Modern Operating Systems

Chapter 4 – File System Implementation

Zhang Yang

Spring 2022

Content of the Lecture

- 4.3 File System Implementation
  - File System Layout
  - Implementing Files
  - Implementing Directory
  - Shared Files

Overview

- Implementer's view on file systems (4.3)
  - How files and directories are stored?
  - How disk space is managed?
  - How to make everything work efficiently and reliably?
  - …

File System Layout (1)

- Sector 0: "Master Boot Record" (MBR)
  - Contains the partition map.
- Rest of disk divided into "partitions".
  - Partition: sequence of consecutive sectors.
- Each partition can hold its own file system.
  - Unix file system
  - Windows file system
  - Apple file system
  - …
- Every partition starts with a "boot block".
  - Contains a small program.
  - This "boot program" reads in an OS from the file system in that partition.
- OS Startup
  - BIOS reads MBR , then reads & execs a boot block.

File System Layout (2)

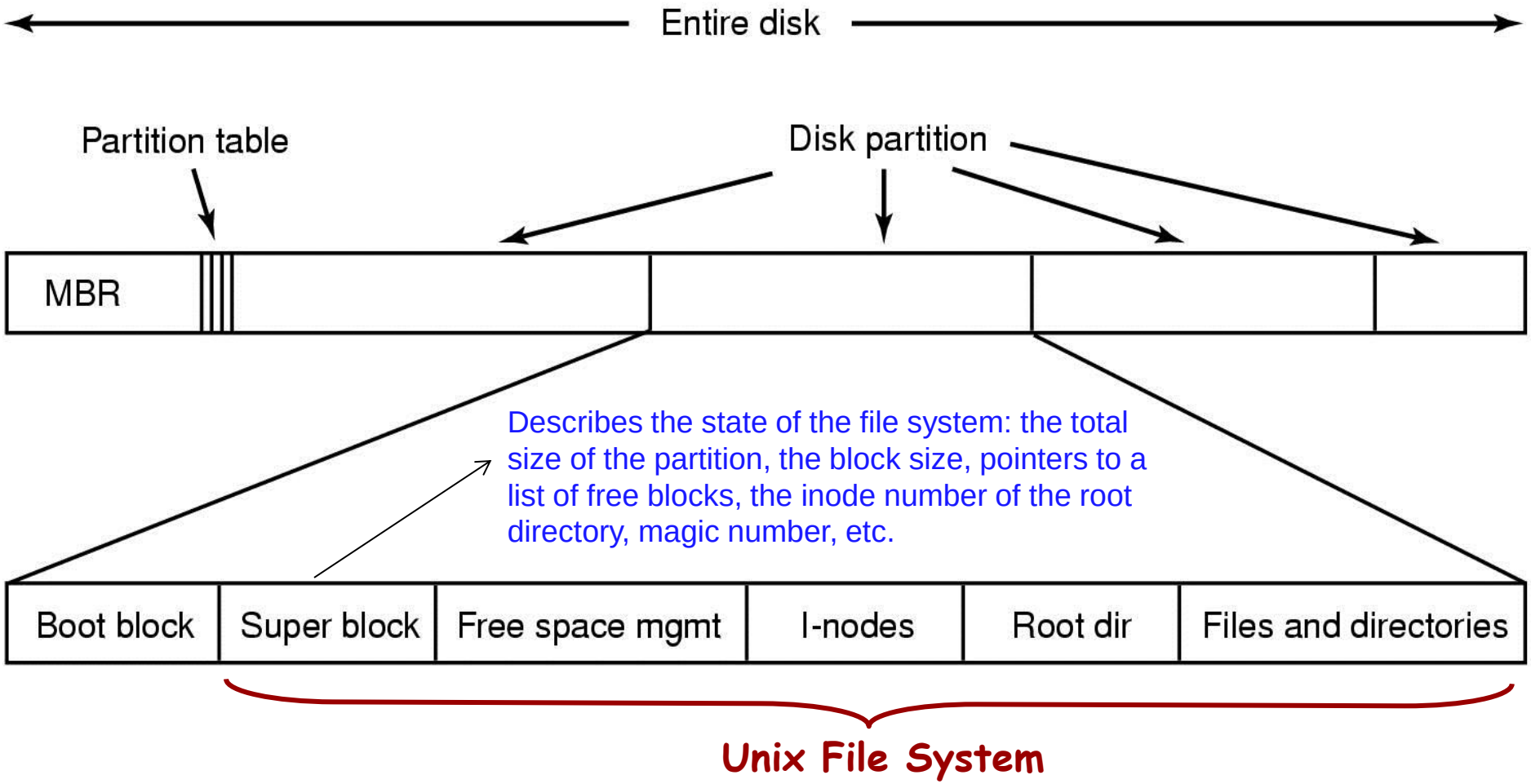


Figure 4-9 A possible file-system layout

File Bytes vs Disk Sectors

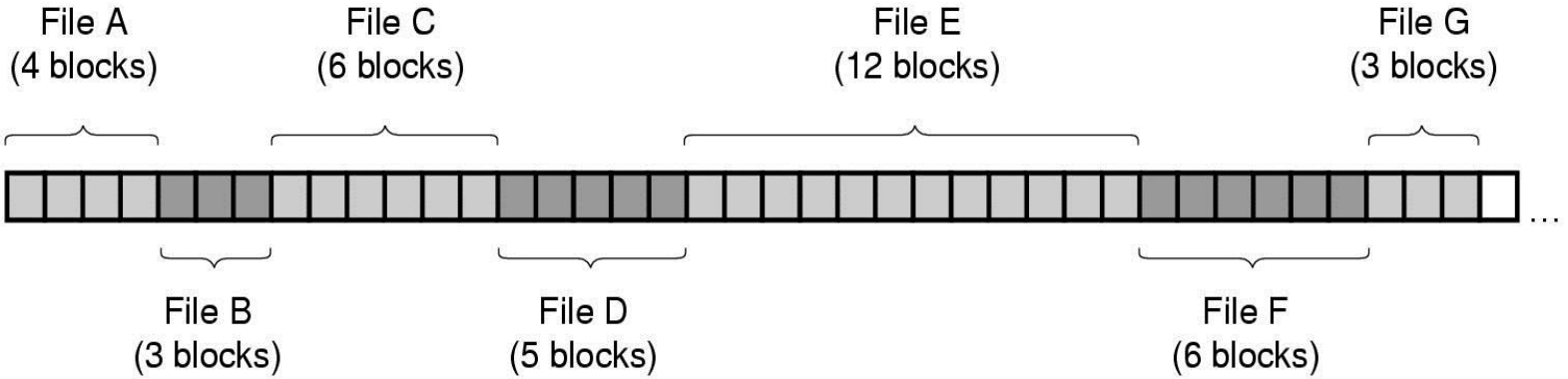
- Files are sequences of bytes
  - Granularity of file I/O is bytes.
- Disks are arrays of sectors (512 bytes)
  - Granularity of disk I/O is sectors.
  - Files data must be stored in sectors.
- File systems define a block size
  - block size = 2^n *sector size
  - Contiguous sectors are allocated to a block.
- File systems view the disk as an array of blocks.
  - Must allocate blocks to file.
  - Must manage free space on disk.

Allocation Methods

- Allocation Method
  - Refers to how disk blocks are allocated for files.
- Three Methods
  - Contiguous Allocation
  - Linked List Allocation
  - Indexed Allocation
- Key Metrics
  - Fragmentation (internal & external)?
  - Grow file over time after initial creation?
  - Fast to find data for sequential and random access?
  - Easy to implement?
  - Storage overhead?

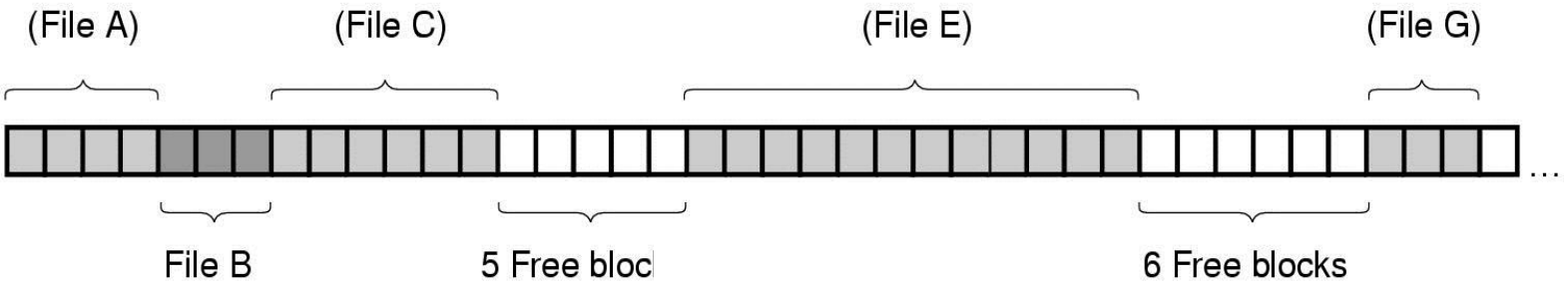
Contiguous Allocation (1)

- Each file occupies a set of contiguous blocks on the disk.
- Used by IBM VM/CMS.



