File Structure and Disk Scheduling (Topic 9)

홍 성 수

서울대학교 공과대학 전기 공학부
Real-Time Operating Systems Laboratory

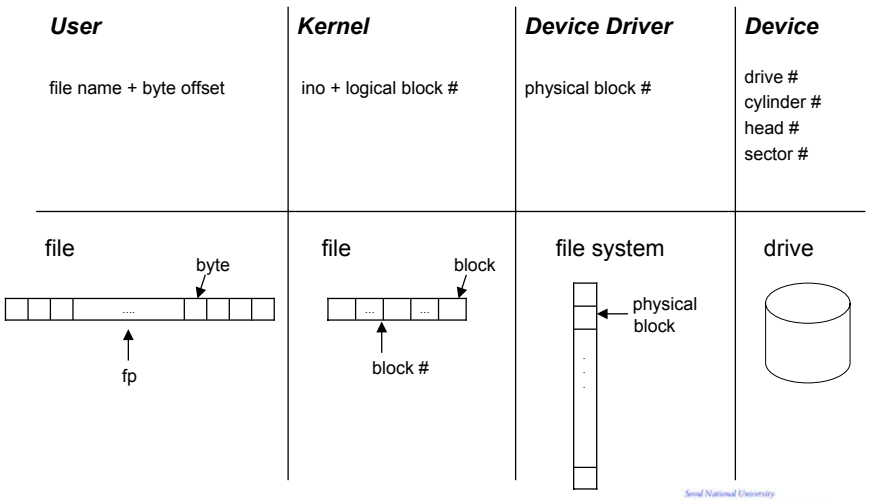
What is File?

- File: A named collection of bytes stored on disk.
  - From the OS'es standpoint, the file consists of a bunch of blocks stored on the device.
  - Programmer may actually see a different interface (e.g., records), but this doesn't matter to the file system (just pack bytes into disk blocks, unpack them again on reading).

Operations on Files

- What operations need the OS support on files:
  - Create and delete files.
  - Open files for reading and writing.
  - Seek within a file.
  - Read from and write to a file.
  - Close files.
  - Create directories to hold groups of files
  - List the contents of a directory
  - Removes files from a directory

Views on File



Addressing within a File (1)

Common addressing patterns:

- Sequential access: Information is processed in order, one piece after the other.
  - This is by far the most common mode.
  - Examples: Editor writes out new file.
    Compiler compiles it, etc.
- Random access: Can address any block in the file directly without passing through its predecessors.
  - Examples: The data set for demand paging.
    Databases.

Addressing within a File (2)

- Keyed access: Search for blocks with particular values.
  - Examples: Hash table.
    Associative database.
    Dictionary.
  - Usually not provided by an operating system.

Functions of File Systems (1)

- Modern file systems must address four general problems:
  - Disk Management: Efficient use of disk space, fast access to files, sharing of space between several users.
  - Naming: How do users select files?
  - Protection: All users are not equal.
  - Reliability: Information must last safely for long periods of time.

Functions of File Systems (2)

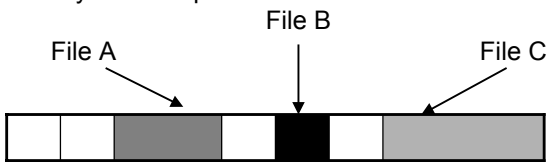
- Disk Management:
  - How should the disk sectors be used to represent the blocks of a file?
  - Need to keep track of where a file is on disk:
    - The data structure used to describe which sectors represent a file is called the file descriptor or the metadata.
    - File descriptors are often stored on disk along with the files.

Allocation Strategies (1)

- Contiguous allocation:
  - Allocate disk space to files like segmented memory.
    (Give each disk sector a number from 0 up.)
  - When creating a file, make the user specify its length, and allocate all the space at once.
  - Keep a free list of unused areas of the disk.
  - File descriptor contents: *Location* and *size*.
  - Advantages: Easy access, both sequential and random.
    Simple. Few seeks.

Allocation Strategies (2)

- Drawbacks
  - Horrible fragmentation will make large files impossible.
  - Hard to predict needs at file creation time.
- Example: IBM OS/360.
- Disk layout example:



- What happens if File C is 3 sectors?

