File System

CS 256/456

Dept. of Computer Science, University of Rochester

Recap of the Last Class: Storage Systems

- Disk Structure
  - mechanical parts (cylinders, tracks, sectors) and how they move to access disk data
  - electronic part (disk controller main) exposes an one-dimensionally addressable set of blocks
- Disk Scheduling
  - goals: overall efficiency; fairness (no starvation)
  - FCFS, SSTF, SCAN, C-SCAN, LOOK

Deadline Scheduling in Linux

- A regular elevator-style scheduler similar to C-LOOK
- Additionally, all I/O requests are put into a FIFO queue with an expiration time (e.g., 500ms)
- When the head request in the FIFO queue expires, it will be executed next (even if it is not next in line according to C-LOOK).
- A mix of performance and fairness.

Concurrent I/O

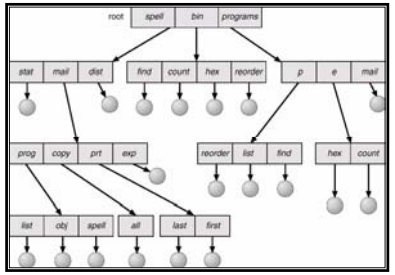
- Consider two request handlers in a Web server
  - each accesses a different stream of sequential data (a file) on disk;
  - each reads a chunk (the buffer size) at a time; does a little CPU processing; and reads the next chunk
- What happens?
- How to deal with it?
- Anticipatory scheduling [Iyer & Druschel, SOSP 2001]
  - at the completion of an I/O request, the disk scheduler will wait a bit (despite there is other work to do), in anticipation that a new request with strong locality will be issued. go ahead to schedule another request if no such new request appears before timeout.
  - default I/O scheduler in Linux 2.6

File System

- File system is the OS abstraction for storage resources
- What is a file?
  - File is a logical storage unit in the OS abstract interface for storage resources
- File structure (mostly concerns of user programs)
  - None - sequence of words, bytes
  - Simple record structure
    - lines/fixed length/variable length
  - Complex Structures
    - formatted document/relocatable load file

Directory

- What is a directory?
  - Part of the file system abstract interface
  - A logical "container" or "cabinet" for a group of files
  - A directory includes file attribute information for files residing in the directory
  - Both the directory structure and the files reside on disk.
- Directory structure:



File Attributes

- **Name** – file identification in human-readable form.
- **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 a file is kept in a file control block (in the directory structure), which is maintained on the storage device.

File Sharing and Protection

- Sharing of files on multi-user systems is desirable.
- Sharing must accompany a *protection* scheme.
  - In general, a protection scheme specifies whether any specific user can access any specific file.
  - What is it done right now?
  - Is it flexible enough?

File System Structure

- Abstractions:
  - file: storage unit
  - directory: container of files
- OS responsibility for file management:
  - Manipulation of files and directories.
  - Map files onto (nonvolatile) secondary storage - disks.

application programs

file system

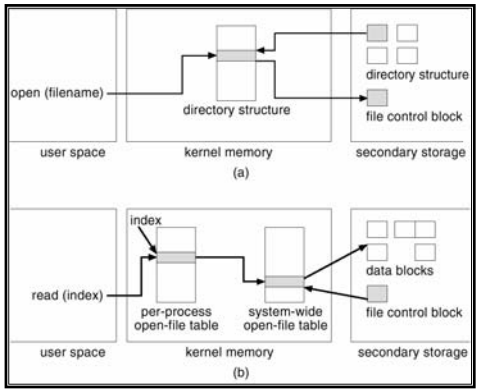
device I/O system

I/O devices, e.g., disks

File Access Model

- Sequential access
  - read all bytes/records from the beginning
  - used in many applications that access a whole file at a time
  - open()/read()/write()/close()
- Random access
  - bytes/records read in any order
  - essential for database systems
  - seek()

In-Memory File System Structures

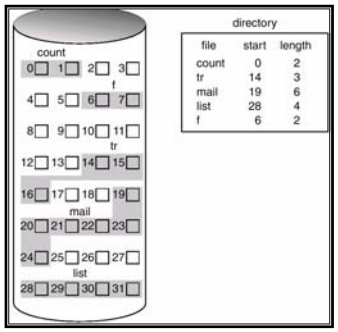


- System-wide info for each file (inode):
  - file open count
  - disk location
- Per-process info for each file (open file):
  - a pointer to the in-memory inode
  - file access offset
  - access rights