(a)

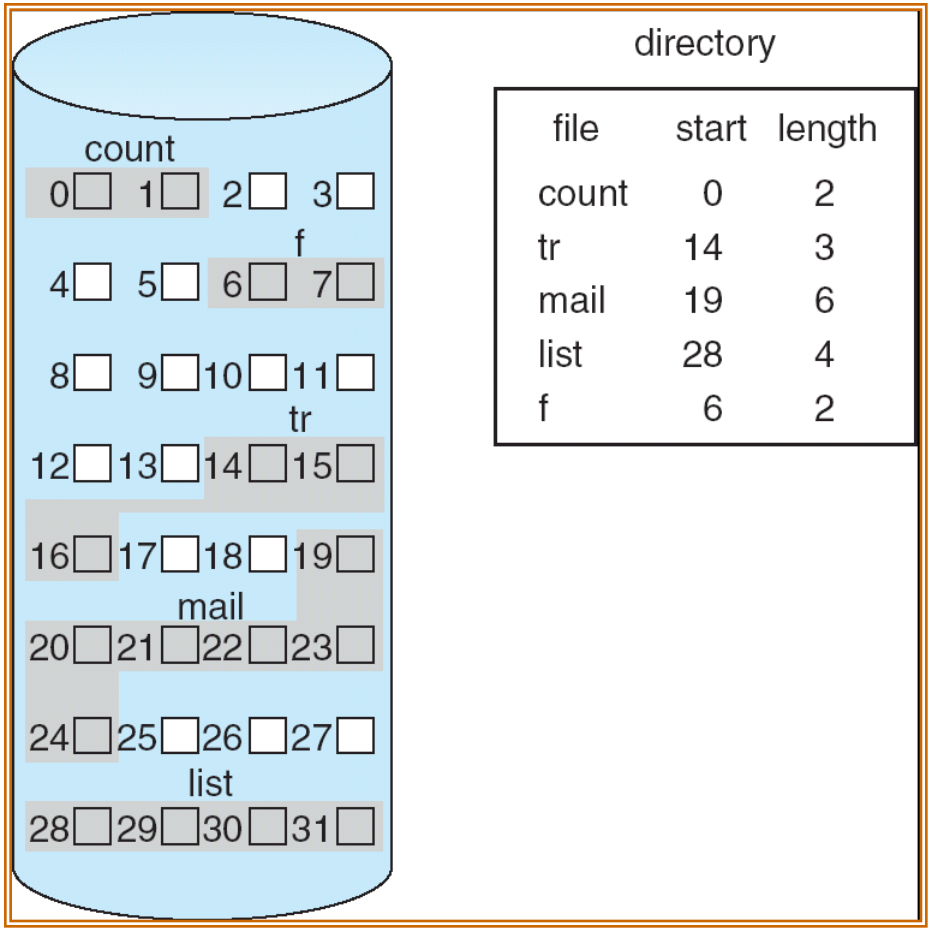
After deleting D and F...



(b)

Contiguous Allocation (2)

- Advantages
  - Simple to implement (Need only starting sector & length of file).
  - Easy to recover in event of system crash.
  - Performance is good (suits for sequential access and direct access).



Contiguous Allocation (3)

- Disadvantages
  - External Fragmentation
    - Will need periodic compaction (time-consuming).
    - Will need to manage free lists.
      - Must know a file's maximum possible size *... at the time it is created!*
  - Files cannot grow.
- Good for CD-ROMs, DVDs and other write-once optical media.
  - All file sizes are known in advance.
  - Files are never deleted.

Linked List Allocation (1)

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

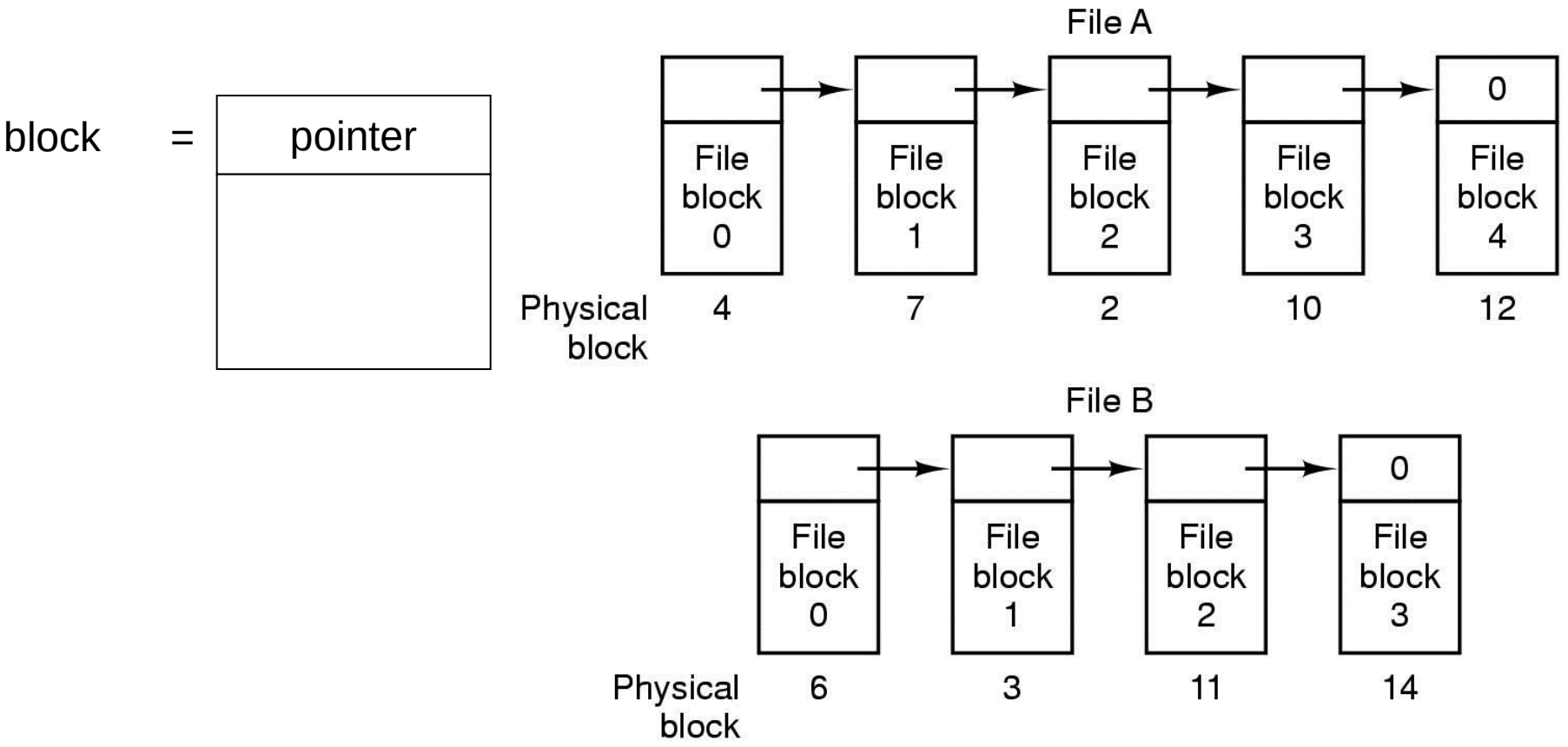
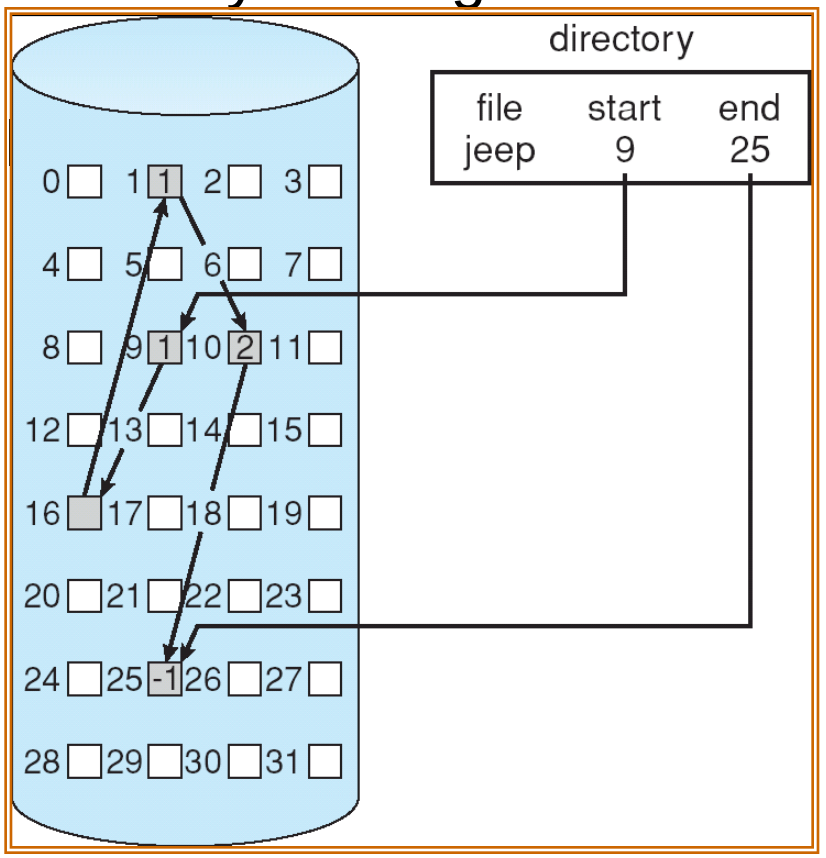


