## File System

1

## Longterm Storage

- store very large amounts of data
- data should not be lost after process terminates
- processes should be able to share access to the data

2

## File System Functions

- file naming
- file access
- file use
- protection and sharing
- implementation

3

## File System Properties

- from the point of view of the user
  - file contents
  - file names
  - file protection and sharing
  - file operations
  - ...

⇒ User interface

- from the point of view of the designer
  - implementation of files
  - free space handling
  - logical block size
  - ....

⇒ File system implementation

4

## File Types

- Files
  - ASCII files
  - binary files
- Catalogs
  - in most operating systems catalogs ≈ files

5

## Access within a File

- sequential access
- random access

6

## File Attributes

- access rights
- password
- creator
- owner
- read-only flag
- hidden flag
- system flag
- archive flag
- ASCII/binary flag
- random access flag
- locking flags
- record length
- key location
- key length
- time of creation
- time of last access
- time of last change
- file size
- maximum file size

7

## File Operations

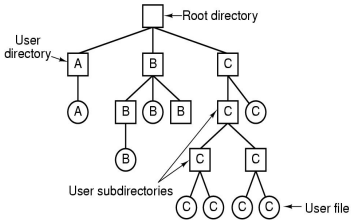
- create / delete
- rename
- open / close
- read / write / append
- position file pointer
- query/change file attributes

⇒ through system calls (*open*, *creat*, *read*, *write*, *close*, .....)

8

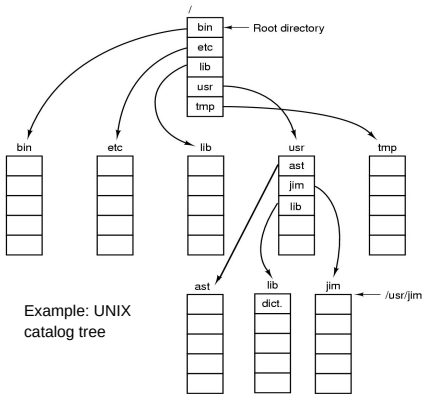
## Hierarchical Catalog Systems

- users wish to keep their files in a logical grouping
- catalog tree
- used in modern operating systems




(**Note:** letters show the owners of the files/directories)

9



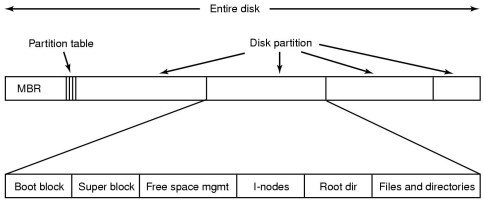

Example: UNIX catalog tree

## Catalog Operations

- create / erase
- open / close
- reading
  - example: list files
  - must open before reading
- rename
- link / unlink
  - in UNIX this is similar to erasing a file

11

## File System Implementation



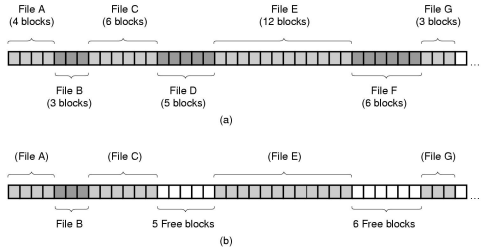

Example file system structure

12

## File System Implementation

- using contiguous allocation
  - keep a list of addresses of first blocks and number of blocks for each file
  - advantages
    - easy implementation
    - more efficient “read” operation
  - disadvantages
    - fragmentation on disk (need to compact disk)
    - keep a list of free spaces
      - file size must be known at creation (cannot change)
      - limited maximum file size
  - good for CD-ROM file systems (only one write)

13

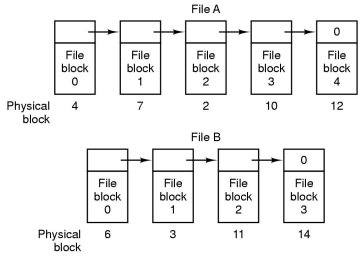



(a) contiguous allocation example: 7 files
(b) view of the disk after files *D* and *E* have been deleted