Allocation Strategies (3)

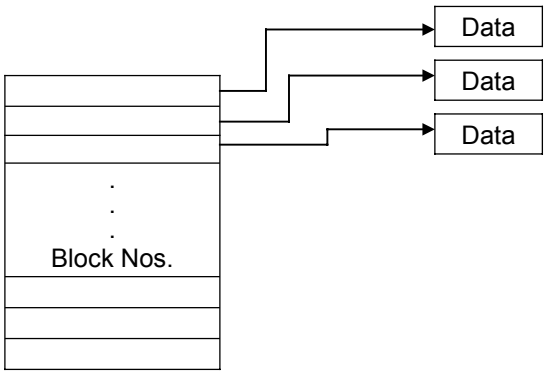
- Block-based allocation:
  - Disk blocks are allocated as they are used.
    - Minimal number of blocks are allocated to a file in an attempt to conserve storage space.
  - When a file is extended, blocks are allocated from a free block map.
    - Blocks are allocated in a random order
  - Each file must contain and maintain block allocation information, often called *metadata.*
  - Advantage: efficient disk space usage
  - Drawbacks:
    - Excessive seeks for sequential accesses
    - Extra disk IO to read and write the metadata

Allocation Strategies (4)

- File system metadata is always written synchronously to the storage device.
  - File size changes must wait for each metadata operation to complete.
  - Metadata operations significantly slow down the overall file system performance.

Allocation Strategies (5)

– Example: traditional Unix file system (UFS)



Block Allocation

Allocation Strategies (6)

- Extent-based allocation:
  - Allocates a large series of contiguous blocks each time the file exhausts the space available in its last *extent*.
  - When created, a file is allocated a large number of blocks (called *cluster*).
  - File system metadata is written when the file is first created.
    - Subsequent writes within the first allocation extent of blocks do not require additional metadata writes until the next extent is allocated.

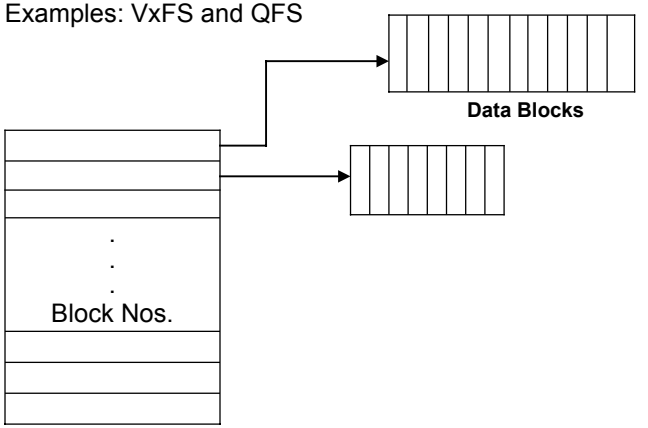
Allocation Strategies (7)

– Advantages:
  - Optimized disk seek patterns:
    – Efficient for sequential accesses
    – Grouping block writes into clusters allows file system to issue larger physical disk writes.
  - Smaller amount of file system metadata
    – Files with on a few very large extents require only a small amount of metadata

– Drawbacks:
  - Bad for random I/O
  - Disk space fragmentation

Allocation Strategies (8)

– Examples: VxFS and QFS



Extent Allocation

File Structures (1)

- Things to think about in designing a file structure:
  - Most files are small.
  - Much of the disk is allocated to large files.
  - Many of the I/O operations are made to large files.

    Thus, per-file cost must be low but large files must have good performance.

File Structures (2)

- Linked files:
  - Keep a linked list of all free blocks.
  - File descriptor contents: A pointer to first block.
  - In each block of file keep pointer to next block.

File Structures (3)

- Advantages:

  Files can be extended.

  No fragmentation problems.

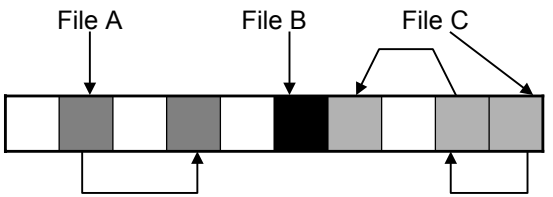
  Sequential access is easy: Just chase links.
- Drawbacks:

  Random access is virtually impossible.