Figure 4-11 Storing a file as a linked list of disk blocks

Linked List Allocation (2)

- Advantages
  - No external fragmentation.
  - Directory entries are simple(Need only starting & end block number).
  - A file can continue to grow as available.
  - Good for sequential access.



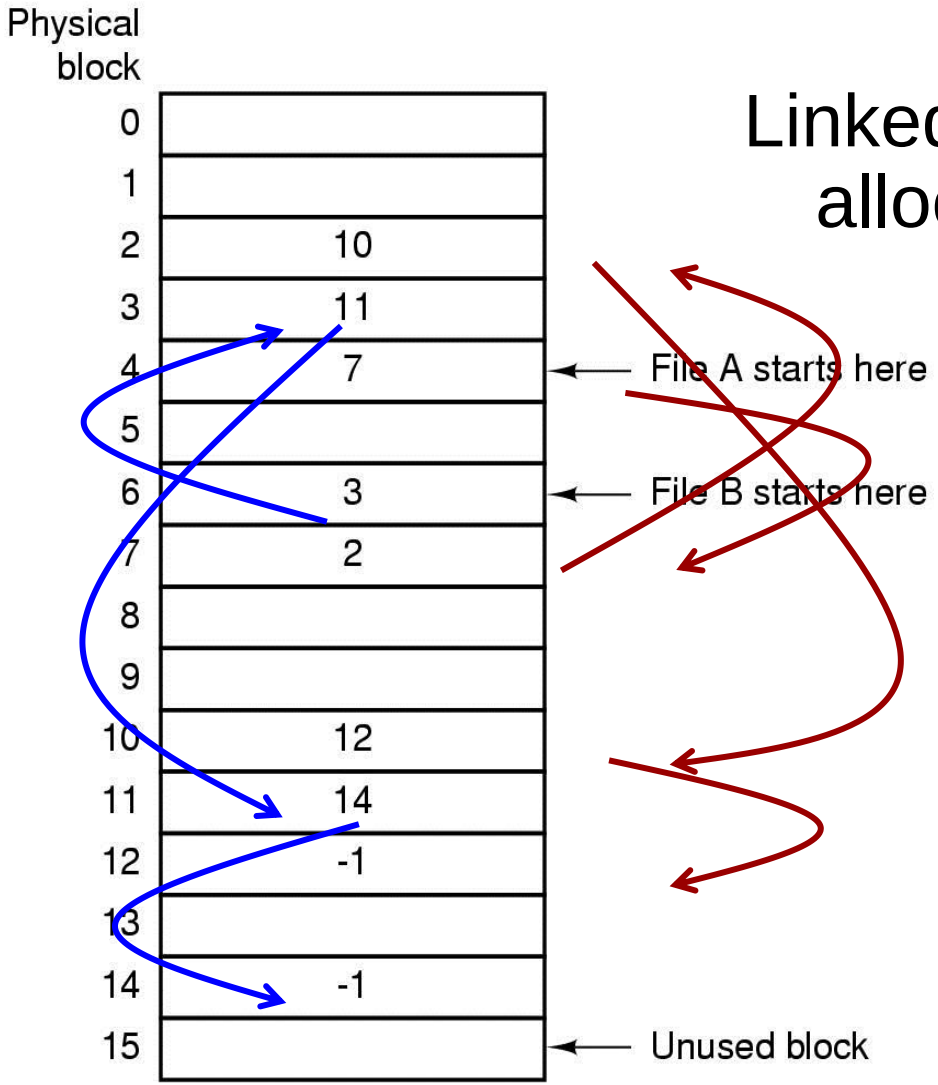
Linked List Allocation (3)

- Disadvantages
  - Random access into the file is slow!
  - The amount of data storage in a block is no longer a power of two.
  - Fragile: a pointer can be lost or damaged.

Linked List Allocation Using FAT(1)

- Important variation on linked list allocation method.
- File Allocation Table (FAT)
  - Keep a table in memory:
    - One entry per block on the disk.
    - Each entry contains the address of the "next" block.
    - End of file marker (-1).
    - A special value (-2) indicates the block is free.
- Used by OS/2 and MS-DOS.

Linked List Allocation Using FAT(2)

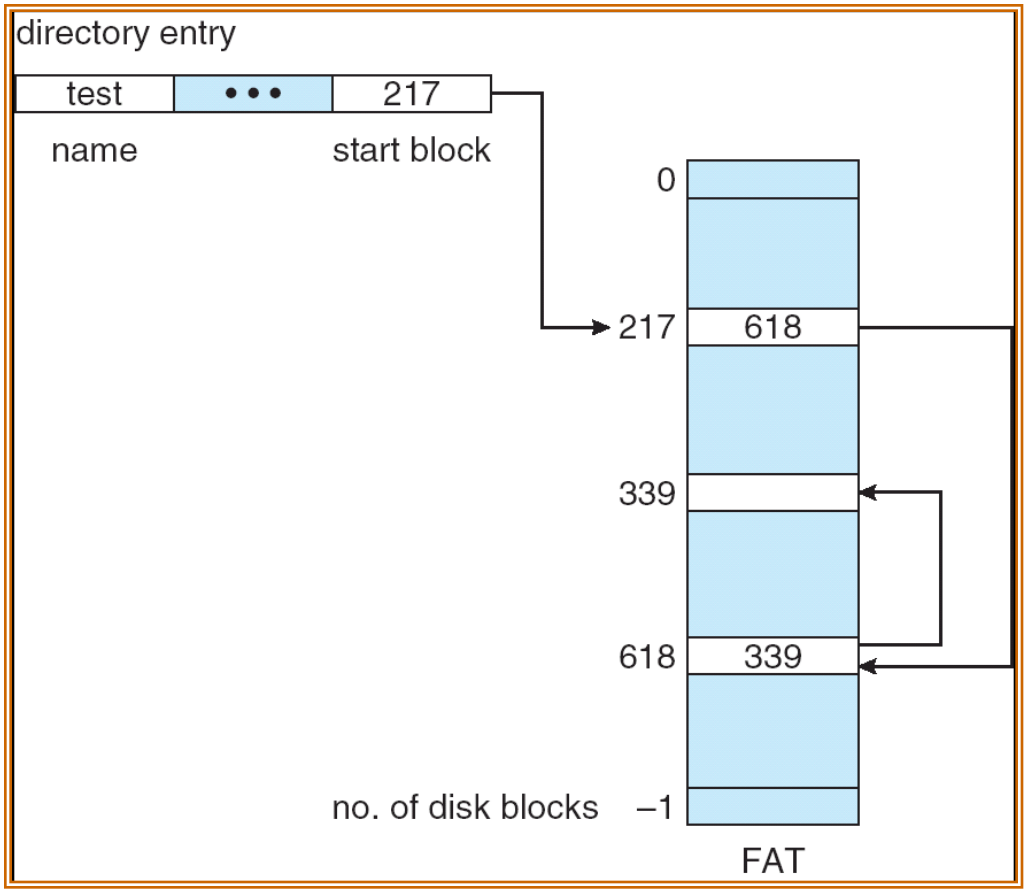


Linked list allocation using a file allocation table in RAM

Figure 4-12 Linked-list allocation using a file-allocation table in main memory.