## File System Implementation

- using linked lists
  - first word of each block is a pointer to the next block
  - no fragmentation (internal fragmentation only in the last block)
  - only the address of the first block of a file is kept
  - access to data in a file
    - easy sequential access
    - random access is harder
  - data size in blocks are no longer a power of 2
    - few bytes taken up by pointer
    - most reads performed in sizes as powers of 2
      - need to read two blocks to achieve the required amount of data

15

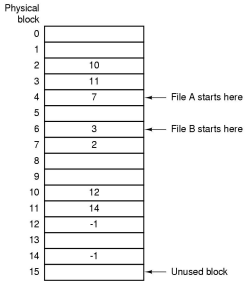



Using linked lists

## File System Implementation

- using file tables in memory
  - keep the pointers in a table in memory (instead of in the blocks on the disk)
  - FAT (File Allocation Table)
  - easier random access
    - since table is in memory
  - only need to know the address of the starting block
  - the whole table must be in memory !
  - size of table depends on size of disk
    - e.g.: for a 20 GB disk and a block size 1K: need 20 million records of a minimum of 3 bytes in the table (20MB)

17

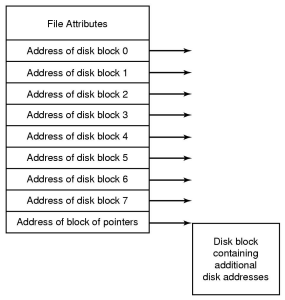



using file tables in memory

## File System Implementation

- keep an i-node (index-node) for each file
  - contains file attributes
  - contains disk addresses of blocks
- keep only the i-nodes of open files in memory
  - total memory size needed is proportional to the number of maximum files allowed to be open at the same time
- in the simplest implementation, the maximum number of blocks for a file is limited
  - solution: reserve the last entry of the i-node for a pointer to a block containing more block addresses

19



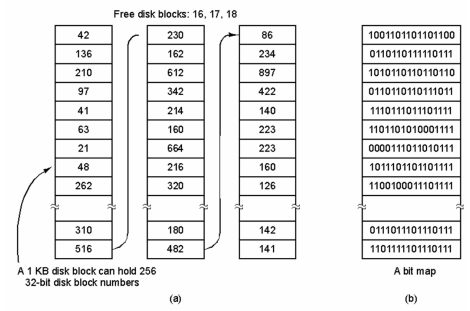

example i-node

## Disk Space Management

- files split up into blocks of fixed size which do not need to be adjacent on disk
- what should the block size be (unit of allocation) ?
  - same as sector, track, cylinder size?
    - device dependent
  - selection of the size of blocks is crucial
    - performance and efficient disk space usage are contradictory objectives
    - better to choose depending on average file size
    - size is usually pre-determined for each system
      - UNIX systems: usually 1K

21

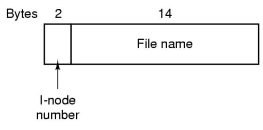
## Keeping Track of Free Blocks on Disk




Storing the free space info: (a) in a linked list (b) as a bitmap

## UNIX V7 File System

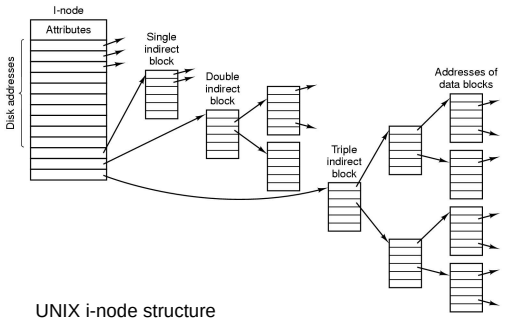
- tree structure starts at root catalog
- file name max. 14 characters
- attributes in i-nodes
  - file size
    - creation time
    - last access time
    - last change time
    - owner
    - group
    - protection bits
    - no of connections




UNIX V7 catalog entry

23

## UNIX V7 File System




UNIX i-node structure