  Lots of seeking, even in sequential access.

File Structures (4)

- Example: TOPS-10, Alto, sort of.
- Disk layout example:



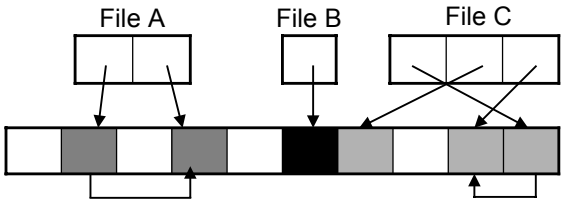
- How do you process (Read block 3)?
- How do figure out the file's size?

File Structures (5)

- Indexed files:
  - Keep an array of block pointers for each file.
  - File maximum length must be declared when it is created.
  - Allocate an array to hold pointers to all the blocks, but don't allocate the blocks. Then fill in the pointers dynamically using a free list.
  - Advantages:
    - Not as much space wasted by overpredicting.
    - Both sequential and random access are easy.
  - Drawbacks:
    - Still have to set maximum file size.
    - There will be lots of seeks.

File Structures (6)

- Disk layout example:



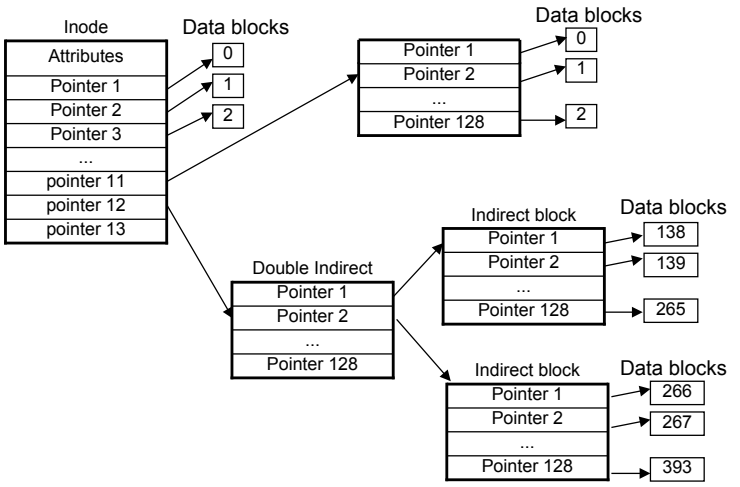
File Structures (7)

- Multi-level indexed files: 4.3 BSD solution:
  - File descriptor = 14 block pointers.
    - The first 12 point to data blocks.
    - The next one to an indirect block.
      - Contains 1024 more block pointers.
    - Last one to a doubly indirect block.
  - Maximum file length is fixed, but large.

File Structures (8)

- Advantages:
  - Simple, easy to implement
  - Incremental expansion.
  - Easy access to small files.
- Potential drawbacks:
  - Indirect mechanism doesn't provide very efficient access to large files: Up to 2 descriptor ops for each real operation.
  - Block-by-block organization of free list may mean that file data gets spread around the disk.

File Structures (9)



File Structures (10)

- Block (buffer) cache:
  - A pool of recently-accessed blocks is kept in main memory.
  - If the same blocks are referenced over and over (such as indirect blocks for large files), there's no need ever to read them from disk.
  - This solves the problem of slow access to large files.
  - How does a block cache compare with virtual memory?

File Structures (11)

- Keeping track of free blocks in UNIX: Use a bit map.
  - Just an array of bits, one per block.

    1 means block free, 0 means block allocated.

    For a 1200 Mbyte drive there are about 307000 4kbyte blocks, so bit map takes up 38400 bytes.

    Nowadays, can usually keep entire bit map in memory most of the time.

File Structures (12)

  - During allocation, try to allocate next block of file close to previous block of file. If disk isn't full, this will usually work well.
  - If disk becomes full, this becomes VERY expensive, and doesn't get much in the way of adjacency.

    Solution: Keep a reserve (e.g. 10% of disk), and don't even tell users about the reserve. Never let the disk get more than 90% full.
  - With multiple surfaces on disk, there are multiple optimal next blocks; with 10% of disk free, can almost always use one of them.