Linked List Allocation Using FAT(3)

- Advantages
  - The entire block is available for data.
  - Random access...
    - Search the linked list (but all in memory)
  - Directory entry needs only one number.
    - Starting block number



Linked List Allocation Using FAT(4)

- Disadvantage
  - Entire table must be in memory all at once!
    - Example

      200 GB = disk size

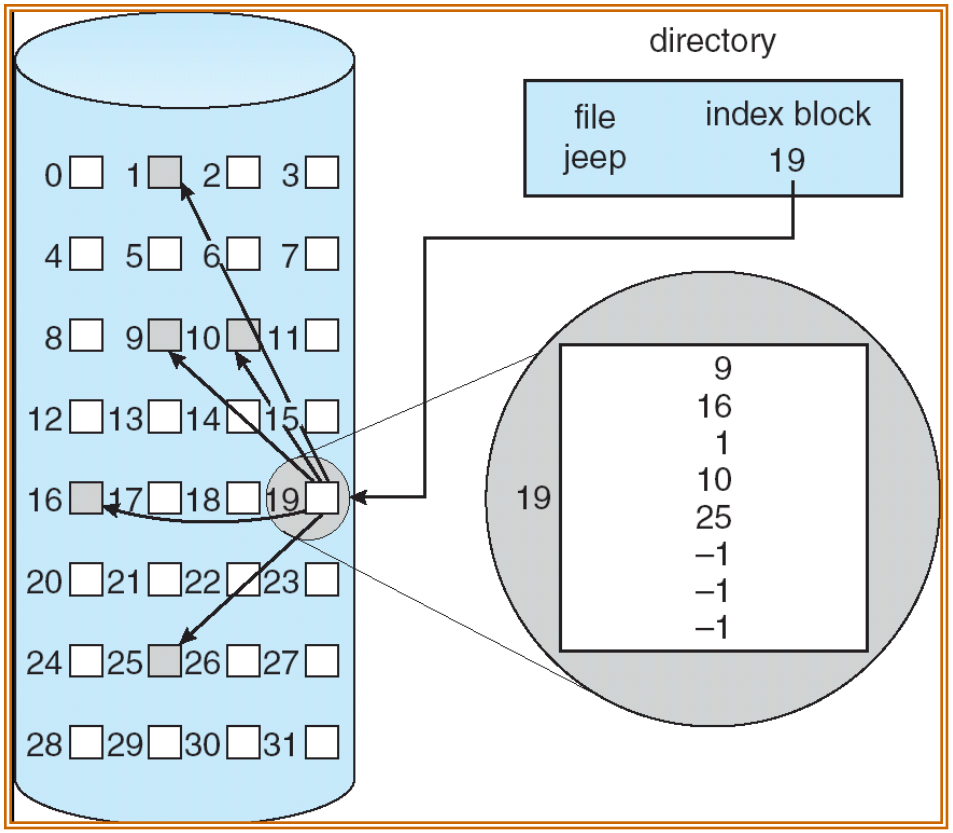
      1 KB = block size

      4 bytes = FAT entry size

      800 MB of memory used to store the FAT

Indexed Allocation (1)

- Bring all the pointers together into one location (index block or i-node)
- Each file has its own index block.
- Loading the index block into memory when the file is opened makes it available all times for random access.
- Example: Index block

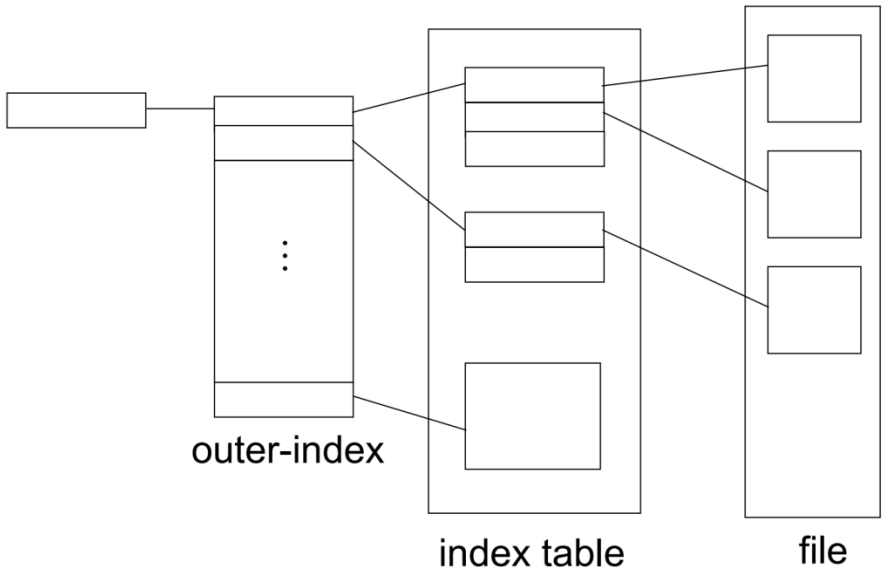


Indexed Allocation (2)

- Advantages
  - Easy to implement.
  - No external fragmentation.
  - Files can be easily grown with the limit of the index block size.
  - Fast lookups and random accesses.
- Disadvantages
  - File blocks may be scattered all over the disk.
    - Poor sequential access.
    - Needs defragmenter.
  - Space overhead for index blocks.
  - Needs to reallocate index block as the file size increases.

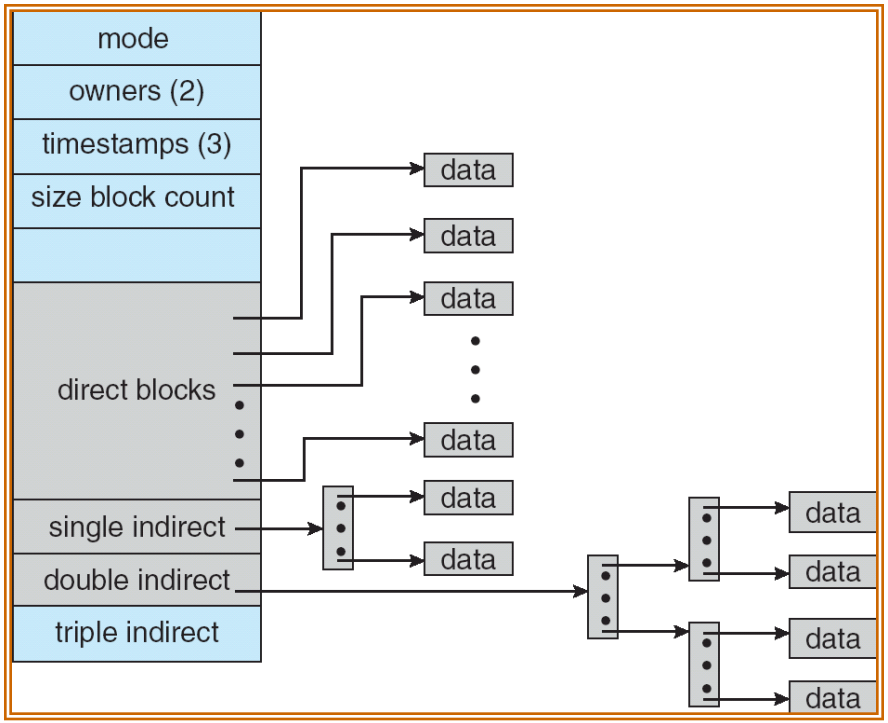
Multi-level Indexed Allocation (1)

- Question: If each i-node has room for a fixed number of disk addresses, what happens when a file grows beyond this limit?
- Certain index entries point to index blocks, as opposed to data blocks (e.g., Linux ext2).
- Block index has multiple levels.



Multi-level Indexed Allocation (2)

- Example
  - A ***single indirect block*** contains pointers to data blocks.
  - A ***double indirect block*** contains pointers to single indirect blocks.
  - A ***triple indirect block*** contains pointers to double indirect blocks.



Multi-level Indexed Allocation (3)

- Advantages
  - No external fragmentation.
  - Files can be easily grown with much larger limit compared to one-level index.
  - Fast random access.
    - Use index block.
- Disadvantages
  - Implementation can be complex.
  - Large space overhead (due to index).
  - Sequential access may be slow.
    - Must allocate contiguous block for fast access.

Implementing Directories (1)

- Directories are stored like normal files.
  - Directory entries are contained inside data blocks.
- The file system assigns special meaning to the content of these files.
  - A directory file is a list of directory entries.
  - In Unix/Linux, a directory entry contains file name, attributes, and the file's i-node number.
    - Maps human-oriented file name to a system-oriented name.

Implementing Directories (2)

- The Location of File Attributes
  - In the directory entry
    - Fixed size entries.
    - Disk addresses and attributes in directory entry.
    - Used by MS-DOS/Windows.

games	attributes
mail	attributes
news	attributes
work	attributes

Figure 4-14 (a) A simple directory, fixed size entries, disk addresses and attributes in directory entry

Implementing Directories (3)

- The Location of File Attributes (ctd.)
  - In the separate data structure (e.g., i-node)
    - Directory entry contains
      - File Name
      - i-node Number
    - i-node contains file attributes.
    - Used by Unix.

