UNIVERSITY of WISCONSIN-MADISON
Computer Sciences Department

CS 537
Intro to Operating Systems

A. Arpaci-Dusseau
Spring 2000

Fast File System for UNIX

Questions answered in these notes

- What were the primary performance problems with the UNIX FS?
- How does FFS minimize internal fragmentation?
- How does FFS organize its freelist?
- How does FFS allocate i-node and data blocks for locality?

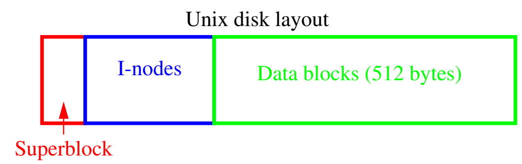
Reading

- "A Fast File System for UNIX" by McKusick, Joy, Leffler, and Fabry

Motivation

Original UNIX File system from Bell Labs

- Simple and elegant
- Problem: Achieves 20 Kb/sec
  - 2% of disk maximum even for sequential disk transfers!



Why such poor performance?

- Three primary reasons...

Why such poor performance?

Blocks too small (512 bytes)

- Fixed costs per transfer
  - Seek time, rotational delay, computation
- More indirect blocks needed for same size file

Poor freelist organization

- Consecutive file blocks not close together
- Pay seek cost between even sequential disk transfers

No locality in allocation to disk

- I-nodes far from data blocks
  - Pay two seeks for every data transfer
- I-nodes of files in directory not close together
  - Pay seek for every i-node (e.g., ls -l)

#1: Larger Block Sizes

Measure FS performance on workload given different block sizes

Block size	Space wasted	Bandwidth
512 bytes	6.9 %	2.6 %
1024 bytes	11.8 %	3.3 %
2048 bytes	22.4 %	6.4 %
4096 bytes	45.6 %	12.0 %
1 MB	99.0 %	97.2 %

BSD: Increase block to 4096 or 8192 bytes

- What is the problem with larger blocks?
- What is the solution?

Solution to Internal Fragmentation

Fragments: Allow large blocks to be chopped into small ones

- Lower bound on size determined disk sector
- Limit number of fragments per block to 2, 4, or 8
- Keep track of free fragments

Beneficial for small files and ends of files

Algorithm for ensuring fragments only used for end of file

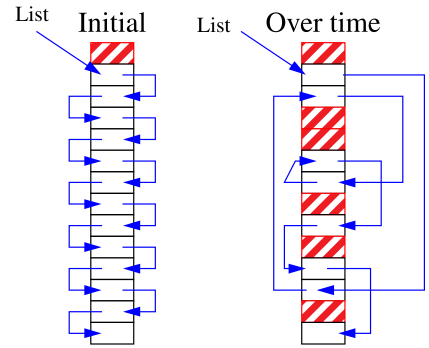
- Only allocate fragments from one block per file
- Coallesce blocks of allocated fragments
- Performance problem if file grows a fragment at a time

Advantages

- Greatly reduces amount of wasted space
- Transfer speeds of larger blocks

#2: Unorganized Freelist

Leads to random allocation of sequential files over time



- Initial performance good
  - ...but FS are long-lived entities

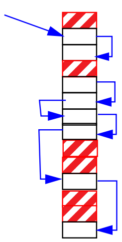
What are possible solutions?

Fixing the Unorganized Freelist

Periodically compact / defragment disk

- Disadvanatage: Disk not accessible during operation

Organize freelist by address



- Disadvantage: Costly to find set of contiguous free blocks

Bitmap of free blocks

Bitmap: 10010000110110101111

- Solution used in BSD

#3: Locality

Techniques for keeping related items together

- Keep freespace on disk
  - Always find free block nearby
  - 90% rule of thumb
- Spread unrelated data far apart
  - Leaves room for related things to be placed together

What new organization to support locality did BSD introduce?

Solution: Cylinder Groups

Divide disk into *cylinder groups*

- Set of adjacent cylinders
- Little **seek** time between cylinders in same group

Each cylinder groups contains:

- Superblock
  - Vary offset within each cylinder group for reliability
- I-nodes
  - Fixed number per cylinder group
- Bitmap of free blocks
- Usage summary for high-level allocation policy
- Data blocks

Goals for Locality

Maintain locality of each file

Maintain locality of files and inodes in a directory

Make room for locality within a directory

- Two requirements

How does BSD achieve each of these goals?

- What heuristics does it use when allocating blocks to disk?

Solution to Achieving Locality

Maintain locality of each file

- Allocate runs of blocks within a cylinder group

Maintain locality of files and inodes in a directory

- Keep files in a directory in same cylinder group

Make room for locality within a directory

- Spread out directories among the cylinders groups
  - Greater than average # of free inodes, smallest # of directories
- Switch to a different cylinder group for large files
  - After 48KB and every 1MB thereafter
  - Prevent one file from filling a cylinder group

Layout: Global vs. Local

Decompose allocation into two steps

Global: Heuristics for allocate files+directories to cylinder groups

- Pick “optimal” next block for allocation

Local: Handles request for specific block

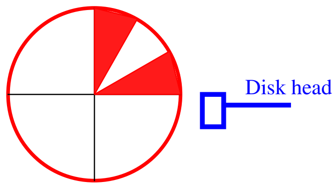
- If block available, use it
- If not free, check a sequence of alternatives

  1) Next rotational block on same cylinder
  2) A block within cylinder group
  3) Rehash on cylinder group to choose another group
  4) Exhaustive search

Rotationally Optimal Placment

Skip-sector allocation

- Based on CPU and device speed
- Do not allocate contiguous sectors if CPU not fast enough



- Problems
  - Cannot achieve full bandwidth from disk
  - Timing may be optimal for reads but not writes

BSD Performance Improvements

Achieve 20-40% of disk bandwidth on large files

- 10x improvement over original Unix file system
- Does not change over lifetime of FS
- Especially good considering skip-sector allocation
  - Could not achieve bettern than 50% of peak

Better small file performance

Other Enhancements

Long file names

File locking

- Old: Create separate lock file; Cleanup if process dies
- New: Lock operations for advisory locking

Symbolic links (in addition to hard links)

- Links across file systems
- Links to directories

Atomic rename capability

- Old: `rm name; ln name newName; rm newName`

Disk quotas