Unix File Systems

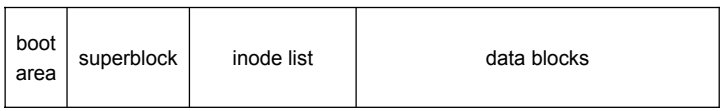
- Unix file systems have evolved:
  - System V file system (s5fs)
  - Berkeley fast file system (FFS) – often called UFS
  - Sun's network file system (NFS)
  - Sun's virtual file system (VFS)
  - Log-structured file system (LFS)

S5FS (1)

- File system structure:
  - File system resides on a single logical disk (partition).
  - Each file system is self-contained.
  - A partition is a linear array of disk blocks.
  - The size of a block is 512 bytes multiplied by some power of two (512, 1024, or 2048 bytes).
  - The physical block number – an index into this array – uniquely identifies a block on a given disk partition.

S5FS (2)

- On-disk layout
  - Boot area: may be empty.
  - Superblock: holds the metadata of the file system itself.
  - Inode list: fixed sized array of inodes
  - Data blocks: holds files, directories, and indirect blocks.



S5FS (3)

- Superblock
  - Size in blocks of the file system
  - Size in blocks of the inode list
  - Number of free blocks and indoes
  - Free block list
  - Free inode list

S5FS (4)

- Drawbacks
  - Reliability concern on superblock:
    - Each file system contains a single copy of its superblock.
  - Low performance
    - Accessing a file requires reading the inode and the file data.
    - Inode is allocated far away from its data.
    - No attempt to group related inodes (files in the same directory).
      - Example: ls –l
  - Suboptimal disk allocation
    - After heavy use, the order of blocks in the free block list is completely random.

S5FS (5)

- Limitations on functionality
  - Max 14 characters for file names
  - Max 65535 indoes per file system

FFS (1)

- On-disk organization
  - A formatted partition holds a self-contained file system.
  - FFS divides the partition into one or more cylinder groups.
    - A cylinder group contains a small set of consecutive cylinders.
    - Allows Unix to store related files in the same cylinder group, thus minimizing disk head movements.
  - Superblock is divided into two structures
    - FFS superblock
    - Per-cylinder group superblock
  - FFS superblock
    - No., sizes, and locations of cylinder groups
    - Block size, total no of blocks and inodes
    - Never change unless the file system is rebuilt

FFS (2)

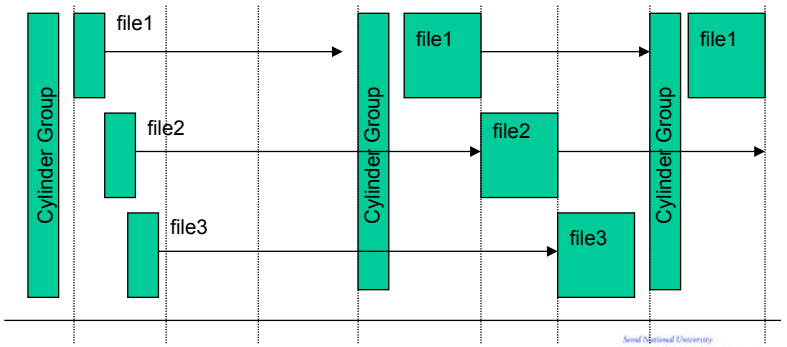
- Per-cylinder group superblock
  - Data structure for the particular cylinder group
  - Duplicate copy of the superblock
    - FFS maintains these duplicates at different offsets in each cylinder group in such a way that no single track, cylinder, or platter contains all copies of the superblock.
- Blocks and fragments
  - FFS blocks are two's power of sectors.
  - Blocks are subdivided into fragments.
  - Fragments are the unit of allocation.
    - Fragments are allocated in a contiguous manner.

FFS (3)

- Allocation policies
  - Attempts to place the inodes of all files of a single directory in the same cylinder group.
  - Creates each new directory in a different cylinder group from its parent, so as to distribute data uniformly over the disk.
  - Tries to place the data blocks of a file in the same cylinder group as the inode.

FFS (4)

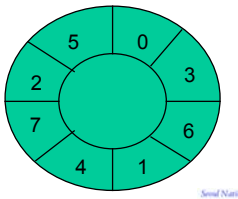
  - Changes the cylinder group when the file size reaches 48KB and again at every megabyte.
    - The 48KB mark was chosen because the inode's direct block entries describe the first 48KB.