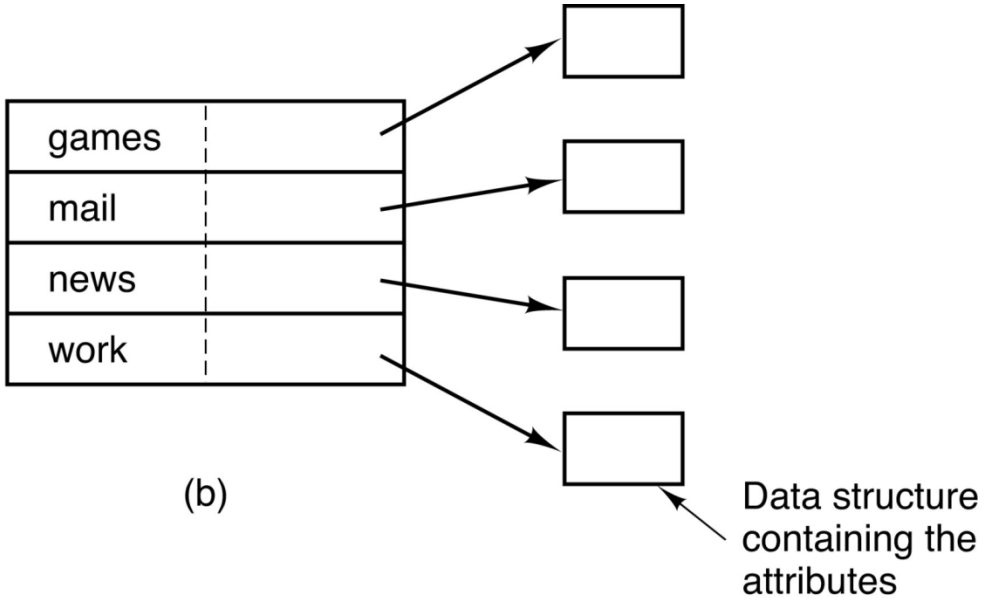


Figure 4-14 (b) Directory in which each entry just refers to an i-node

Implementing Directories (4)

- File Names
  - Short, Fixed Length Names
    - MS-DOS
      - File name: 1-8 chars, file extension:1-3 chars
      - Each directory entry has 11 bytes for the name.
    - Unix V7
      - Max 14 chars.
  - Variable Length Names
    - Unix (today)
      - Max 255 chars.
      - Directory structure gets more complex.

Implementing Directories (5)

- What if files have large, variable-length names?
- Handling Long File Names
  - Solution1: Set a limit on file name length, use one of the designs of figure 4-14.
    - Typically 255 chars.
    - Simplest.
    - Waste a great deal of directory space.

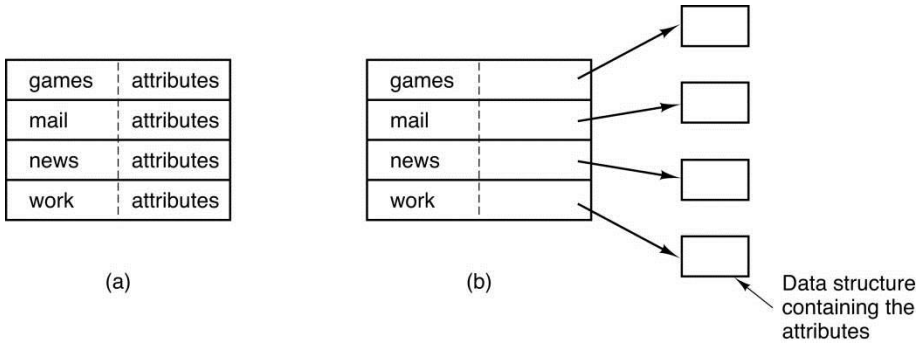


Figure 4-14

Implementing Directories (6)

- Handling Long File Names (ctd.)
  - Solution 2: Directory entry comprises fixed and variable portion.
    - Fixed part starts with entry size, followed by attributes.
    - Variable part has the file name.
    - Save space.
    - Holes on removal, need compaction.
    - A single directory entry may span multiple pages, so a page fault may occur while reading a file name.

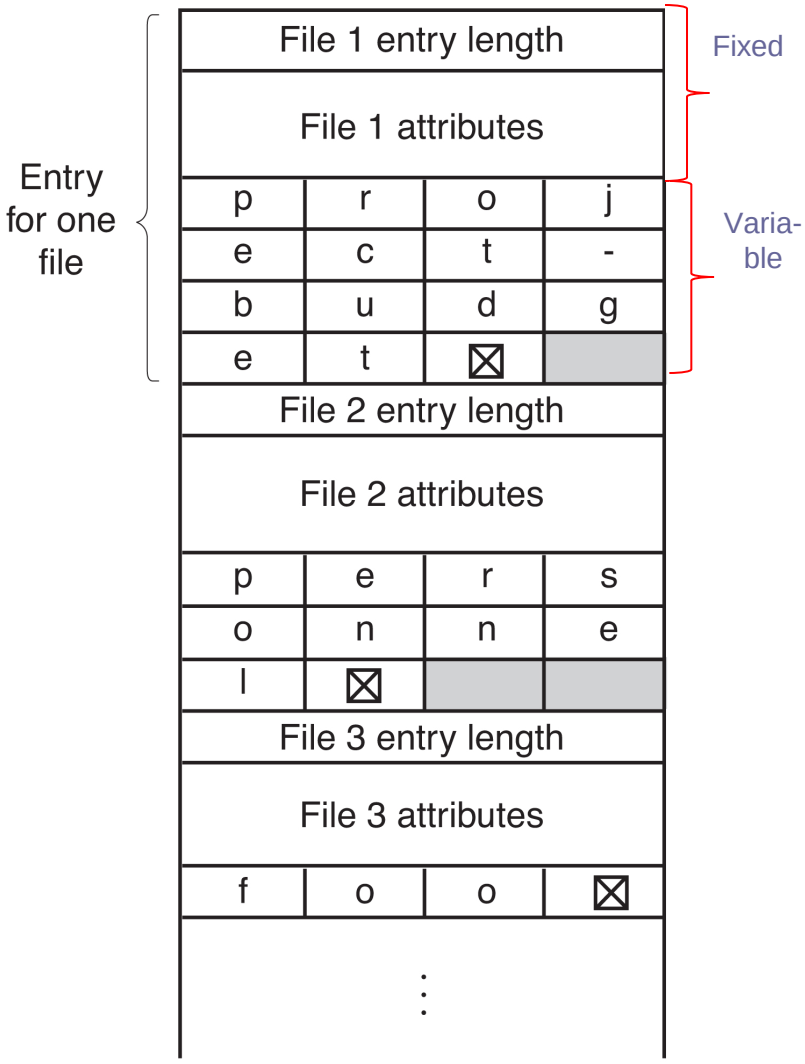


Figure 4-15 (a) In line

Implementing Directories (7)

- Handling Long File Names (ctd.)
  - Solution3: Directory entries are fixed in length, pointer to file name in heap.
    - Easy removal, no compaction.
    - No filler characters are needed after file names.
    - The heap must be managed.
    - Page faults can still occur while processing file names.

