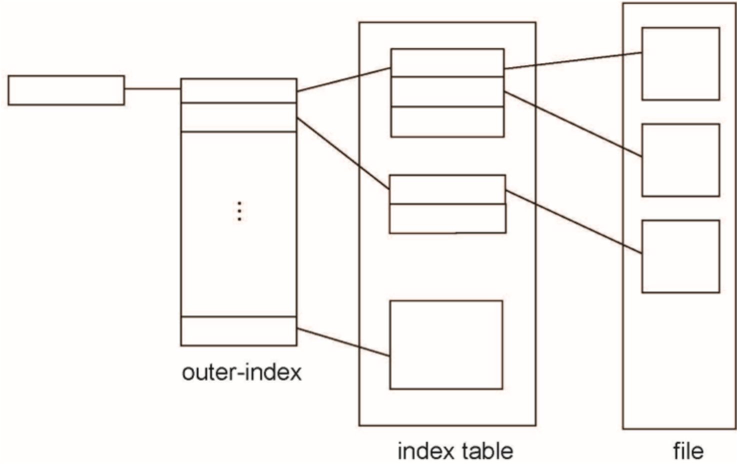
$$LA / (512 \times 512) \begin{cases} Q_1 \\ R_1 \end{cases}$$

Q1 = displacement into outer-index
R1 is used as follows:

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

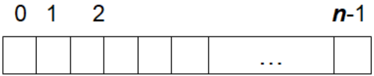
Q2 = displacement into block of index table
R2 displacement into block of file:

Indexed Allocation Mapping (Cont.)



Free-Space Management

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

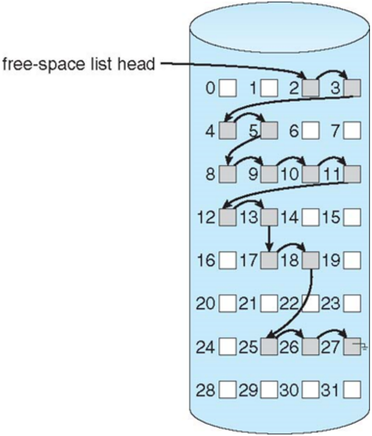


$$\text{bit}[i] = \begin{cases} 1 \Rightarrow \text{block}[i] \text{ free} \\ 0 \Rightarrow \text{block}[i] \text{ occupied} \end{cases}$$

Block Number Calculation

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

- Linked list (free list)
  - Cannot get contiguous space easily
  - No waste of space
  - No need to traverse the entire list (if # free blocks recorded)



Free-Space Management (Cont.)

- Grouping
  - Modify linked list to store address of next n-1 free blocks in first free block, plus a pointer to next block that contains free-block-pointers (like this one)
- Counting
  - Because space is frequently contiguously used and freed, with contiguous-allocation allocation, extents, or clustering
  - Keep address of first free block and count of following free blocks
  - Free space list then has entries containing addresses and counts

Free-Space Management (Cont.)

- Space Maps
  - Used in ZFS
  - Consider meta-data I/O on very large file systems
  - Full data structures like bit maps couldn't fit in memory $\rightarrow$ thousands of I/Os
  - Divides device space into metaslab units and manages metaslabs (Given volume can contain hundreds of metaslabs)
  - Each metaslab has associated space map (Uses counting algorithm)
  - But records to log file rather than file system (Log of all block activity, in time order, in counting format)
  - Metaslab activity $\rightarrow$ load space map into memory in balanced-tree structure, indexed by offset

Thank You!!!