FFS (5)

  - Allocate sequential blocks of a file at rotationally optimal position.
    - ***rotdelay*** factor or disk's ***interleave***
      - The time it takes for the kernel to issue the next read and computes the number of sectors the disk head passes over in that time.
    - Blocks are interleaved on disk such that consecutive logical blocks are separated by ***rotdelay*** blocks on that track.

8 sectors/track
rotdelay=2



FFS (6)

- Functionality enhancements
  - Long file names
    - The max size of the filename is 256 characters.
  - Symbolic links
  - Disk quota
    - Applies to both inodes and disk blocks.
    - Have both a soft limit and a hard limit.

FFS (7)

- Comparison with S5FS
  - Performance gain
    - Read throughput:
      - 29KB/s (1KB blocks in S5FS) vs 221KB/s (4KB blocks and 1KB fragments in FFS)
    - Write throughput
      - 48 KB/s vs 142 KB/s
  - Disk space wastage
    - With free space reserve of 5%, the percentage of waste in S5FS with 1KB blocks approximately equals that in FFS with 4KB blocks and 512B fragments.

FFS (8)

- Limitations
  - Performance
    - Problem in sequential reads
    - Predominance of writes
  - Slow crash recovery
    - Fsck causes unacceptable crash recovery time.
  - Limited security policies and mechanisms
  - Size restrictions

LFS (1)

- Crash recovery
  - Metadata updates must be ordered appropriately.
    - Deleting a file in a directory
      1. Remove the directory entry
      2. Free the inode
      3. Free the disk blocks used by the file
    - What will happen if the kernel free the inode before removing the directory entry?
    - In traditional file systems, such ordering is achieved through synchronous writes -- serious performance penalty.

LFS (2)

- Operations performed by fsck
  1. Read and check all inodes and build a bitmap of used blocks
  2. Record inode numbers and block addresses of all directories.
  3. Validate the structure of the directory tree, making sure that all links are accounted for.
  4. Validate directory contents to account for all the files.
  5. If any directories could not be attached to the tree in phase 2, put them in the lost+found directory.
  6. If any file could not be attached to a directory, put in the lost+found dirctory.
  7. Check the bitmaps and summary counts for each cylinder group.

LFS (3)

- Basic concepts
  - Record all file system changes in an append-only log file.
  - The log is written sequentially, in large chunks at a time.
    - Results in efficient disk utilization and high performance .
  - After a crash, only the tail of the log needs to be examined.
    - Results in quick recovery and high reliability.
- Design considerations
  - What to log
  - Operations or values

LFS (4)

- Design considerations
  - What to log
  - Operations or values
  - Redo and undo logs
  - Gargabe collection
  - Group commit
  - Retrieval

Disk Scheduling (1)

- In timesharing systems, it may sometimes be the case that there are several disk I/O's requested at the same time.
- Disk scheduling strategies:
  - First com first served (FIFO, FCFS): May result in a lot of unnecessary disk arm motion under heavy loads.

Disk Scheduling (2)

- Shortest seek time first (SSTF): Handle nearest request first.

  This can reduce arm movement and result in greater overall disk efficiency, but some requests may have to wait a long time.
- Scan: Like an elevator. Move arm back and forth, handling requests as they are passed. This algorithm doesn't get hung up in any place for very long. It works well under heavy load, but not as well in the middle (about ½ the time it won't get the shortest seek.)

Disk Scheduling (3)

- Example:

  Requested cylinder numbers 0, 53, 14, 27, 2, 31, 85, 30 when head is at 1 initially.

  – FCFS: 0, 53, 14, 27, 2, 31, 85, 30

    Total seek distance: 269

  – SSTF: 0, 2, 14, 27, 30, 31, 53, 85

    Total seek distance: 86

  – Scan: 2, 14, 27, 30, 31, 53, 85, 0

    Total seek distance: 169

Disk Scheduling (4)

- Many more possible scheduling algorithms.
- Newer disks require looking at rotational latencies as well.
- Most of the time there are not very many disk requests in the queue, so this is not a terribly important decision.