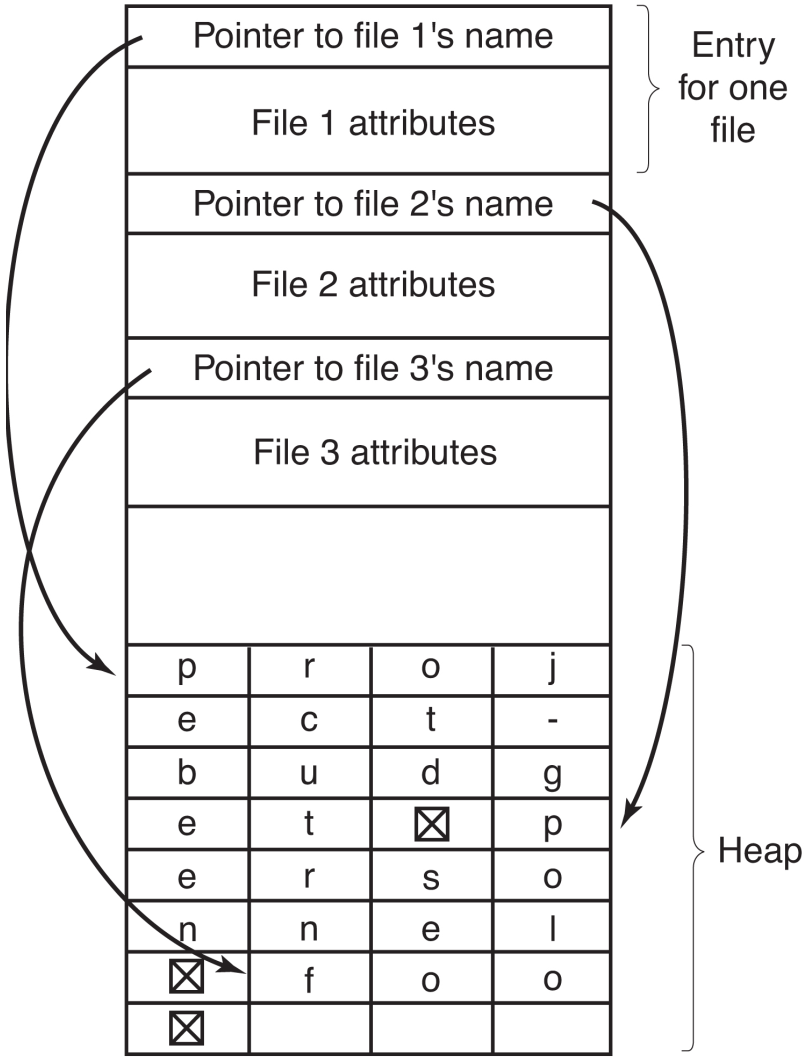
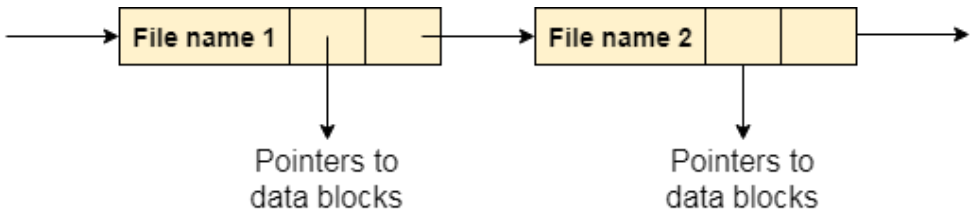


Figure 4-15 (b) In a heap

Implementing Directories (8)

- Directory Implementation
  - Linear List
    - All the files in a directory are maintained as singly linked list. Each file contains the pointers to the data blocks which are assigned to it and the next file in the directory.
    - When a new file is created, then the entire list is checked whether the new file name is matching to a existing file name or not. In case, it doesn't exist, the file can be created at the beginning or at the end.



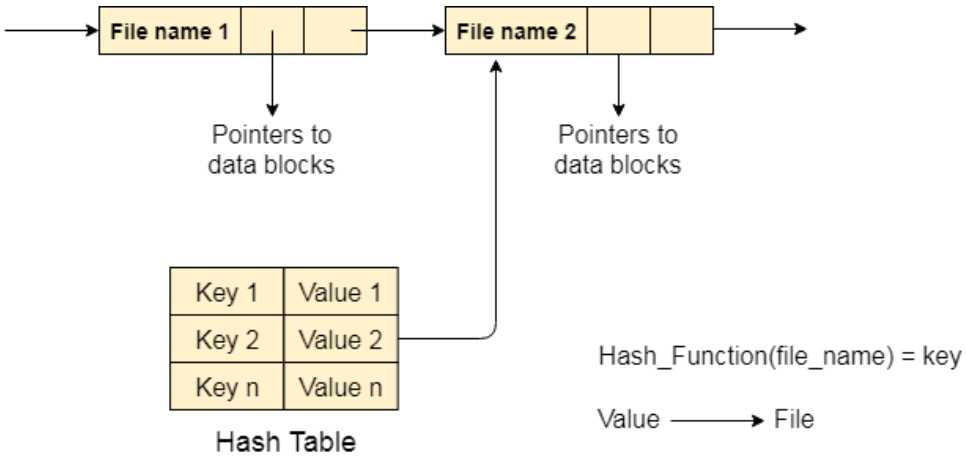
Linear List

Implementing Directories (9)

- Directory Implementation (ctd.)
  - Linear List (ctd.)
    - Requires a linear search to find a particular entry.
    - Simple to program but time-consuming to execute.
  - Hash Table
    - This approach suggests to use hash table along with the linked lists.
    - A key-value pair for each file in the directory gets generated and stored in the hash table. The key can be determined by applying the hash function on the file name while the key points to the corresponding file stored in the directory.

Implementing Directories (10)

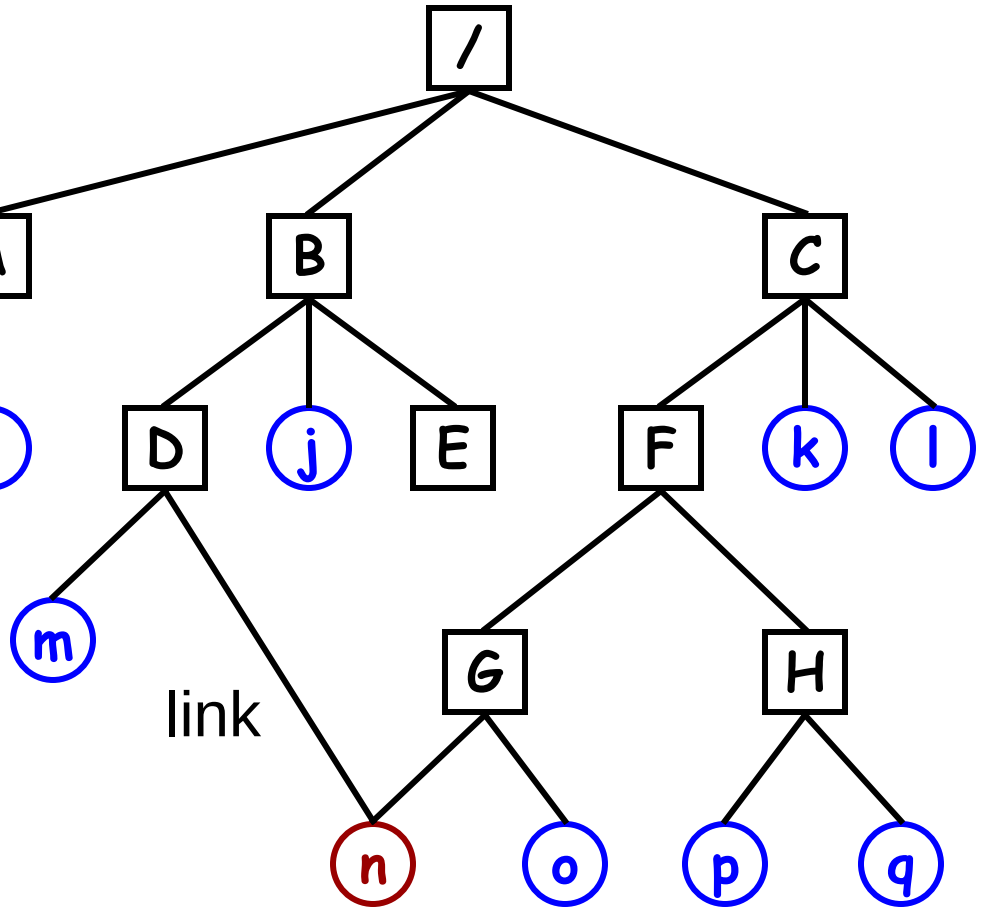
- Directory Implementation (ctd.)
  - Hash Table (ctd.)



  - Hashing can greatly decrease the directory search time.
  - Major difficulties with a hash table are its fixed size and dependence on the hash function.
  - Handle collisions – situations where two file names hash to the same location

Shared Files (1)

- One file appears in several directories.
- Tree → DAG (Directed Acyclic Graph)



What if the file changes?
New disk blocks are used.
Better not store this info in the directories!!!

Shared Files (2)

- Link
  - The connection between a directory and the shared file is called a link.
- In Unix
  - Hard Links
    - Both directories point to the same i-node.
    - *ln file link*
  - Symbolic Links (Soft Links)
    - One directory points to the file's i-node.
    - Other directory contains the "path".

