

CS534 Operating Systems – More on UNIX File Systems

File System Implementation

- The primary issue of file implementing file storage is how to keep track of file blocks

Strategies

1. Contiguous Allocation

- The idea is to store each file as a contiguous block of data on disk
- Thus, if a disk has 1k blocks, a 50kb file would be given 50 consecutive blocks

Advantages of Contiguous Allocation

- Simple to implement because keeping track of where the file's blocks are is reduced to knowing the address of the first block
- Excellent performance

Disadvantages of Contiguous Allocation

- Not feasible unless the maximum size of the file is known in advance
- Disk fragmentation and wasted space are imminent as not all blocks can necessarily be allocated contiguously

2. Linked List Allocation

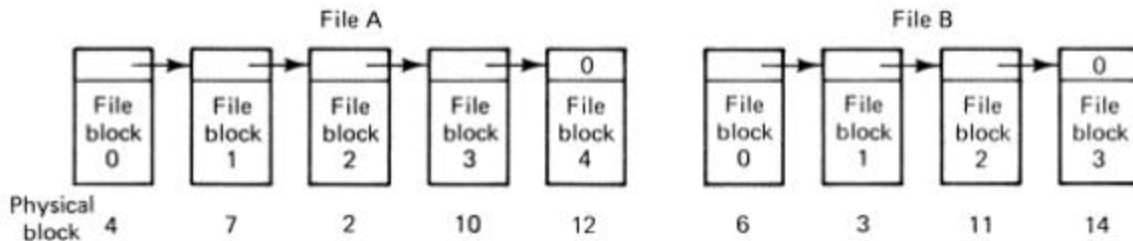
- A file is represented a linked list of blocks
- The first two (or more) bytes in each block contain a pointer to the next block, and the remainder of the space in the block is used for data

Advantages of Linked List Allocation

- Every disk block can be used
- Space is not lost to fragmentation

Disadvantages of Linked List Allocation

Random access is a relatively slow process



Example of a Linked List of File Blocks

3. Linked List Allocation with an Index

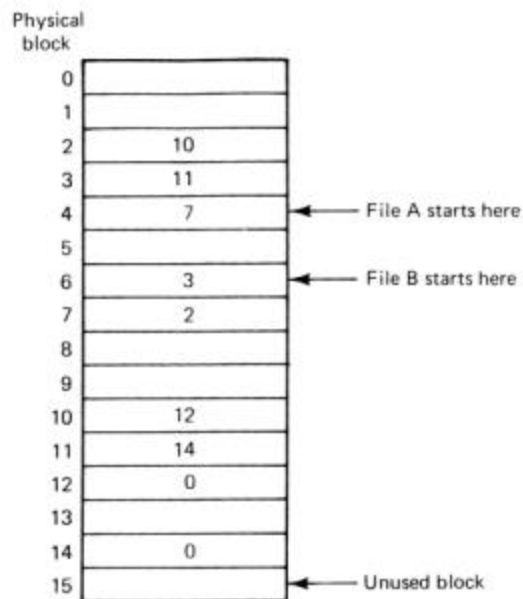
- In this strategy, the pointer is moved from the block itself to a table
- The table contains the location of all blocks in the file

Advantages of Linked List Allocation with an Index

- The entire block can be used for data
- Since the table is held in memory, random access is more efficient since references to disk are not needed

Disadvantages of Linked List Allocation with an Index

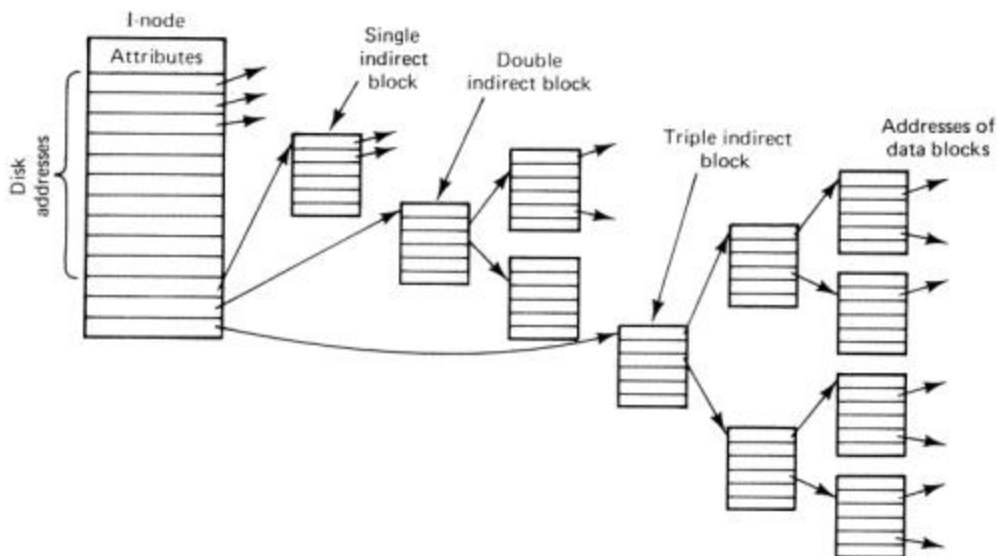
- For speed of lookup, table entries can contain multiple bytes potentially making the table large
- The table can be optimized for either size or speed...not both