Implementation on the Disk

- Directory implementation
  - Linear list of file names with pointer to the data blocks.
    - time-consuming to search an item
  - Hash Table – linear list with hash data structure.
    - decreases directory search time
    - a little waste of space
- File allocation methods
  - How disk blocks are allocated for files
  - Contiguous allocation, linked allocation, indexed allocation
  - Metrics: access speed (sequential & random), space utilization

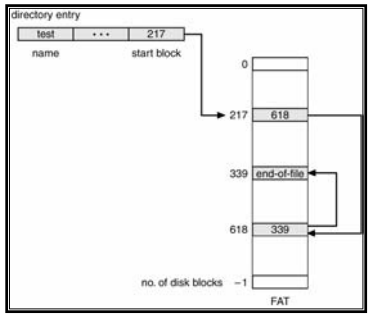
Contiguous Allocation



- Each file occupies a set of contiguous blocks on the disk.
- Advantage:
  - Simple – only starting location (block #) and length (number of blocks) are required.
  - Fast sequential/random access.
- Disadvantage:
  - Inflexible
  - External fragmentation

Linked Allocation

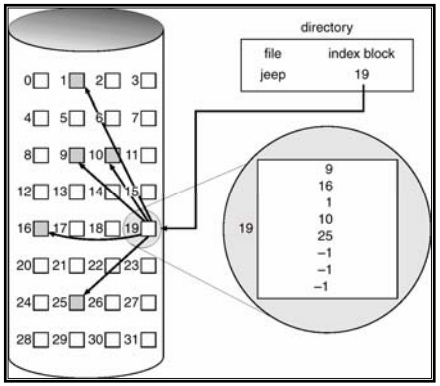
- Each file is a linked list of disk blocks
  - each block contains a next pointer
  - directory only needs to store the pointer to the first block
  - blocks may be scattered anywhere on the disk.



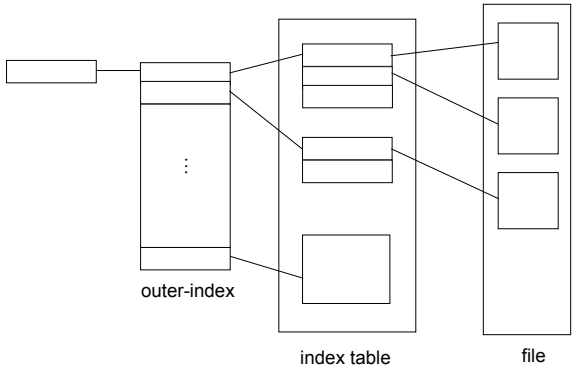
  - Advantage
    - Simple – need only starting address
    - Space efficient
  - Disadvantage:
    - Poor access speed (sequential & random)

Indexed Allocation

- Brings all pointers together into the *index block.*



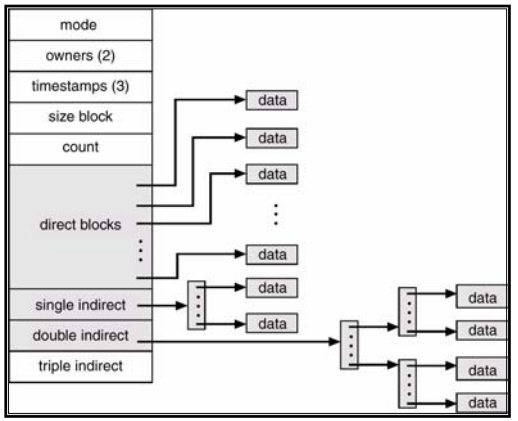
Multi-level Indexed Allocation



Indexed Allocation (pros and cons)

- Space efficiency
  - no external fragmentation
  - overhead of index blocks
- Access speed
  - random access
  - sequential access

UNIX (4K bytes per block)



Disclaimer

- Parts of the lecture slides contain original work of Abraham Silberschatz, Peter B. Galvin, Greg Gagne, Andrew S. Tanenbaum, and Gary Nutt. The slides are intended for the sole purpose of instruction of operating systems at the University of Rochester. All copyrighted materials belong to their original owner(s).