Shared Files (3)

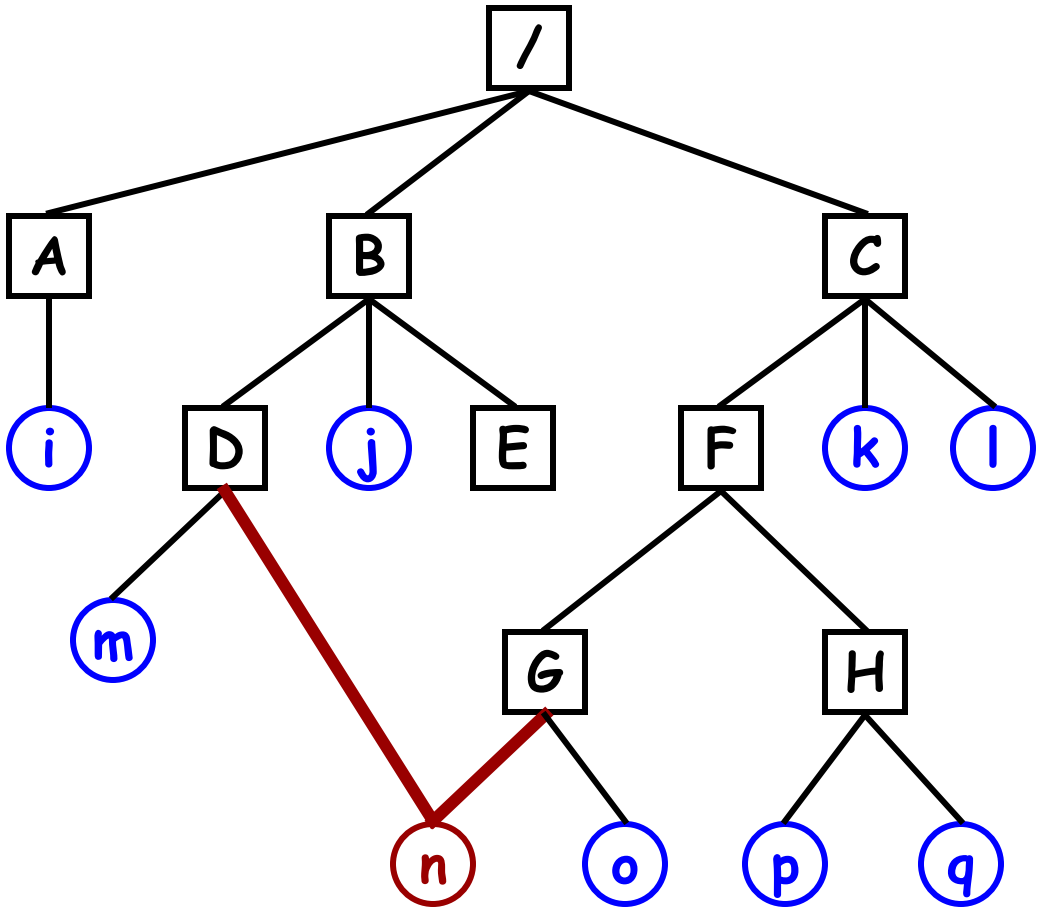
- Hard Links
  - Assume i-node number of "n" is 45.

Directory "D"

"m"	123
"n"	45
⋮	⋮

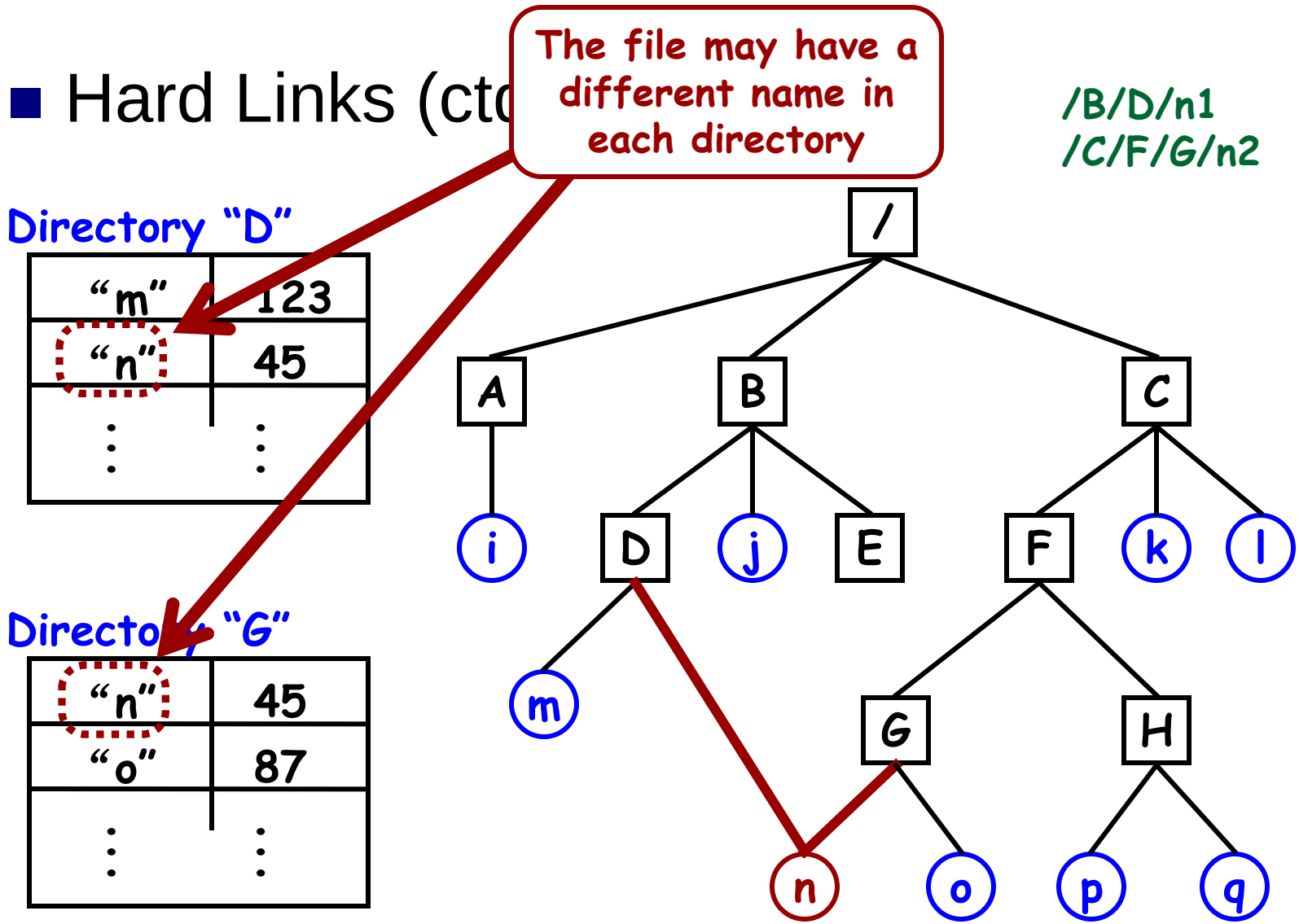
Directory "G"

"n"	45
"o"	87
⋮	⋮



Shared Files (4)

- Hard Links (ctd)



Shared Files (5)

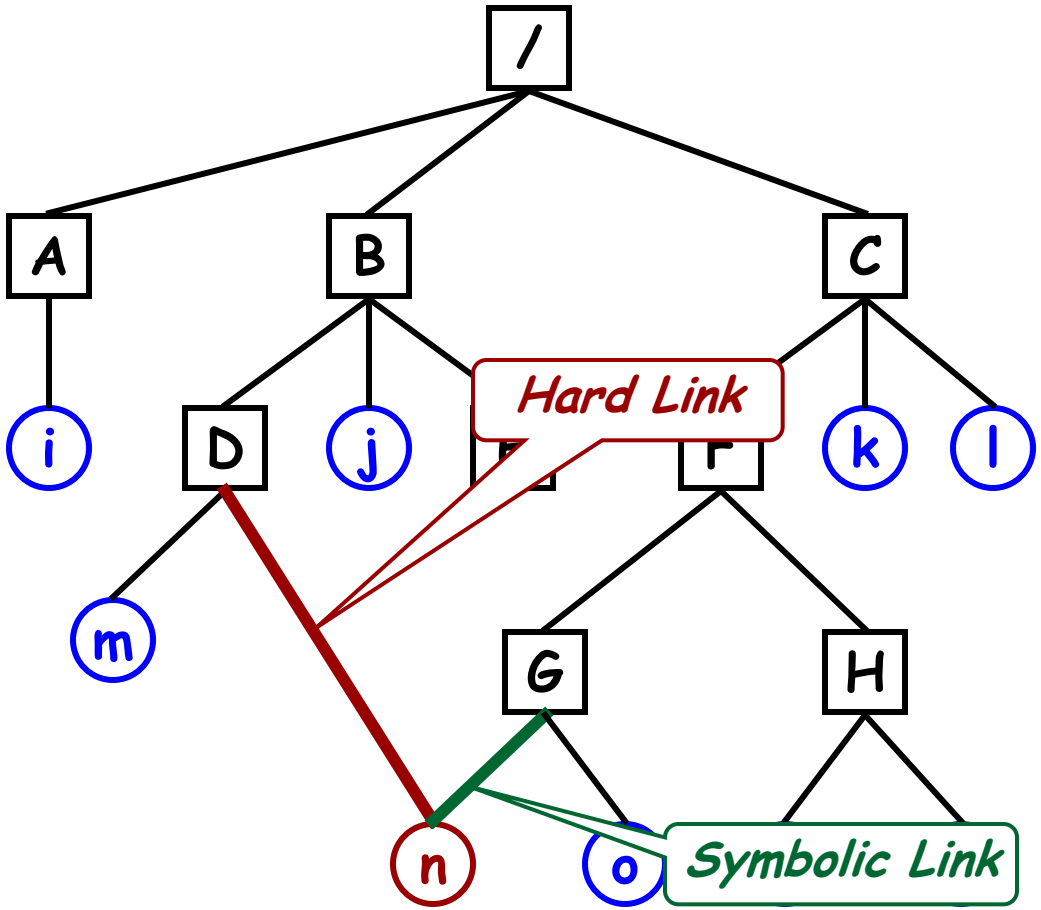
- Symbolic Links
  - Assume i-node number of "n" is 45.