Example of Linked List Allocation with an Index

4. i-nodes

- Incorporates the association of each file with a small table called an **i-node**, (**I**ndex **N**ode)
- An **i-node** contains the file attributes and disk addresses of the file's data blocks
- The **i-node** contains the first few data block addresses
- For **small files**, this means all necessary information about the file is contained in the i-node and can be fetched from disk to RAM when the file is opened
- For **larger files**, one of the addresses in the i-node is the address of a **single indirect block**
- The **single indirect block** contains additional addresses of file data blocks
- When this is not enough, another address in the i-node contains the address of a **double indirect block** that contains a list of the addresses of **single address blocks**
- Each single indirect block can contain pointers to **several hundred data blocks**
- For huge files, **triple indirect blocks** can be used

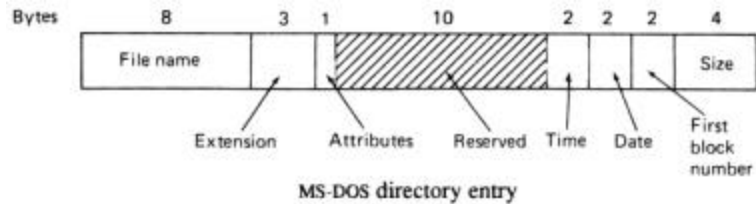


Example of Allocation Using I-Nodes

Implementing Directories

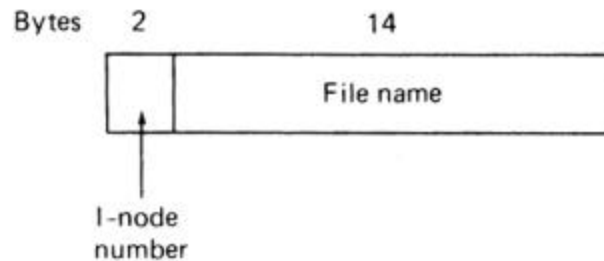
Directories in DOS

- A DOS directory entry contains the filename, extension, an attribute byte, 10 reserved bytes, time, date, file size, and a pointer to the location in the index table (FAT) containing the address of the first data block of the file



Directories in UNIX

- Each UNIX directory entry contains only the file name and the i-node number
- All information about the file type, size, times, dates, ownership, and disk blocks is contained in the i-node

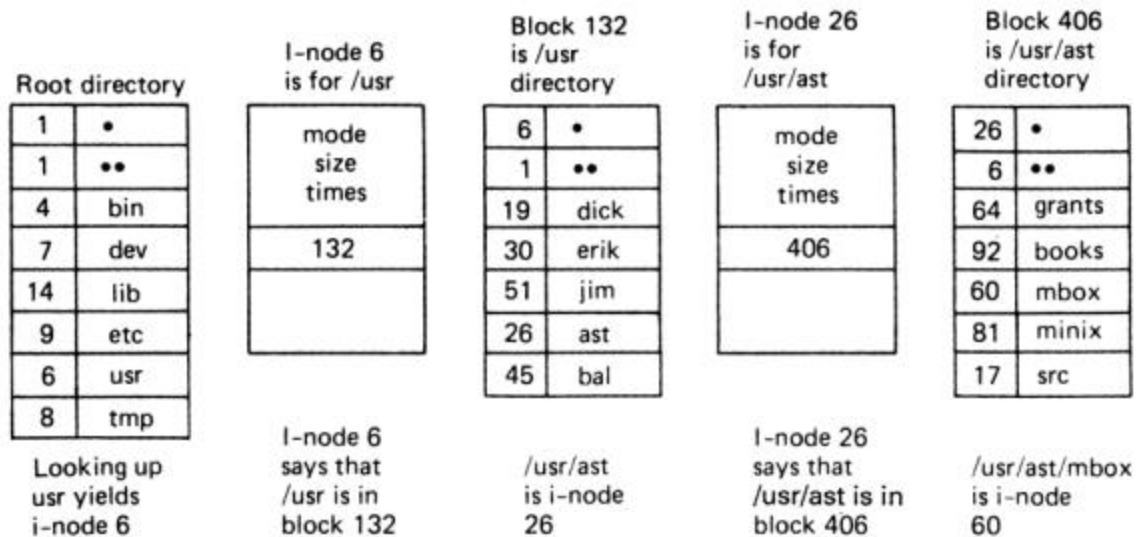


- When a file is opened, the file system takes the file name that was supplied, and locates the disk blocks belonging to the file

How UNIX Looks Up a Path Name

1. The file system locates the root directory. The i-node of the root directory is located in a fixed place on the disk
2. The root directory provides the location of the i-node number of the first part of the path
3. This i-node contains a pointer to the directory of the next portion of the path
4. The process continues until the file is found and its i-node read into memory

Example: Finding `/usr/ast/mbox`



The steps in looking up `/usr/ast/mbox`