Directory "D"

"m"	123
"n"	45
⋮	⋮

Directory "G"

"n"	/B/D/n
"o"	87
⋮	⋮



Shared Files (6)

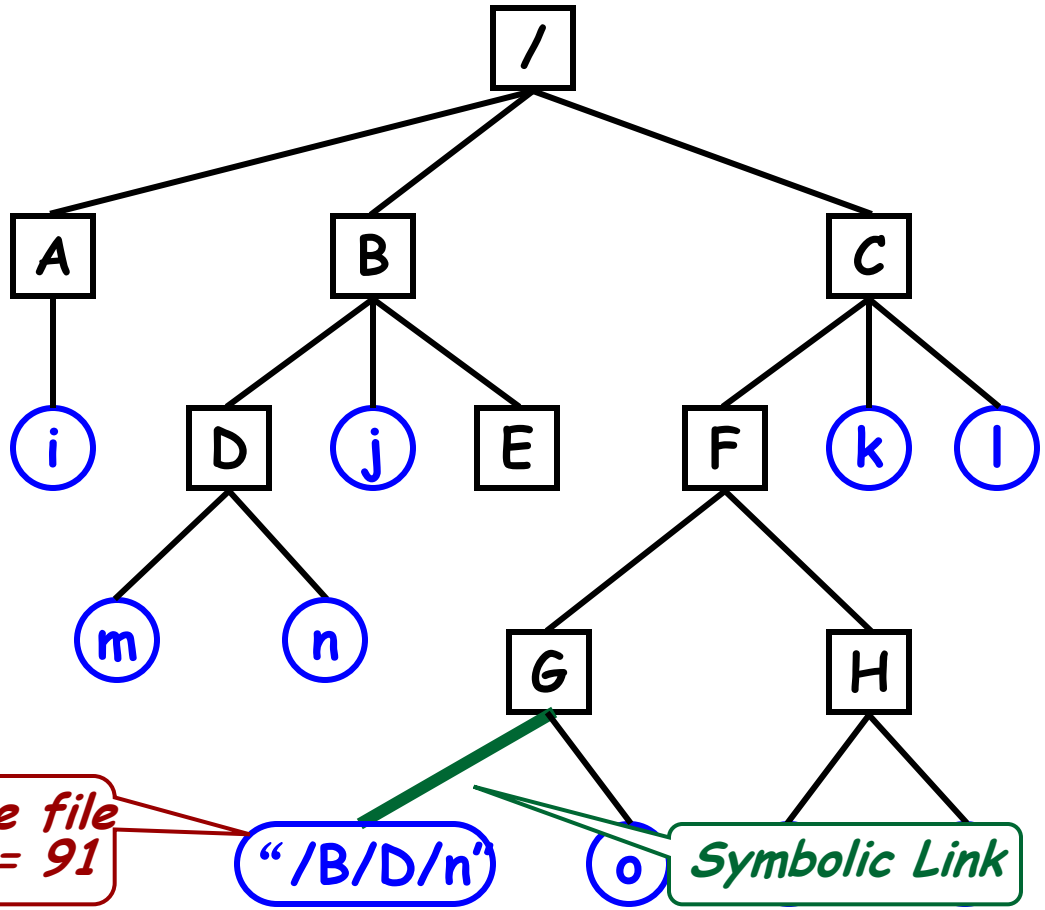
- Symbolic Links (ctd.)

Directory "D"

"m"	123
"n"	45
⋮	⋮

Directory "G"

"n"	91
"o"	87
⋮	⋮



Shared Files (7)

- When deleting a file
  - Directory entry is removed from directory.
  - All blocks in file are returned to free list.
  - *What about sharing???*
    - Multiple links to one file (in Unix)

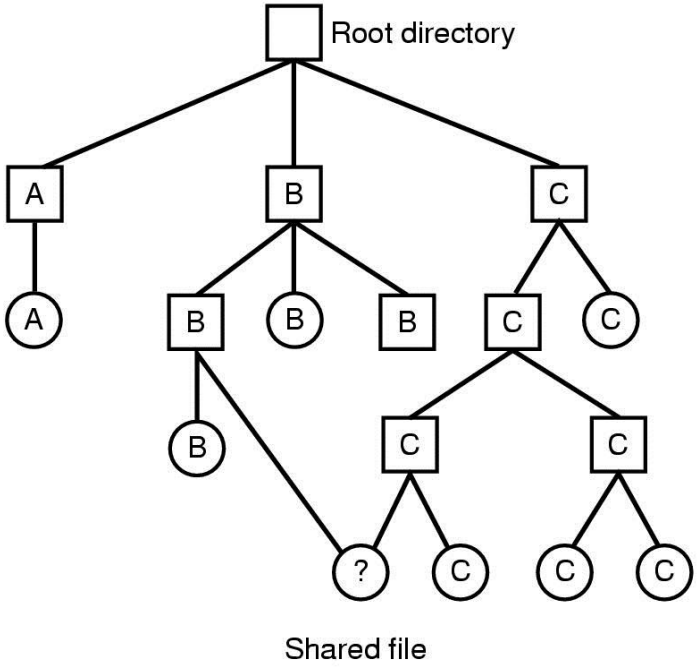


Figure 4-16 File system containing a shared file

Shared Files (8)

- Hard Links
  - Put a "reference count" field in each i-node.
  - Counts number of directories that point to the file.
  - When removing file from directory, decrement count.
  - When count goes to zero, reclaim all blocks in the file.
  - Example

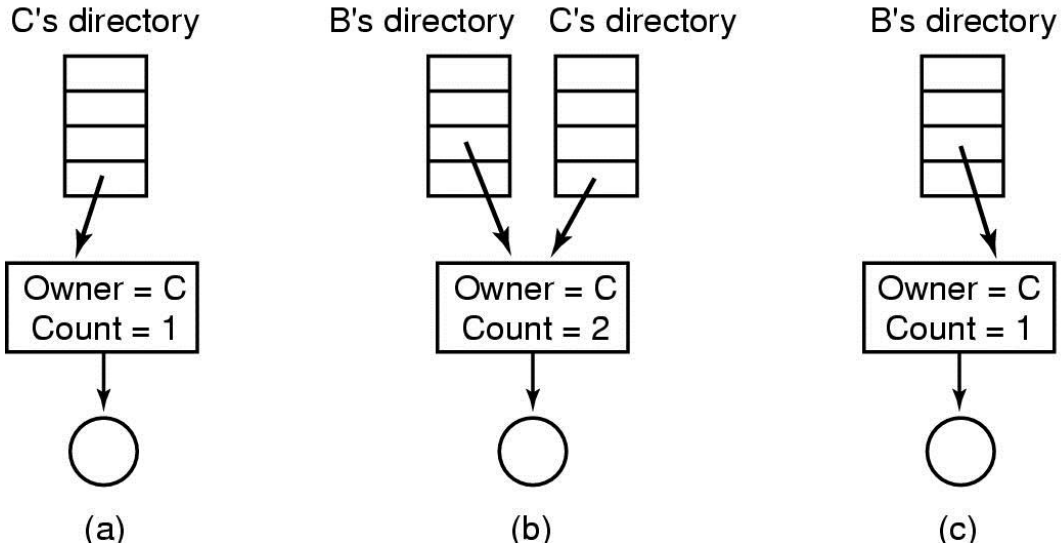


Figure 4-17
(a) Situation prior to linking
(b) After the link is created
(c) After the original owner removes the file

Shared Files (9)

- Symbolic Link
  - When removing the real file... (normal file deletion), symbolic link becomes "broken".
  - Removing a symbolic link does not affect the file at all.
- Limitations of Hard Link
  - Links cannot be established across file systems.
  - If one of the files is moved to a different file system, it is copied instead, and the link counts of both files adjusted accordingly.
  - Only superusers can create hard links to directories.

Shared Files (10)

- Drawbacks of Symbolic Link
  - Extra space on disk and extra inode to store the link file.
  - Extra time required for access to the original file: the link file has to be read first, then path followed to target file.
  - If the original file is moved to a different location, it can no longer be accessed via the symbolic link (dangling link).

Homework

- P333 12, 18, 20, 21